Enterprise Architect

User Guide Series

SysML Parametric Simulation

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: 13/11/2023
Version: 16.1

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

SysML Parametric Simulation 3
Configure SysML Simulation 7
Creating a Parametric Model 18
Model Analysis using Datasets 35
SysML Simulation Examples 40
Electrical Circuit Simulation Example 42
Mass-Spring-Damper Oscillator Simulation Example 53
Water Tank Pressure Regulator 63

SysML Parametric Simulation

Enterprise Architect provides integration with both OpenModelica and MATLAB Simulink to support rapid and robust evaluation of how a SysML model will behave in different circumstances.

The OpenModelica 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 these Libraries.

Enterprise Architect's MATLAB integration connects via the MATLAB API, allowing your Enterprise Architect simulations and other scripts to act based on the value of any available MATLAB functions and expressions. You can call MATLAB through a Solver Class, or export your model to MATLAB Simulink, Simscape and/or Stateflow.

SysML Simulation features

These sections describe 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 of the results.

Section	Description
Introduction to SysML Parametric	SysML Parametric models support the engineering analysis of critical system parameters, including the evaluation of key metrics such as performance,

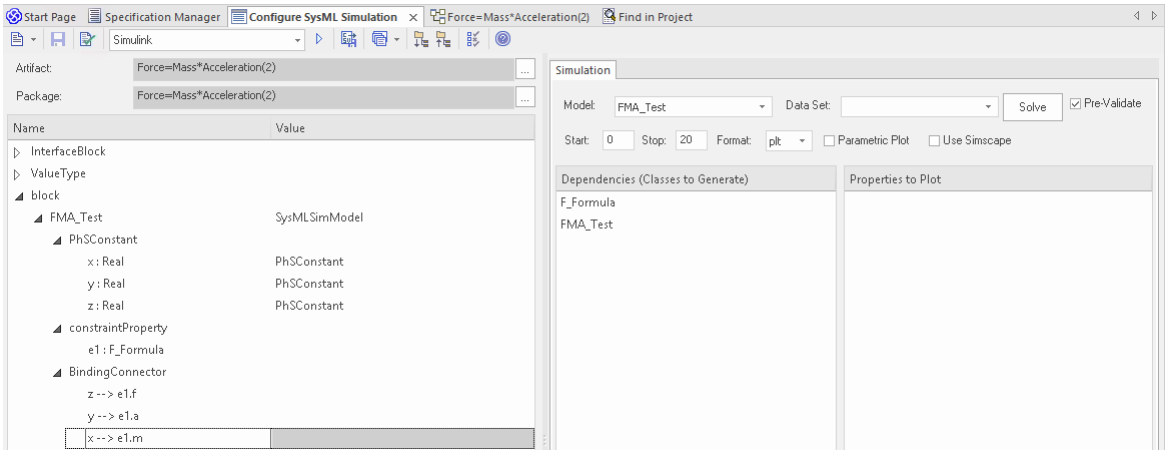
Models	reliability and other physical characteristics. These models combine Requirements models with System Design models, by capturing executable constraints based on complex mathematical relationships. Parametric diagrams are specialized Internal Block diagrams that help you, the modeler, to combine 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.
Creating a Parametric Model	An overview on developing SysML model elements for simulation, configuring these elements in the Configure SysML Simulation window, and observing the results of a simulation.
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 model that can be solved

	(simulated) using either MATLAB Simulink or 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.
Model Analysis using Dataset	Using the Simulation configuration a SysML Block can have multiple datasets defined against it. This allows for running repeatable variations on a simulation of the SysML model.
SysPhS Standard Support	The *SysPhS Standard* is a *SysML Extension for Physical Interaction and Signal Flow Simulation*. It defines a standard way to translate between a SysML model and either a Modelica model or a Simulink/Simscape model, providing a simpler model-based method for sharing simulations. See the *SysPhS Standard Support* Help topic.

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. All three examples happen to use the OpenModelica libraries. These examples and what you are able to learn from them are described in the *SysML Simulation Example*s topic.

Configure SysML Simulation

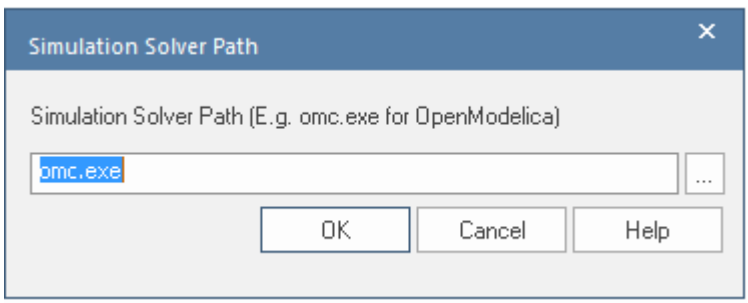
The Configure SysML Simulation window is the interface through which you can provide run-time parameters for executing the simulation of a SysML model. The simulation is based on a simulation configuration defined in a SysMLSimConfiguration Artifact element.



Access

Ribbon	Simulate > System Behavior > Modelica/Simulink > 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: • Select Artifact — Select and load an existing configuration from an Artifact with the SysMLSimConfiguration stereotype (if one has not already been selected) • Create Artifact — Create a new SysMLSimConfiguration or select and load an existing configuration artifact • Select Package — Select a Package to scan for SysML elements to configure for simulation • Reload — Reload the Configuration Manager with changes to the current Package • Configure Simulation Solver — Display the 'Simulation Solver Path' dialog, in which you type or browse for the path to the Solver to use 

	Click on this button to save the configuration to the current Artifact.
	Click on this icon to specifically validate the model against the SysML configuration now. The results of the validation display in the 'SysML Simulation' tab of the System Output window. You can also select an option to automatically pre-validate the model before each simulation is executed. See the 'Pre-validate' option in the *Simulation Tab* table.
	Click on this icon to expand every item in the hierarchy in the 'Name' column of the window.
	Click on this icon to collapse all the expanded items in the model hierarchy in the 'Name' column of the window.
	Click on this icon to display a list of object types that can be suppressed in the simulation. Click on the checkbox against each object to suppress, or click on the All button to select all items for suppression.

	You can also use the Filter Bar at the top of the 'Option' column to only display items having the specified letter or text string in the name.
	Click on the drop-down arrow and select the application under which the simulation is being run - such as OpenModelica or Simulink.
	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: • ModelName_res.mat (the default for OpenModelica) • ModelName_res.plt or • ModelName_res.csv 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: • Run Last Code - Execute the most

	recently generated code • Generate Code — Generate the code without compiling or running it • Open Simulation Directory — Open the directory into which OpenModelica or Simulink code will be generated • Edit Templates — Customize the code generated for OpenModelica or Simulink, using the Code Template Editor

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	If you have specified an existing SysMLSimConfiguration Artifact, this field defaults to the Package containing the SysML model associated with that Artifact. 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.

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 elements either generalize

ValueType	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 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'). 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.
Properties	The preferred way to specify constants or variables and their settings is to use the SysPhS stereotypes PhSConstant and PhSVariable on the Properties themselves. The PhSVariable stereotypes have built-in properties for *isContinuous*, *isConserved*, and *changeCycle*. The Properties will be listed under either PhSConstant or PhSVariable and the Value cannot be changed. It's also possible to define the settings within the Configure SysML Simulation

	window. In this case they will be listed under 'Properties'. Properties within a Block can be configured to be either SimConstants or SimVariables. For a SimVariable, you configure these attributes: • isContinuous — determines whether the property value varies continuously ('true', the default) or discretely ('false') • 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 • changeCycle — specifies the time interval at which a discrete property value changes; the default value is '0' - changeCycle can be set to a value other than 0 only when isContinuous = 'false' - The value of changeCycle must be positive or equal to 0
Port	No configuration required.
SimFunction	Functions are created as operations in Blocks or ConstraintBlocks, stereotyped

	as 'SimFunction'. 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. No configuration required; however, if the properties are different, the system provides an option to synchronize them.
Connector	Connects two Ports. No configuration required in the Configure SysML Simulation view. 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 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.
Pre-Validate	Select this checkbox to automatically validate the model before each simulation of the model is executed.
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. • Deselect the checkbox to plot Legend(s) on the y-axis against time on the x-axis Note: With the checkbox selected, you must select two properties to plot.
Use Simscape	(if the selected math tool is Simulink) Select the checkbox if you also want to process the simulation in Simscape.
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.

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 ConstraintBlocks, 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

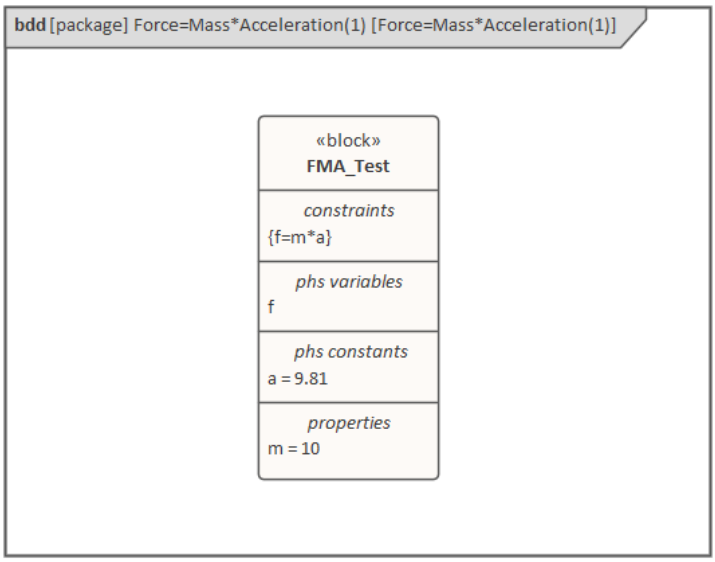
Ribbon	Simulate > System Behavior > Modelica/Simulink > SysMLSim

	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.

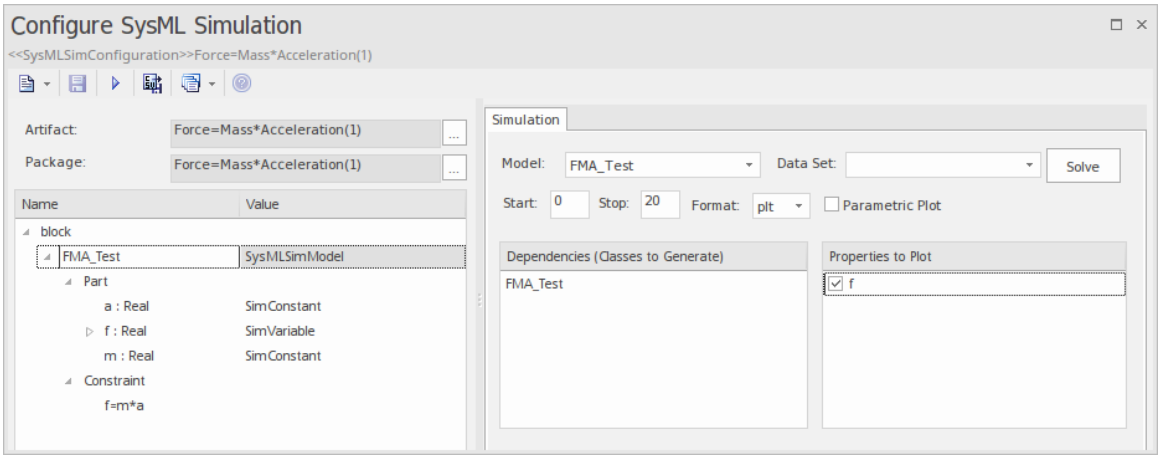


Tip: You can define multiple constraints in one Block.

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 'PhSConstant' and (optionally) set 'f' to

'PhSVariable'.

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.

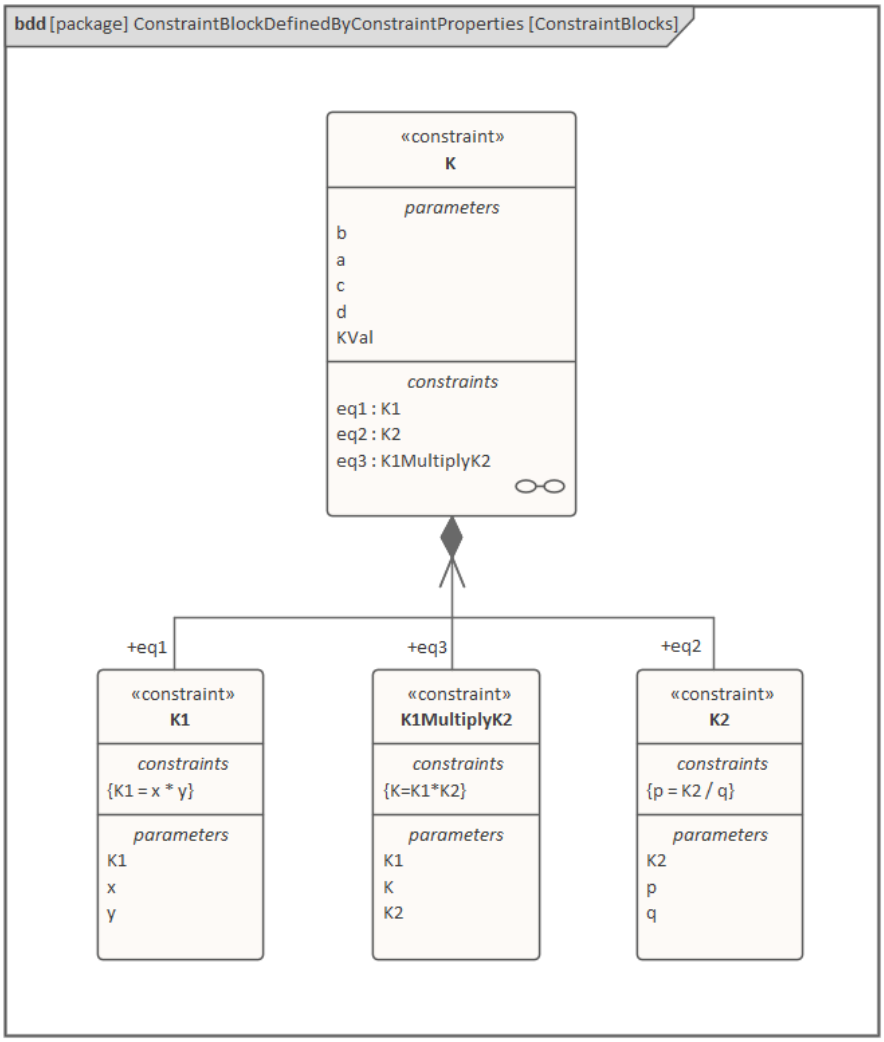


A chart should be plotted with f = 98.1 (which comes from 10 * 9.81).

