Enterprise Architect

User Guide Series

# Parametric Simulation using OpenModelica

Author: Sparx Systems
Date: 15/07/2016
Version: 1.0

# Table of Contents

# 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 document 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 and can be solved 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 Reference

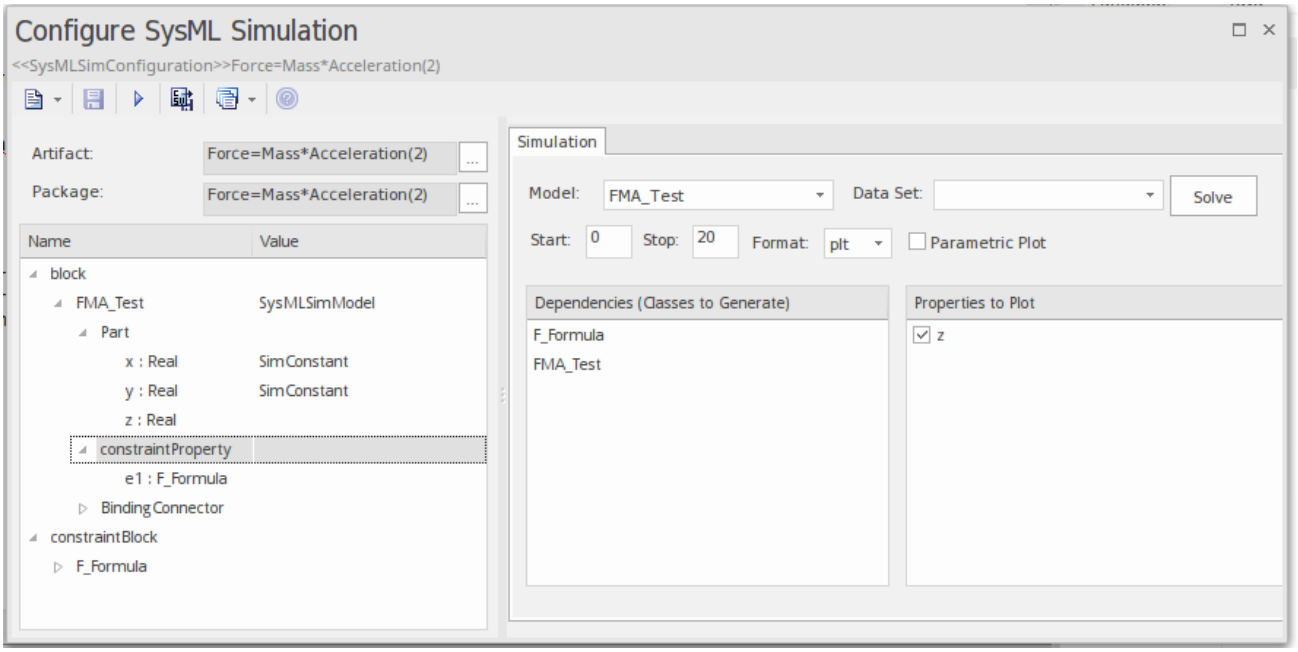
The user interface for the SysML simulation is described in the *SysML Simulation Configuration* topic.

## OpenModelica Example

To aid your understanding of how to create and simulate a SysML parametric model, three examples have been provided to cover three different domains. These examples and what you are able to learn from them are described in the *SysML Simulation Example*s topic.

# 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.

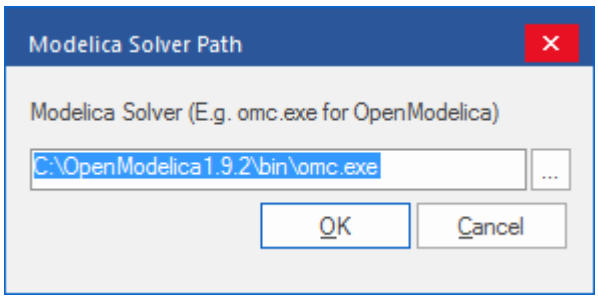



## Access

| | |
|---|---|
| Ribbon | Simulate > SysMLSim > Manage > **SysMLSim Configuration Manager** |
| Other | Double click on an Artifact with the SysMLSimConfiguration stereotype. |

## Toolbar Options

| Option | Description |
|---|---|
|  | Click on the drop-down arrow and select from these options:<br>• Select Artifact - Select and load an existing configuration from an Artifact with the SysMLSimConfiguration stereotype (if one has not already been selected)<br>• Create Artifact - Create a new SysMLSimConfiguration or select and load an existing configuration artifact<br>• Select Package - Select a Package to scan for SysML elements to configure for simulation<br>• Reload - Reload the Configuration Manager with changes to the current Package<br>• Configure Modelica Solver - Display the 'Modelica Solver Path' dialog, in |

| | |
|---|---|
| | which you type or browse for the path to the Modelica solver to use<br>  |
|  | Click on this button to save the configuration to the current Artifact. |
|  | Click on this button to generate, compile and run the current configuration, and display the results. |
|  | After simulation, the result file is generated in either plt, mat or csv format. That is, with the filename:<br>• ModelName_res.plt<br>• ModelName_res.mat or<br>• ModelName_res.csv<br><br>Click on this button to specify a directory into which Enterprise Architect will copy the result file. |
|  | Click on this button to select from these options:<br>• Generate Modelica Code - Generate the code without compiling or running it<br>• Open Modelica Simulation Directory - Open the directory into which Modelica code will be generated<br>• Edit Modelica Templates - Customize the code generated for Modelica, using the **Code Template Editor** |

## Simulation Artifact and Model Selection

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

## Package Elements

This table discusses the element types from the SysML model that will be listed under the 'Name' column in the Configure SysML Simulation window, to be processed in the simulation.

| Element Type | Behavior |
|---|---|
| ValueType | ValueType elements either generalize from a primitive type or are substituted by SysMLSimReal for simulation. |
| Block | Block elements mapped to SysMLSimClass or SysMLSimModel elements support the creation of data set(s). In the case of multiple data sets, a SysMLSimClass must define one of them as the default; a SysMLSimModel is a possible top level element for a simulation, and a data set is chosen on simulation. |
| Properties | Properties within a block can be configured to be either SimConstants or SimVariables. For a SimVariable, you configure these attributes:<br>• isContinuous - determines whether the property value varies continuously ('**True**', the default) or discretely ('**False**')<br>• 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<br>• changeCycle - specifies the time interval at which a discrete property value changes; the default value is '0'<br>- changeCycle can be set to a value other than 0 only when isContinuous = 'False'<br>- The value of changeCycle must be positive or equal to 0 |
| Port | No configuration required. |
| SimFunction | Functions are created as operations in Blocks or Constraint Blocks, stereotyped as 'SimFunction'.<br>No configuration is required in the Configure SysML Simulation window. |
| Generalization | No configuration required. |
| Binding Connector | Binds a property to a parameter of a constraint property.<br>No configuration required. |
| Connector | Connects two Ports.<br>No configuration required in the Configure SysML Simulation window. However, you might have to configure the properties of the Port's type by determining whether the attribute 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 window.

| Field | Meaning |
|---|---|
| Model | Click on the drop-down arrow and select the top level type for the simulation. The list is pre-populated with the blocks marked as a SysMLSimModel. |
| 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 | • Select this checkbox to plot Legend A on the y-axis against Legend B on the x-axis.<br>• Deselect the checkbox to plot Legend(s) on the y-axis against time on the x-axis<br><br>Note: With the checkbox selected, you must select two properties to plot. |
| 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, from which you select those to plot. |

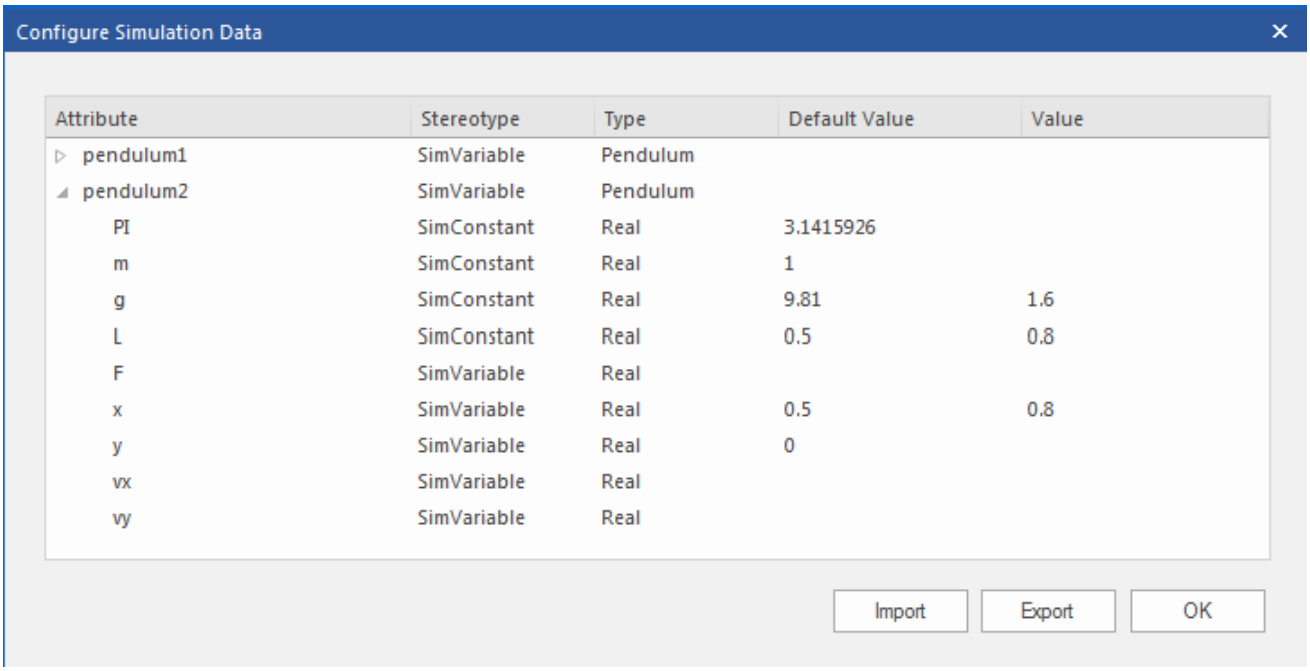
# Model Analysis using Datasets

Every SysML block used in a Parametric model can have multiple datasets defined against them. This allows for repeatable simulation variations using the same SysML model. When running a simulation, a user is able to select from the datasets specified against the model before starting.

## Dataset Management

| | |
|---|---|
| Create | New Datasets can be created by right clicking on a block selecting 'Create Simulation Dataset'. |
| Duplicate | To duplicate an existing dataset as a base for creating a new dataset, right click on the dataset and select 'Duplicate'. |
| Delete | To remove a dataset that is no longer required, right click on the dataset and select 'Delete Dataset'. |
| Set Default | The default dataset used by a SysMLSimClass when used as a property type or inherited can be set by right clicking on the dataset and selecting 'Set as Default'. The properties used by a model will use this default configuration unless the model overrides them explicitly. |

## Configure Simulation Data




| | |
|---|---|
| Attribute | The Attribute column provides a tree view of all the properties in the Block being edited. |

| | |
|---|---|
| Stereotype | The stereotype column specifies 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 (eg. 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 the user to override the default value for each primitive value. |
| Export / Import | The export and import buttons allow the user to modify the values in the current dataset using an external application such as a spreadsheet before re-importing them. |

# SysML Simulation Examples

This section provides three worked examples of how to create a SysML model for a domain, simulate it and evaluate the results of the simulation. These examples apply the information discussed in the earlier topics.

## Electrical Circuit Simulation Example

The first example, is for simulation of an electrical circuit. The example starts with an electrical circuit diagram, 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.

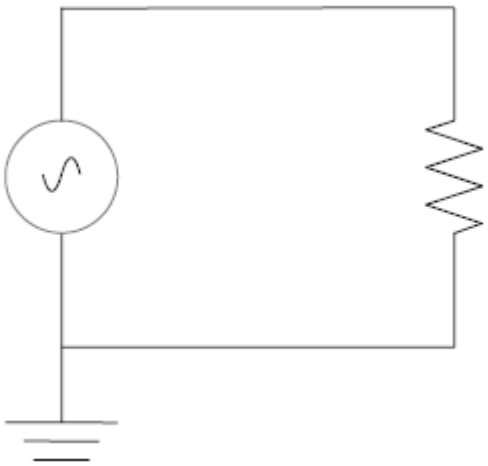
Water Tank Pressure Regulator

# 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

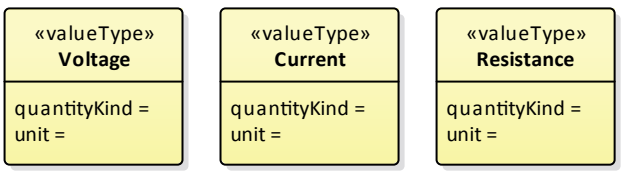
The electrical circuit we are going to model is shown below using a standard electrical circuit notation.

