Enterprise Architect

User Guide Series

Mathematical Simulations

Author: Sparx Systems
Date: 2026-05-28
Version: 17.1

Table of Contents

Mathematical Simulations

Enterprise Architect provides a wide range of options for introducing advanced mathematical tools and capabilities into your simulations.

You can bring the power of integrated external tools such as MATLAB into your models through the use of Solver Classes, and can also export your models for execution in other external tools such as MATLAB Simulink, Stateflow and Simscape, or OpenModelica.

Enterprise Architect includes an extensive library of mathematical functions within the JavaScript engine, providing the benefits of a significantly expanded Simulation capability.

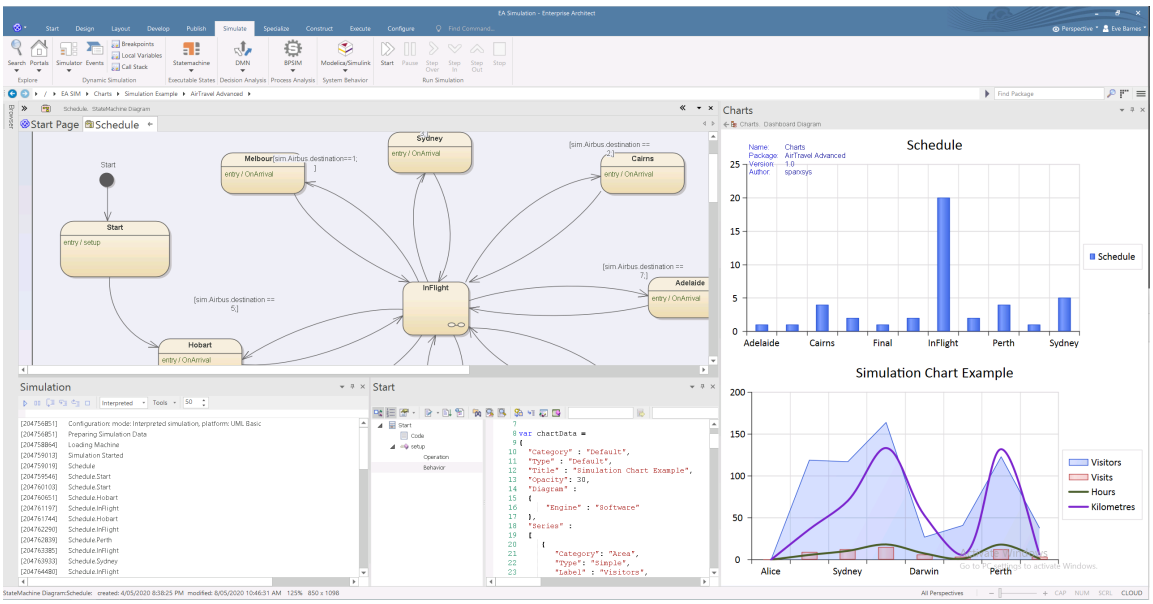
Enterprise Architect also provides a wide range of Dynamic Charts; without the need for external tools, you can configure these Charts to extract and plot information from Simulations that have been directly executed inside Enterprise Architect.

Explore the:

- Solver classes in Enterprise Architect that call MATLAB or Octave to incorporate complex mathematics into your model-based simulations
- Extensive internal Math Library based on the popular Cephes function library
- Integration with the OMG SysPhS standard, enabling you to configure your model for export to common tools
- Support for exporting models to MATLAB Simulink,

Simscape and Stateflow; you can create your model in Enterprise Architect and execute it in MATLAB

- Extensive support for Modelica; you can create and configure your model in Enterprise Architect and execute it in Modelica
- Presentation of the results of your modeling and simulation in Chart formats, either within a dedicated graphics presentation tool or through the Dynamic Charting facilities of Enterprise Architect



Available Integrations

Product	Description
MATLAB	MATLAB is a popular and widely-used numerical computing environment and programming language, developed by MathWorks. It provides a wealth of mathematical expressions and formulae

	that can be processed within the application itself or called into other applications such as Enterprise Architect. Enterprise Architect's MATLAB integration connects via the MATLAB API, allowing your Enterprise Architect simulations and other scripts to execute based on the values of the selected MATLAB functions and expressions. You can call MATLAB through the Solver classes, or export your model to MATLAB Simulink, Simscape and/or Stateflow. Note: Integration with MATLAB requires MATLAB version R2018b or higher.
Simulink	Simulink is a core MATLAB application, for running SysML simulations of directed messages between Blocks. Enterprise Architect can translate a SysML model into the Simulink format, automatically run the simulation, and plot the outputs of the selected variables as Charts. You can also open the generated Simulink file directly in Simulink, allowing you to modify and fine-tune the simulation settings and output functionality. You can drag-and-drop common built-in

	Simulink library Blocks directly from the Enterprise Architect Simulink patterns, or reference your own custom-built Blocks with new SysPhS standard stereotype parameters. Simulink is an alternative option to OpenModelica for developing and running simulations in Enterprise Architect.
Simscape	Simscape is an optional extension to MATLAB Simulink, allowing you to model physical systems and instruct MATLAB to simulate and plot the requested outputs, using Simscape's vast array of library Blocks across many different physical domains. Enterprise Architect can translate SysML Internal Block diagrams into Simscape.
Stateflow	Stateflow is also an optional extension to MATLAB Simulink, providing the ability to generate MATLAB Stateflow diagrams to be run under Simulink. Within Enterprise Architect, this helps you to guide your SysML simulations using StateMachines modeled in Enterprise Architect, which are translated to Stateflow diagrams.

Modelica	Modelica is an open language standard for modeling, simulating, optimizing and analyzing complex dynamic systems. It defines and provides a file structure that can be accessed and operated on by applications such as OpenModelica (free open source) and Dymola and Wolfram Modeller (commercially available; these can work with Enterprise Architect but have not been tested or integrated with Sparx Systems software).
OpenModelica	OpenModelica is a free and open source environment based on the Modelica open language standard; OpenModelica enables you to read, edit and simulate Modelica files. Enterprise Architect is integrated with OpenModelica, and supports its use under the SysPhS Standard for defining constants and variables in the simulation of StateMachine diagrams and Parametric diagrams. You can also display the SysML Block diagrams from your models in Enterprise Architect in the OMEdit - OpenModelica Connection Editor, which displays the Blocks' aliases and notes.

	OpenModelica is an alternative option to Simulink for developing and running simulations in Enterprise Architect.
GNU Octave	GNU Octave is a library of mathematical functions. From Enterprise Architect's JavaScript engine, you can integrate with an Octave interpreter to use any of the available Octave functions. Octave provides an alternative to MATLAB functions, with a special emphasis on sequences and matrices.
JavaScript Math Library	The JavaScript Math Library is an implementation of the Cephes mathematical library built directly into JavaScript within Enterprise Architect, to facilitate the use of advanced mathematical functions within a scripted Simulation (or within a Dynamic Chart, Model based Add-In or many other scenarios).

Solvers

Product	Description

The Solver Class	The Solver Class provides a common API to a variety of external tools; it is available in any JavaScript engine used by Enterprise Architect, and is of particular value in calling in mathematical functions from MATLAB or Octave. You can review the results of processing within the external tool, or bring them into the JavaScript engine for presentation within Enterprise Architect.
MATLAB	The MATLAB solver is available when MATLAB is installed on your computer. The solver uses the MATLAB API to provide access to the wide array of available MATLAB functions.
Octave	The Octave solver is available when Octave is installed on your computer. The solver directly communicates with the Octave interpreter to allow you to access functions and data within an Octave environment.

Configuring Simulations

Type	Description

Configuration Artifacts	The SysMLSim Configuration Artifact is a purpose-designed Artifact for specifying the characteristics and parameters of a SysML simulation in Enterprise Architect. You set up the specification through the Configure SysML Simulation window.
The SysML Extension for Physical Interaction and Signal Flow Simulation (SysPhS) Standard	The SysPhS Standard provides an easier, model-based method of sharing simulations, defining variables, constants and initial values within each element rather than through a configuration file. This enables a visual approach to setting up a simulation, as the variables, constants and initial values can be made visible in diagrams in additional compartments on the SysML Blocks.
Define Multiple Datasets	Multiple datasets can be defined against SysML Blocks used within a Simulation configuration in a Parametric model. This allows repeatable simulation variations using the same SysML model.

Common Use Cases

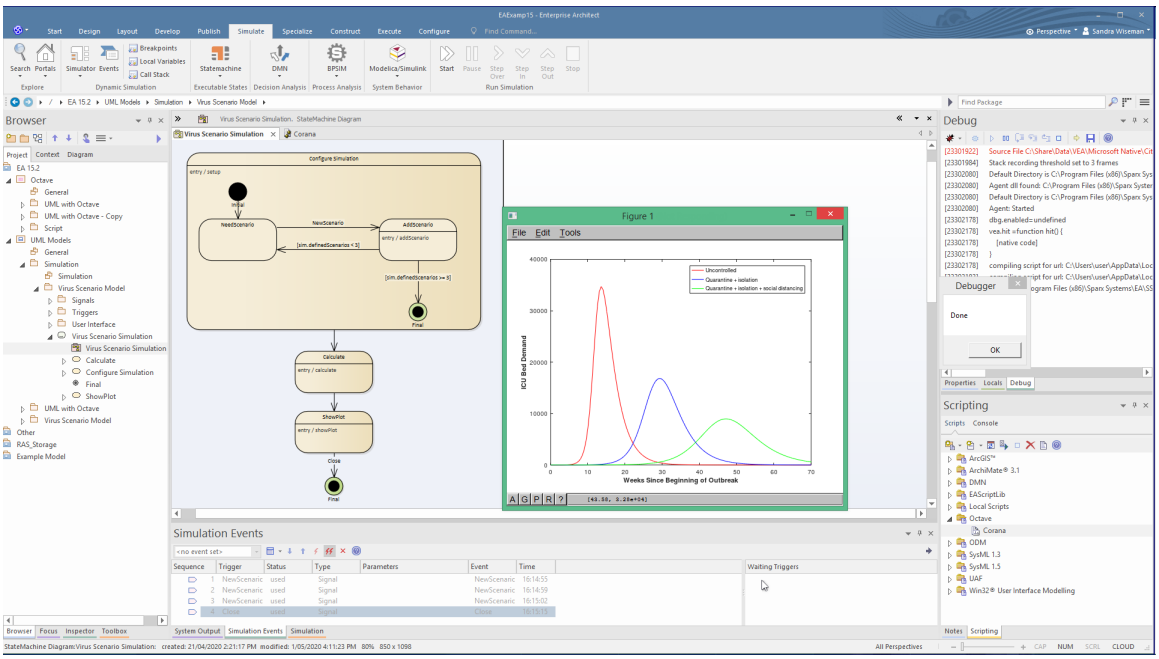
Name	Description
Solver Consoles	• Quickly test out commands to be used in a script or simulation • Call a MATLAB function to see if it returns what you expect and runs without errors • Cut and paste a JavaScript snippet that only ever needs to be run once, rather than creating a script and then deleting it
Solvers in Simulations	• Call a complex mathematical function that was defined as an Octave function • Call MATLAB's API routines to determine a decision flow
SysML SysPhS Simulation	• Model a new automotive ABS system in Enterprise Architect SysML and simulate using Simulink • Design and model a hydraulics system in Enterprise Architect and simulate the system in OpenModelica using an existing library of Modelica components
Generate a StateChart and Refine	• Create a SysML StateMachine to quickly define the action of a user turning the system on and off

and Debug in StateFlow	repeatedly; generate the simulation and open in Stateflow to view the state parameters in 'real-time' and tweak Stateflow settings
Model and Test a StateChart	• Fully model a StateChart before simulating it in Stateflow

Solvers

From Enterprise Architect Release 15.2 onwards you can use JavaScript to invoke 'Solvers' that provide integration with external tools such as MATLAB and Octave. The Solver Class provides a real-time connectivity to your external mathematical tool, through which you call complex and high order mathematical functions during the execution of Simulations within Enterprise Architect.

Solvers help you bring the computational power of MATLAB and Octave into Enterprise Architect in simulations, model-based Add-Ins and custom scripts. The Solver Class is a built-in construct within Enterprise Architect's JavaScript engine.



Getting Started with Solvers

The Solver Class variable can be created once per simulation or, if required, multiple Solver Class instances can be created. Note that starting a new Solver Class instance can sometimes take many seconds. It is generally recommended to create one Solver instance at the start of the simulation and reuse it when required.

To use the Solver Class, you need to have a knowledge of the functions available in your preferred Math Library and the parameters they use, as described in the Library product documentation.

You first define the Math Library (or libraries) that you intend to use in your script. For MATLAB you type:

```
var matlab = new Solver('matlab');
```

For Octave you type:

```
var octave = new Solver('octave');
```

Then, wherever you need to use a maths function within your script, for MATLAB you type:

```
matlab.exec('complexMathsFunction', parameters);
```

For Octave, you type:

```
octave.exec('complexMathsFunction', parameters);
```

These two lines of script execute the function within the appropriate tool and display the results there. If you want to bring the results back into Enterprise Architect, you precede the line with:

```
var resultFrom'Toolname';
```

That is:

```
var resultFromMatlab =
matlab.exec('complexMathsFunction', parameter1,
parameter2);
```

or

```
var resultFromOctave =
octave.exec('complexMathsFunction', parameter1,
parameter2);
```

Being part of the JavaScript engine, the Solver Classes are also immediately accessible to Add-In writers creating model-based JavaScript Add-Ins.