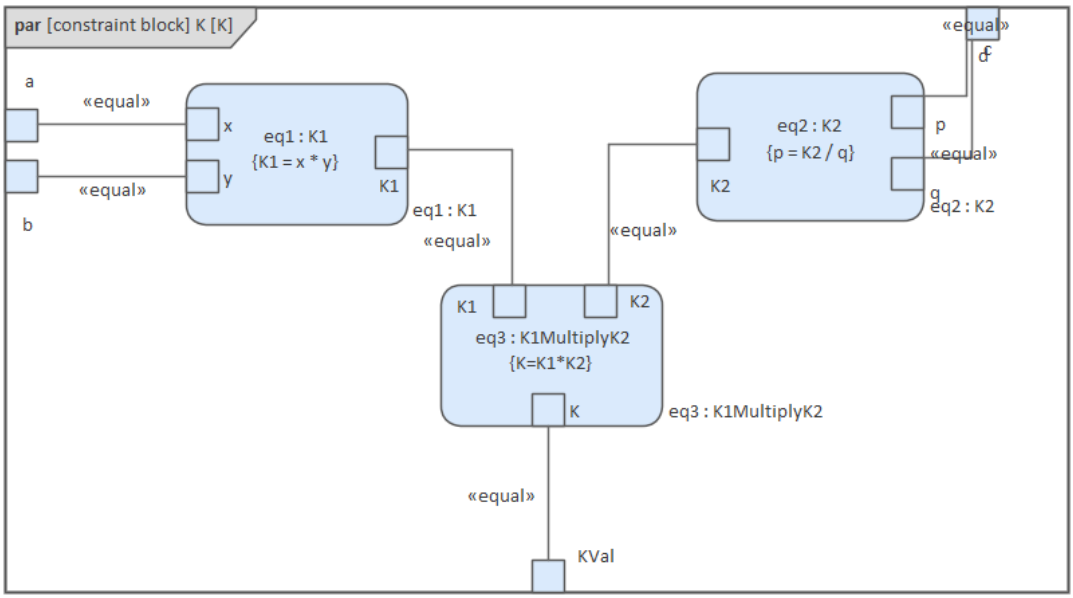
Connected Constraint Properties

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

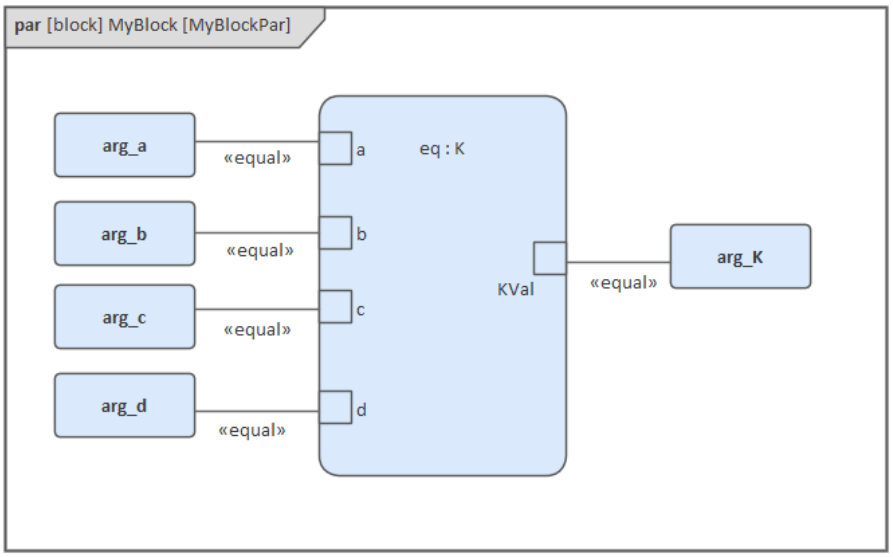
In this figure, ConstraintBlock 'K' defines parameters 'a', 'b', 'c', 'd' and 'KVal', and three constraint properties 'eq1', 'eq2' and 'eq3', typed to 'K1', 'K2' and 'K1MultiplyK2' respectively.



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

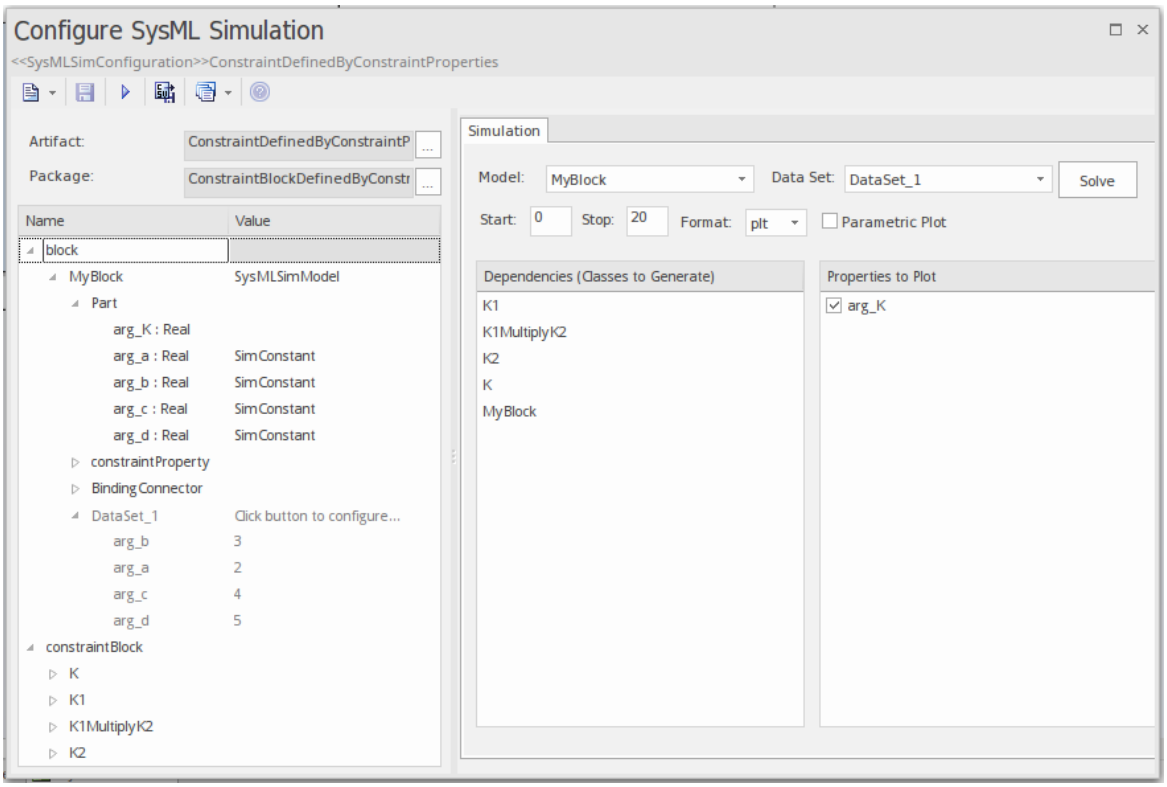


- Create a model MyBlock with five Properties (Parts)
- Create a constraint property 'eq' for MyBlock and show the parameters
- Bind the properties to the parameters



- Provide values (arg_a = 2, arg_b = 3, arg_c = 4, arg_d = 5) in a data set

- In the 'Configure SysML Simulation' dialog, 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.

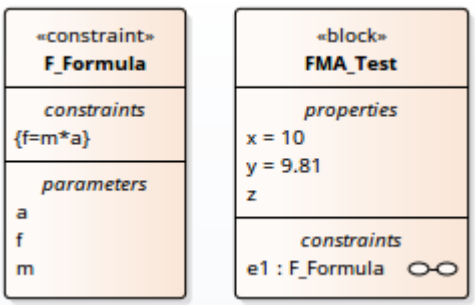
We can easily solve K2 to be p * q in this example, but in some complex examples it is extremely hard to solve a variable from an equation; however, the Enterprise Architect SysMLSim can still get it right.

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

Creating Reuseable ConstraintBlocks

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

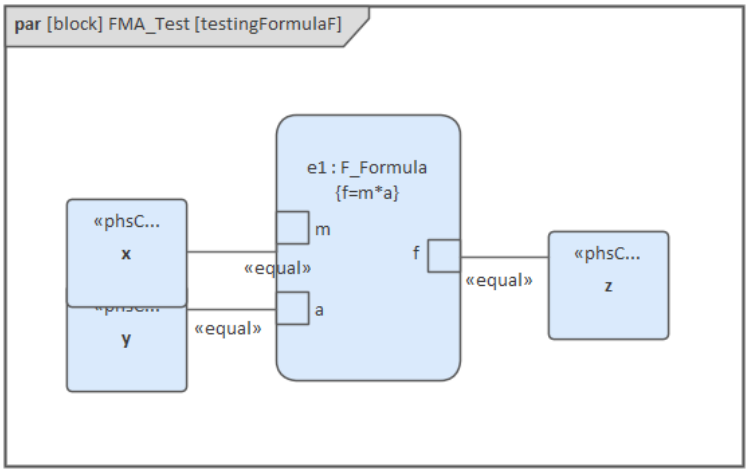
- Create a ConstraintBlock element 'F_Formula' with three parameters 'a', 'm' and 'f', and a constraint 'f = m * a'



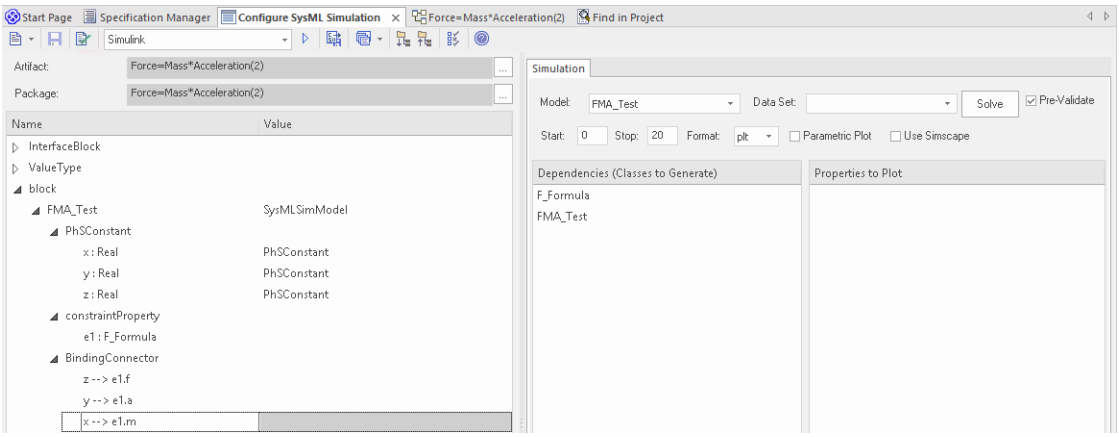
Tip: Primitive type 'Real' will be applied if property types are empty

- 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 ConstraintProperty '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 'PhSConstant'
  - 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 OpenModelica 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:

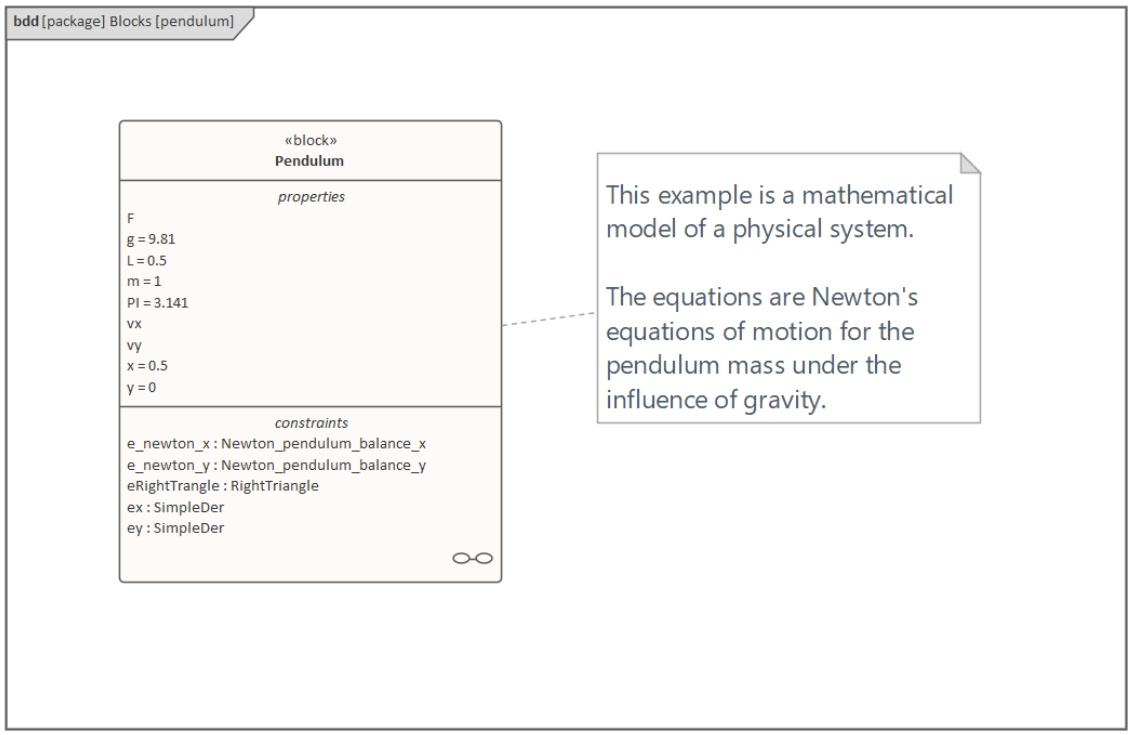
```
source.p.v = resistor.n.v;
source.p.i + resistor.n.i = 0;
```