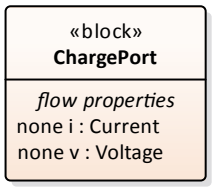
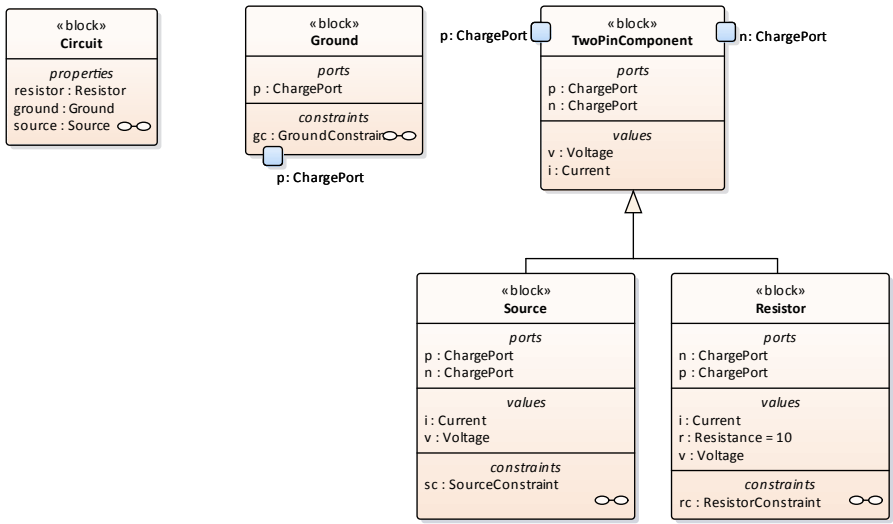


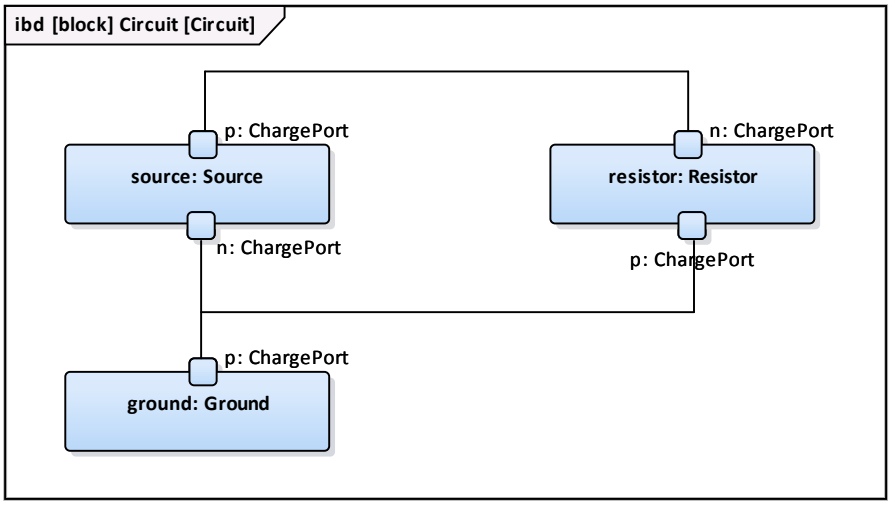
The circuit includes an AC power source, a ground and a resistor. Each of these being connected via wires.

## Create SysML Model

The following 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.

| | |
|---|---|
| Types | Define Value Types for each of Voltage, Current and Resistance. Unit and quantity kind are not important for the purpose 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.<br><br>  <br><br>Additionally, we 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. |

| | |
|---|---|
| |  |
| Blocks | 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 charges) is transmitted through the pins. Create an abstract block TwoPinComponent with two ports (pins). The two ports are named p and n, and they are of type ChargePort.<br><br>The following figure shows what the BDD should look like, with the blocks Circuit, Ground, TwoPinComponent, Source and Resistor.<br><br>  |
| 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. |

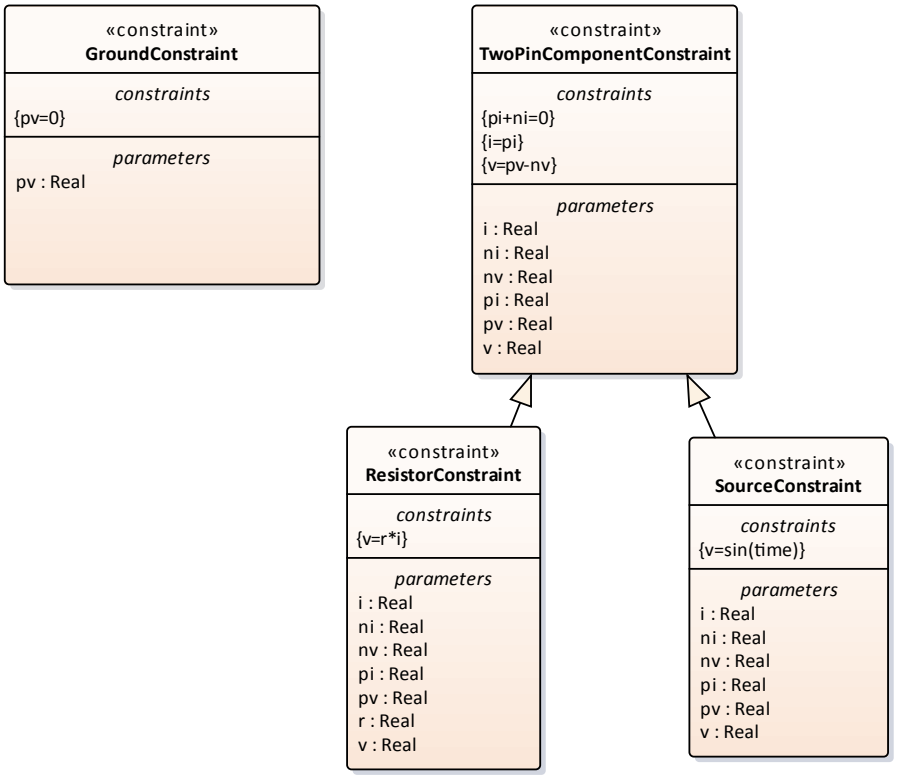



Notice that this follows the same structure as the original circuit diagram, but the symbols for each component have been replaced with properties typed by the blocks defined above.

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 SimConstant 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).

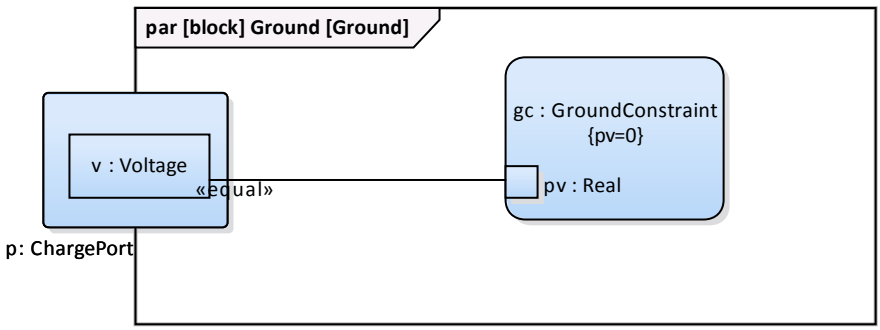
Create an constraint block TwoPinComponentConstraint to define parameters and equations common to sources and resistors. The equations should state that the voltage of the component is equal to the difference between the voltage at the positive and negative pin. The current of the component is equal to the current going through the positive pin. The sum of the current 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 parameter. The following figure shows what these constraints should look like in a BDD.



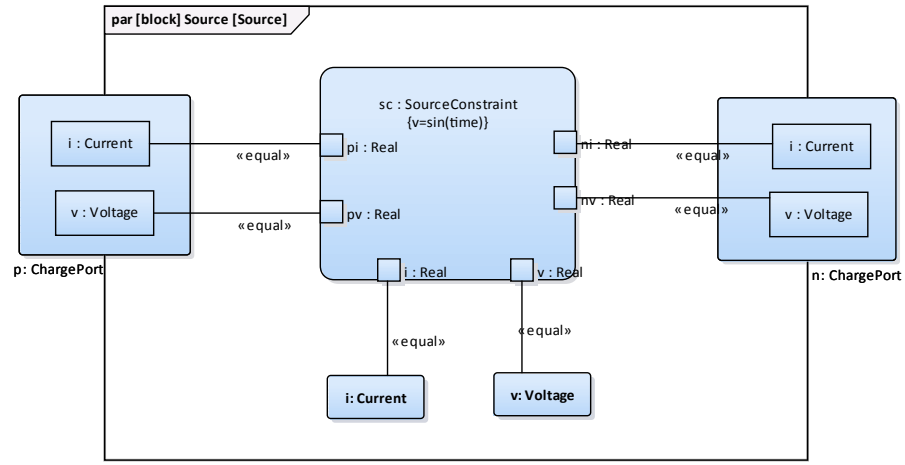

Bindings

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. block. The following figures show the bindings for the ground, the source, and the resistor respectively.

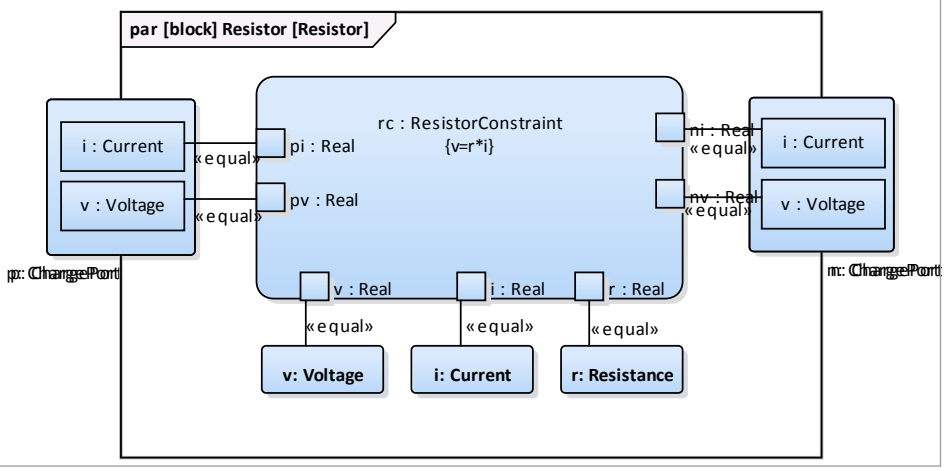
For the ground constraint, bind gc.pv to p.v.




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

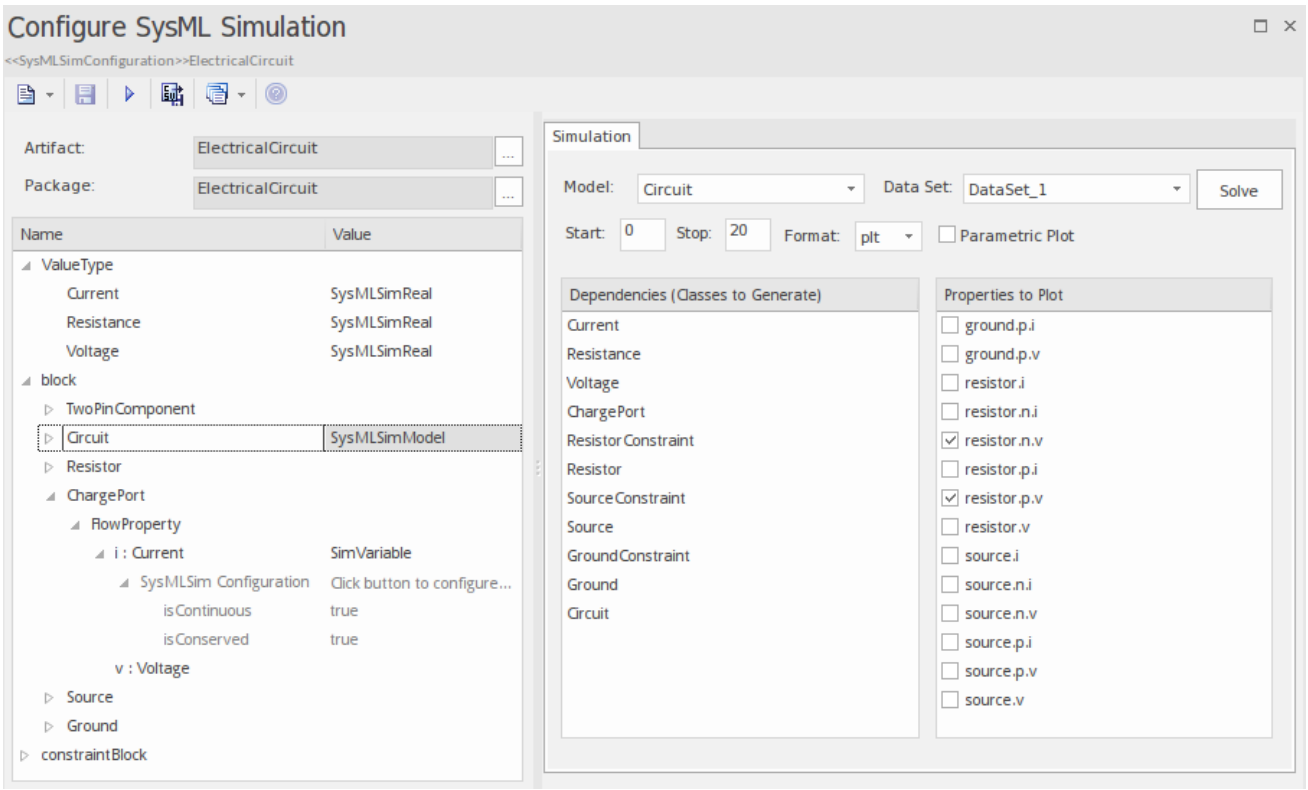
The following table shows the detailed steps of the configuration of SysMLSim.

| | |
|---|---|
| SysMLSimConfiguration Artifact | • Simulate > SysMLSim > Manage > **SysMLSim Configuration Manager**<br>• From the toolbar, Create Artifact<br>• Select Package that owns this SysML Model |
| Root elements in Configuration Manager | • ValueType<br>• block<br>• constraintBlock |
| ValueType Substitution | • Expand to ValueType \| Current, select "SysMLSimReal" from the combo box<br>• Expand to ValueType \| Resistence, select "SysMLSimReal" from the combo box<br>• Expand to ValueType \| Voltage, select "SysMLSimReal" from the combo box |
| | |