Note: If a new Solver is created in a section of JavaScript that is called multiple times, simulation performance will be greatly decreased (for example, on a StateMachine node that is entered multiple times).

See the *GNU Octave Solver* and *MATLAB Solver* Help topics for further information on these two Solvers.

Solver Constructor

Constructor	Description
Solver(string solverName)	Creates a new Solver connected to a new instance of the specified helper application.

Solver Methods

Method	Description
get(string name)	Retrieves a named value from within the solver environment.
set(string name, object value)	Assigns a new value to a named variable in the solver environment.
exec(string name, string arguments, int returnValues)	Executes a named function. The actual functions will depend on the type of solver being used.

Solver Consoles

The Solver Console is an easily-accessed and simple-to-use editor. It is primarily for managing MATLAB and Octave Solvers, providing simple and quick access to test functions supported in these external applications. The Solver Console can be used for creating, checking and maintaining calls used in simulations, as well as in scripts written using JavaScript.

Access

Ribbon	Simulate > Console > Solvers > MATLAB Simulate > Console > Solvers > Octave

Solver Initialization

The chosen Solver is set automatically to one of these:

MATLAB:

- var matlab = new Solver("matlab");

Octave:

- var octave = new Solver("octave");

If the background Octave or MATLAB is accessible, the Console will prompt with the starting detail and be ready to

accept any input.

Usage

The Console accepts single line JavaScript commands that will be executed one at a time. Enter new commands in the bottom text-entry section and press the Enter key to execute the command.

The executed command will be added to the main output panel, with any output from the command.

The Solvers can then be accessed using the Solver Class functions; for example:

MATLAB:

- matlab.set('<variable_name>', <value>)
- matlab.get('<variable_name>')
- matlab.exec(<function>, <parameters>)

Octave:

- octave.set('<variable_name>', <value>)
- octave.get('<variable_name>')
- octave.exec(<function>, <parameters>)

For more details see the *Octave Solver* and the *MATLAB Solver* Help topics.

Previous commands that have been entered can be re-used by pressing the Up Arrow key or Down Arrow key.

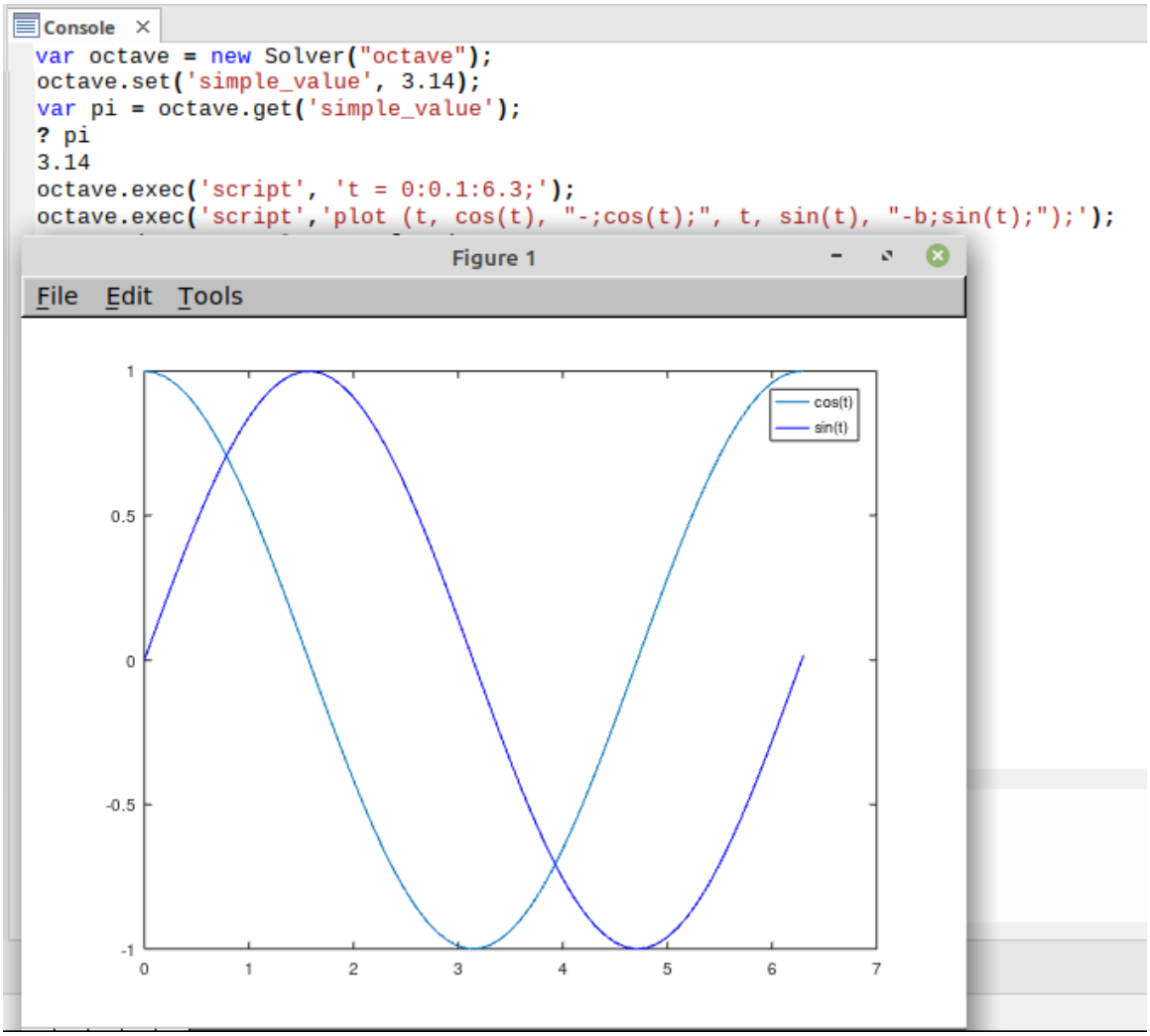
Output

To output the value of a variable or a function you can use:

- ? <variable_name>
- ? <function>

Example

This illustration shows some examples of using the Octave Solver's Set, Get and Exec, as well as the output using ?, in the Octave Console:



The Octave plot is the generated output of the two ocatave.exec() statements.

Console Commands

Console commands are preceded by the ! character and instruct the Console to perform an action. The Console commands include:

- !clear - clears the Console display
- !save - saves the Console display to a file
- !help - prints a list of commands

- !close - closes the Console
- !include <scriptname> - executes the named script item; scriptname is of the format GroupName.ScriptName (spaces are allowed in names)
- ? - lists commands (same as !help)
- ? <variable or function> - outputs the value

Solvers in Simulations

A Solver within a simulation helps you to include behavior from mathematical models, querying the state as necessary to adjust the behavior of the corresponding UML model.

Creating and Initializing a Solver

A Solver is best created once at the start of a simulation. This allows it to be used throughout the simulation without incurring the delay of repeated construction time. This is also a good time to load any required modules, define functions and define an initial state for the model.

In a StateMachine simulation, good places to perform this initialization are in the Effect for the Transition leading from the Initial State, or in the Entry for a State that is not re-entered during simulation.

In an Activity simulation, you will need to add the Solver to the Effect for an Action.

Updating a Solver Model

As a simulation progresses you can use any of the Effect fields to update parameters of the mathematical model. That might consist of updating rates or changing between algorithms. In a StateMachine simulation:

- A Transition Effect can be used when the change is needed for a particular flow

- A State Entry can be used to ensure the behavior is changed while the simulation is in a State, regardless of how it gets there; this includes any nested States
- A State Exit can be used to ensure the behavior is changed any time the simulation leaves a State, regardless of how it leaves
- A State Do Activity behaves similarly to an Entry action.

In an Activity simulation there are only Action Effects.

Querying a Solver Model

With a Transition guard, call *solver.get()*. You can also call *solver.exec()* to call a function with no side effects.

GNU Octave Solver

GNU Octave is a library of mathematical functions, with a special emphasis on sequences and matrices. You can call each function into a script written in JavaScript.

You can invoke arbitrary mathematical functions from Octave at run time, using a simple construct called a Solver Class, written in JavaScript; a Solver Class for Octave can call into the external Octave tool and link the advanced mathematical functions directly into your running simulation. For example:

```
var octave = new Solver(octave);
var resultFromOctave =
octave.exec('complexMathsFunction', parameter1,
parameter2);
```

See the Help topic *Solvers in Simulations*.

Features include:

- Calling in vectors, matrices, numbers, and strings from Octave
- Octave vectors return as JavaScript one-dimensional arrays (and JavaScript one-dimensional arrays return as Octave vectors)
- Octave matrices return as JavaScript two-dimensional arrays (and JavaScript two-dimensional arrays return as Octave matrices)
- You can retrieve variable values from Octave using:
  octave.get(<name>)

- You can call any Octave function with JavaScript values using:
  octave.exec(<name>, [<arguments>], 0/1)
- All arguments are passed inside a JavaScript array
- You can also use the result in JavaScript; pass 1 if you want a result, 0 if you do not want a result
- You can execute any Octave statement using:
  octave.exec("script", <statement>, 0/1)

Octave is available as an alternative to the MATLAB library and can be used in all the same contexts as MATLAB.

Setup and Configuration

After you install Octave, Enterprise Architect will read the location from the registry to provide automatic integration.

Usage

Task	Description

Assigning Values	Use the octave.set command; for example:
	octave.set("simple_value", 3.14);
	octave.set("example_sequence", [0, 1, 2]);
	octave.set("identity", [[1, 0], [0, 1]]);

Constructing	Within the JavaScript editor, create a new Solver connected to Octave by passing "octave" to the Solver constructor. var octave = new Solver("octave");

Retrieving Values	Use the octave.get command; for example:
	var simple_value = octave.get("simple_value");
	var example_sequence = octave.get("example_sequence");
	var identity= octave.get("identity");

Calling Functions	Pass the name of the function as the first argument to Solver.exec. Pass all the parameters to that function in an array as the second argument. When you want a value returned by that function inside JavaScript, pass a non-zero value as the third argument. For example: var sequence = octave.exec("linspace", [0, 10, 1001],1);
Executing Statements	Pass 'script' as the name for the first argument to Solver.exec. Pass an entire Octave statement in a string as the second argument.

	octave.exec('script', 't = 0:0.1:6.3;'); octave.exec('script','plot (t, cos(t), "-;cos(t);", t, sin(t), "-b;sin(t);");');

Trace()	The Trace(statement) method allows you to print out trace statements at any arbitrary point in your simulation. This is an excellent means of supplementing the Simulation log with additional output information during execution.
	The Trace() output is written to the Simulation window.
	This is an example of using the Trace() method:
	octave.set('simple_value', 3.14);
	var pi = octave.get('simple_value');
	Trace("PI = " + pi); // output that value to the Simulation Window

Videos

Sparx Systems provide a YouTube video of using a Solver to generate a plot with Octave. See:

- Plotting with Octave

MATLAB Solver

MATLAB is a numerical computing environment and programming language that includes a large library of mathematical functions, each of which can be called from a script written in JavaScript.

You can invoke arbitrary mathematical functions from MATLAB at run time using a simple construct, called a Solver Class, written in JavaScript. A Solver Class for MATLAB can call into the external MATLAB tool via its API and link the mathematical functions directly into your running simulation. For example:

```
var matlab = new Solver("matlab");
var resultFromMatlab =
matlab.exec('complexMathsFunction', parameter1,
parameter2);
```

See the *Solvers in Simulations* Help topic.

Features include:

- Retrieving vectors, matrices, numbers, and strings from MATLAB
- MATLAB vectors return as JavaScript one-dimensional arrays (and JavaScript one-dimensional arrays return as MATLAB vectors)
- MATLAB matrices return as JavaScript two-dimensional arrays (and JavaScript two-dimensional arrays return as MATLAB matrices)
- You can retrieve variable values from MATLAB using:

matlab.get(<name>)

- You can call any MATLAB function with JavaScript values using:
  matlab.exec(<name>, [<arguments>])
- All arguments are passed inside a JavaScript object
- You can also use the result in JavaScript
- You can execute any MATLAB statement using:
  matlab.exec("script")

MATLAB is available as an alternative to the GNU Octave library and can be used in all the same contexts as GNU Octave.

Note: Integration with MATLAB requires MATLAB version R2018b or higher.

Setup and Configuration

After you install MATLAB, Enterprise Architect will read the location from the registry to provide automatic integration.

If MATLAB does not load automatically, set the path (as in the *Configure SysML Simulation Window* Help topic) to the value obtained by running 'matlabroot' in the MATLAB console.