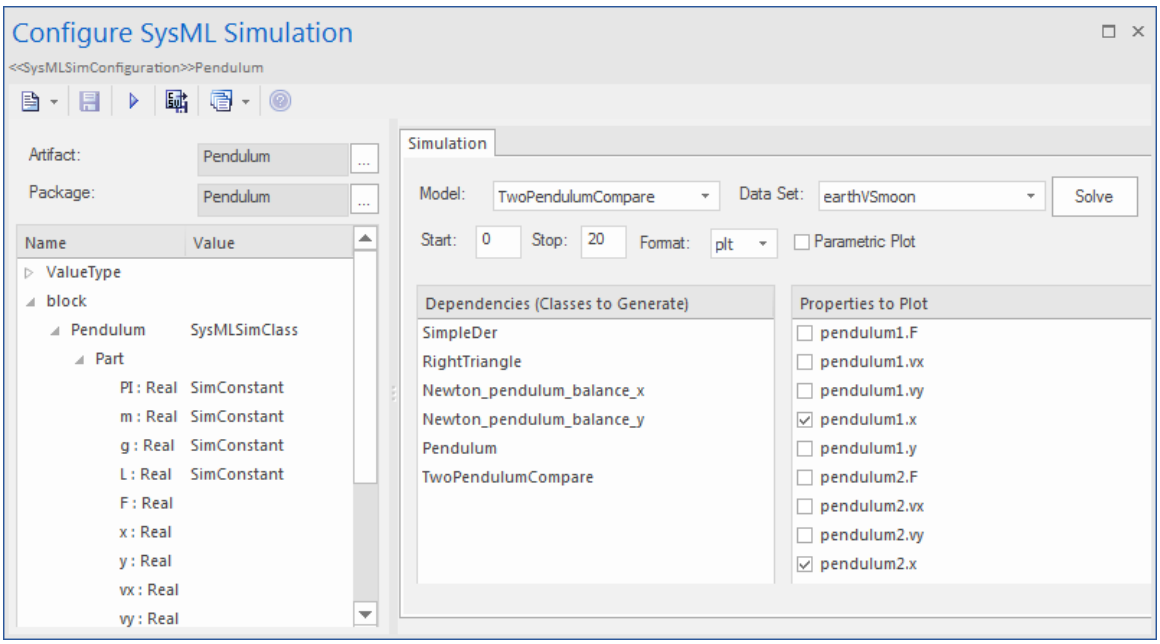
Default Value and Initial Values

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 PhSConstant or the initial value for a PhSVariable.

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 'PhSConstant'.





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;
```

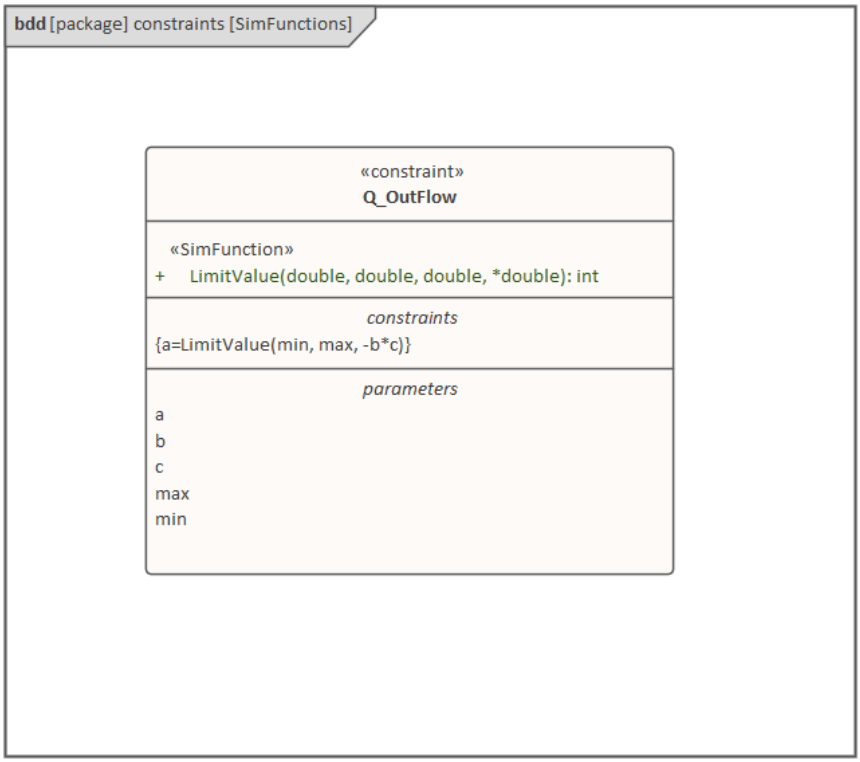
- Properties 'PI', 'm', 'g' and 'L' are constant, and are generated as a declaration equation
- Properties 'x' and 'y' are variable; their starting values are 0.5 and 0 respectively, and the initial values are generated as modifications

Simulation Functions

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

In the ConstraintBlock 'Q_OutFlow', a function 'LimitValue'

is defined and used in the constraint.



- On a Block or ConstraintBlock, 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;
```