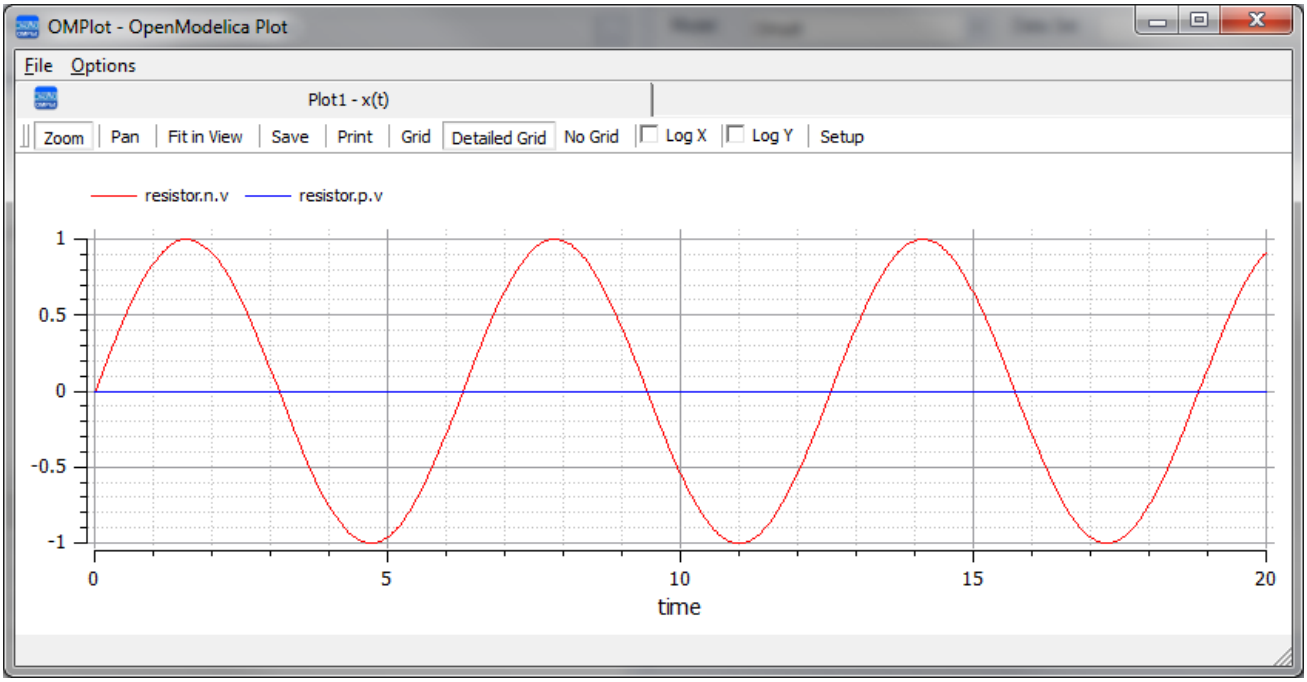
| | |
|---|---|
| Set property as flow | • Expand to block \| ChargePort \| FlowProperty \| i : Current, Select "SimVariable" from the combo box<br>• Click the "..." button to open the "Element Configurations" dialog<br>• Set "isConserved" to "true" |
| SysMLSimModel | This is the model we want to simulate: Set block "Circuit" to be SysMLSimModel |

## Run Simulation

In the Simulation page, configure resistor.n.v and resistor.p.v for plot and click the "Solve" button




Two legends resistor.n.v and resistor.p.v are plotted as the following figure:

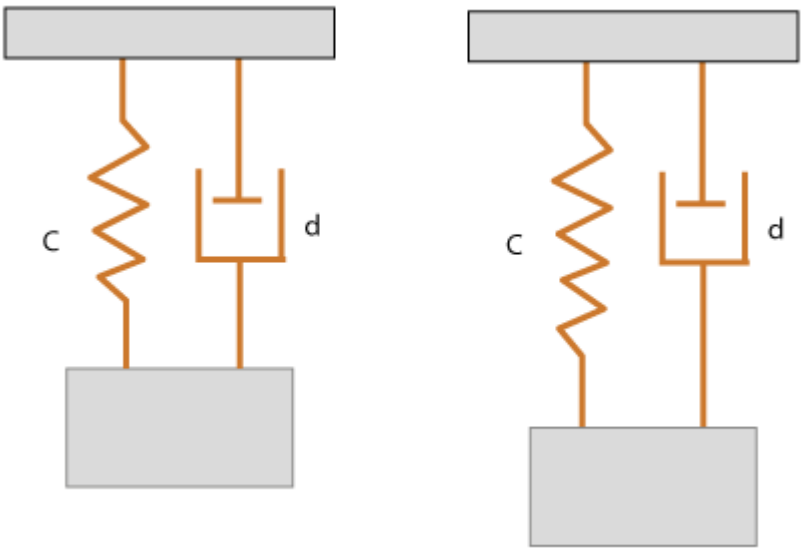

OMPlot - OpenModelica Plot
File Options
Plot1 - x(t)
Zoom Pan Fit in View Save Print Grid Detailed Grid No Grid Log X Log Y Setup
resistor.n.v
resistor.p.v
1
0.5
0
-0.5
-1
0
5
10
15
20
time

# 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 mass-spring-damper and then use a parametric simulation to predict and chart the behavior of this mechanical system. Finally, we do some what-if analysis by compare two oscillators provided with different parameter values through data sets.

## System being modeled

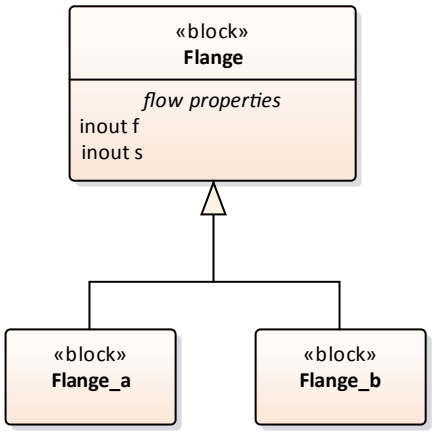
A mass body hanging on a spring damper. The first state represents the initial point at time = 0 just when the body is released. The second state represents the final state when the body is at rest and the spring forces are in equilibrium with the gravity force.

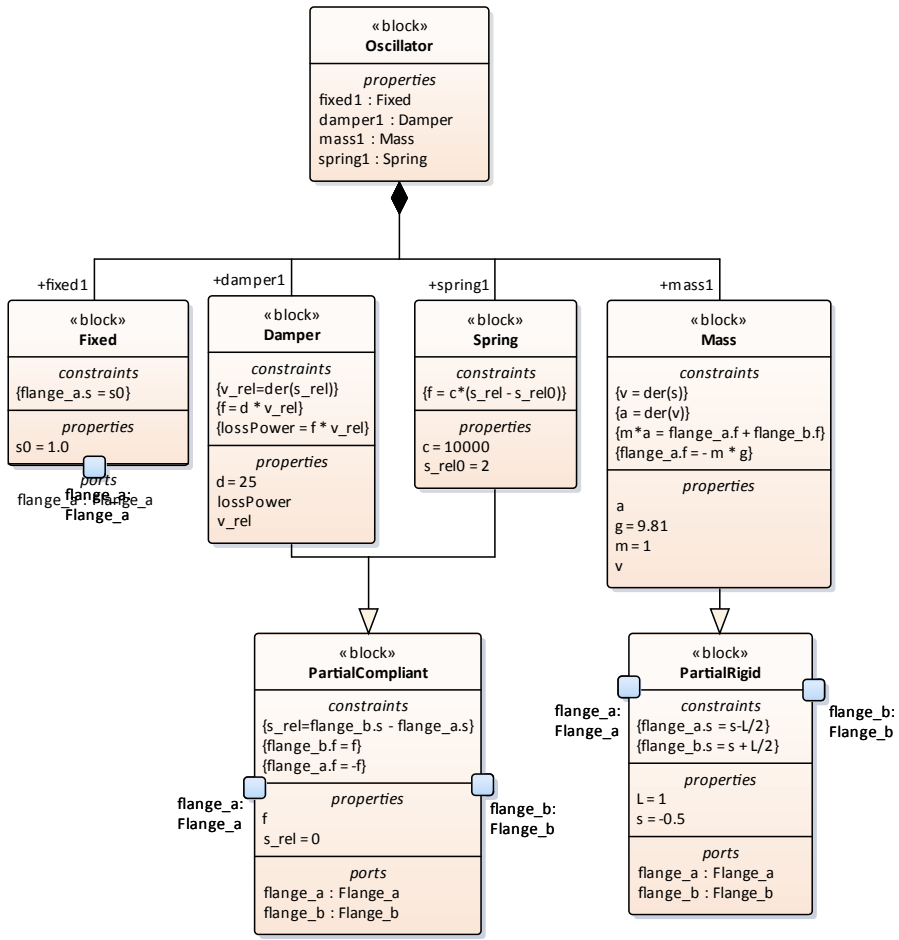



## Create SysML Model

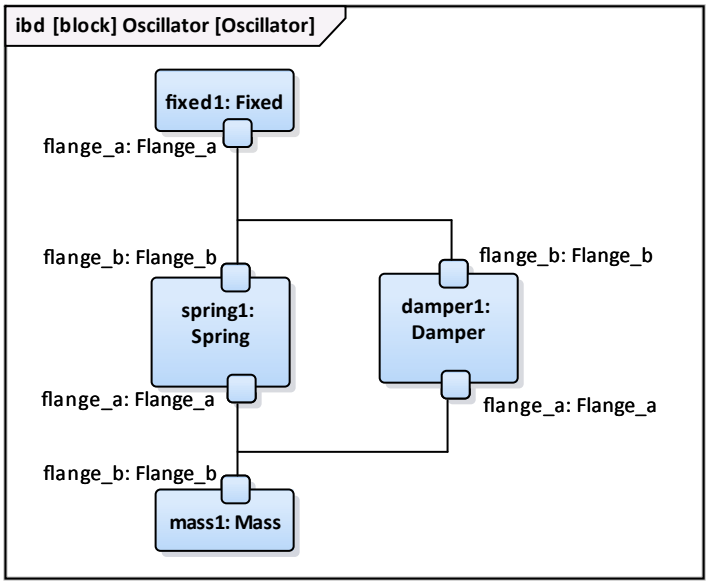
The MassSpringDamperOscillator model in SysML has a main block, the *Oscillator*. The oscillator has four parts: a fixed *ceiling*, a *spring*, a *damper* and a *mass body*. Create a block for each of these part types. The four parts of the Oscillator block are connected through ports, which represent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

| | |
|---|---|
| Port Types | The block Flange_a and Flange_b used for flanges in the 1D translational 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 block Spring, Damper, Mass, Fixed to represent spring, damper, mass body and the ceiling respectively.<br>• Create an block PartialCompliant with two ports (flanges), named flange_a and flange_b, and they are of type Flange_a and Flange_b respectively. The Spring and Damper blocks generalize from PartialCompilant.<br>• Create an block "PartialRigid" with two ports (flanges), named flange_a and flange_b, and they are of type Flange_a and Flange_b respectively. The Mass blocks generalize from PartialRigid.<br>• Create a "Fixed" block with only one flange for the ceiling, which only have 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.<br>• The flang_a of fixed1 connect to flange_b of sprint1<br>• The flange_b of damper1 connect to flange_b of spring1<br>• The flange_a of damper1 connect to flange_a of spring1<br>• The flange_a of spring1 connect to flange_b of mass1 |

| | |
|---|---|
| |  |
| Constraints | For simplicity, we defined the constraints directly in the block elements; optionally, you can define constraint blocks and use constraint properties in the blocks and bind their parameters to block's properties. |

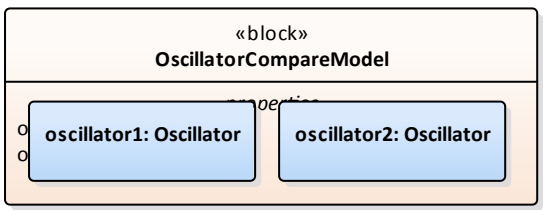
## Two Oscillator Compare Plan

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

- What's the difference between two oscillators whose dampers are different?
- What if there is no damper?
- What's the difference between two oscillators whose springs are different?
- What's the difference between two oscillators whose mass are different?