Usage

Use	Discussion

Constructing	Create a new Solver connected to MATLAB by passing *'matlab'* to the Solver constructor. That is: var matlab = new Solver('matlab');

Assigning Values	Assign values using the matlab.set function. For example:
	matlab.set('simple_value', 3.14);
	or
	var myString = "this is an example string";
	matlab.set('myString', myString);

Retrieving Values	Retrieve results from MATLAB using the matlab.get function. For example:
	var simple_value = matlab.get('simple_value');
	var myString = matlab.get('myString');

Calling Functions	Pass the name of the function as the first parameter to Solver.exec. Either: • For functions that take a single parameter, simply pass the value as the second parameter; for example: var result = matlab.exec('ceil', 7.4); or

	• If two or more parameters are required, pass all the parameters as a JavaScript Object as the second parameter; this can be done inline such as:
	var result = matlab.exec('minus', {0: 8, 1: 4.5});
	Note: the order of the parameters is determined by the alphabetical order of the object names
	A helper function can be used here to avoid errors:
	// Wrap a variable number of arguments into an object to be passed to solver.exec
	function args() {
	var obj = {};
	for (var i = 0; i < arguments.length; i++) {
	obj[i] = arguments[i];
	}
	return obj;
	}
	var result = matlab.exec('minus', args(8, 4.5));

Trace()	The Trace(statement) method allows you to print out trace statements at any arbitrary point in your simulation. This is

	an excellent means of supplementing the Simulation log with additional output information during execution.
	The Trace() output is written to the Simulation window. For example:
	matlab.set('simple_value', 3.14);
	var pi = matlab.get('simple_value');
	Trace("Simple Value = " + pi);

Videos

Sparx Systems provide a YouTube video of using the MATLAB console to create a MATLAB Solver. See:

- The MATLAB Console

Two further videos illustrate the use of the MATLAB Solver in performing a StateMachine Simulation of a Vaccine Trial, and a Lottery Draw. See:

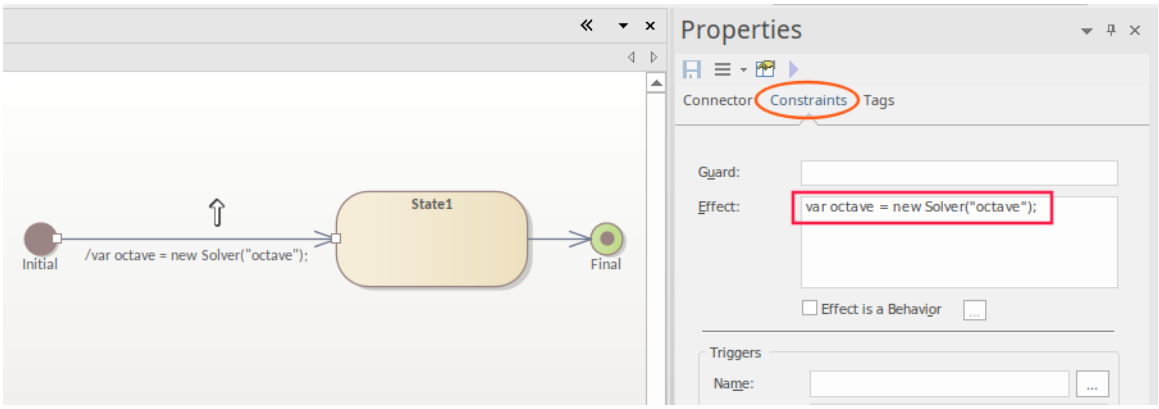
- Matlab Solver
- StateMachine Simulation - Vaccine Trial
- StateMachine Simulation - Lottery Draw

StateMachine Solver Examples

This topic provides some simple examples of where and how to apply Solvers in StateMachine Simulations, which includes using scripts in StateMachine operations as well as in Transition connectors. The images in this topic show Octave Solver examples; however, unless otherwise stated, the same script can be used for MATLAB in the MATLAB Solver.

Initializing the Solver

The script to initialize a Solver in a StateMachine can be placed either in the Effect of a Transition or in the Entry operation of a State. It is usually placed in the Effect of the first Transition exiting the Initial.



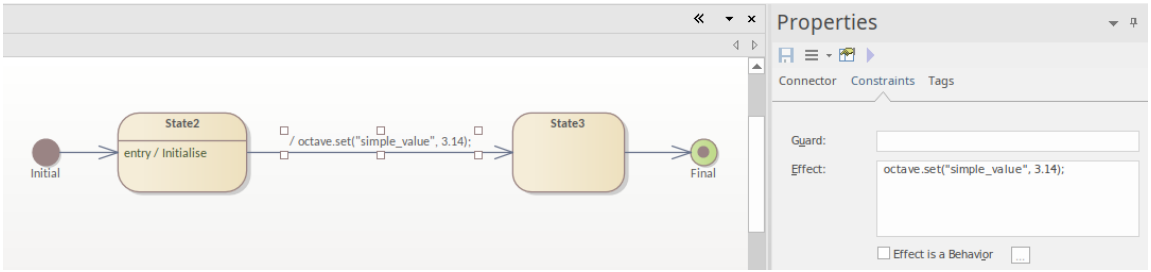
Assigning Values and Executing Commands

To assign a value, use the octave.set() or matlab.set() function.

For StateMachines the method can be placed in the Effect of

a Transition or in the Behavior (Entry/Do/Exit) of an operation.

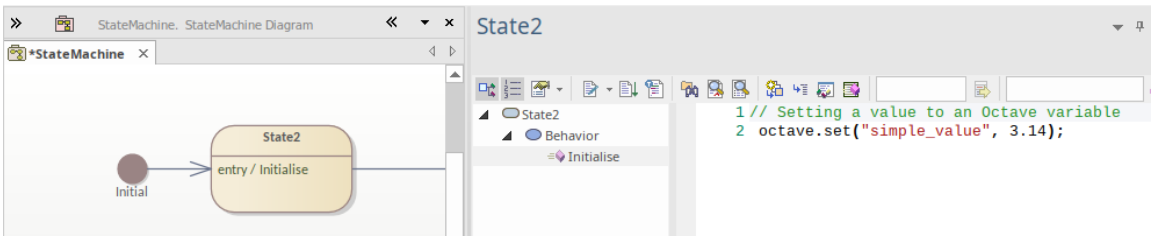
To edit the Effect of a Transition you use the 'Constraints' page of the 'Properties' dialog.



To place the method in an operation, open the operation's Behavior script:

1. Create a new Entry operation in the State from either the diagram or the Browser window, by right-clicking on the element and selecting the 'Features > Operations' context menu option.
2. Click on the new Entry operation and press Alt+7.

This will open the behavior script in the editor.

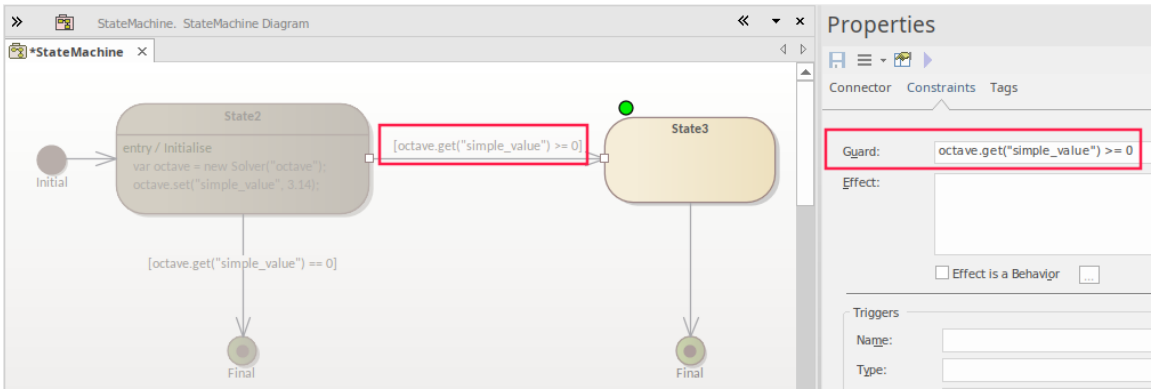


When executing MATLAB or Octave commands using the .exec() function, the commands can be placed in the Effect of the Transition or the Behavior script of the operation.

Conditional Branching

When using conditional branching in a StateMachine, the condition can be placed in the Guard of a Transition and can

contain a script calling MATLAB or Octave functions. For example:



Note:

- The condition can also be set using a Choice; the placement of the condition statement is the same, in that it is defined in the Guard of the exiting Transition
- The example shows a simulation at the Breakpoint set on State3

Getting Results

When returning the results from external function calls, you can use the Solver's get() function to return the results in the script. There are three key options for then delivering the results from the script to the user:

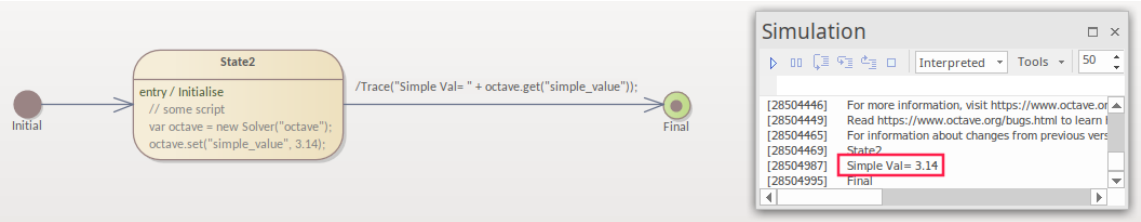
- Trace()
- Plot
- Win32 display

With both the Trace and Win32 options you must return a local copy in your JavaScript, using the Solver's .get() command. As an example of using the .get() command, see

the Guard in the previous image.

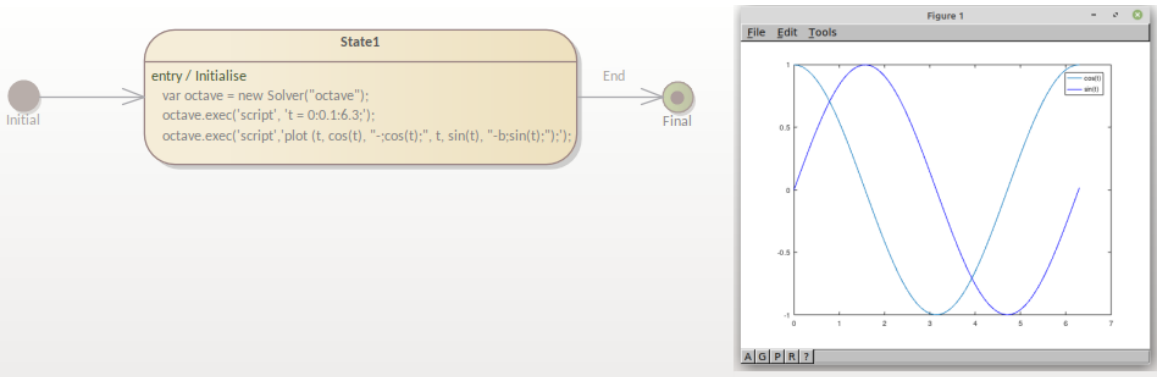
Using Trace

The Trace() command is useful when initially writing and debugging a simulation, as it allows you to check the results of your script at different stages. The results are output in the Simulation window.



Executing Plots

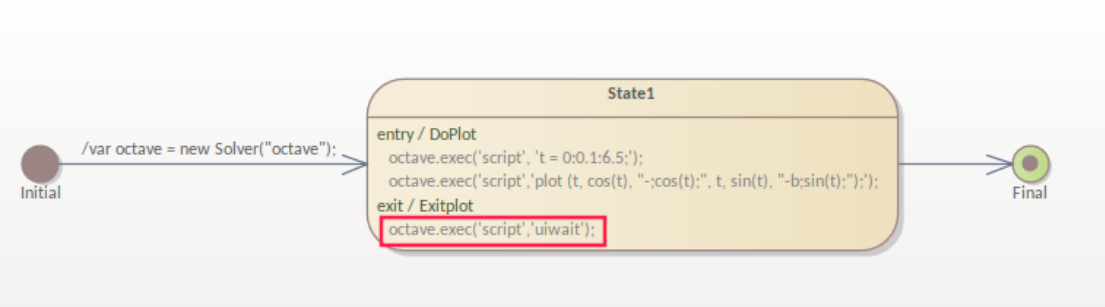
Octave and MATLAB have a strong emphasis on the ability to generate a plot, as this is a key method of outputting results. To generate a plot you use the Solver's .exec() function to call a plot generation function.