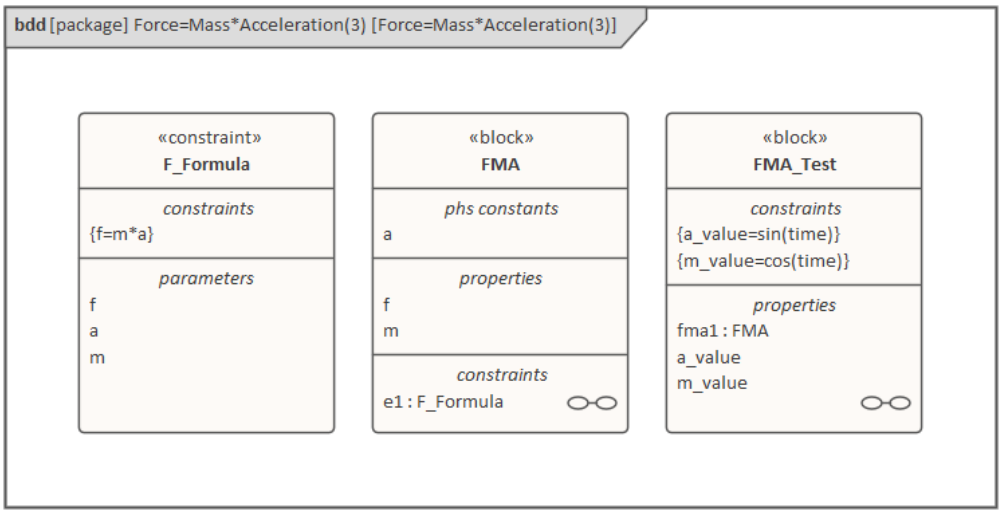
When generating code, Enterprise Architect will collect all the operations stereotyped as 'SimFunction' defined in ConstraintBlocks and Blocks, then generate code resembling this:

```
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;

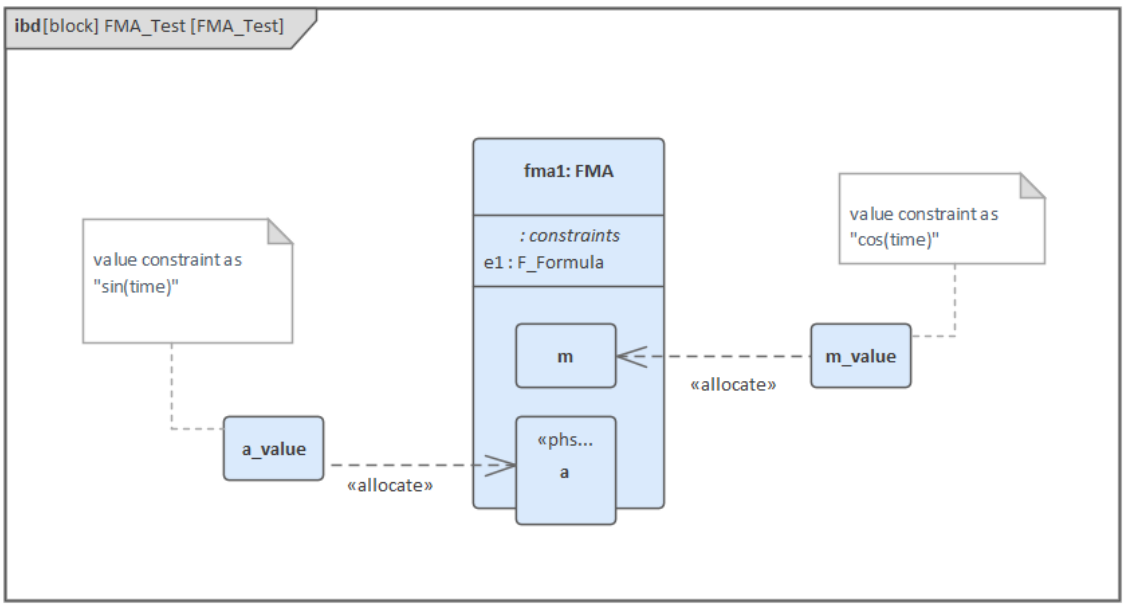
Value Allocation

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

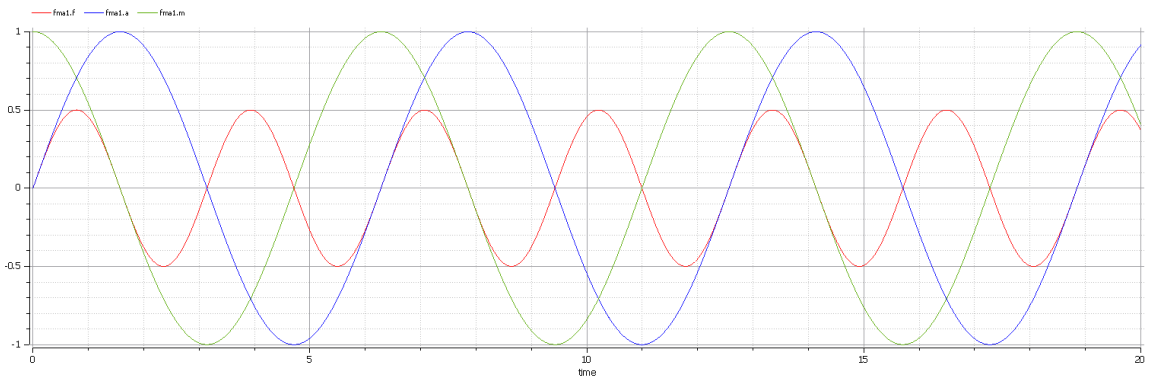


- A Block 'FMA' is modeled with properties 'a', 'f', and 'm' and a constraintProperty 'e1', typed to Constraint Block 'F_Formula'
- The Block 'FMA' does not have any initial value set on its properties, and the properties 'a', 'f' and 'm' are all variable, so their value change depends on the environment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

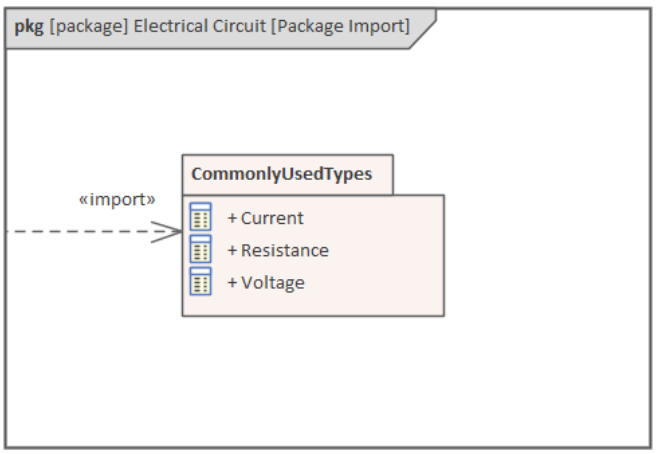


Packages and Imports

The SysMLSimConfiguration Artifact collects the elements (such as Blocks, ConstraintBlocks 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.



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/Simulink > SysMLSim Configuration Manager > in "block" group > Name column > Context menu on block element > Create Simulation DataSet

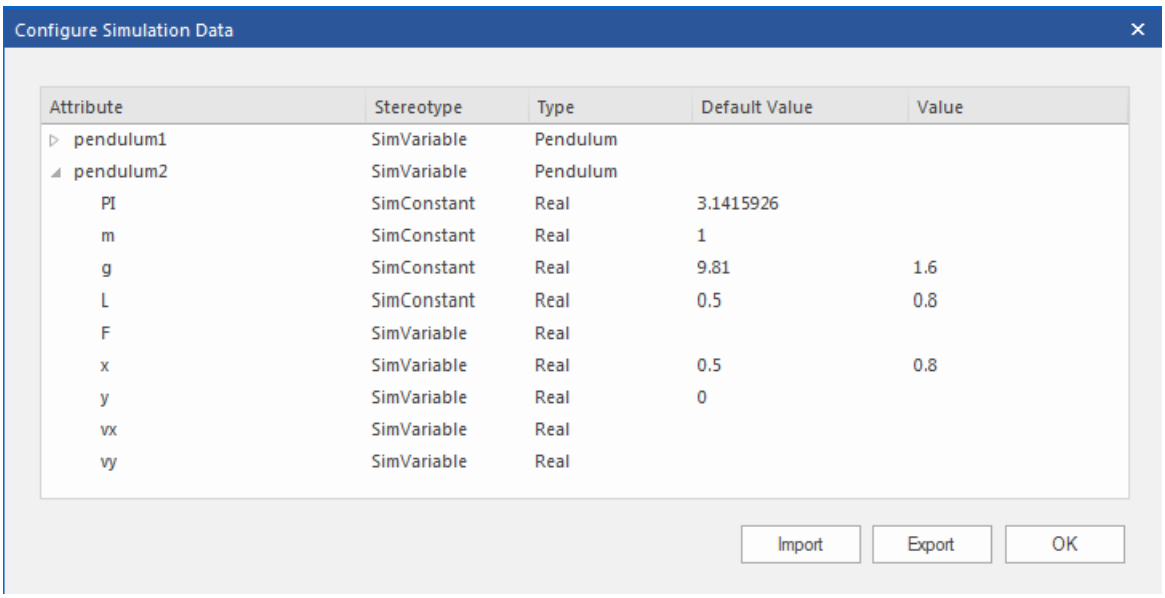
Dataset Management

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 *Configure Simulation Data* 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 *Configure Simulation Data* 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.



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.

SysML Simulation Examples

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

Examples

Model	Description
Electrical Circuit Simulation Example	The first example is of the simulation of loading 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 terminals of a resistor are evaluated and compared to the expected values.
Mass-Spring-Damper Oscillator Simulation Example	The second example uses a simple physical model to demonstrate the oscillation behavior of a mass-spring-damper system.
Water Tank Pressure	The final example shows the water levels of two water tanks where the water is

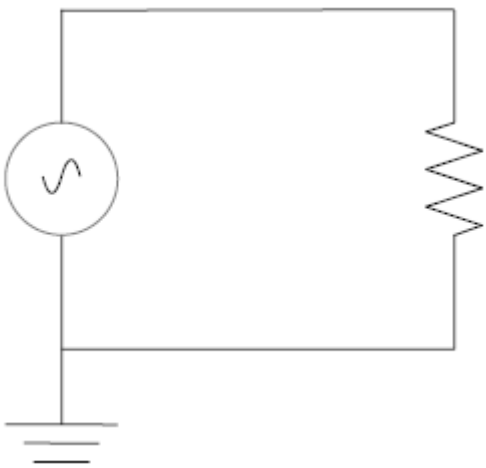
Regulator	being distributed between them. We first simulate a well-balanced system, then we simulate a system where the water will overflow from the second tank.

Electrical Circuit Simulation Example

For this example, we 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.

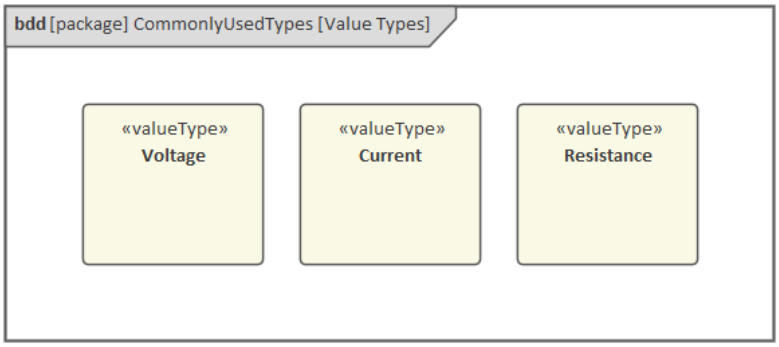


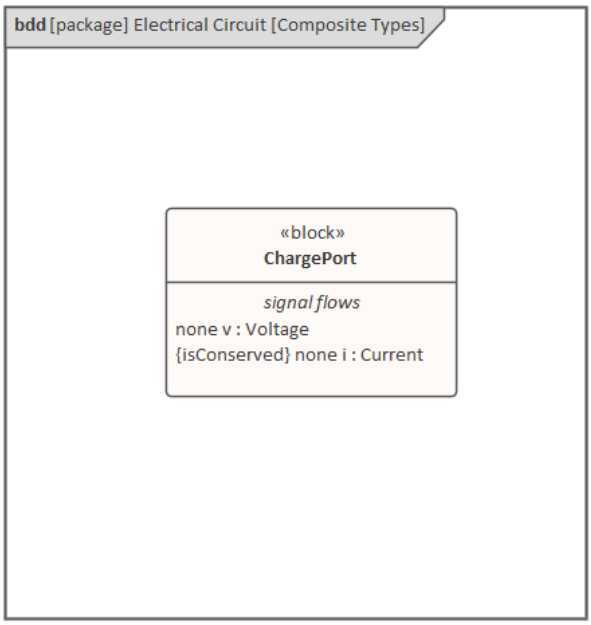
The circuit includes an AC power source, an earth and a resistor, connected to each other by electrical wire.

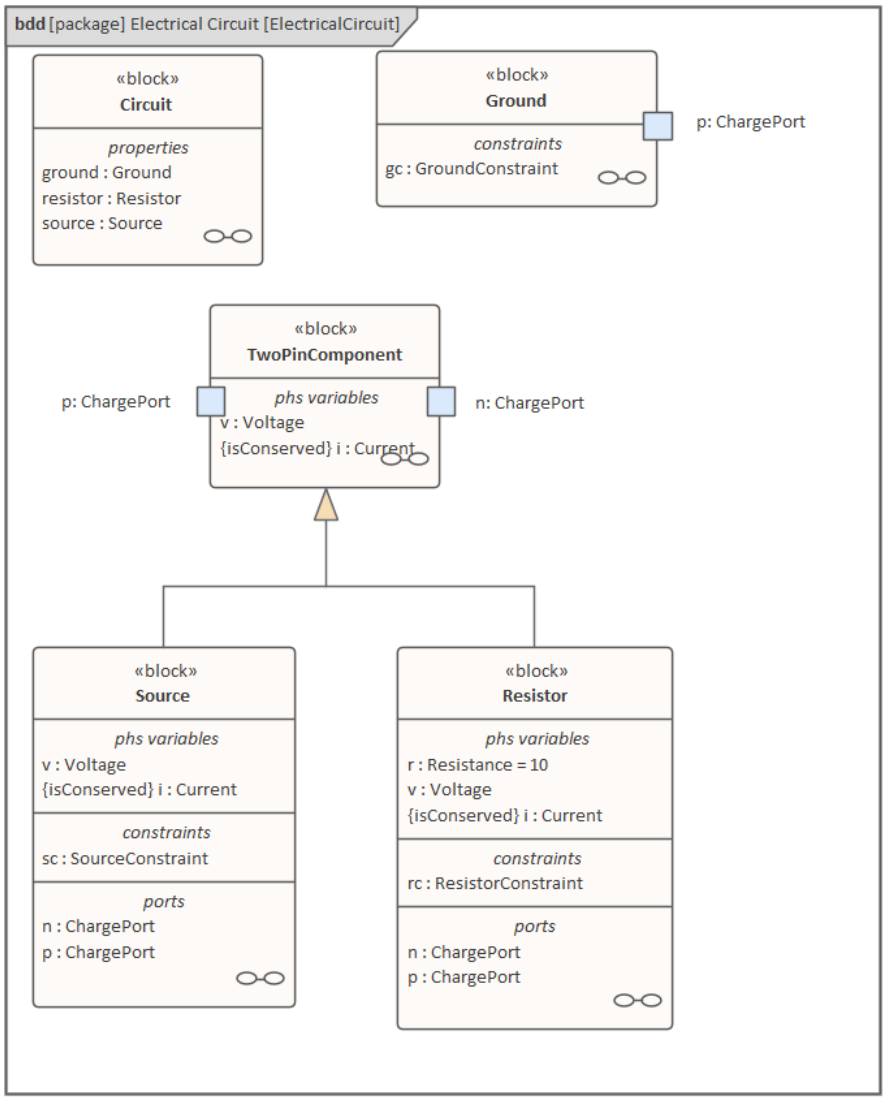
Create SysML Model

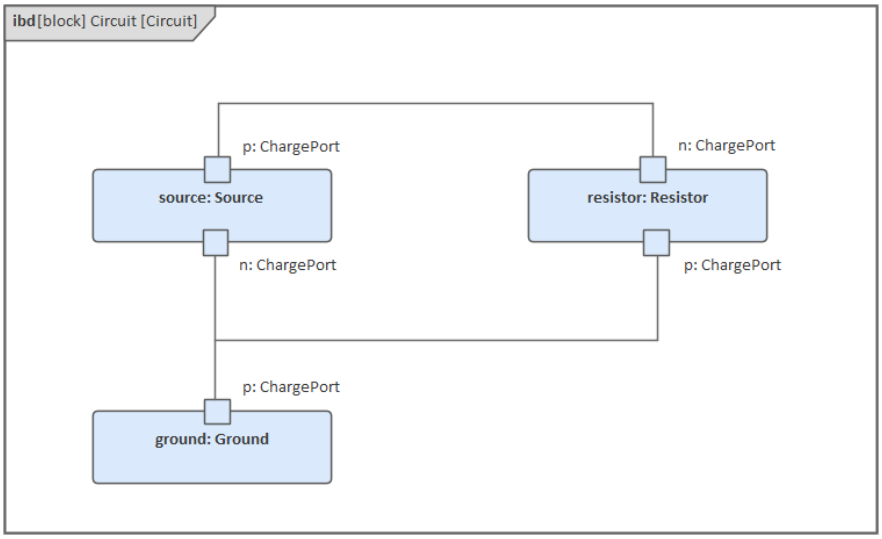