Here are the steps of creating a compare model:

- Craete a block named "OscillatorCompareModel"
- Create two properties for "OscillatorCompareModel", give them name *oscillator1* and *oscillator2,* type them with Block *Oscillator*




## Setup DataSet and Run Simulation

Create a SysMLSim Configuration artifact and setup to this package. Then create the following data sets:

- Damper: small VS big

provide "oscillator1.damper1.d" with smaller value 10 and "oscillator1.damper2.d" with bigger value 20

- Damper: no vs yes

provide "oscillator1.damper1.d" with value 0; ("oscillator2.damper2.d" will use the default value 25)

- Spring: small VS big

provide "oscillator1.spring1.c" with smaller value 6000 and "oscillator1.spring1.c" with bigger value 12000

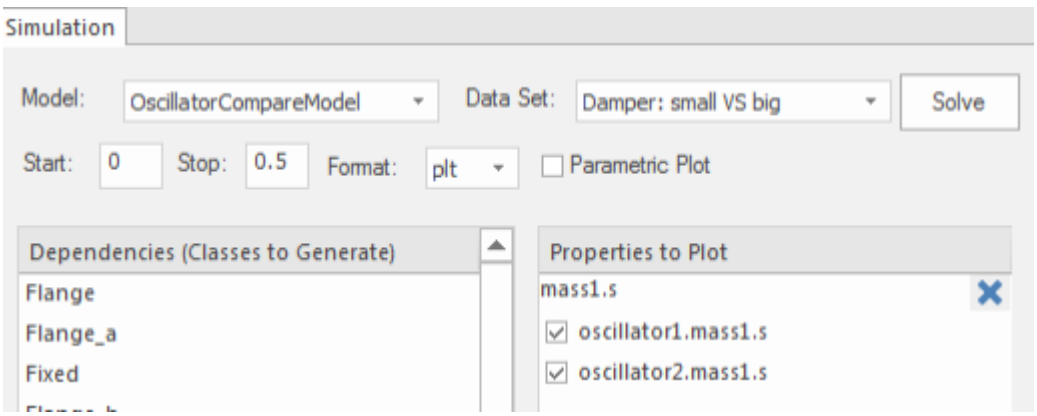
- Mass: light VS heavy

provide "oscillator1.mass1.m" with smaller value 0.5 and "oscillator1.mass1.m" with bigger value 2

**The configured page looks like this:**

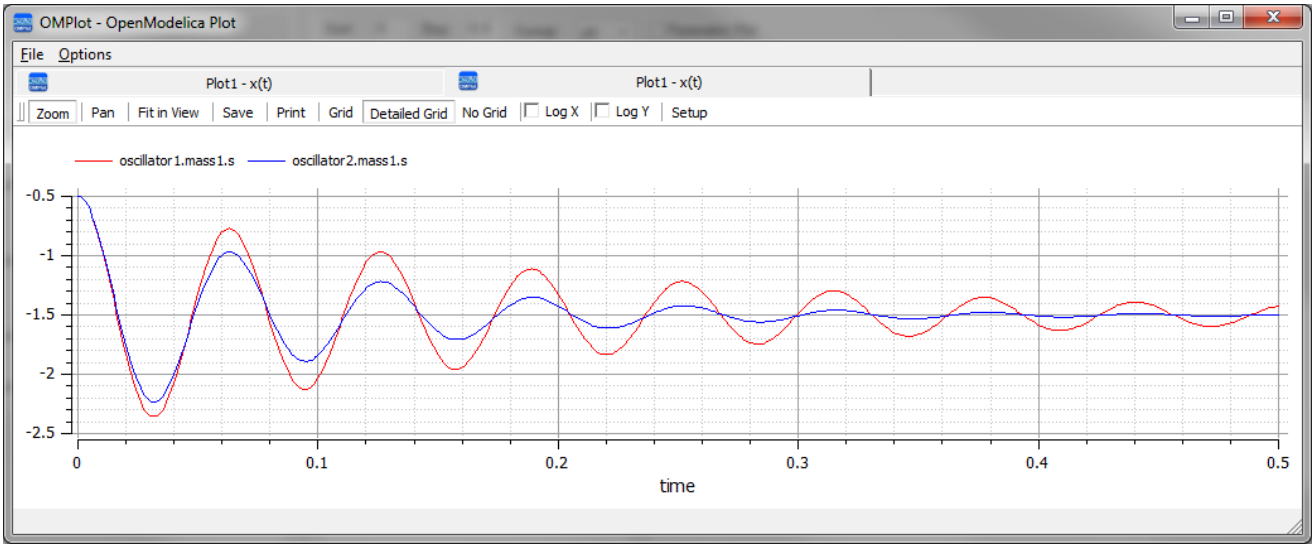

At the simulation page, select "OscillatorCompareModel", plot for "oscillator1.mass1.s" and "oscillator2.mass1.s" then choose one of the created datasets and simulate:

*Tips: if there are too many properties in the plot list, you can toggle the filter bar by context menu on the header of the list, then input "mass1.s" in this example.*

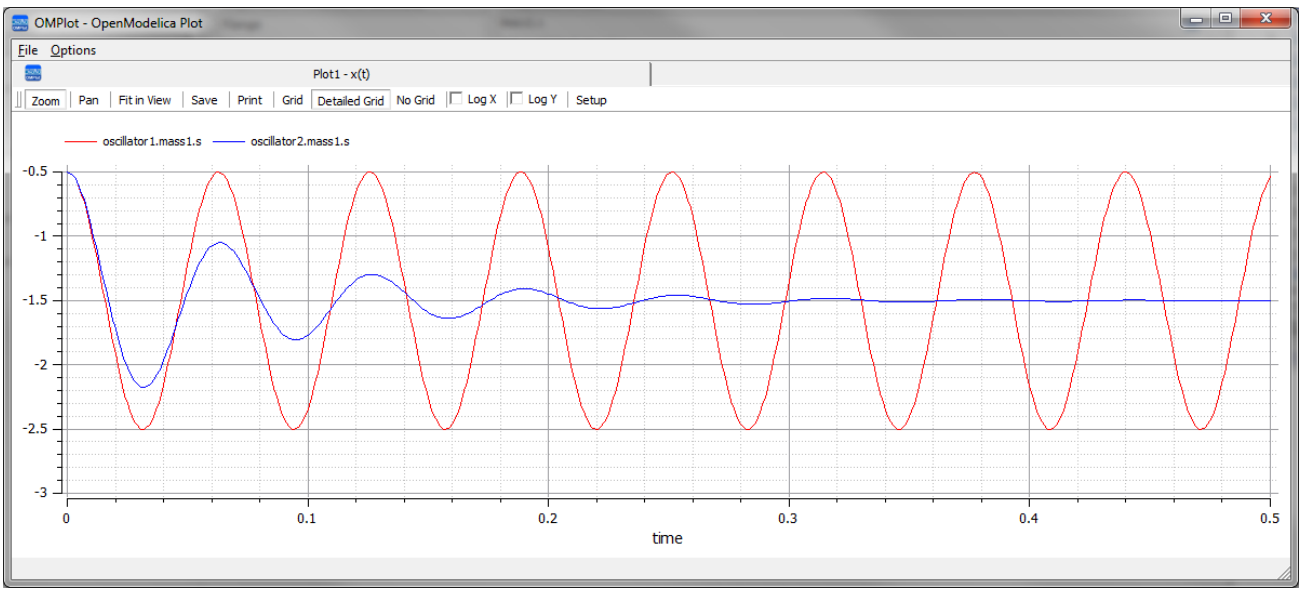



**Here is the simulation result:**

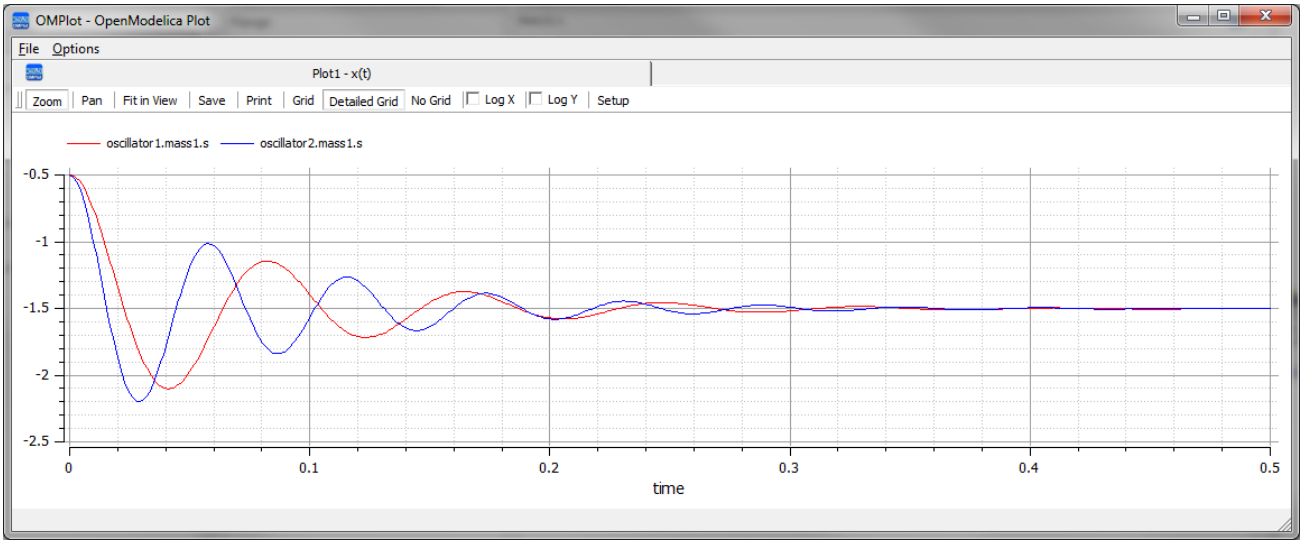
- Damper: small VS big: The smaller damper make the body oscillate more.



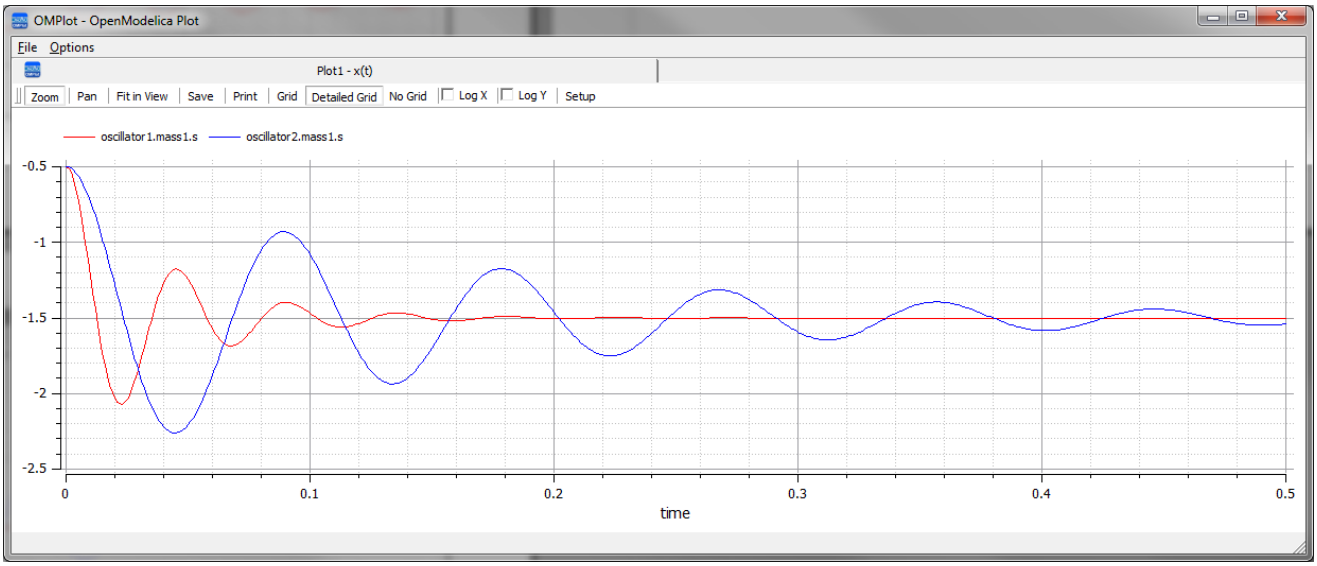

- Damper: no vs yes : the oscillator never stops without a damper




- Spring: small VS big: the spring with smaller "c" will oscillate slower




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



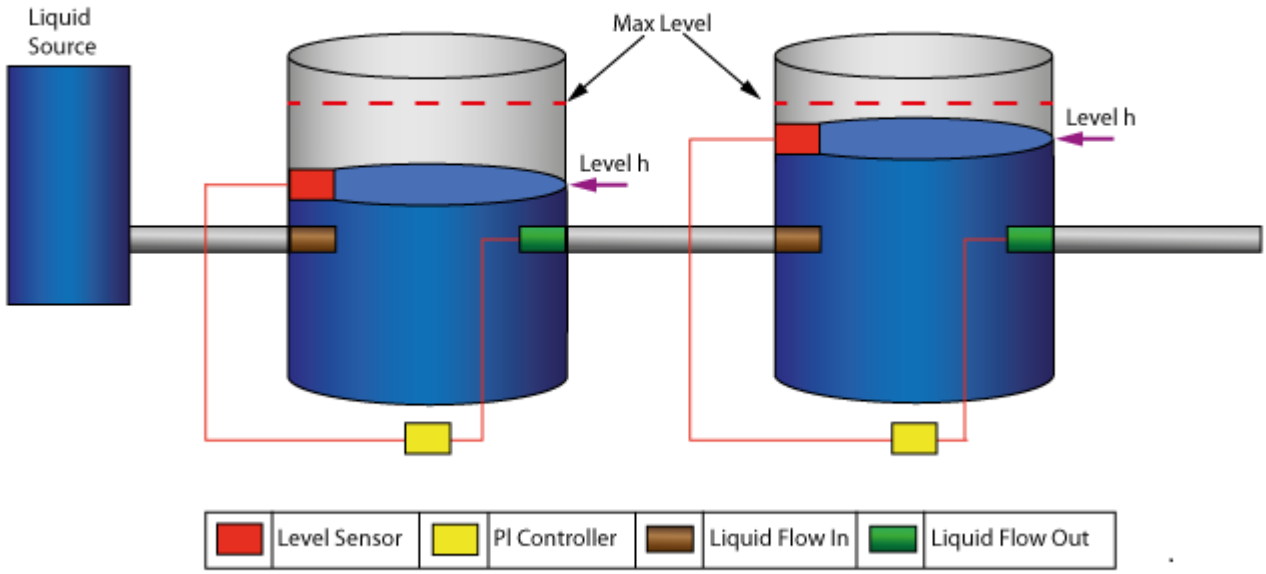
# Water Tank Pressure Regulator

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 monitor the water level and control the valve to regulate the system.

We will get the model explained, create the SysML model and setup the SysMLSim Configurations. Then run the simulation with OpenModelica.