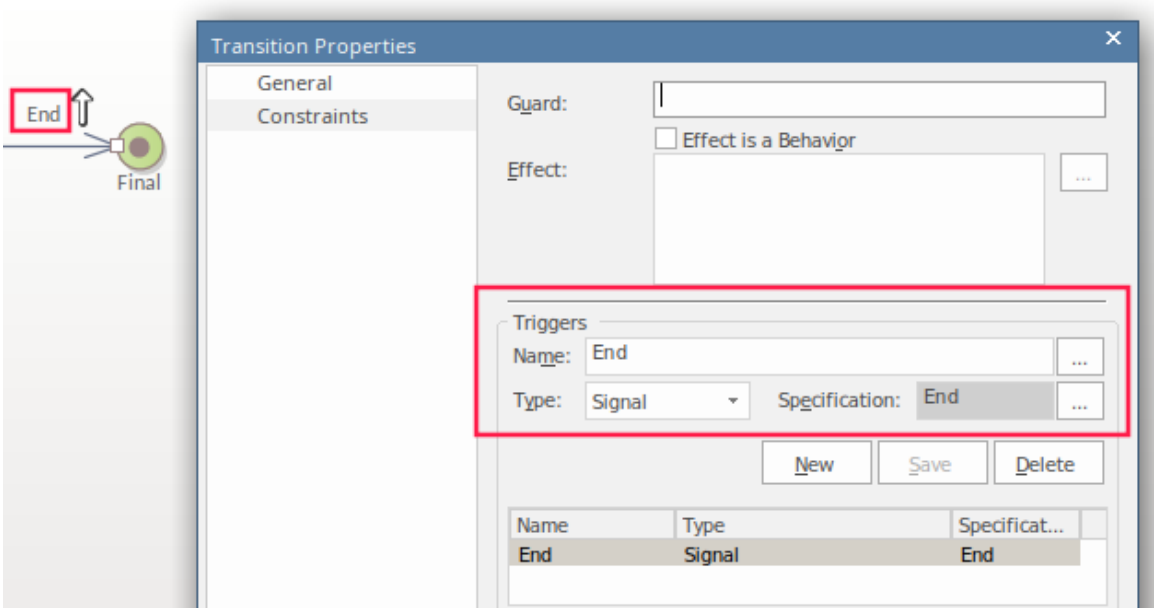
Holding the Plot

When executing a plot in a simulation, if the simulation is not paused the plot will only be viewable for a brief moment. So, for StateMachines, there are two options for pausing the flow while the plot is being viewed:

1. Using the uiwait in Octave or MATLAB. For example this can be set in the Exit Operation or an in a Transition Effect. Here is an example using Octave:
   octave.exec('script','uiwait');



2. Setting a Trigger/Event sequence to pause the simulation in a Transition out of the State where the plot was generated. In the example case this is in the final Transition.

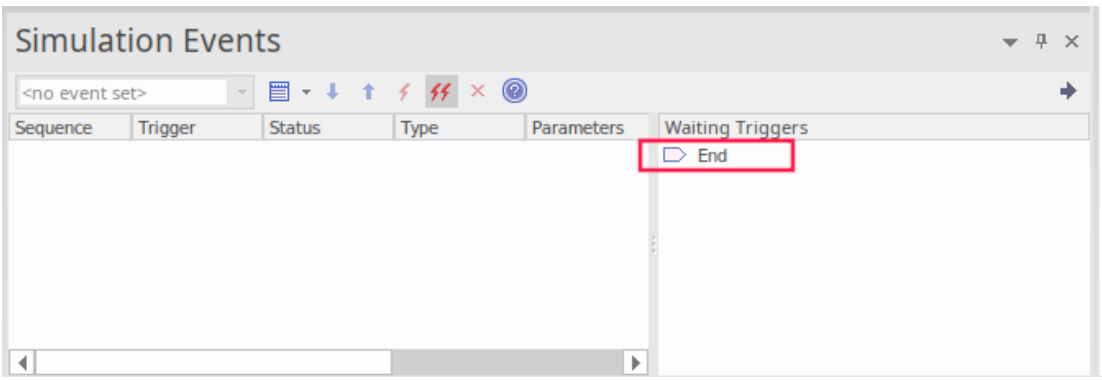


To progress past the Transition you can either:

- Click on the Trigger showing in the Simulation Events

window or

- Use a BroadcastSignal() from, say, a button in the Win32 screen (this is discussed in the *Using the Win32 Interface* section, next)



Using the Win32 Interface

When using the Win32 interface the broad steps to perform are:

- Create a Win32 dialog
- Set a script line to open the dialog
- Get a value from a field in the dialog
- Pass that value to the solver
- Use a button to trigger the plot

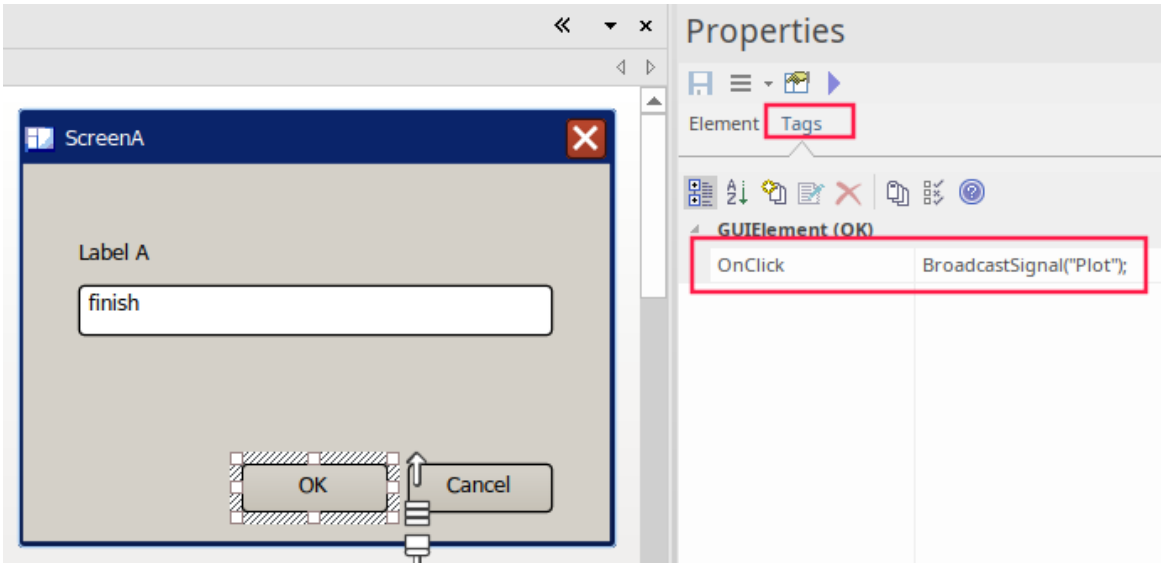
The steps to configure a Win32 interface are:

- Create a ‘Starter Win32 Model’ using the Model Builder - select the 'UX Design' Perspective Set and the 'Win32 UI Models' Perspective
- Change the name of the <<Win32 Screen Dialog>> to 'ScreenA'

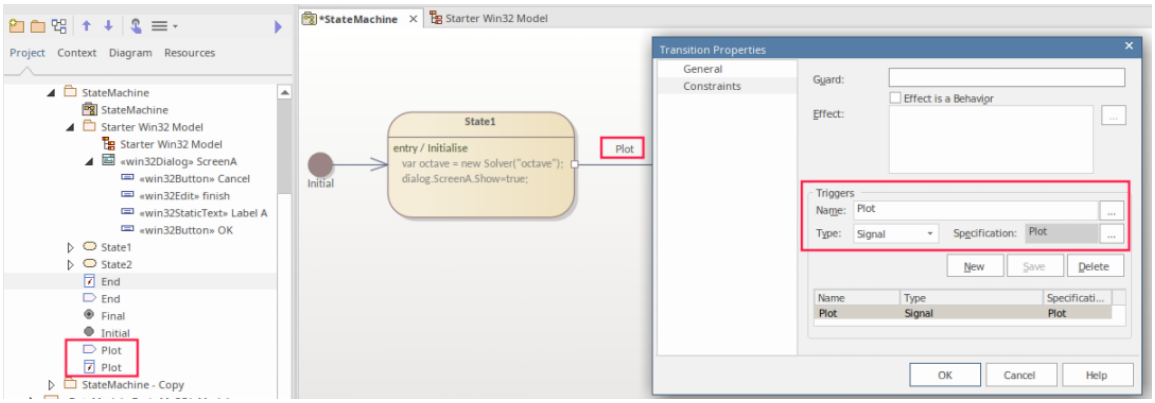
- Change the name of ‘Edit Control A’ to something more meaningful, such as ‘finish’
- Click on State1 and press Alt+7 to add - in the Entry Operation's script - a call to open the dialog:
  dialog.ScreenA.Show=true;

To get a valid user input value, the user must click on the OK button, so from the OK button you must Broadcast an event for a Trigger to start the process. To receive the Trigger you must have a Signal and set a Transition to be triggered by that Broadcast to the Signal.

See the steps to create and set the Trigger and Signal 'End', as shown in the *Using a Trigger to Hold the Plot* section, earlier. In this case we are setting the same Transition, but for a new Trigger called 'Plot'. This is sending a BroadcastSignal(‘Plot’) using the OnClick Tagged Value on a button:



The Transition exiting State1 is set to be triggered by the BroadcastSignal:



On the Transition you must create a Trigger, set the Trigger-Type to Signal and create the Signal. For more detail see the *Win32 User Interface Simulation* Help topic.

The script to run the plot will be in the Entry operation of State2. During the simulation the transition into State2 will only occur after the Win32 OK button is clicked and the Plot trigger is sent. So we now:

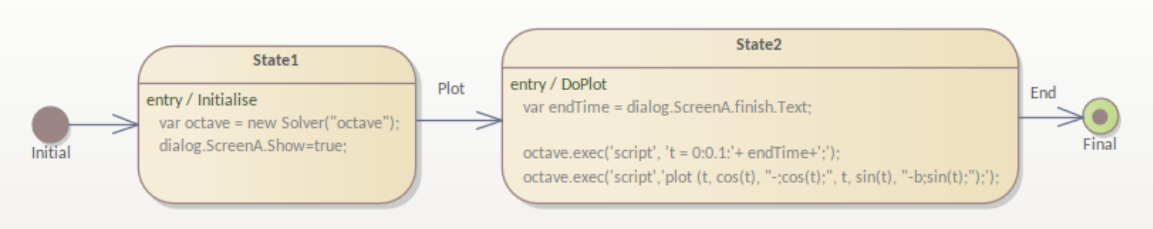
- Add an Entry Operation and open the Behavior using Alt+7
- In this state's Entry Operation we get the value of the field using:

  ```
  var endTime = dialog.ScreenA.finish.Text;
  ```

  This sets a variable with the value for the last parameter to be sent to Octave for the plot
- In the octave.exec() statement, we place the variable to set the number of seconds to plot too:

  ```
  octave.exec('script', 't = 0:0.1:'+ endTime+';');
  ```

This gives two States with the scripts in the Entry Operations:

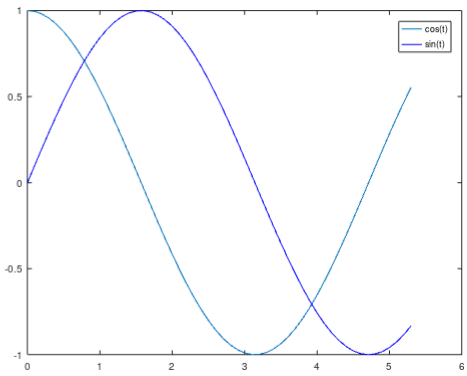


Running the Simulation

To run the simulation:

- Select the Simulate ribbon and click on 'Run Simulation > Start'

When you enter a value and click on the OK button, this returns:

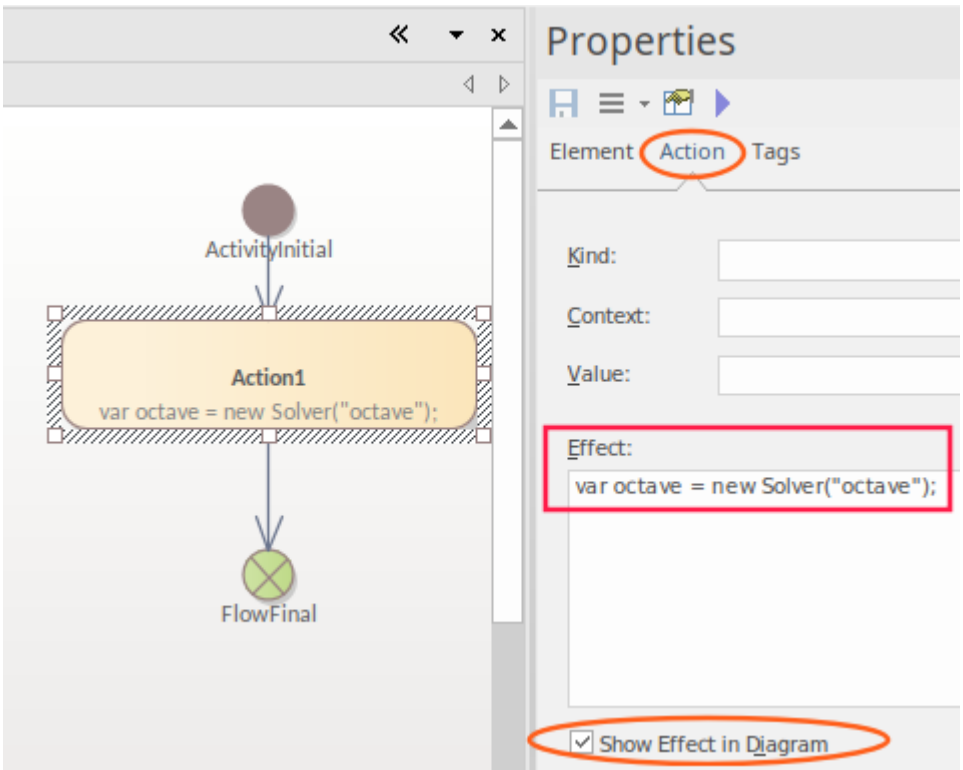


Activity Diagram Solver Examples

In this topic we run through some simple examples of where and how to apply Solvers in Activity diagram simulations. This includes using scripts in Action Effects, as well as in ControlFlow Guards. The images in this topic show Octave Solver examples; however, the same script can be used for MATLAB by replacing the Octave Solver with the MATLAB Solver.