This table shows how we can build up a complete SysML model to represent the circuit, starting at the lowest level types and building up the model one step at a time.

Component	Action

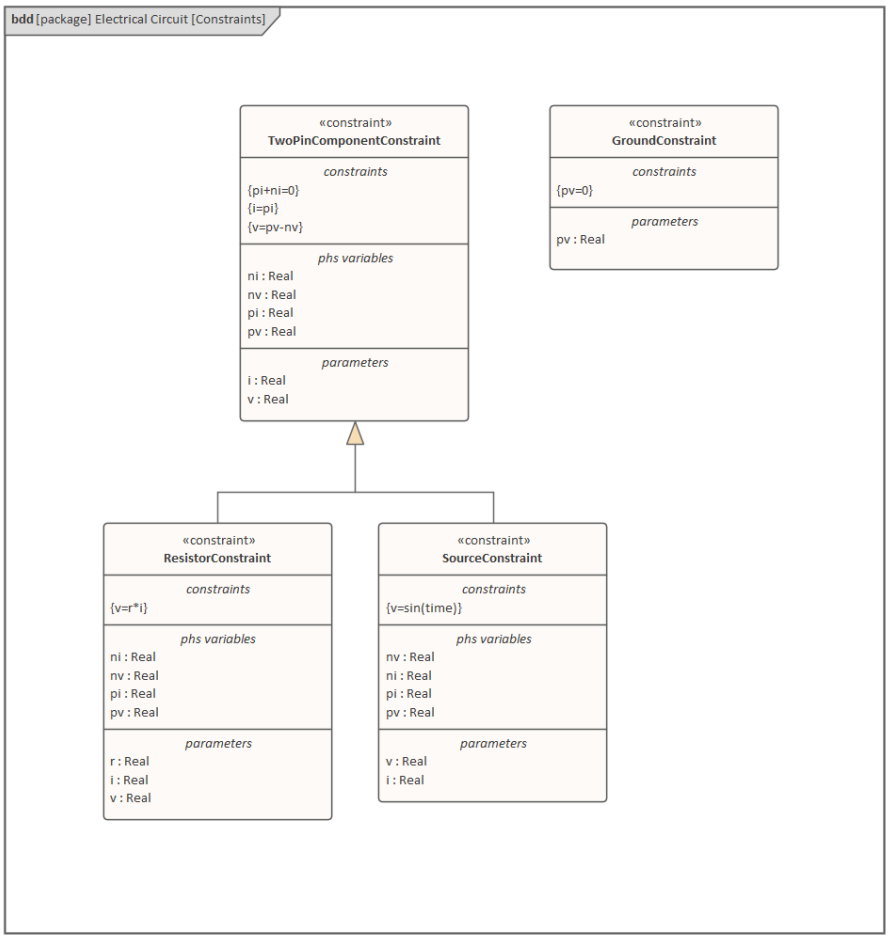
Types	Define Value Types for the Voltage, Current and Resistance. Unit and quantity kind are not important for the purposes of simulation, but would be set if defining a complete SysML model. These types will be generalized from the primitive type 'Real'. In other models, you can choose to map a Value Type to a corresponding simulation type separate from the model.  Additionally, define a composite type (Block) 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. In a Block Definition Diagram (BDD), create a Circuit Block. 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 the 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

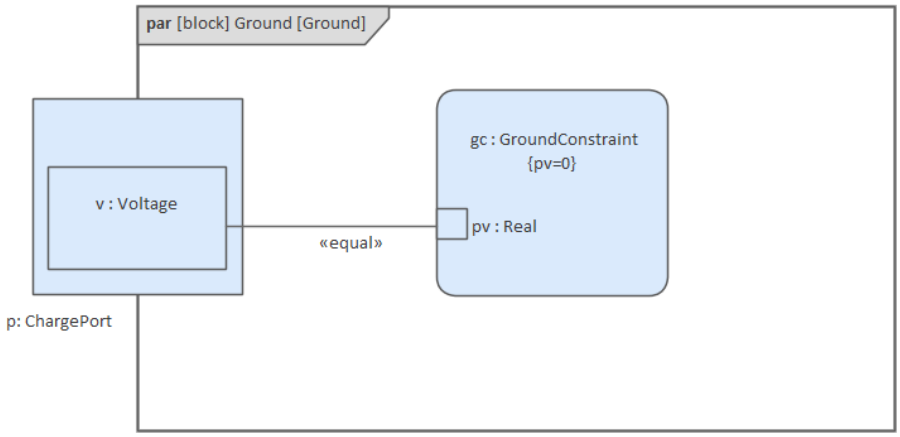
	named 'p' (positive) and 'n' (negative), and they are of type ChargePort. This figure shows the BDD, with the Blocks Circuit, Ground, TwoPinComponent, Source and Resistor. 
Internal Structure	Create an Internal Block Diagram (IBD) for Circuit. Add properties for the Source, Resistor and Ground, typed 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 ConstraintBlocks. Parameters of ConstraintBlocks correspond to PhSVariables and PhSConstants of Blocks ('i', 'v', 'r' in this example), as well as to PhSVariables

	present in the type of the Ports ('pv', 'pi', 'nv', 'ni' in this example). Create a ConstraintBlock '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 are rendered in a BDD.

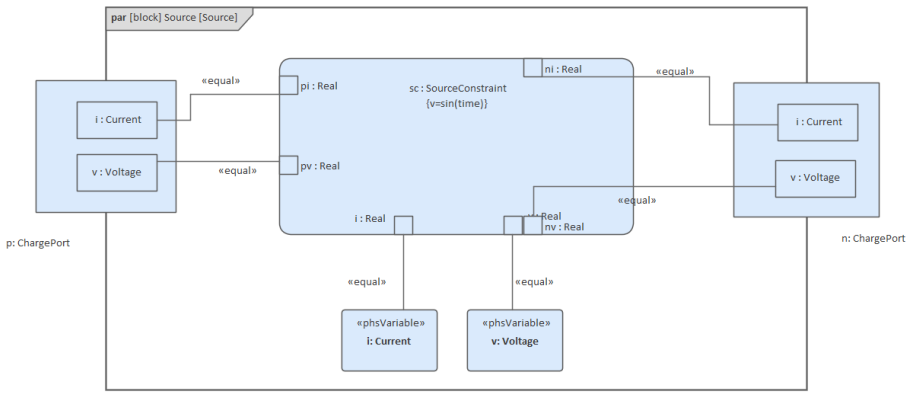


Bindings	The values of Constraint parameters are equated to variable and constant values with binding connectors. Create Constraint properties on each Block (properties typed by ConstraintBlocks) and bind the Block variables and constants to the Constraint parameters to apply the Constraint to the Block. These figures show the bindings for the Ground, the Source and the Resistor respectively. For the Ground constraint, bind gc.pv to p.v.



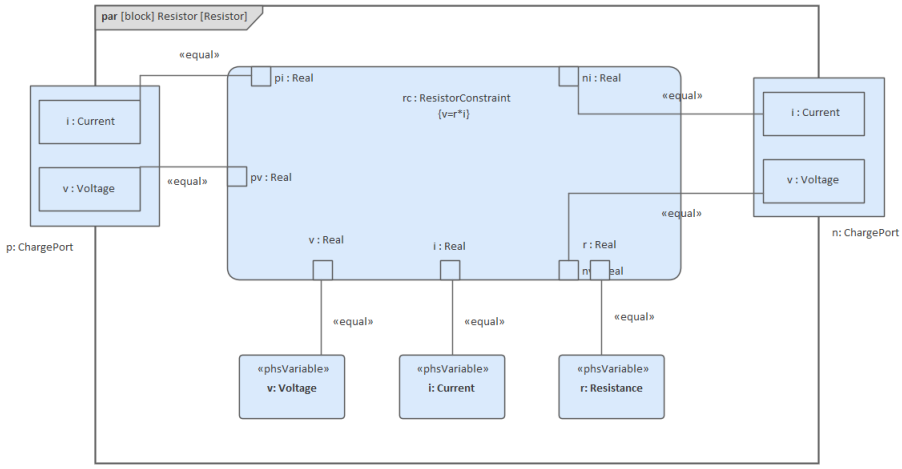
For the Source constraint, bind:

- 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.

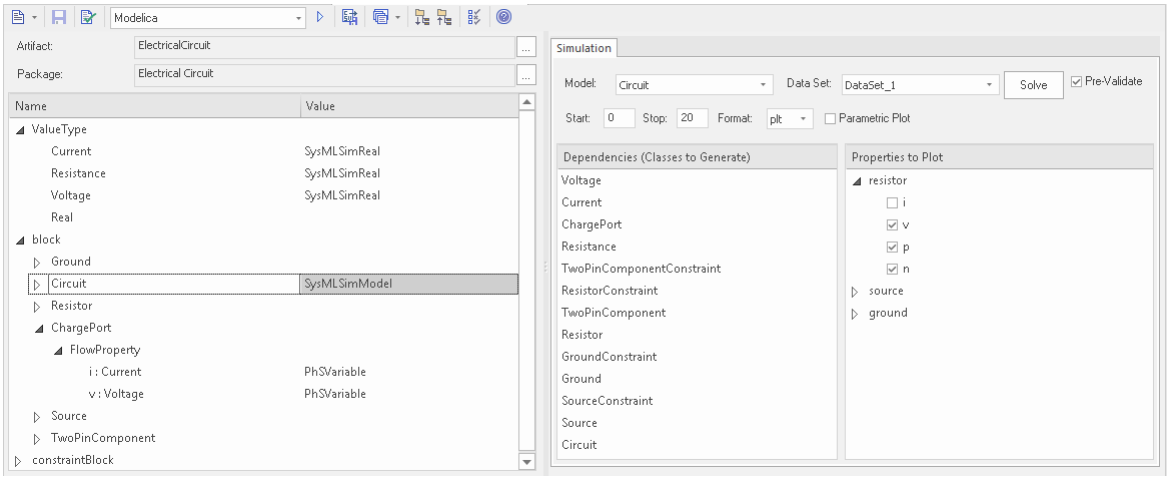
Step	Action
SysMLSimConfiguration Artifact	• Select 'Simulate > System Behavior > Modelica/Simulink > SysMLSim Configuration Manager' • From the first toolbar icon drop-down, select 'Create Artifact' and create the Artifact element • Select the Package that owns this SysML model

Create Root elements in Configuration Manager	• ValueType • Block • constraintBlock
ValueType Substitution	Expand ValueType and for each of Current, Resistance and Voltage select 'SysMLSimReal' from the 'Value' combo box.