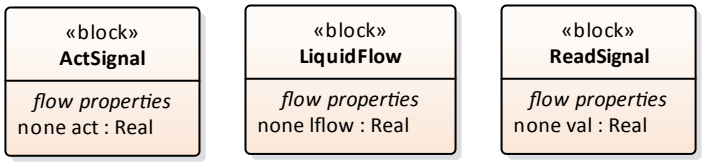
## System being modeled

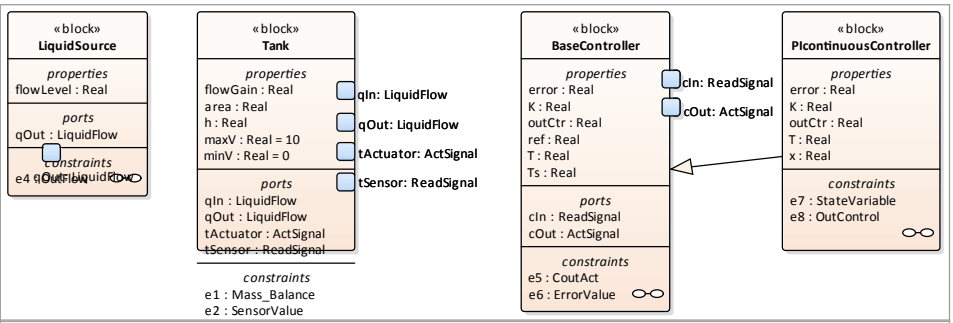
The following figure represents two tanks connected together, and a source water will fills the first tank. Each tank has a continuous proportional–integral (PI) controller connected to it, which regulates the level of water contained in the tanks to a reference level. While the source fills the first 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

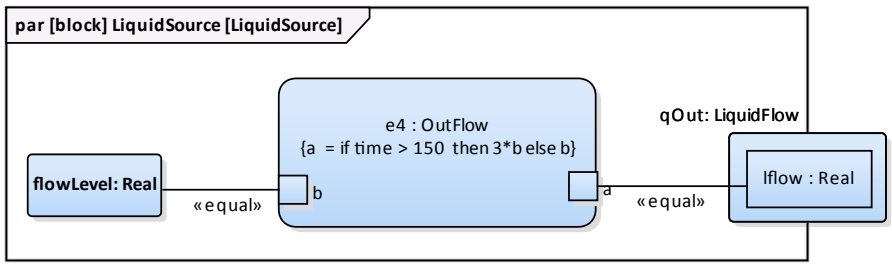
| | |
|---|---|
| Port Types | The tank has four ports, they are typed to the following three blocks:<br>• ReadSignal: Reading fluid level. It has a property val with unit "m"<br>• ActSignal: Signal to actuator for setting valve position<br>• LiquidFlow: Liquid flow at inlets or outlets. It has a property lflow with unit "m3/s" |

| | |
|---|---|
| |  |
| Block Definition Diagram | • LiquidSource: The water entering the tank must come from somewhere. Therefore, we have a liquid source component in the tank system. Property *flowLevel* has a unit of "m3/s"; A port *qOut* typed to LiquidFlow.<br>• Tank: Tanks are connected to controllers and liquid sources through ports.<br>- The Tank has four ports:<br>- qIn: for input flow<br>- qOut: for output flow<br>- tSensor: for providing fluid level measurements<br>- tActuator: for setting the position of the valve at the outlet of the tank<br>- Properties:<br>- area (unit="m2"): area of the tank, involved in the *mass balance* equation<br>- h (unit = "m"): water level, involved in the *mass balance* equation; its value is read by the sensor<br>- flowGain (unit = "m2/s" ): the output flow is related to the valve position by *flowGain*<br>- minV, maxV: Limits for output valve flow<br>• BaseController: This block could be super of a PI continuous Controller and PI Discrete Controller.<br>- Ports:<br>- cIn: Input sensor level<br>- cOut: Control to actuator<br>- Properties:<br>- Ts (unit = "s"): Time period between discrete samples (not used in this example)<br>- K: Gain factor<br>- T(unit = "s"): Time constant of controller<br>- ref: reference level<br>- error: difference between the reference level and the actual level of obtained from the sensor<br>- outCtr: control signal to the actuator for controlling the valve position<br>• PIcontinuousController: generalize from BaseController<br>- Properties:<br>- x: the controller state variable |




Constraint Blocks

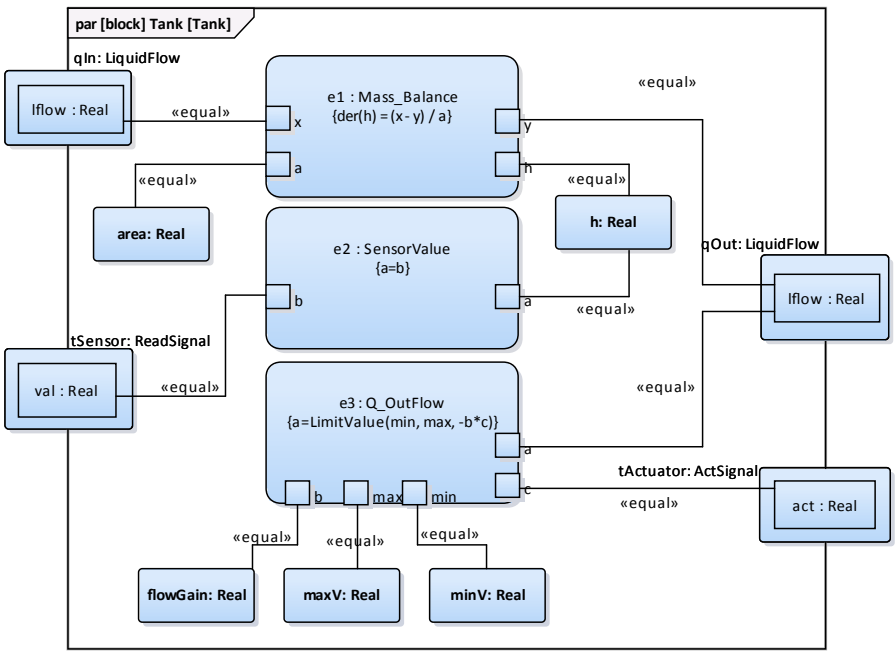
The flow increases sharply at time = 150 to factor of three of the previous flow level, which creates an interesting control problem that the controller of the tank has to handle.



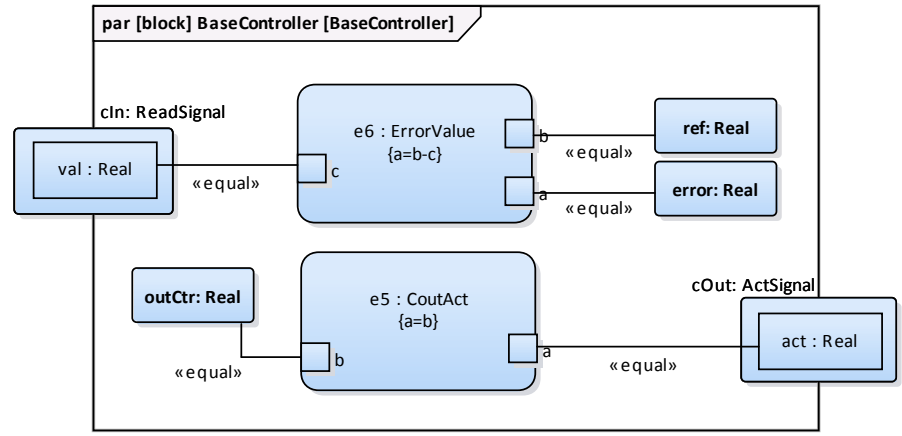

The central equation regulating the behavior of the tank is the *mass balance* equation.

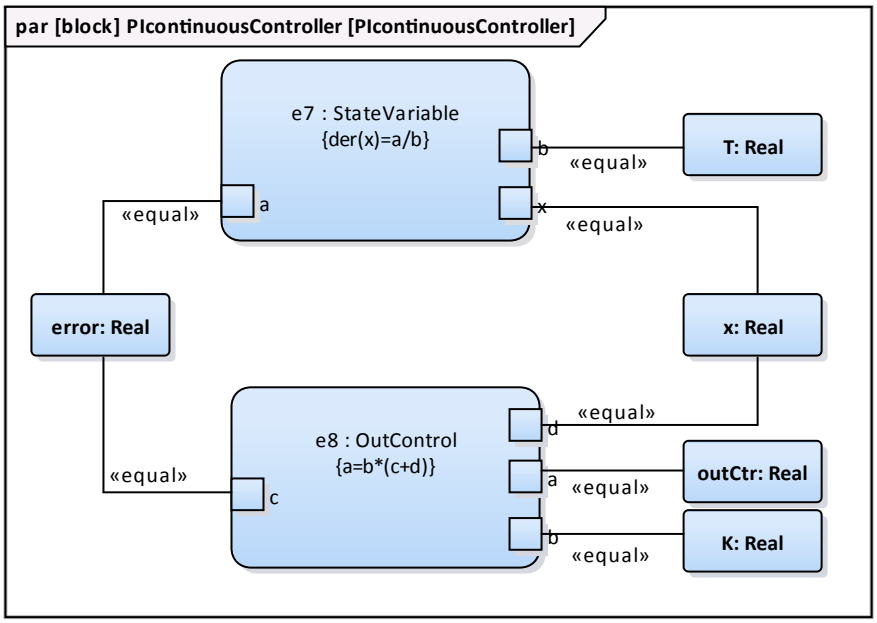
The output flow is related to the valve position by a flowGain parameter.

The sensor simply read the level of the tank.



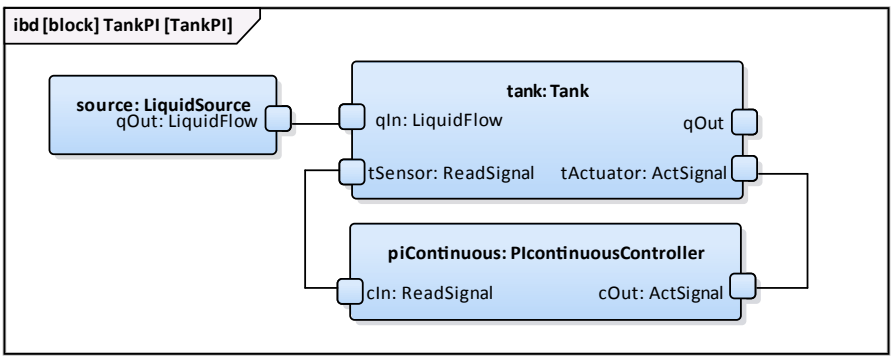

The Constrains defined for BaseController and PIcontinuousController are illustrated in the following figures.



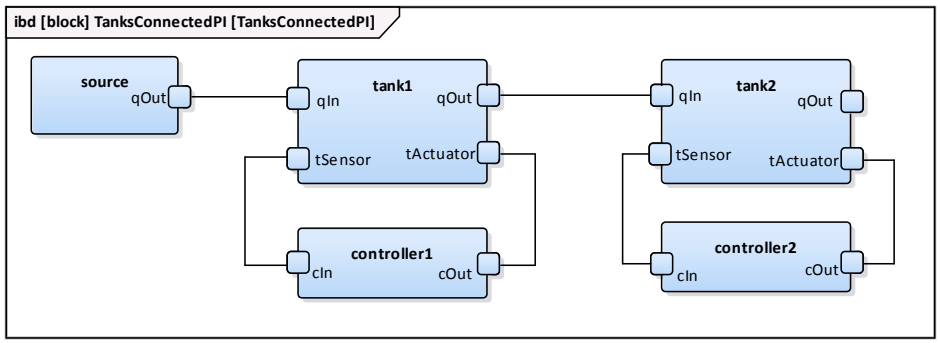




Internal Block Diagram

Here is the **Internal Block Diagram** for a system with a single tank.




Here is the Internal Block Diagram for a system with two connected tanks.




## Setup DataSet and Run Simulation

## Run Simulation

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

Select *TanksConnectedPI*, then the following GUI change will happen:

- Data Set Combobox: will be filled with all the data sets defined in *TanksConnectedPI*
- Dependencies list: will automatically collect all the blocks, constraints, SimFunctions, ValueTypes directly or indirectly referenced by *TanksConnectedPI*. (These elements will be generated as Modelica code.)
- Properties to Plot: A whole list of "leaf" variable properties(leaf means they don't have properties) will be collected. You can choose one or multiple to simulate, they will become legends of the plot.

## Create Artifact and Configure

- Ribbon | Simulate | Manage | **SysMLSim Configuration Manager**
- Toolbar |Element Menu | Create Artifact | Select the package containing this TwoTanks model

Then the elements in the package will be scanned loaded into the manager.

Configure the following blocks and their properties as shown in the following table.