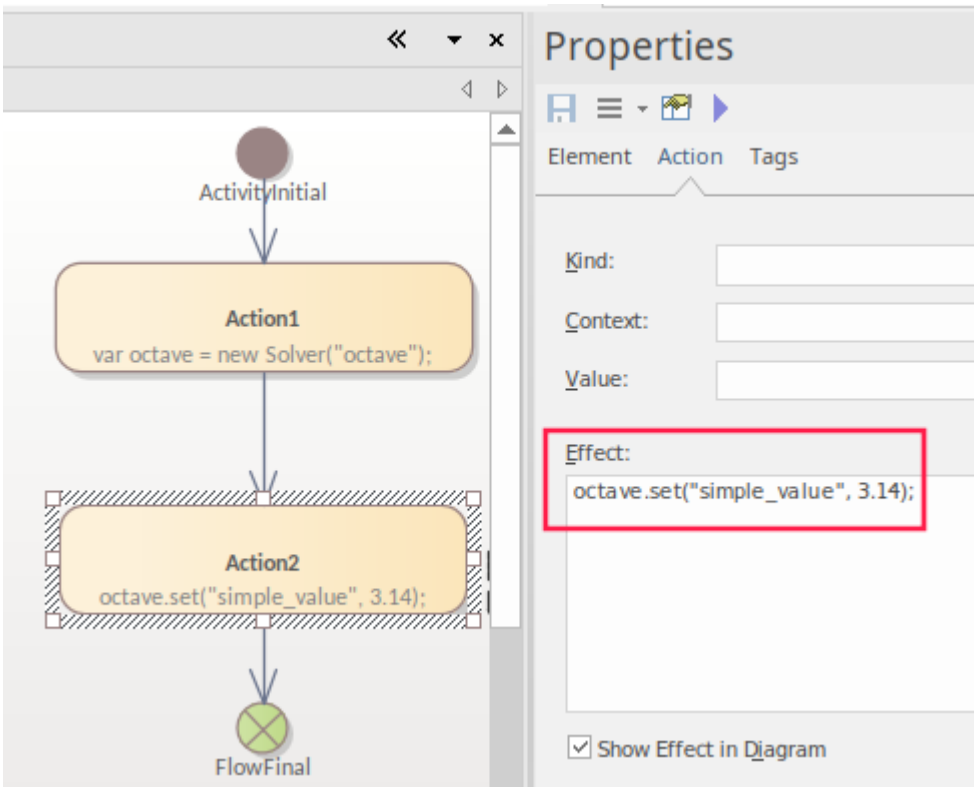
Initializing the Solver

The script to initialize a Solver in an Activity diagram can be placed in the Effect of an Action, usually the Effect of the first Action exiting the ActivityInitial.



Assigning Values and Executing Commands

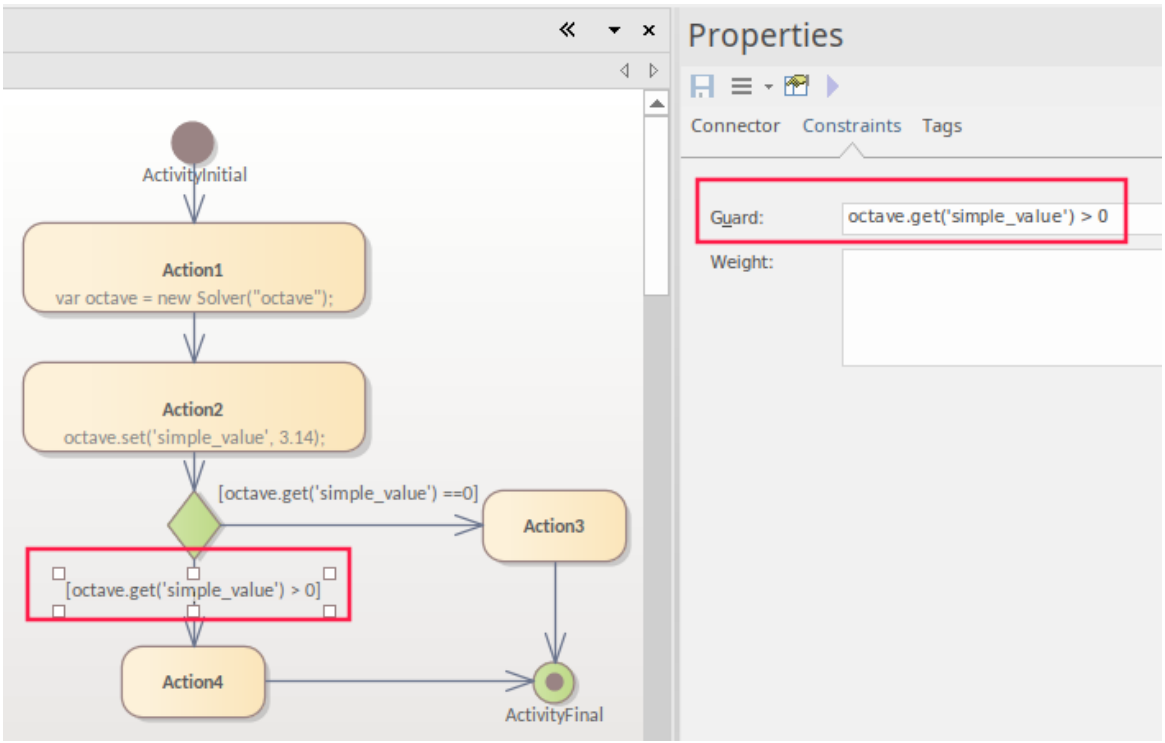
To assign a value, use the octave.set() or matlab.set() functions. For Activity diagrams the function can be placed in the Effect of an Action.



When executing MATLAB or Octave commands using the .exec() function, the commands can again be placed in the Action's Effect.

Conditional Branching

For conditional branching in a StateMachine, the condition can be placed in the Guard of a ControlFlow and contain a script that calls any MATLAB or Octave function. For example:



Getting Results

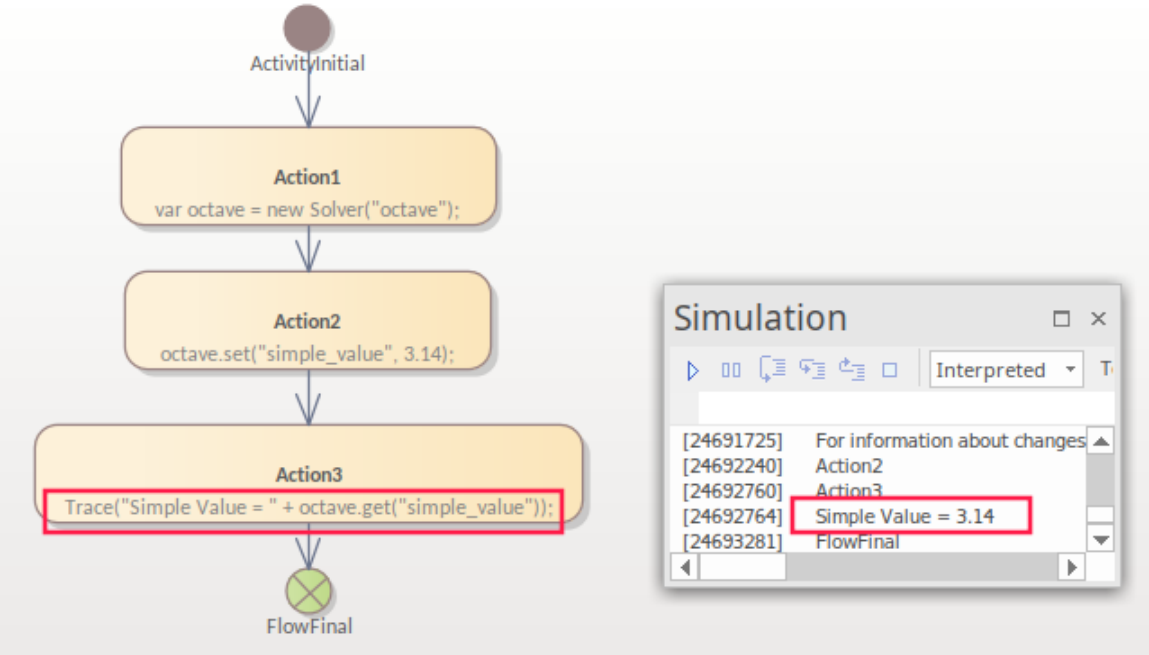
To return the results from external function calls you use the Solver's .get() function. There are three key options for how the results are then delivered from the script to the user:

- Trace()
- Plot
- Win32 display

With both the Trace and Win32 options you must return a local copy in your JavaScript using the Solver .get() function. The previous image includes an example of using the .get() function in a Guard.

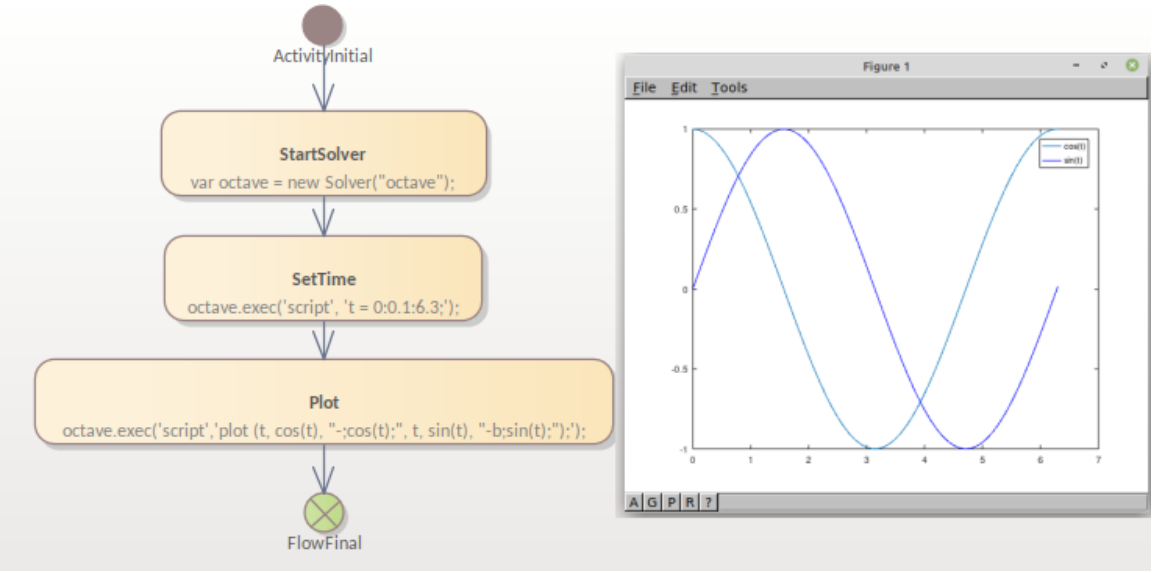
Using Trace

The Trace() command is useful when initially writing and debugging a simulation, as it allows you to check the results of your script at different stages. The results are output in the Simulation window.



Executing Plots

Octave and MATLAB have a strong emphasis on generating plots, as this is a key method of outputting results. To generate a plot you use the Solver's .exec() function to call a plot generation.