Set property as flow	• Expand 'block' to ChargePort | FlowProperty | i : Current and select 'SimVariable' from the 'Value' combo box • For 'SysMLSimConfiguration' click on the ... button to open the 'Element Configurations' dialog • Set 'isConserved' to 'True'
SysMLSimModel	This is the model we want to simulate: set the Block 'Circuit' to be 'SysMLSimModel'.

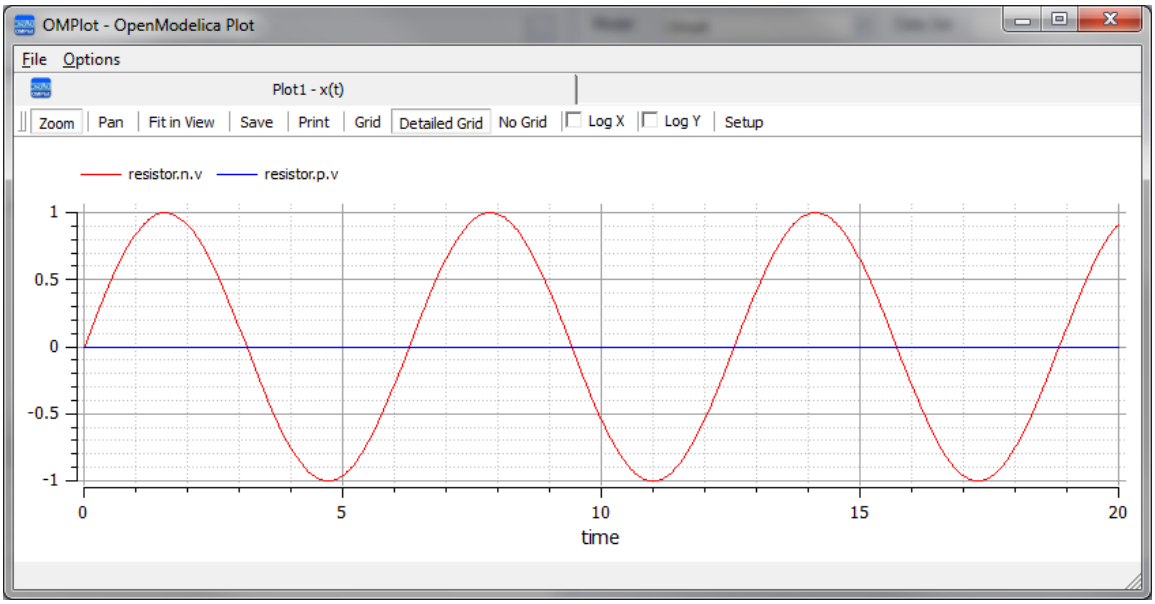
Run Simulation

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

button.



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

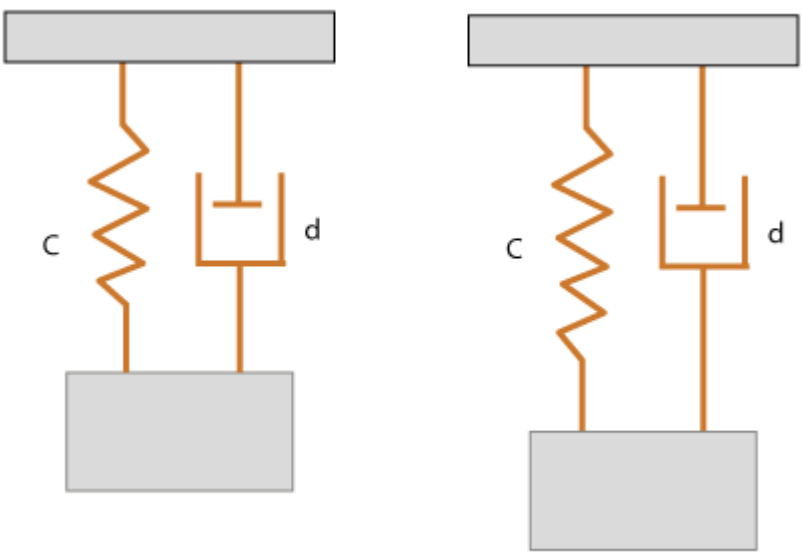


Mass-Spring-Damper Oscillator Simulation Example

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

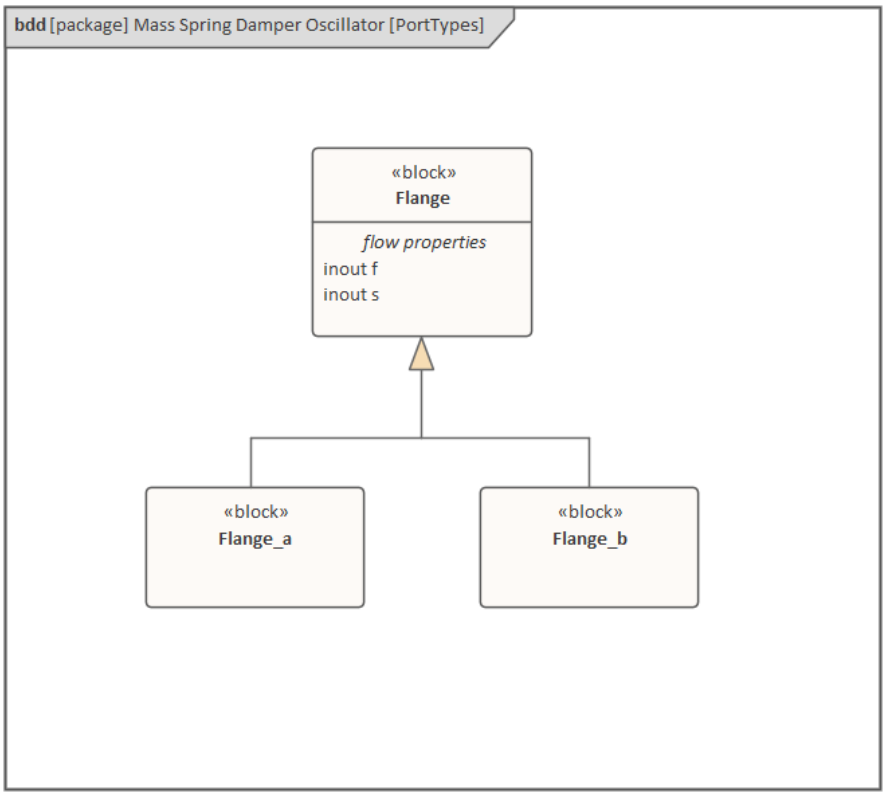
A mass is hanging on a spring and damper. The first state shown here represents the initial point at time=0, just when the mass is released. The second state represents the final point when the body is at rest and the spring forces are in equilibrium with gravity.



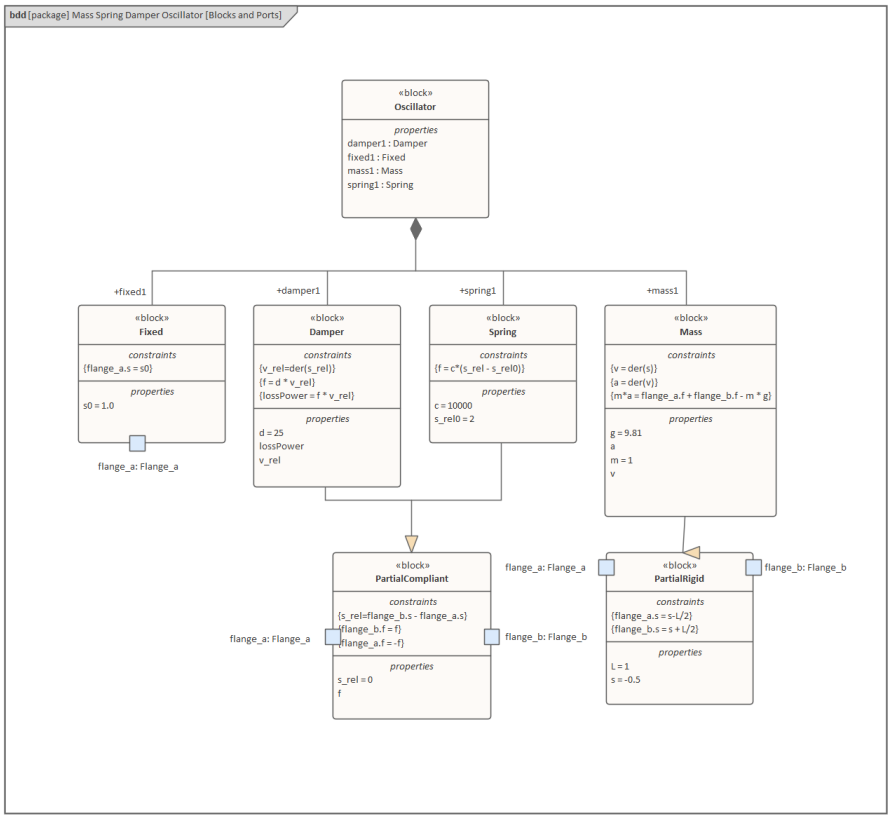
Create SysML Model

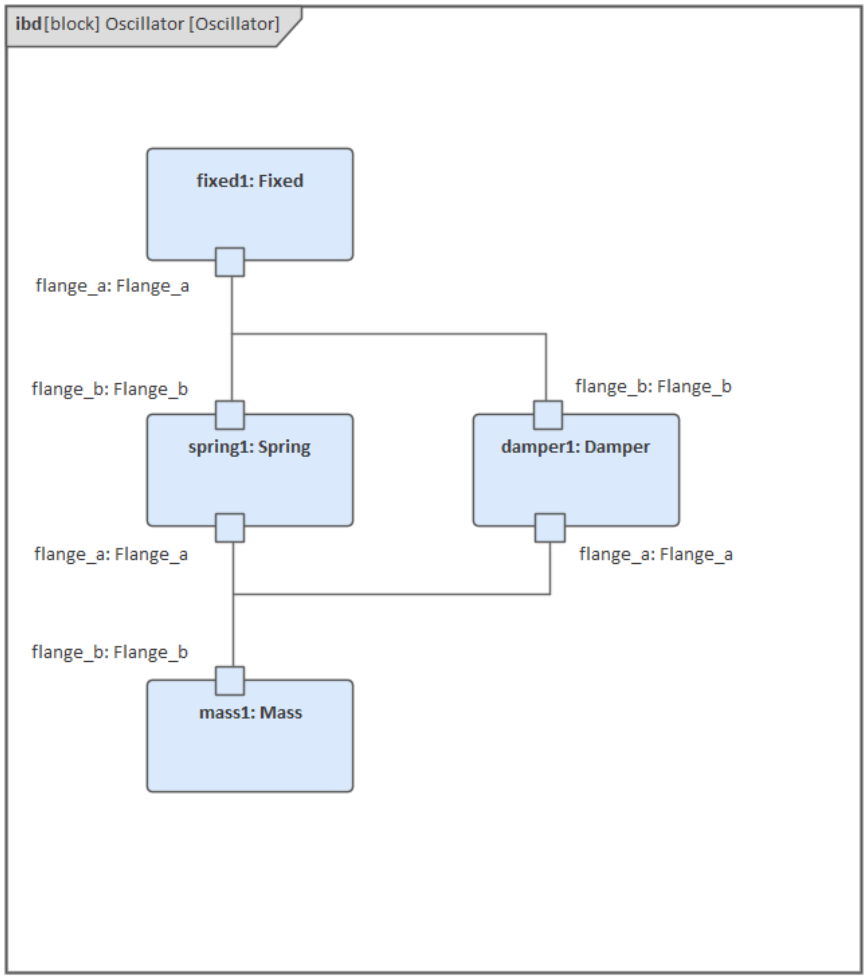
The MassSpringDamperOscillator model in SysML has a main Block, the *Oscillator*. The Oscillator has four parts: a fixed *ceiling*, a *spring*, a *damper* and a *mass body*. Create a Block for each of these parts. The four parts of the Oscillator Block are connected through Ports, which represent mechanical 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. Forces are 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 ConstraintBlocks, use constraint properties in the Blocks, and bind their parameters to the Block's properties.

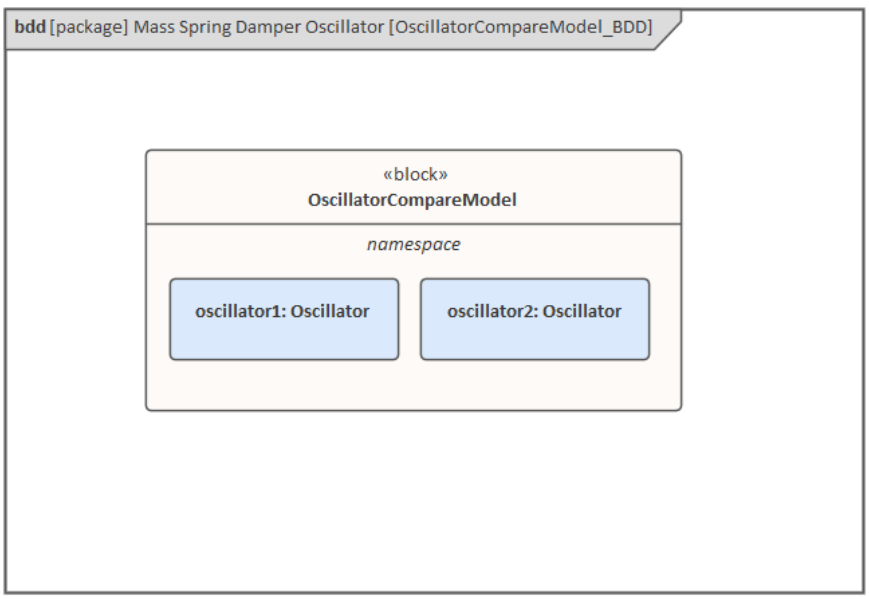
Two Oscillator Compare Plan

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*

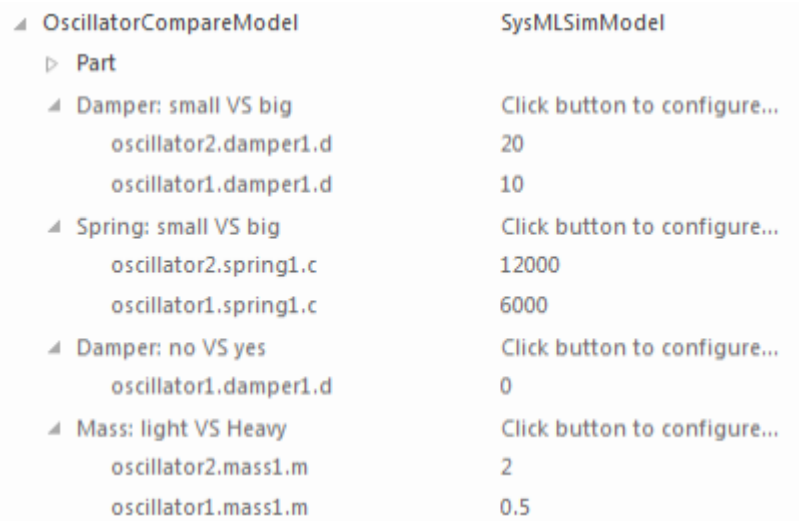


Setup DataSet and Run Simulation

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

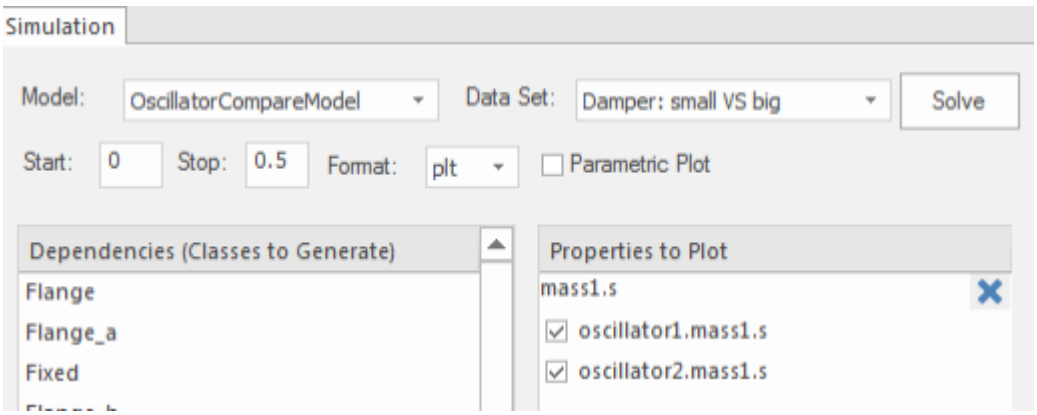
- Damper: small vs big
  provide 'oscillator1.damper1.d' with the value 10 and 'oscillator2.damper1.d' with the larger value 20
- Damper: no vs yes
  provide 'oscillator1.damper1.d with the value 0; ('oscillator2.damper1.d' will use the default value 25)
- Spring: small vs big
  provide 'oscillator1.spring1.c' 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:



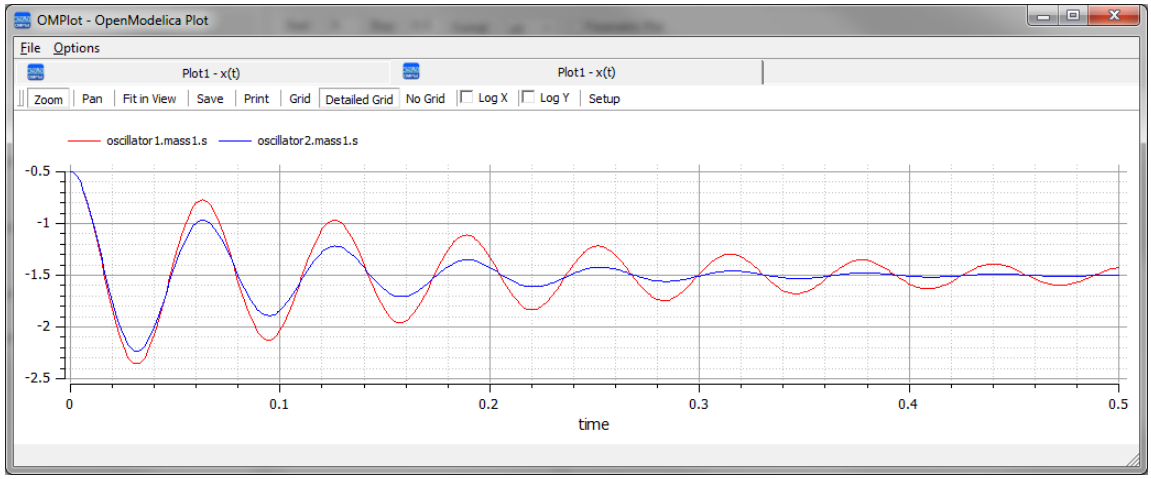
On the 'Simulation' page, select 'OscillatorCompareModel', plot for 'oscillator1.mass1.s' and 'oscillator2.mass1.s', then choose one of the created datasets and run the simulation.

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.

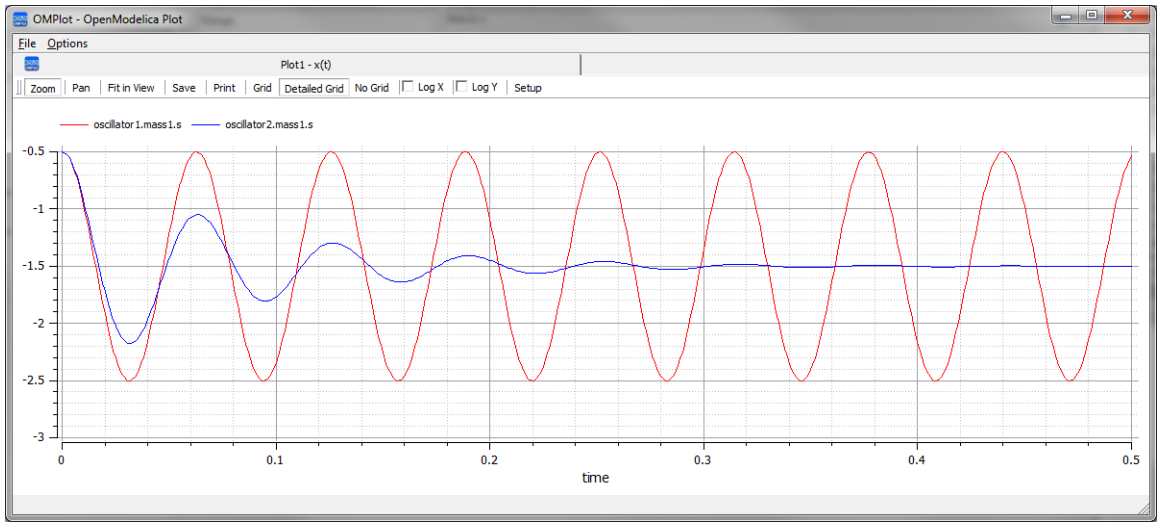


These are the simulation results:

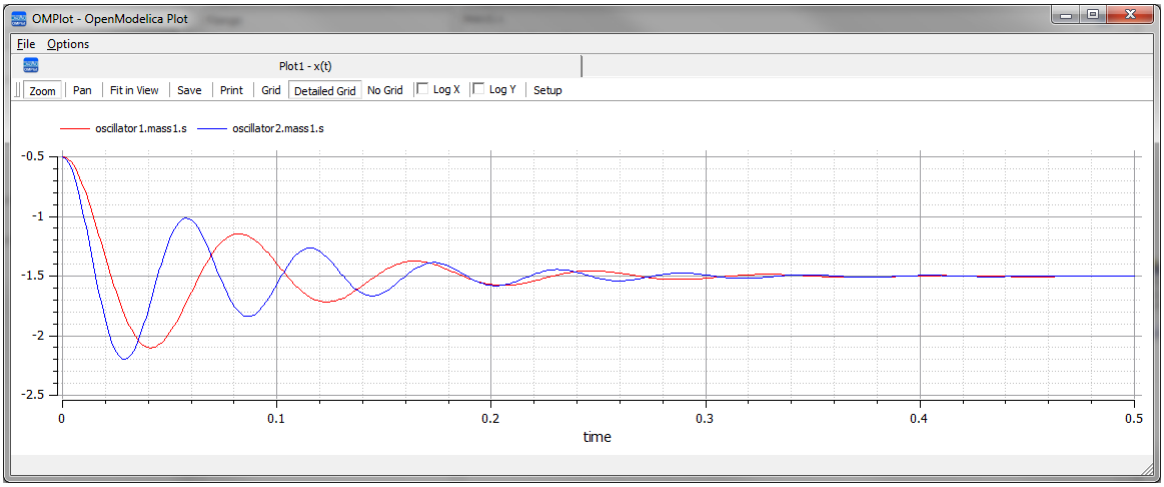
- Damper, small vs big: the smaller damper allows the body to oscillate more