Note: properties not configured as "SimConstant" are "SimVariable" by default

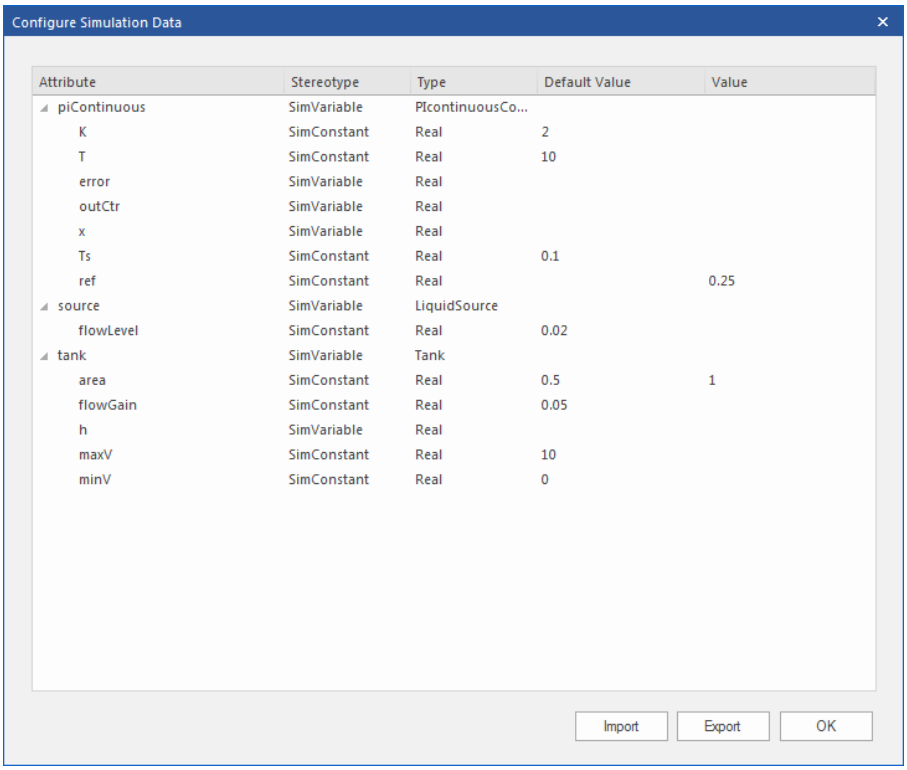
| | |
|---|---|
| LiquidSource | Configure as "SysMLSimClass"<br><br>Properties configuration:<br>• flowLevel: set as SimConstant |
| Tank | Configure as "SysMLSimClass" |

| | |
|---|---|
| | Properties configuration:<br>• area: set as SimConstant<br>• flowGain: set as SimConstant<br>• maxV: set as SimConstant<br>• minV: set as SimConstant |
| BaseController | Configure as "SysMLSimClass"<br><br>Properties configuration:<br>• K: set as SimConstant<br>• T: set as SimConstant<br>• Ts: set as SimConstant<br>• ref: set as SimConstant |
| PIcontinuousController | Configure as "SysMLSimClass" |
| TankPI | Configure as "SysMLSimModel" |
| TanksConnectedPI | Configure as "SysMLSimModel" |

## Setup DataSet

Context menu on the Element in the following table | Create Simulation DataSet | Then Click buttons to configure...

| | |
|---|---|
| LiquidSource | flowLevel: 0.02 |
| Tank | h.start: 0<br>flowGain: 0.05<br>area: 0.5<br>maxV: 10<br>minV: 0 |
| BaseController | T: 10<br>K: 2<br>Ts: 0.1 |
| PIcontinuousController | no configure needed;<br><br>By default, the specific block will use the configured value from super block's default DataSet. |
| TankPI | What's interesting here is that the default value could be loaded in the Configure Simulation Data Dialog.<br><br>For example, the values we configured as default DataSet on each block element were loaded as default values for the properties of TankPI, click the icon on each |

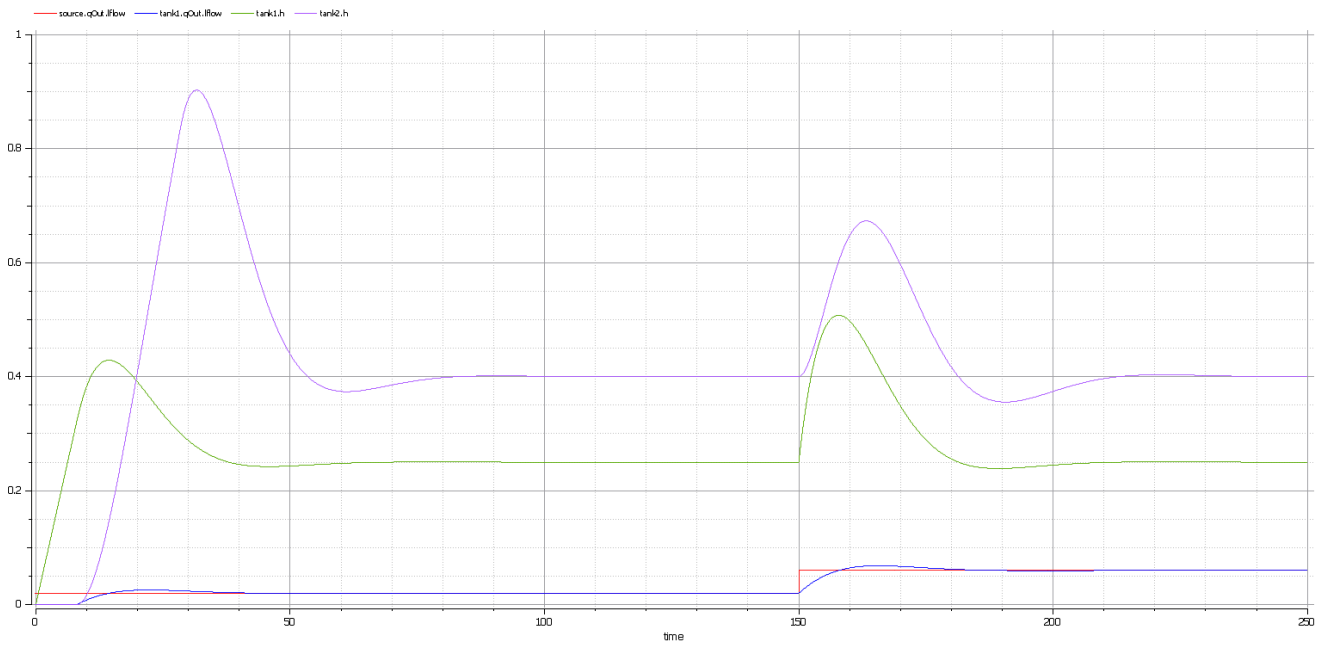
<table>
<tr><td></td><td>row will expand the property's internal structures to arbitrary depth.<br><br> 

<br><br>Click "OK" and return to the manager. Then following values are configured:<br><br>tank.area: 1 (This override the default value 0.5 defined in block Tank's data set )<br>piContinuous.ref: 0.25</td></tr>
<tr><td>TanksConnectedPI</td><td><ul><li>controller1.ref: 0.25</li><li>controller2.ref: 0.4</li></ul></td></tr>
</table>

## Simulation and Analysis 1

Select the following variables and click Solve, the following plot should prompt

- source.qOut.lflow
- tank1.qOut.lflow
- tank1.h
- ank2.h




**Here are the analysis from the result:**

- the liquid flow increases sharply at time=150 to 0.06 m3/s, factor of three of the previous flow level (0.02 m3/s).
- tank1 regulated at height 0.25 and tank2 regulated at height 0.4 as expected (we set the parameter value through data set)
- both tank1 and tank2 regulated twice during the simulation. First time regulated with the flow level (0.02 m3/s); second time regulated with the flow level (0.06 m3/s)
- tank2 was empty before flow came out from tank1

## Simulation and Analysis 2

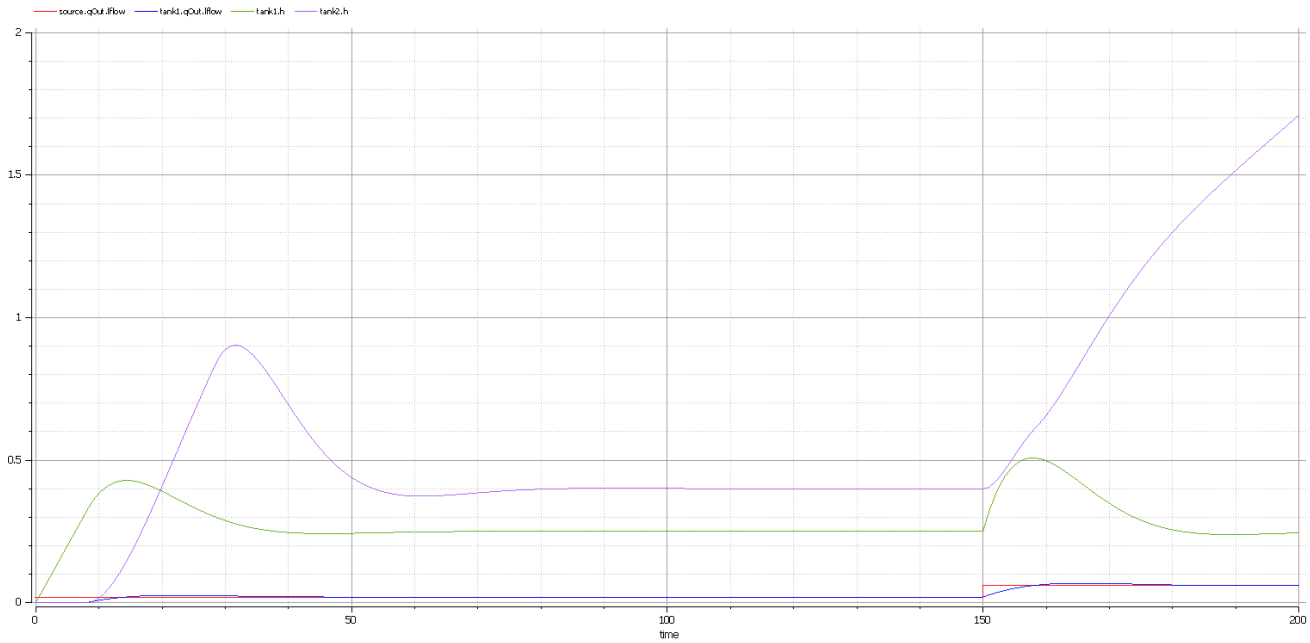
We have given the tank's property minV and maxV with value 0 and 10 in the above example.

In real world, flow speed of 10 m3/s may require a BIG valve installed on the tank.

What would happen if we change the value of maxV to 0.05 m3/s ?

Based on the previous model, we make following changes:

- On the existing "DataSet_1" of TanksConnectedPI | Context menu | Duplicate DataSet | Re-name to "Tank2WithLimitValveSize"
- Click button to configure | Expand tank2 | put value 0.05 in the Value column for property maxV
- Select "Tank2WithLimitValveSize" in the simulation page and plot for the properties
- Click Solve to simulate




**Here are the analysis from the result:**

- Our change only apply to tank2, tank1 can regulate as before on 0.02 m3/s and 0.06 m3/s
- When the source flow is 0.02 m3/s, tank2 can regulate as before
- However, when the source flow increased to 0.06 m3/s, its valve is too small to let the out flow catch up the in flow; the only result is: the water level of tank2 increase.
- Then it's up to the user to fix this problem. For example: change to a larger valve, reduce the source flow, make an extra valve....

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

# Creating a Parametric Model

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:

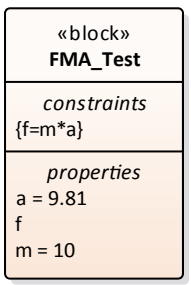
- Flows in physical interactions
- Default Values and Initial Values
- Simulation Functions
- Value Allocation, and
- Packages and Imports

## Access

| | |
|---|---|
| Ribbon | Simulate > SysMLSim > Manage > **SysMLSim Configuration Manager** |

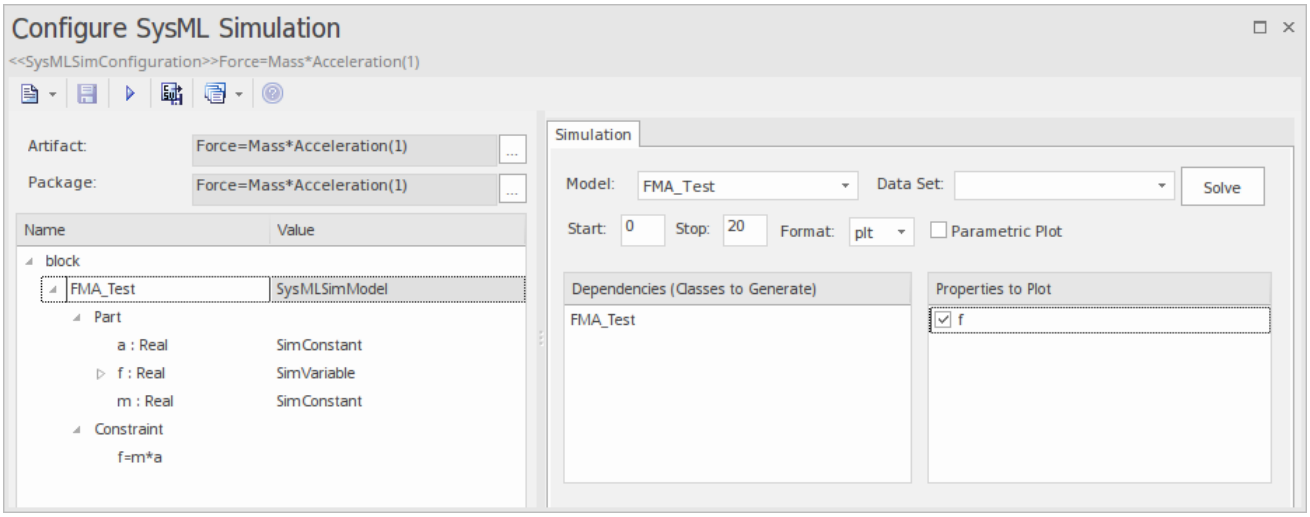
## Defining inline Constraint Equations on a Block

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.




*Tips: You can define multiple constraints in one block.*

1. Create a SysMLSimConfiguration element 'Force=Mass*Acceleration(1)' and point it to the Package 'FMA_Test'.
2. Set 'FMA_Test' to be 'SysMLSimModel'.
3. Set part 'a' and 'm' to be 'SimConstant'; optionally set part 'f' to be 'SimVariable'.
4. Plot for 'f' and simulate.