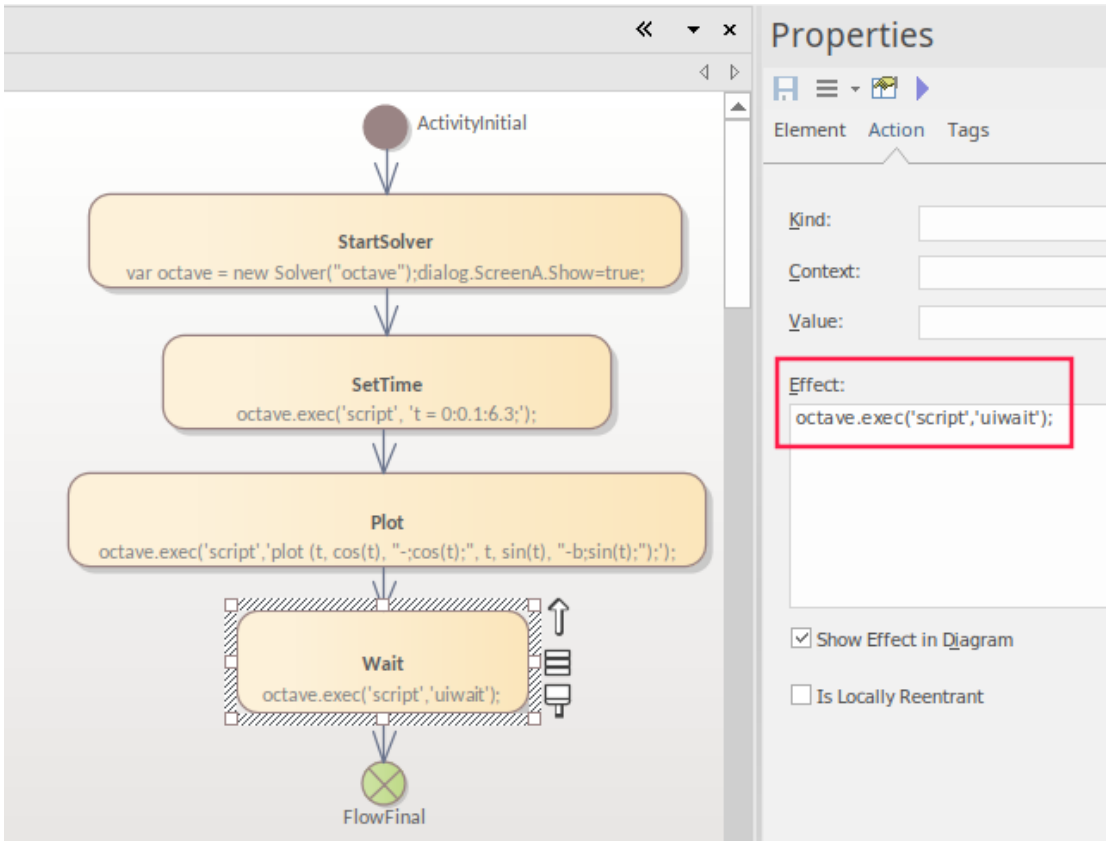


Holding the Plot

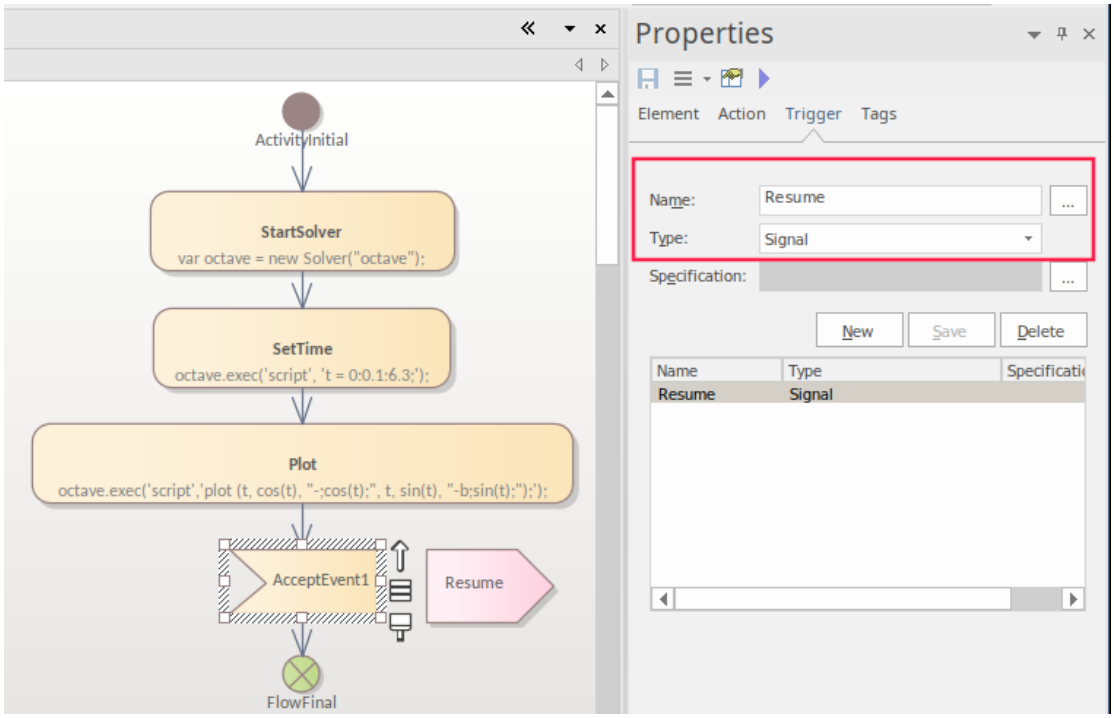
When executing a plot in a simulation, if the simulation is not paused the plot will only be briefly visible. For Activity diagrams there are two options for pausing the flow while the plot is being viewed:

1. Using the uiwait function in Octave or MATLAB, setting it in the Effect of the Action. Here is an example using Octave:

   octave.exec('script','uiwait');

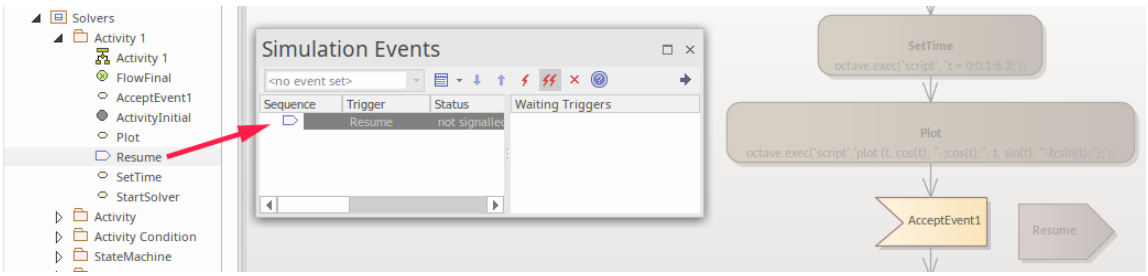


2. Set an Action of type AcceptEvent to pause simulation after the plot is generated, which requires a reference to an Activity Trigger to be set on the AcceptEvent. This can be created and referenced using the ... button, which displays the 'Select Trigger' dialog. Click on the Add New button to create the Trigger.



To progress past the AcceptEvent you:

- Drag the Trigger (Resume) from the Browser onto the Simulation Events window
- Double-click on that transition



Using the Win32 Interface

When using the Win32 interface, broadly the steps to perform are:

1. Create a Win32 dialog.
2. Set a script line to open it.

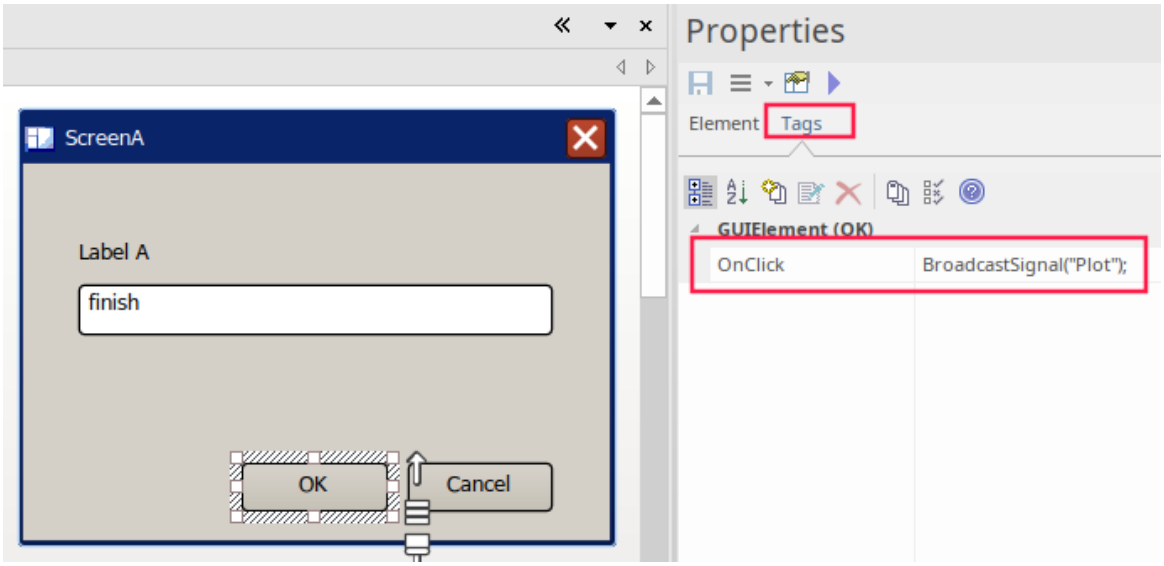
3. Get a value from a field in the dialog.
4. Pass that value to the Solver.
5. Use a button to trigger the plot.

These are the steps to configure a Win32 interface:

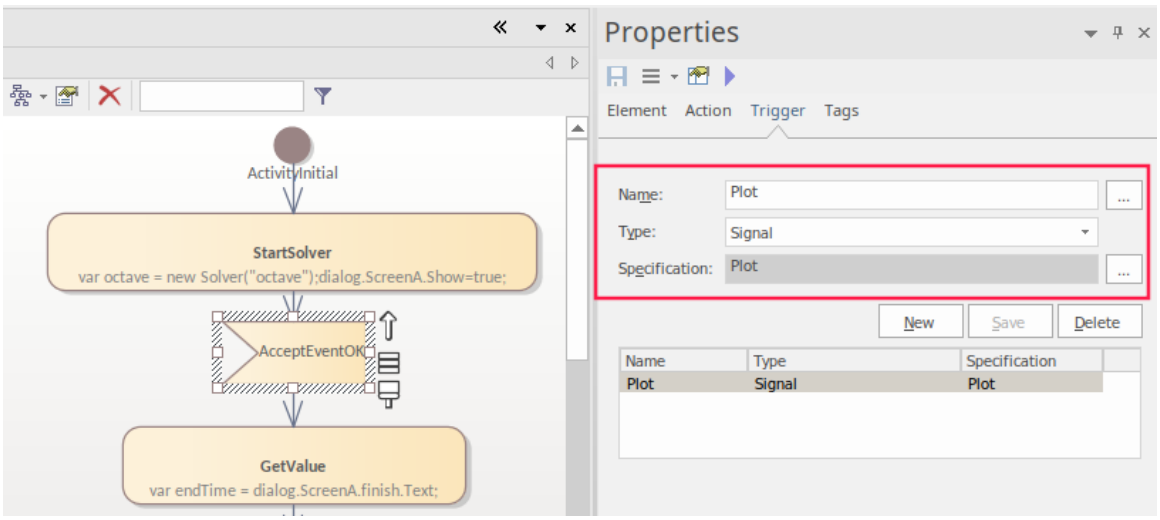
1. Create a ‘Starter Win32 Model’ using the 'Model Builder' dialog (Ctrl+Shift+M).
2. Change the name of the <<Win32 Screen Dialog>> to 'ScreenA'.
3. Change the name of ‘Edit Control A’ to something more meaningful such as ‘finish’.
4. Press Alt+7 on State1 to add a call to open the dialog, in the Entry Operation's script:
   dialog.ScreenA.Show=true;

To get a valid user input value, the user must click on the OK button, so from the OK button you need to Broadcast an event for a Trigger to start the process. To receive the Trigger you set a Signal and a Transition to be triggered by that Broadcast.

See the steps to create and set the Trigger and Signal 'End', as shown in *Holding the Plot* earlier. In this case we are setting the same thing, but for a new Trigger called 'Plot'. This illustration is sending a BroadcastSignal(‘Plot’) using the OnClick Tagged Value on a button.



On the ControlFlow exiting StartSolver there is now an Action 'AcceptEvent'. This is set to be triggered as shown:



On the Properties window 'Trigger' tab you create a Trigger, set the Trigger-Type to 'Signal' and create a Signal for it. For more detail see the *Win32 User Interface Simulation* Help topic.

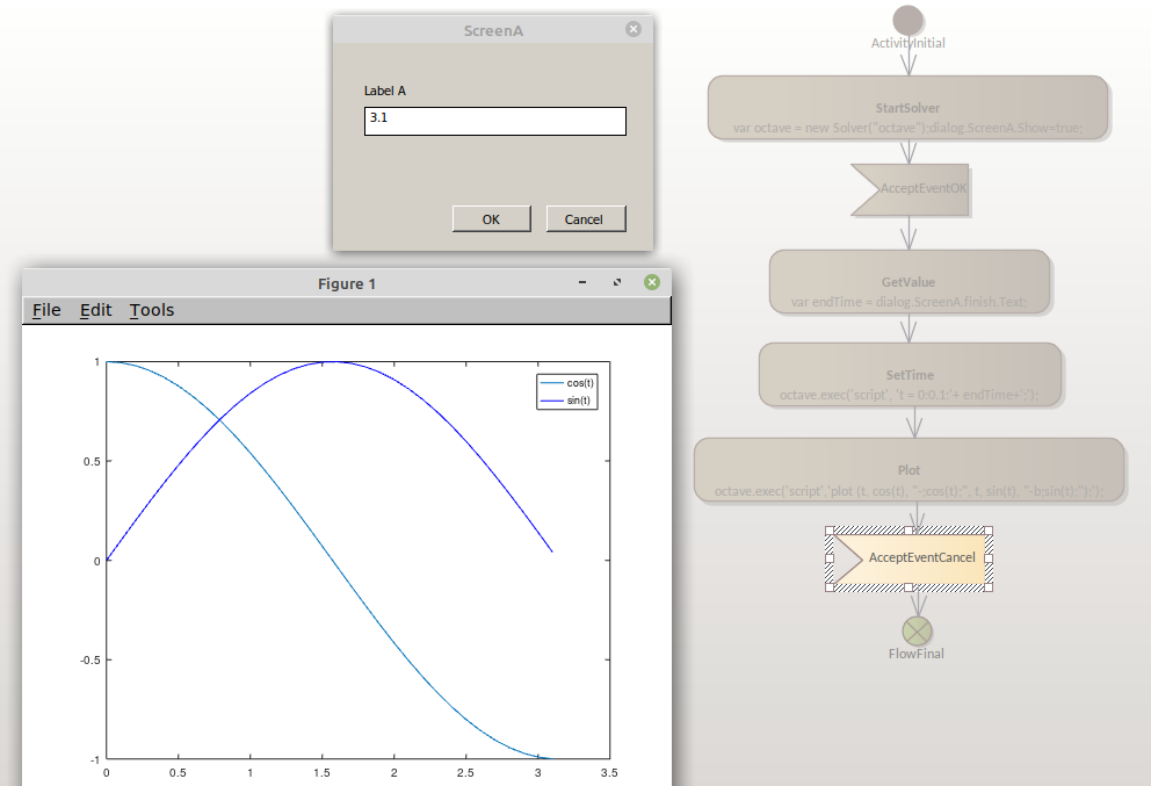
The Cancel button is configured in the same way as the OK button (with a Broadcast signal) and Trigger references are set on the AcceptEventCancel to use the 'cancel' Trigger/Signal.