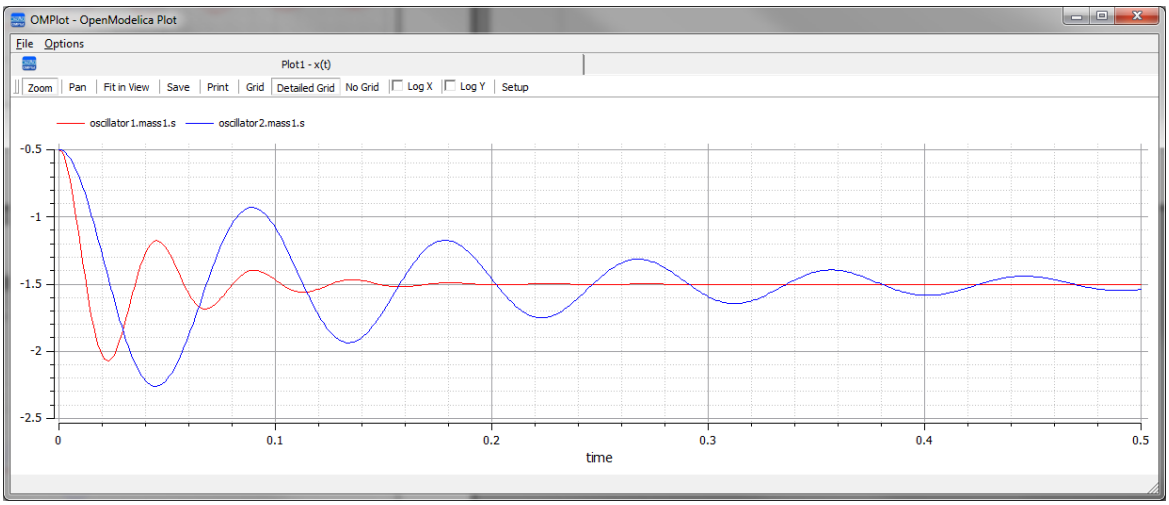
- Damper, no vs yes: the oscillator never stops without a damper



- 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 more quickly



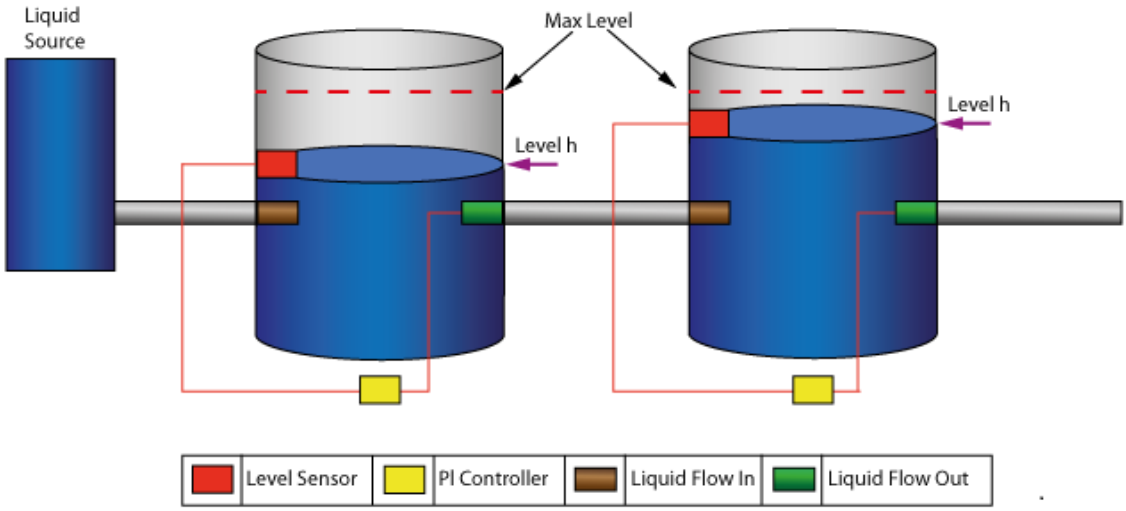
Water Tank Pressure Regulator

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.

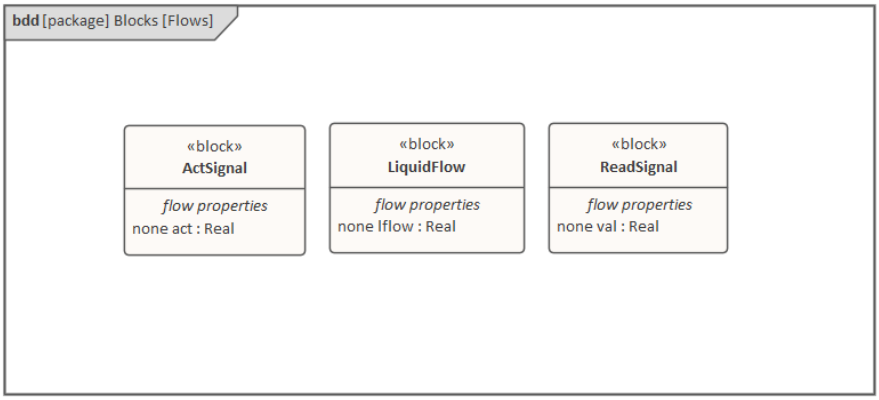
System being modeled

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, not domain-specific physical problem.



Create SysML Model

Component	Discussion
Port Types	The tank has four Ports that are typed to these three Blocks: • ReadSignal: Reading the fluid level; this has a property 'val' with unit 'm' • ActSignal: The signal to the actuator for setting valve position • LiquidFlow: The liquid flow at inlets or outlets; this has a property 'lflow' with unit 'm³/s"



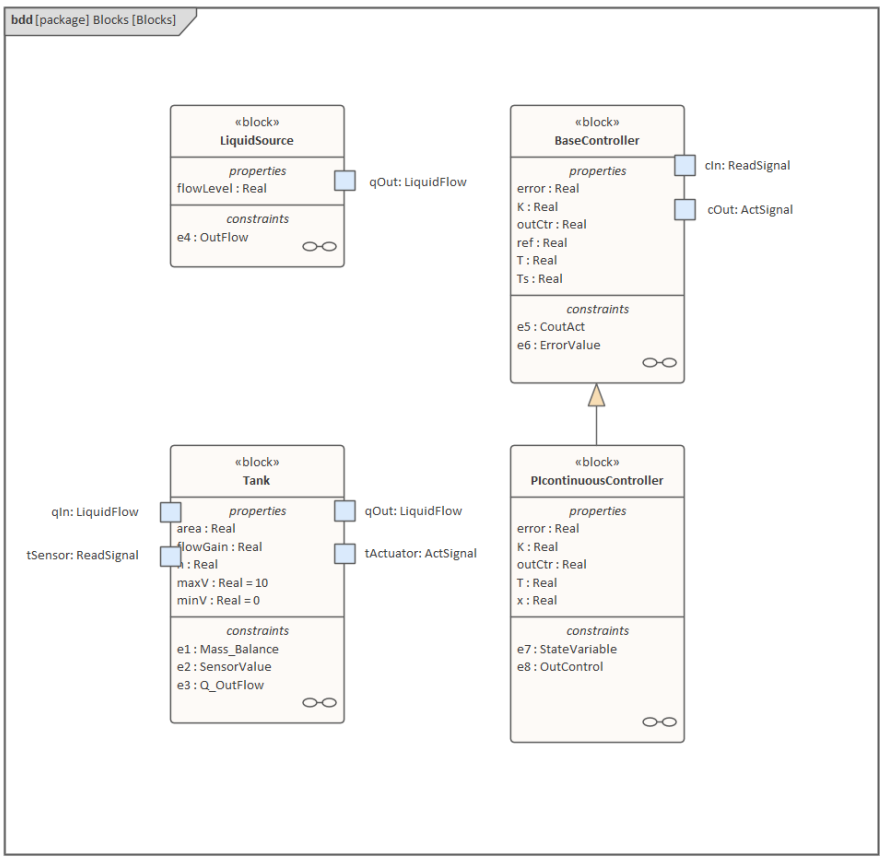
Block Definition Diagram

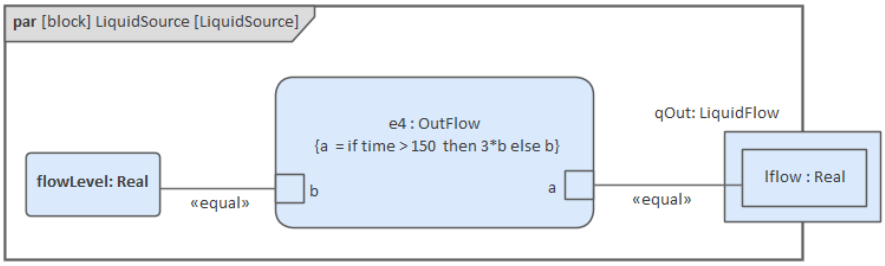
LiquidSource: The water entering the tank must come from somewhere, therefore we have a liquid source component in the tank system, with the property *flowLevel* having a unit of 'm³/s'. A Port 'qOut' is typed to 'LiquidFlow'.

Tank: The tanks are connected to controllers and liquid sources through Ports.

- Each Tank has four Ports:
  - qIn: for input flow
  - qOut: for output flow
  - tSensor: for providing fluid level measurements
  - tActuator: for setting the position of the valve at the outlet of the tank
- Properties:
  - volume (unit='m³'): capacity of the tank, involved in the *mass balance* equation

	- h (unit = 'm'): water level, involved in the *mass balance* equation; its value is read by the sensor - flowGain (unit = 'm³/s'): the output flow is related to the valve position by *flowGain* - minV, maxV: Limits for output valve flow BaseController: This Block could be the parent or ancestor 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

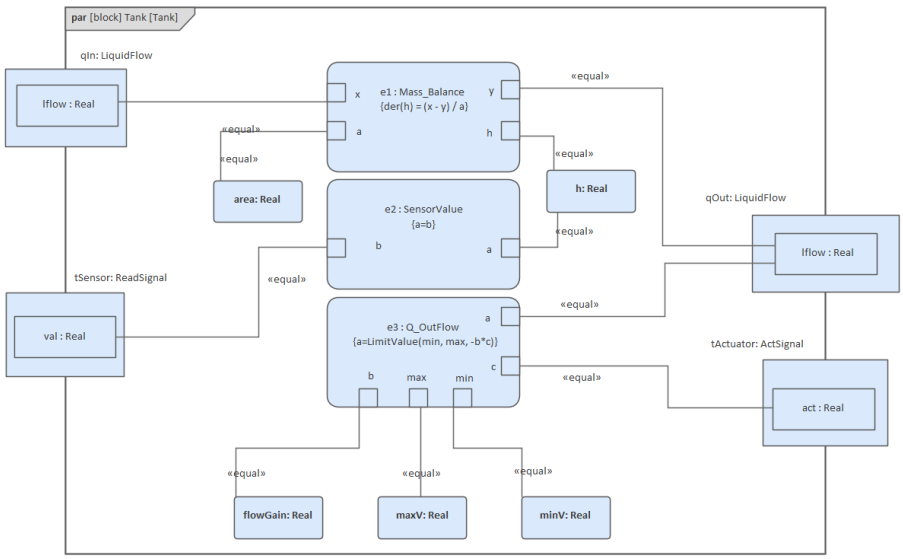
	position PIcontinuousController: specialize from BaseController • Properties: - x: the controller state variable 
ConstraintBlocks	The flow increases sharply at time=150 to a factor of three of the previous flow level, which creates an interesting control problem that the controller of the tank has to handle.



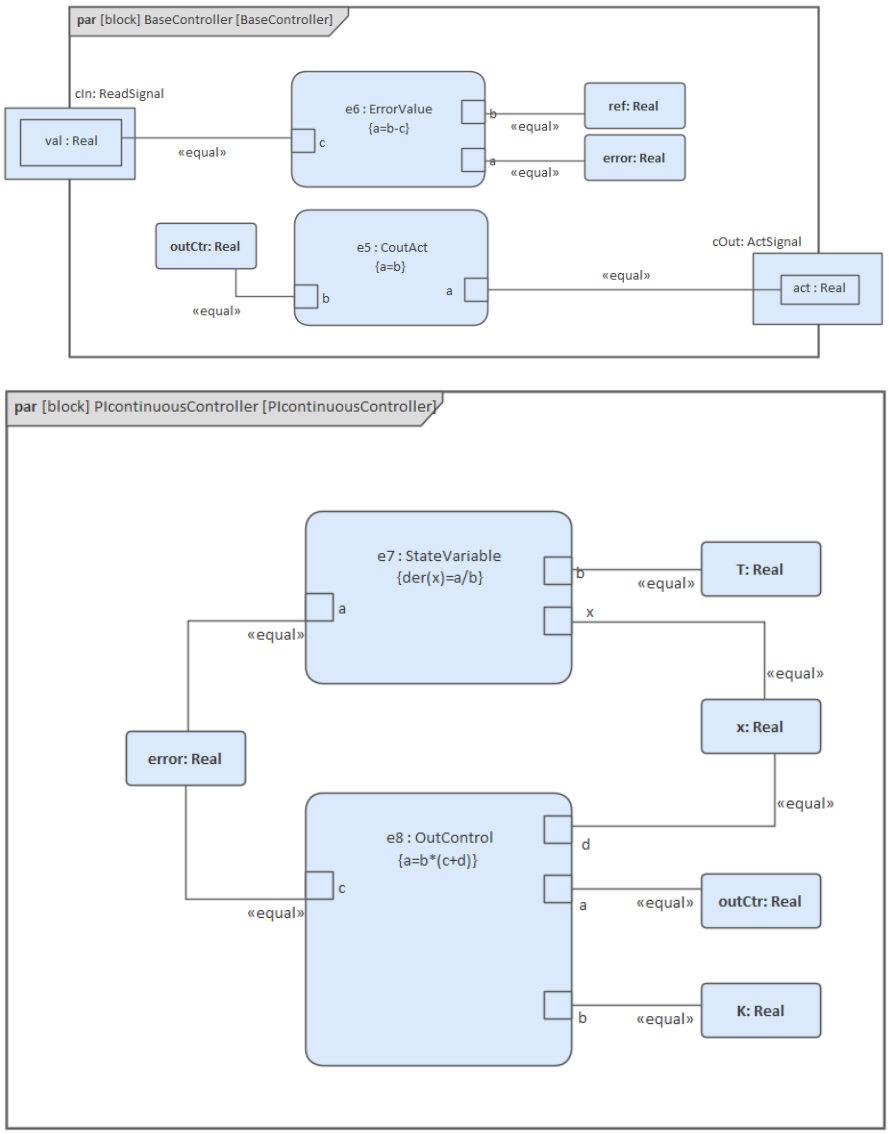
The central equation regulating the behavior of the tank is the *mass balance* equation.

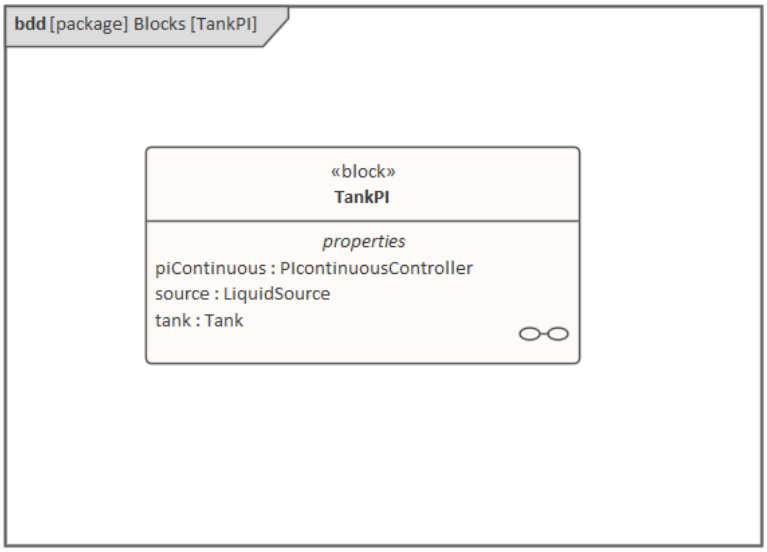
The output flow is related to the valve position by a 'flowGain' parameter.

The sensor simply reads the level of the tank.

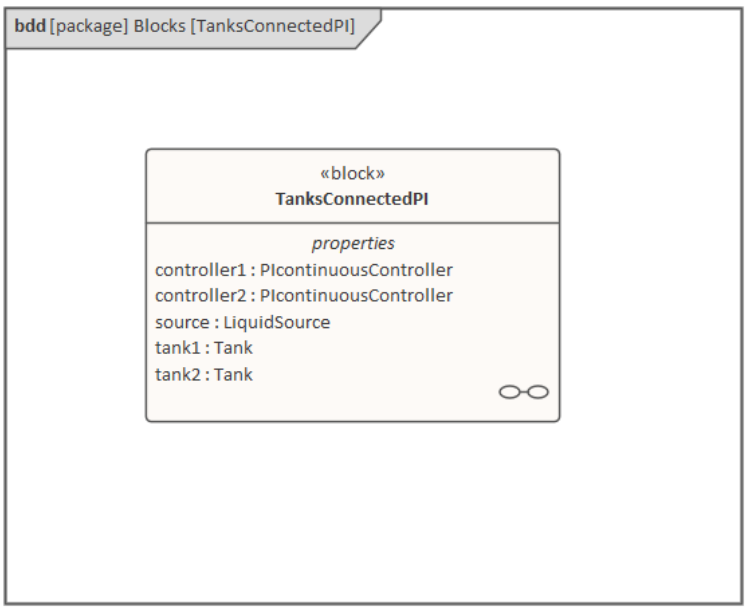


The Constraints defined for 'BaseController' and 'PIcontinuousController' are illustrated in these figures.

	
Internal Block Diagram	This is the Internal Block diagram for a system with a single tank.