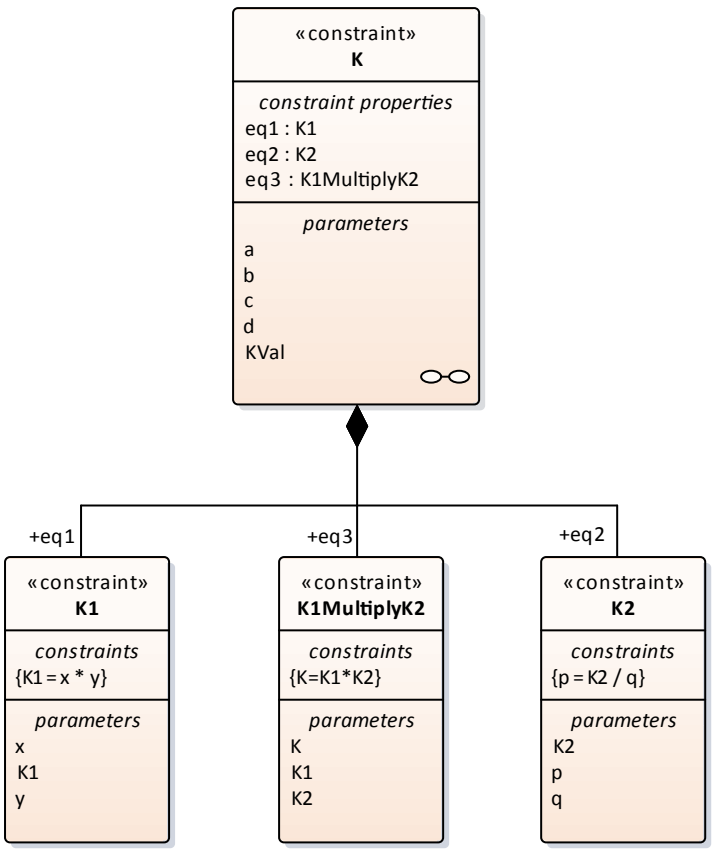



A chart should be plotted with f = 98.1 (which comes from 10 * 9.81).

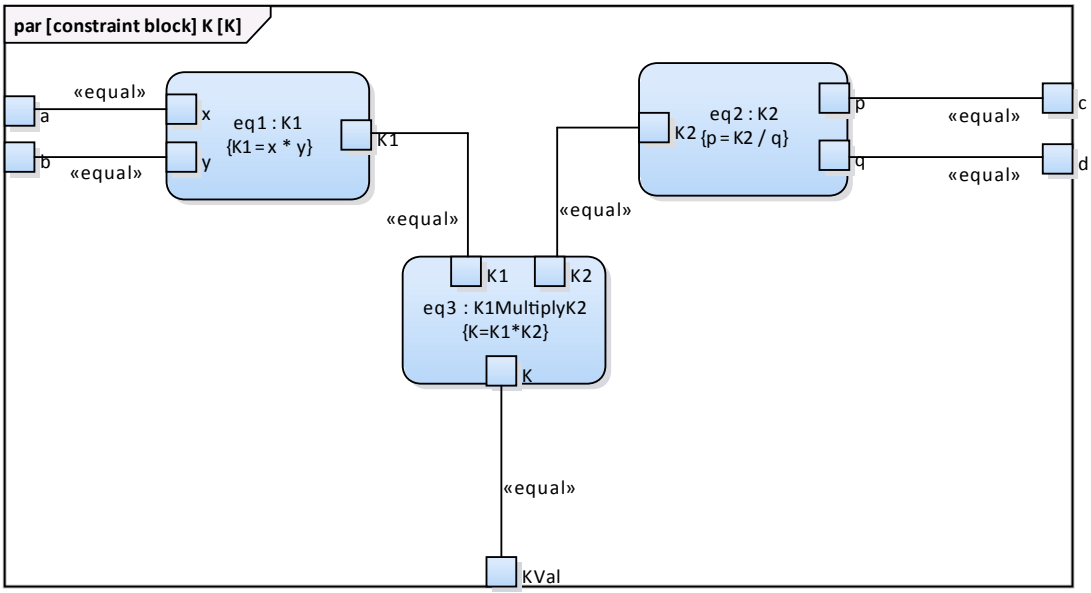
## Connected Constraint Properties

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

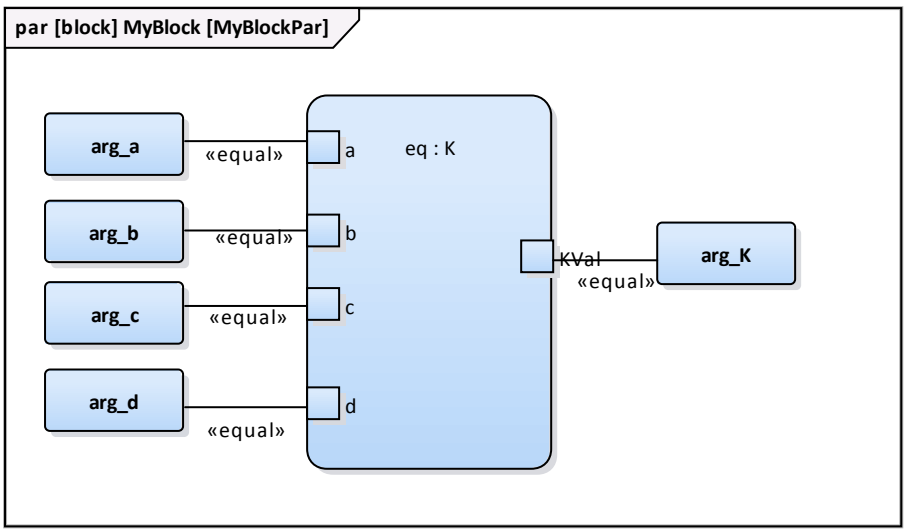
In this figure, Constraint Block 'K' defines parameters 'a', 'b', 'c', 'd' and 'KVal', and three constraint properties 'eq1', 'eq2' and 'eq3', typed to 'K1', 'K2' and 'K1MultiplyK2' respectively.



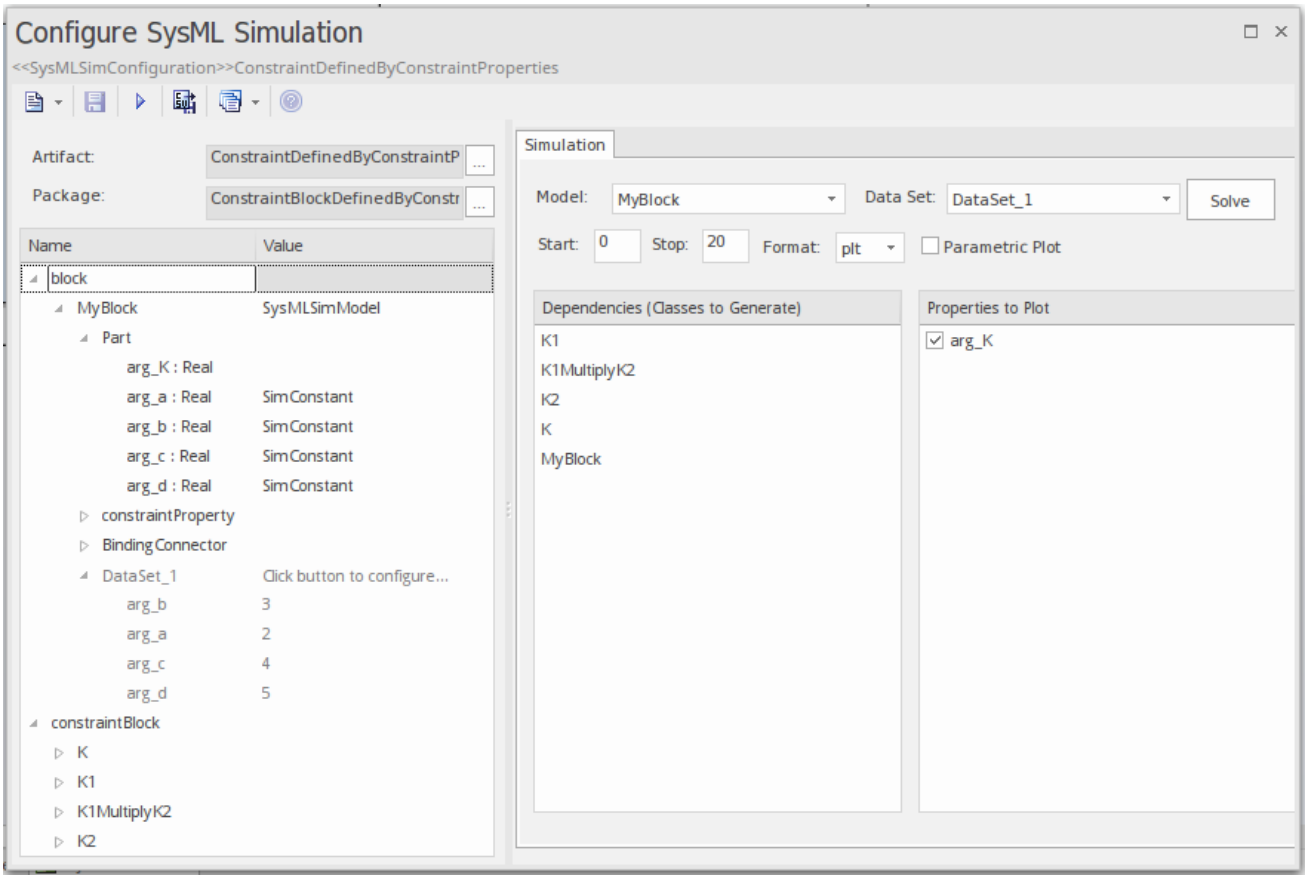
Create a Parametric diagram in Constraint Block 'K' and connect the parameters to the constraint properties with binding connectors, as shown:




- 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 (arg_a = 2, arg_b = 3, arg_c = 4, arg_d = 5) in a data set
- In the 'Configure SysML Simulation' dialog, select MyBlock > DataSet_1 and plot for 'arg_K', and 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.

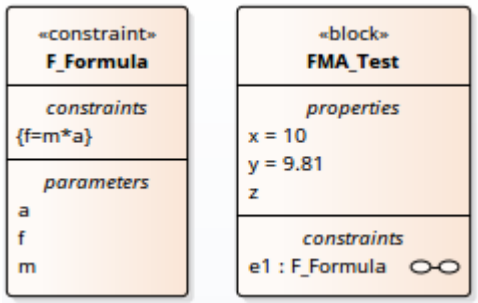
We can easily solve K2 to be p * q in this example, but in some complex examples it is extremely hard to solve a variable from an equation; however, the EA SysMLSim can still get it right.

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

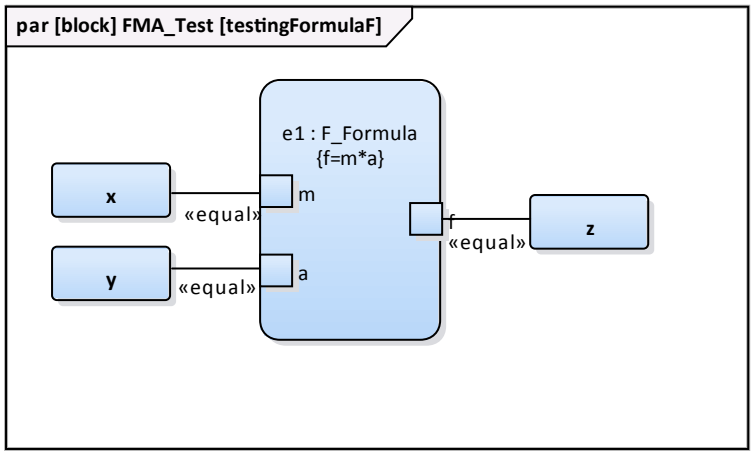
If one equation is commonly used in many Blocks, a Constraint Block can be created for use as a constraint property in each Block. These are the changes we make, based on the previous example:

- Create a Constraint Block element 'F_Formula' with three parameters 'a', 'm' and 'f', and a constraint 'f = m * a

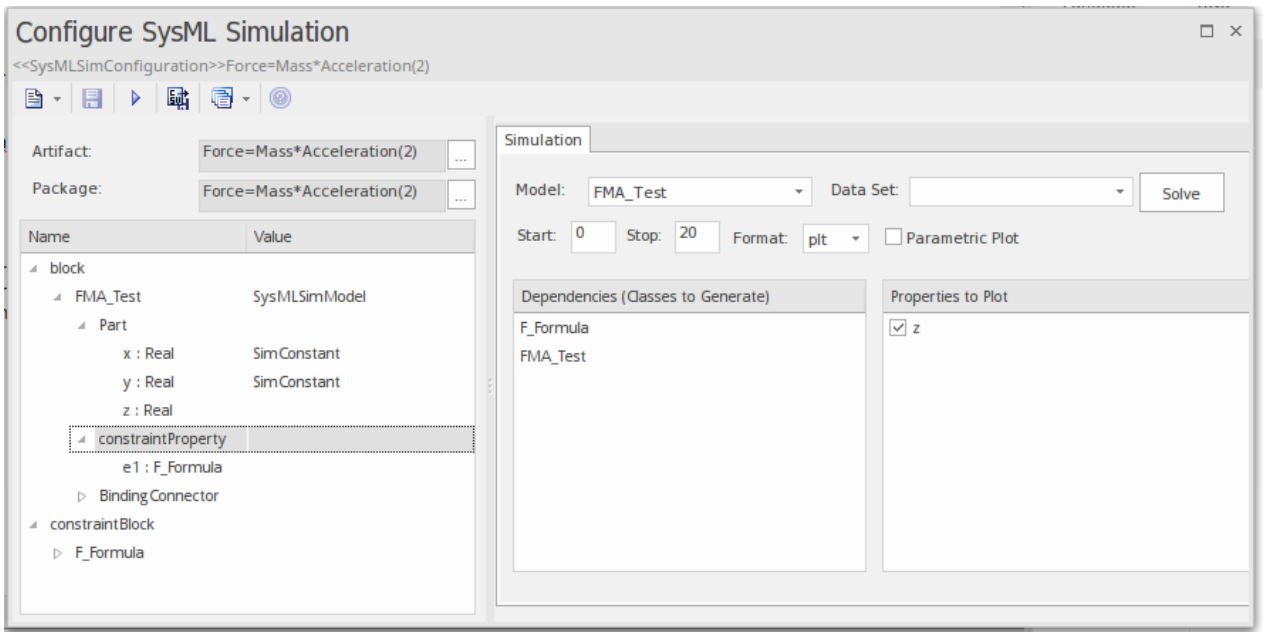



*Tip: Primitive type 'Real' will be applied if property types are empty*

- 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 like this:
- Set 'FMA_Test' to be 'SysMLSimModel'
- Set 'x' and 'y' to be 'SimConstant'
- Plot for 'z' and simulate



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 rates hould be modeled as flows and the attribute 'isConserved' should be set on the flow variables.