Running the Simulation

To run the simulation:

- Select the Simulate ribbon
- Click on 'Run Simulation > Start'

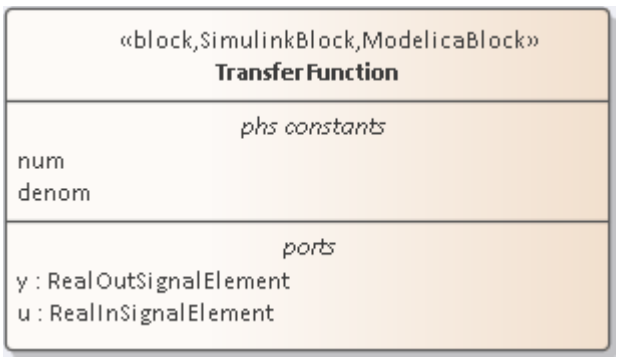
When you enter a value and click on the OK button, this returns:



SysPhS Simulation

The *SysML Extension for Physical Interaction and Signal Flow Simulation* specification (SysPhS), is an Object Management Group (OMG) specification that extends SysML to provide a common modeling platform for defining consistent models. These models can be translated out to either of two key simulation platforms, Modelica and MATLAB's Simulink/Simscape.

The OMG SysPhS stereotypes help you to define the characteristics of the model simulation within the model itself, rather than in a simulation configuration specification. They provide greater visibility of the type of object or property in the Browser window and Properties window, and in the diagram with specific element compartments for the Property types and for initial values.



The standard is represented in Enterprise Architect by the OMG SysPhS Profile, along with:

- SysPhS Libraries of elements for signal flow and for physical interaction (necessary to perform simulations under the SysPhS standard)
- A dedicated Toolbox page

- A wide range of component element patterns from which to generate common simulation elements such as electronic, logic, and fluid components; the patterns reference the library Blocks in either the OpenModelica or Simulink standard libraries
- Features for Simulating plots using Modelica or MATLAB's Simulink, Simscape and Stateflow.

SysPhS Features

Feature	Description
Referencing SysPhS Libraries	Key resources for working with SysPhS are the SysPhS Simulation Libraries, which include reusable resources that you must reference within your model.
SysPhS Toolbox	The SysPhS pages of the Diagram Toolbox contain basic SysML elements for both OpenModelica and MATLAB Simulink.
SysPhS Patterns	The SysPhS Patterns provide pre-defined SysPhS Blocks that reference equivalent MATLAB and Modelica components. These simple Blocks can be used as starters when working with SysPhS models.

SysPhs Components	SysPhS components enable you to set references to both Modelica and Simulink components.
Simulation	You can define IBD or Parametric models with additional information to drive a simulation, then use the simulation configuration to generate the model in Modelica, Simulink or Simscape, to produce a graph of the results.
SysPhS Examples	There are several examples of using SysPhS for setting up simulations.
Updating SysMLSim for SysPhys	You can update older simulation configurations (pre Enterprise Architect 15.2), to reflect the use of the SysPhS standard.

Options

Extra options for variables and constants, such as isContinuous and isConserved, are automatically set as Tagged Values, again avoiding the need to define them in the configuration specification. These options are also visible on the Block itself and in the docked Properties window.

Videos

- SysPhS Patterns for the Simulation of an Electrical Circuit
- Simulating Digital Electronics using SysPhS and Modelica

Referencing the SysPhS Simulation Libraries

The SysPhS specification provides a definition of a Platform-independent Component Library that consists of two component groups: Signal Flow and Physical Interaction. As this component library is an underlying resource used for modeling, a copy of this library must be contained in the repository you are working on, for reference in your SysPhS models.

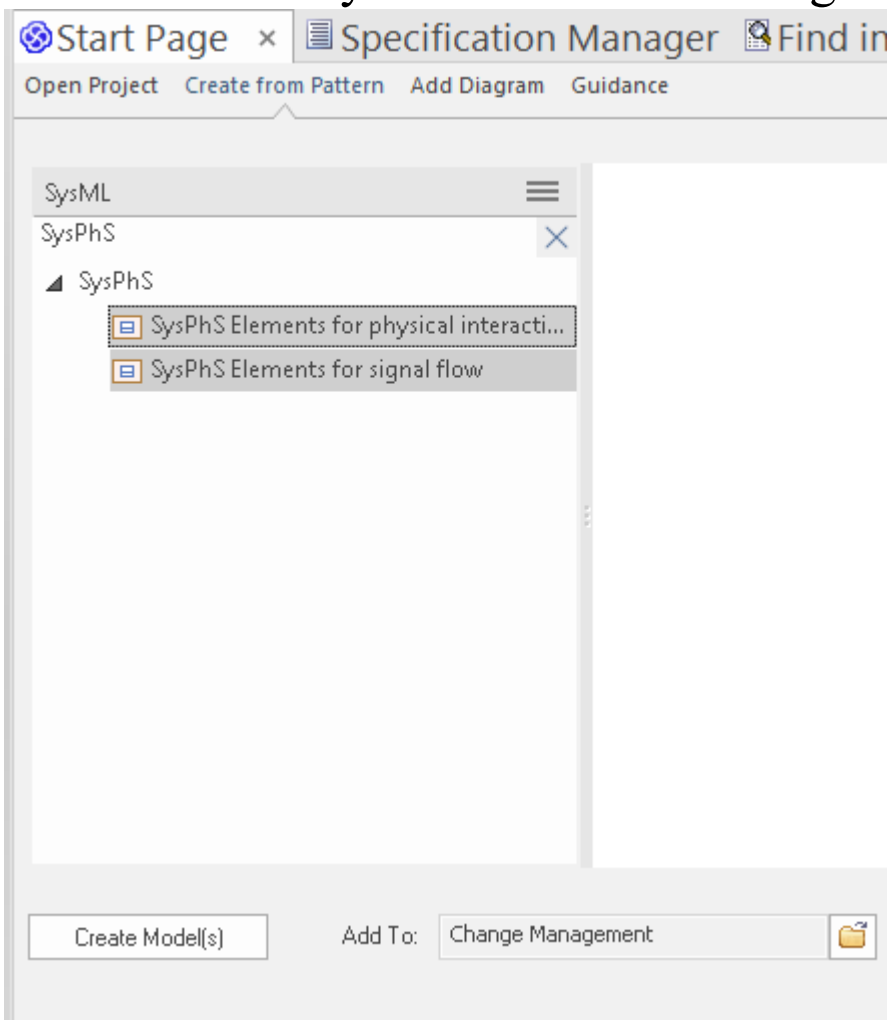
In order to perform simulations under the SysPhS standard, you must download two SysPhS libraries from the Model Builder:

- SysPhS Elements for physical interaction and
- SysPhS Elements for signal flow
- These libraries can be used on multiple SysPhS models created in the repository. It is recommended that they are downloaded to a unique Package, after which a single reference to that Package can be set in any SysPhS model.

To download the libraries:

1. Create a Package to hold the libraries.
2. Select the Package in the Browser.
3. Set the Perspective ( , top right-hand corner of the workspace) to 'Systems Engineering > SysML'.
4. Press Ctrl+Shift+M to open the Model Builder dialog.
5. In the Filter bar, underneath the Perspective name, type 'SysPhS'.

6. Click on 'SysPhS Elements for physical interaction' and Ctrl+click on 'SysPhS Elements for signal flow'.



7. Click on the Create Model button.

The libraries are loaded into the selected Package.

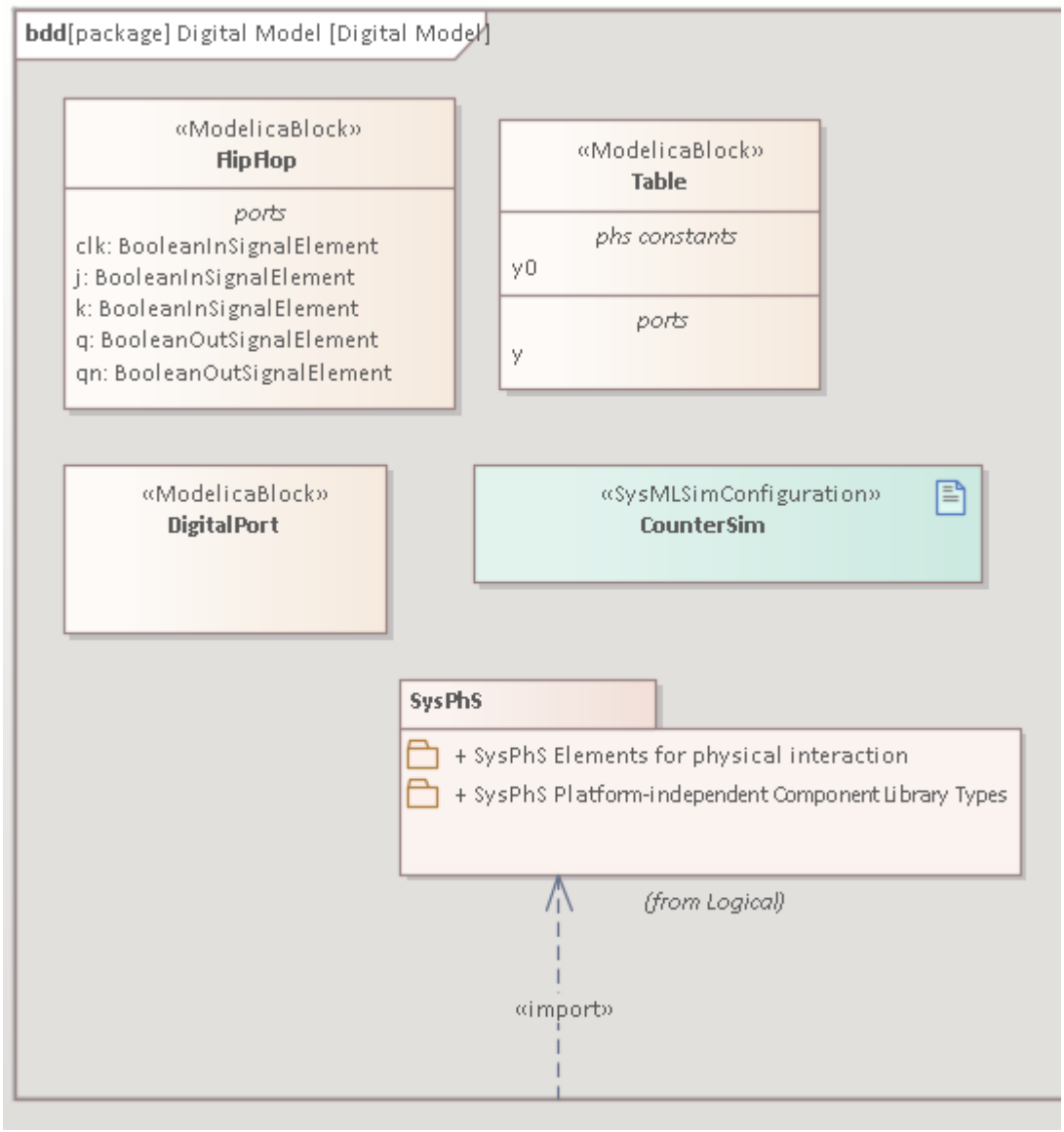
For each SysML simulation using SysPhS, the Block Definition diagram frame must have a reference to the library Package. To set this, link the library Package to the SysML diagram frame with an Import connector:

1. Create a Block Definition diagram for the simulation. This will automatically have a Boundary frame around the elements.
2. Drag the SysPhS library Package onto the diagram.

A prompt displays for the type of 'add' operation to perform.

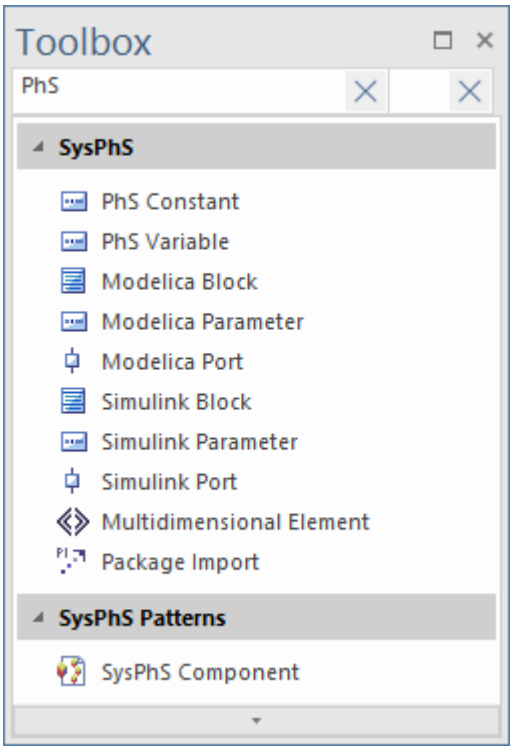
3. Select the 'Package Element' option.
   The library Package element is added to the diagram within the Boundary frame.
4. Right-click on the Boundary frame to display the context menu, and ensure the 'Selectable' option is ticked.
5. Change the Toolbox to the 'SysPhS' page:
   Click on the 'Package Import' relationship icon, and drag the cursor between the Boundary frame and the library Package to create an Import connector.
6. Save the diagram (press Ctrl+S).