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 listed 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 OpenModelica 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 several to simulate, and the Properties will be shown in the Legend for the plot

Create Artifact and Configure

Select 'Simulate > System Behavior > Modelica/Simulink > 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.

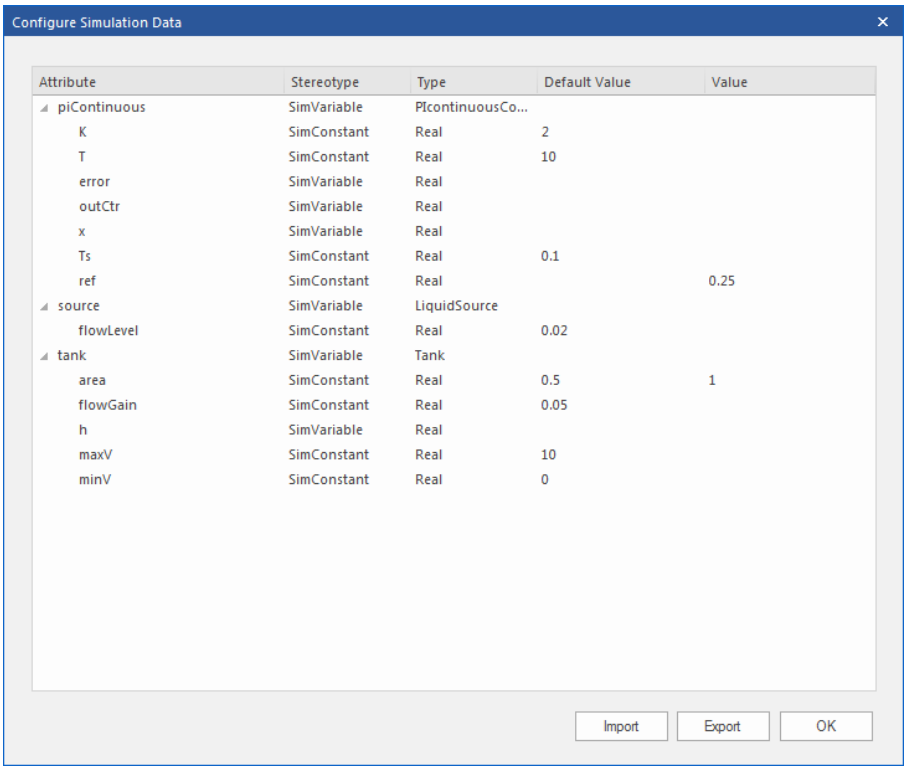
Block	Properties
LiquidSource	Configure as 'SysMLSimClass'. Properties configuration:

	• flowLevel: set as 'SimConstant'
Tank	Configure as 'SysMLSimClass'. Properties configuration: • area: set as 'SimConstant' • flowGain: set as 'SimConstant' • maxV: set as 'SimConstant' • minV: set as 'SimConstant'
BaseController	Configure as 'SysMLSimClass'. Properties configuration: • K: set as 'SimConstant' • T: set as 'SimConstant' • Ts: set as 'SimConstant' • ref: set as 'SimConstant'
PIcontinuousController	Configure as 'SysMLSimClass'.
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.

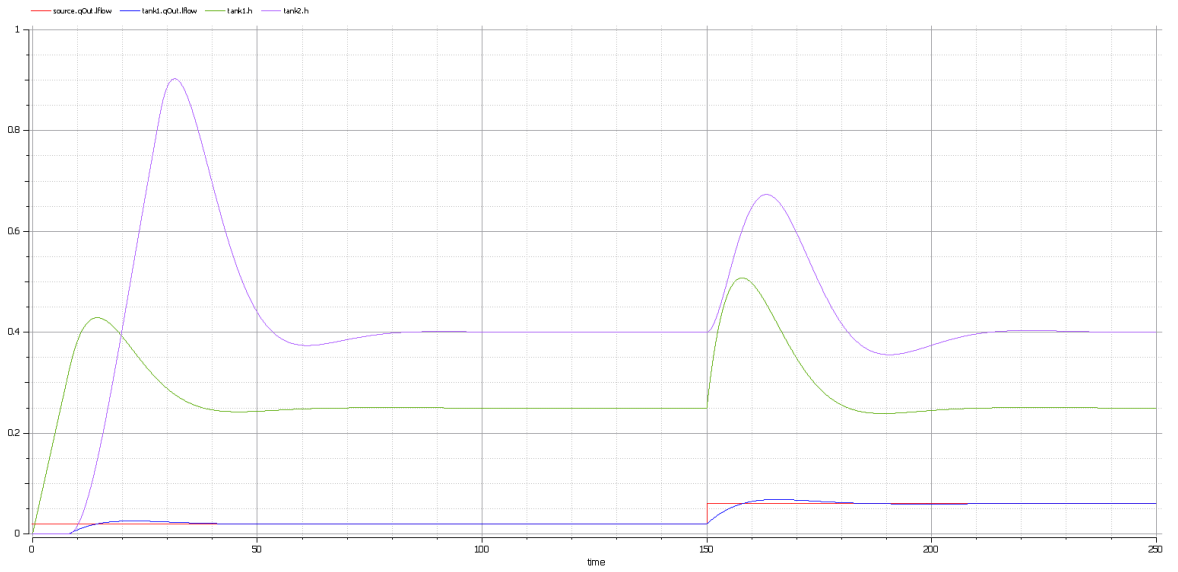
Element	Dataset
LiquidSource	flowLevel: 0.02
Tank	h.start: 0 flowGain: 0.05 area: 0.5 maxV: 10 minV: 0
BaseController	T: 10 K: 2 Ts: 0.1
PIcontinuous Controller	No configuration needed. By default, the specific Block will use the configured values from super Block's default dataSet.
TankPI	What is interesting 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.  Click on the OK button and return to the Configuration Manager. Then these values are configured: • tank.area: 1 this overrides the default value 0.5 defined in the Tank Block's data set • piContinuous.ref: 0.25
TanksConnectedPI	• controller1.ref: 0.25 • controller2.ref: 0.4

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:

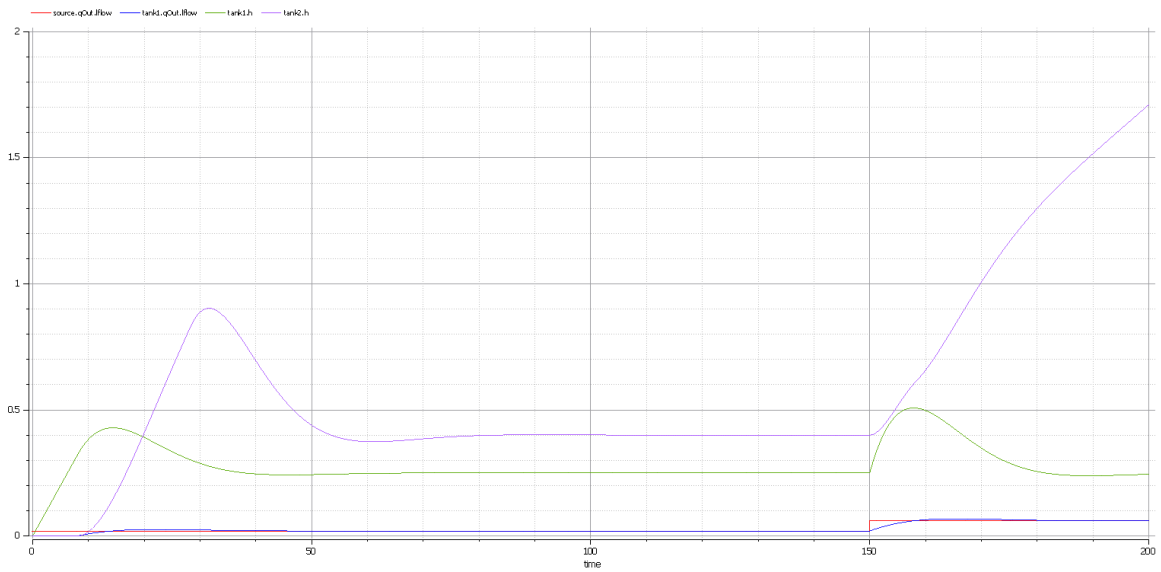
- The liquid flow increases sharply at time=150, to 0.06 m^3/s, a factor of three of the previous flow (0.02 m^3/s)
- Tank1 regulated at height 0.25 and tank2 regulated at height 0.4 as expected (we set the parameter value through the data set)
- Both tank1 and tank2 regulated twice during the simulation; the first time regulated with the flow 0.02 m^3/s; the second time regulated with the flow 0.06 m^3/s
- Tank2 was empty before there was any flow 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 of 10 m^3/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^3/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 results:

- Our change only applies to tank2; tank1 can regulate as before on 0.02 m^3/s and 0.06 m^3/s
- When the source flow is 0.02 m^3/s, tank2 can regulate as before
- However, when the source flow increases to 0.06 m^3/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.