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:

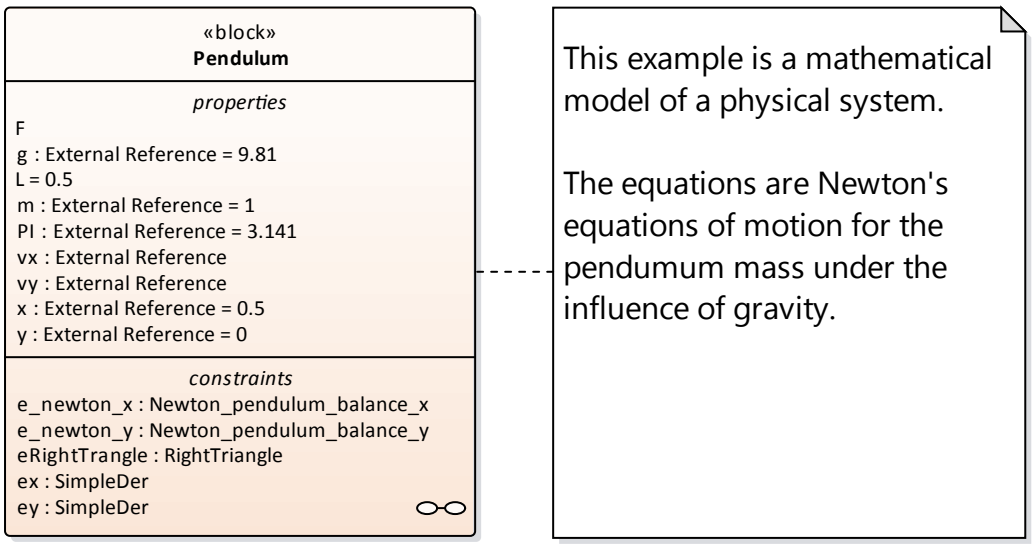
```
source.p.v = resistor.n.v;
source.p.i + resistor.n.i = 0;
```

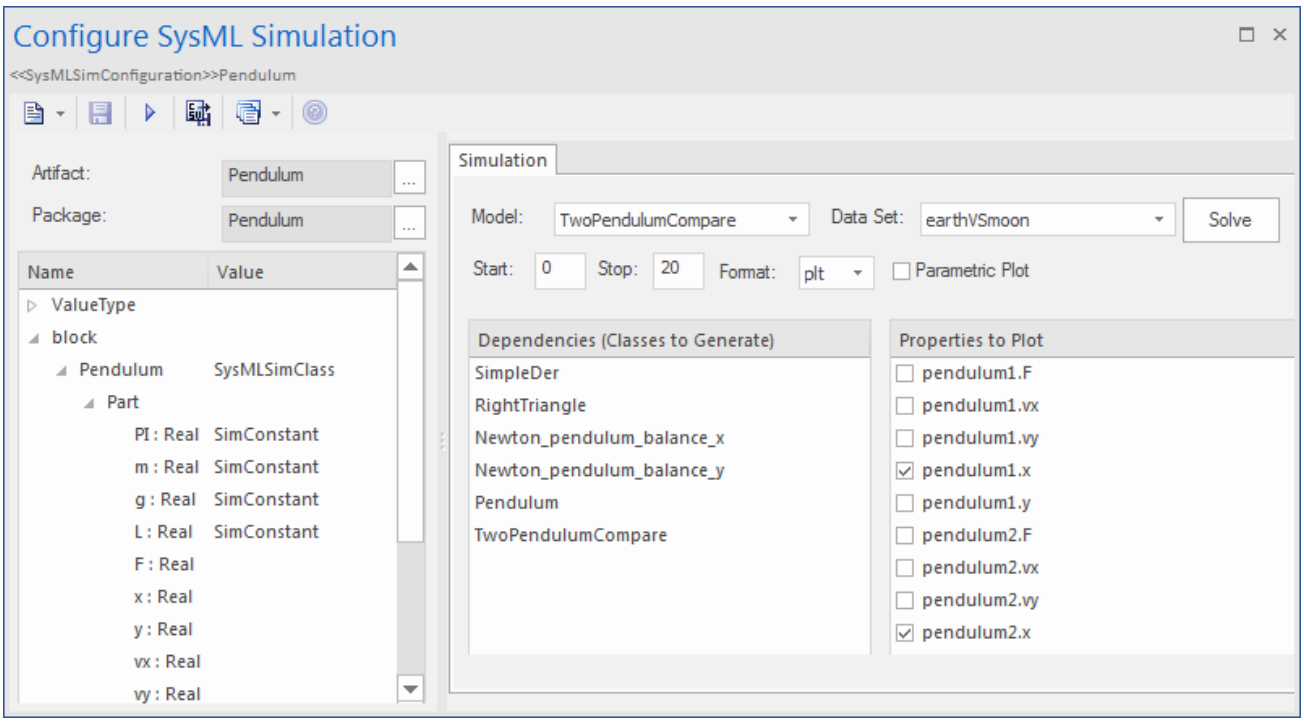
## Default Value and Initial Values

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

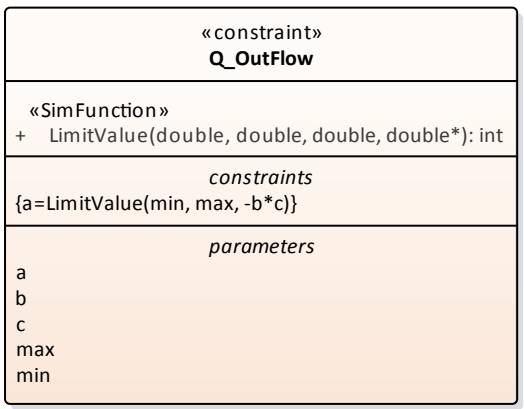
```
    Real x (start=0.5);
    Real y (start=0);
    Real vx;
    Real vy;
    ......
equation
    ......
end Pendulum;
```

- 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' dialog.
- 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 'Initial Code' field of the 'Behavior' tab

The function body is defined as shown:

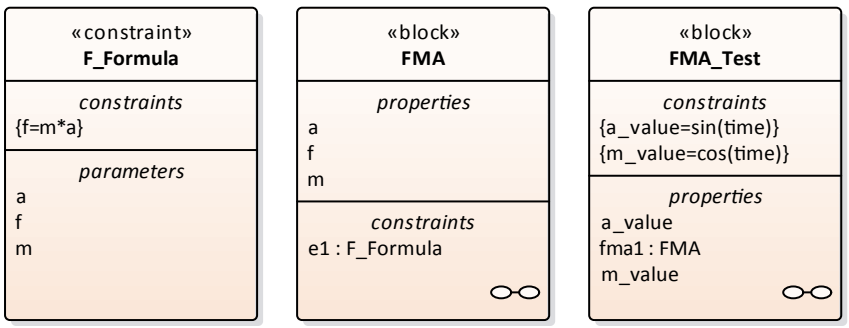
```
pLim :=
  if p > pMax then
    pMax
  else if p< pMin then
    pMin
  else
    p;
```

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

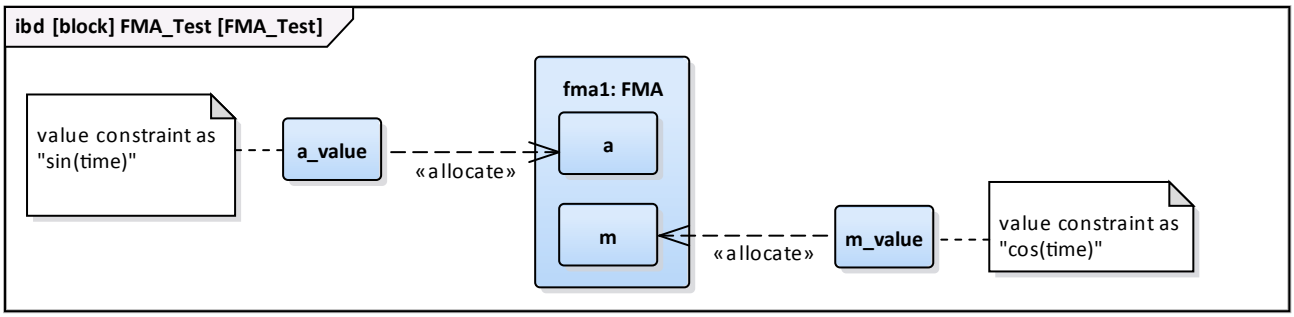
This figure shows a simple model called 'Force=Mass*Accelaration'.



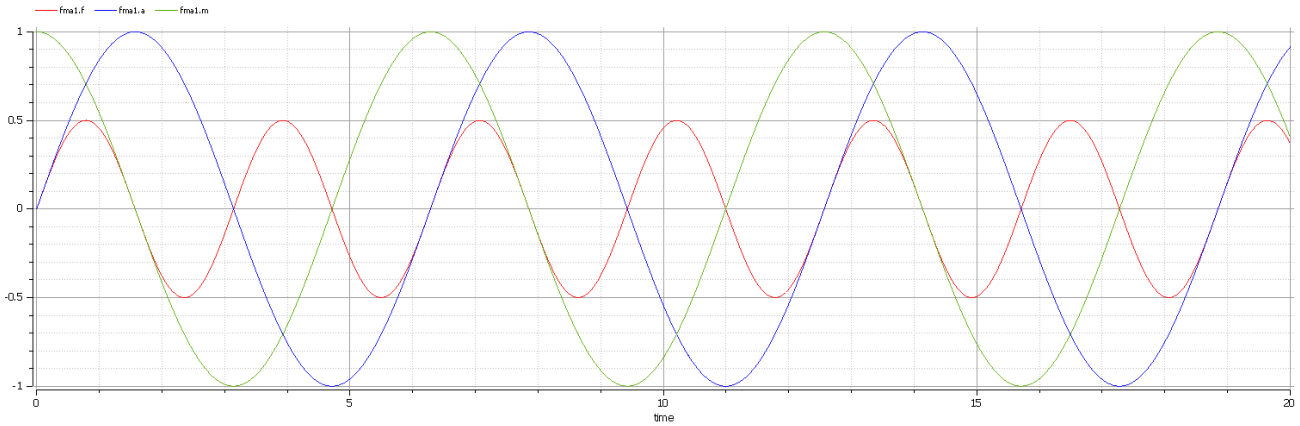

- A block 'FMA' is modeled with properties 'a', 'f', and 'm' and a constraintProperty 'e1', typed to Constraint Block 'F_Formula'
- The block 'FMA' does not have any initial value set on its properties, and the properties 'a', 'f' and 'm' are all variable,

so their value change depends on the environment in which they are simulated

- Create a block 'FMA_Test' as 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'



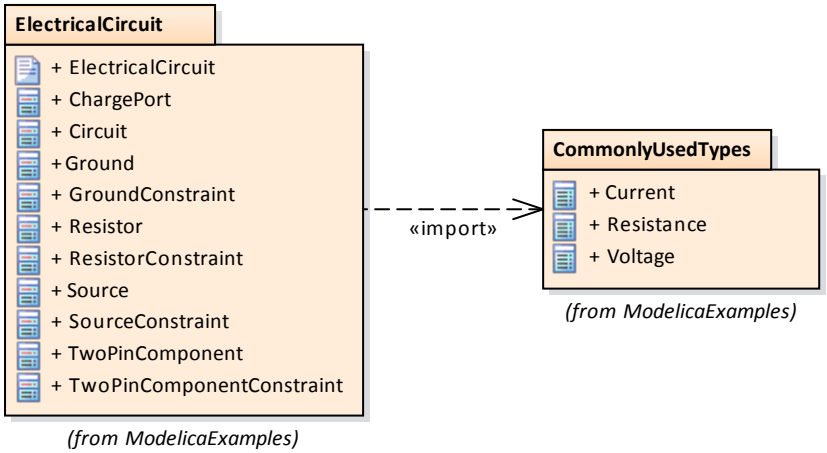

- Plot for 'fma1.a', 'fma1.m', 'fma1.f' and simulate the model




## 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 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**.


ElectricalCircuit
+ ElectricalCircuit
+ ChargePort
+ Circuit
+ Ground
+ GroundConstraint
+ Resistor
+ ResistorConstraint
+ Source
+ SourceConstraint
+ TwoPinComponent
+ TwoPinComponentConstraint
(from ModelicaExamples)
«import»
CommonlyUsedTypes
+ Current
+ Resistance
+ Voltage
(from ModelicaExamples)

# Troubleshooting OpenModelica Simulation

## Common Simulation Issues

The following table describes some of the common issues that can prevent a model being simulated.

| |
|---|
| Check the output at "**System Output** \| Build" window, the messages were dumped from OpenModelica compiler (omc.exe), it normally points you to the lines of the modelica source code. This will help you pick up most of the errors. |
| The number of equations are fewer than the number of variables: You might forgot to set some of the properties to be SimConstant, which means the value doesn't change during simulation. You might need to provide the SimConstant properties value before the simulation started. (Set the values through Simulation Data Set) |
| The blocks that are typing to ports may contain conserved properties. For example, a block ChargePort may contain two parts: "v : Voltage" and "i: Current", the property "i : Current" should be defined with SimVariable with attribute "isConserved = true". |
| SimConstants should be provided default values. |
| A SimVariable may need an initial value to start with. |
| The properties may typed by elements (blocks or value type) outside of the configured package; Use a package import connector to fix this. |