This is an example of the SysPhS Package reference using the <<import>> connector:


bdd[package] Digital Model [Digital Model]
«ModelicaBlock»
FlipFlop
ports
clk: BooleanInSignalElement
j: BooleanInSignalElement
k: BooleanInSignalElement
q: BooleanOutSignalElement
qn: BooleanOutSignalElement
«ModelicaBlock»
Table
phs constants
y0
ports
y
«ModelicaBlock»
DigitalPort
«SysMLSimConfiguration»
CounterSim
SysPhS
+ SysPhS Elements for physical interaction
+ SysPhS Platform-independent Component Library Types
(from Logical)
«import»

Using the SysPhS Toolbox

The SysPhS profile provides a dedicated Toolbox page, containing basic SysML elements for both OpenModelica and MATLAB Simulink and, on a 'SysPhS Patterns' page, the 'SysPhS Component' icon.



SysPhS Elements

Element Icon	Description
PhS Constant	PhS Constant defines values that are constant during a simulation run. For Modelica, these correspond to parameter variables. For Simscape these correspond to

	(constant) parameters.
PhS Variable	PhS Variable defines values that are variable during a simulation run. For Modelica, PhSVariables with: • isContinuous=true correspond to Modelica *continuous* components • isContinuous=false correspond to *discrete* components For Simulink, PhSVariables correspond to Simscape *variables*.
Modelica Block	The Modelica Block element-type is used for defining a corresponding component in the Modelica library. To define a Modelica Block to correspond to a component in the Modelica library requires that the value of the Modelica SysPhS Name property is set to the fully qualified component name from Modelica (the Class Path).
Modelica Parameter	A Modelica Parameter element-type is used to define a parameter of a Modelica Library component. These are defined in the Properties window > Element tab > «ModelicaParameter» (from SysPhS) which has two parameters: • Name: Corresponds to the Modelica

	parameter name (Parameters Name) • Value: The Typing of this parameter and its initial value can be defined on the Properties > Property tab in the fields: • Type • Initial
Modelica Port	A Modelica Port corresponds to a Port defined in the Modelica Library. The corresponding Modelica Port name is defined in: Properties > Element > «*ModelicaPort» (from SysPhs)* > Name.
Simulink Block	The Simulink Block element-type is used for defining a corresponding component in the Simulink library. To define a Simulink Block that corresponds to a component in the Simulink library requires that the value of the Simulink SysPhS Name property is set to the fully qualified component name from Simulink (found in the Simulink Model Explorer as 'Contents of').
Simulink Parameter	A Simulink Parameter element-type is used to define a parameter of a Simulink Library component. These are defined in the Properties window > Element tab >

	«SimulinkParameter» (from SysPhS), which has two parameters: • Name: Corresponds to the parameter name in the Simulink library • Value: The Typing of this parameter and its initial value can be defined on the Properties > Property tab in the fields: • Type • Initial
Simulink Port	A Simulink Port corresponds to a Port defined in the Simulink Library. The corresponding Simulink Port name is defined in: Properties > Element > *«SimulinkPort» (from SysPhs*) > Name.
Multidimensional Element	In systems modeling, while a vector can be specified using multiplicity, for multi-dimensional arrays we need multiple multiplicities. The MultidimensionalElement stereotype provides for this, effectively supporting an array of multiplicities defining the dimensions. This is applied to a Block by dragging on the Multidimensional Element icon.

Package Import	The Package Import connector is required for setting a reference from the main Block to the SysPhS simulation libraries. For more details see the *Referencing the SysPhS Simulation Libraries* Help topic.

SysPhS Patterns

In the OMG SysPhS specification there is a list of SysPhS components that are common to both Modelica and Simulink. Enterprise Architect provides this series of components as a set of SysPhS *Patterns*. These core components are useful starting blocks when creating a new model. For more details on using these see the *Using the SysPhS Patterns* Help topic.

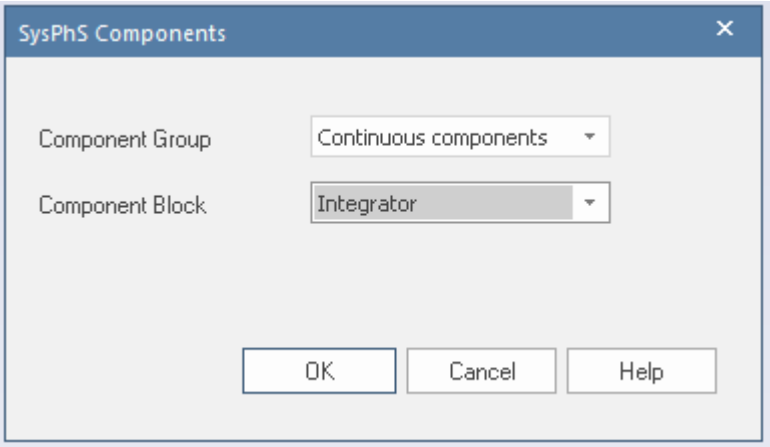
Using the SysPhS Patterns

The OMG SysPhS specification lists a series of SysPhS components as the *Platform Independent Component Library*. What the Enterprise Architect *SysPhS Patterns* provide is this series of components that are common to both Modelica and Simulink. When an item is selected it creates a Block that is typed for both Modelica and Simulink. Then, when running a simulation that is set to either Modelica or Simulink, this item-type will be created in the respective tool.

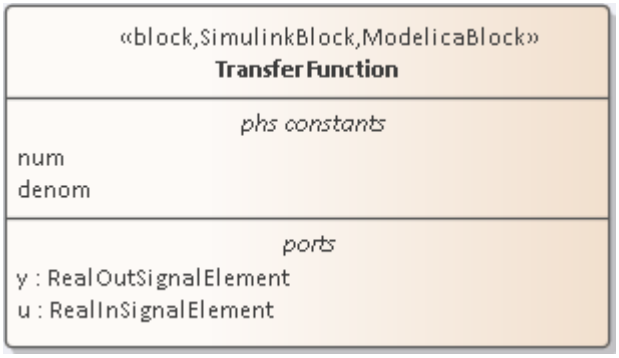
The general grouping of these components is

- Continuous components
- Discrete components
- Non-linear components
- Mathematical components
- Sources and sinks
- Routing components
- Logical components
- Electrical components

When you drag the 'SysPhS Component' icon onto the diagram, the 'SysPhS Components' dialog displays.



On this dialog you firstly select the category of element to create, then the type of element to create, both by selecting from a drop-down list in the 'Component Group' or 'Component Block' field. The drop-down list in each field is populated from the OpenModelica and Simulink standard libraries. For example:



Notice that the element has stereotypes for Simulink and Modelica, so can be worked on in either of those tools. For each element, the correct Properties and Ports for the element type are automatically included in the element, as items in the element compartments. You can drag the actual structural elements from the Browser window onto the element if you prefer their physical presence on the diagram.

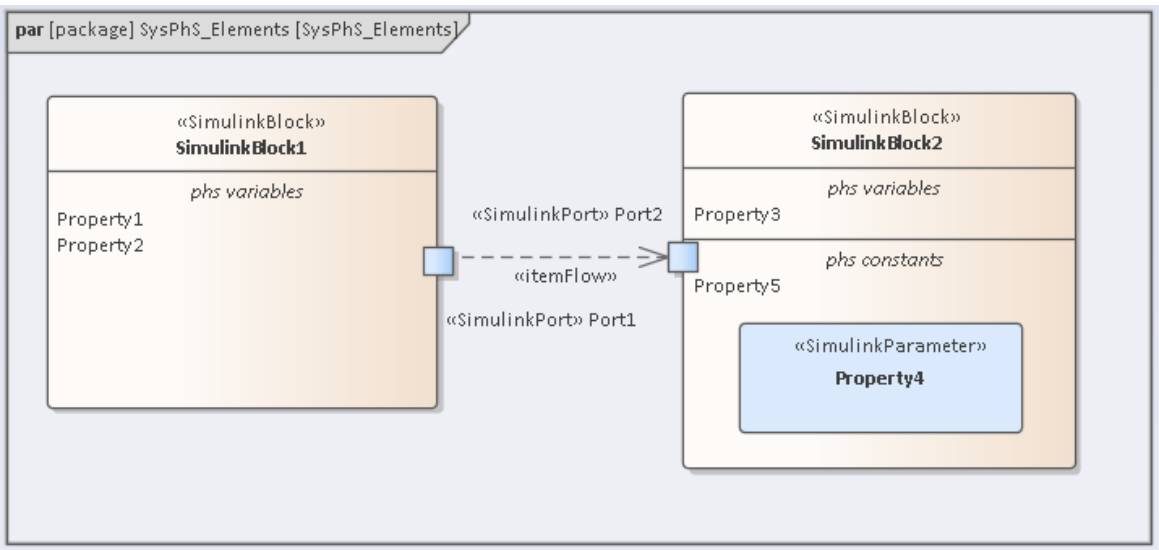
For a listing of all supported components and more details, see section: *11.3.2 Real-valued components* in the *OMG SysPhS 1.0 PDF*.

Working with SysPhS Components

If you are working with an example model held in Modelica or Simulink and you want to refer - in Enterprise Architect - to components that already exist in that model, you can drag the appropriate Block, Parameter or Port element-types from the Toolbox onto a diagram to create the referring elements.

Displaying Properties and Parts

A point to note when working with Properties and Parts in a SysML diagram is that their default display is as Part object. They can be left in that rendering or set to show as text in compartments. Here is an example of both:

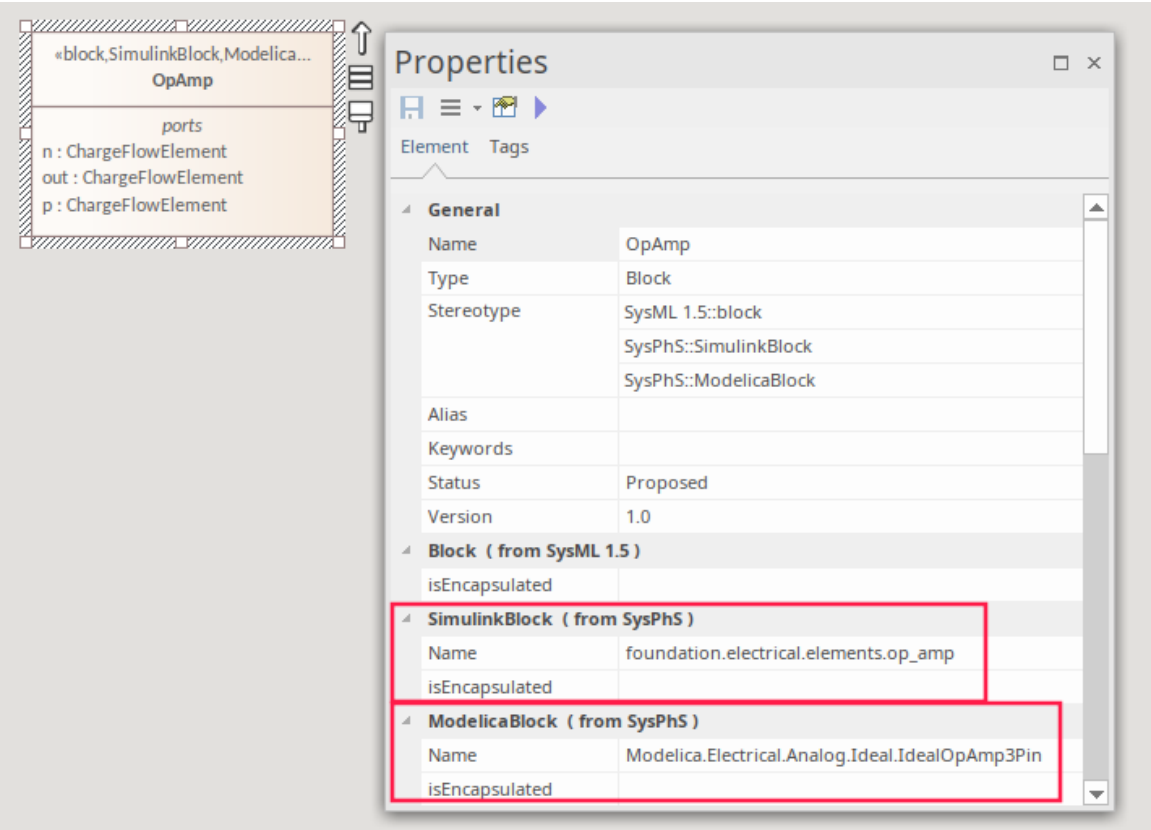


Properties 1, 2, 3 and 5 have been dragged onto the diagram as elements that were then deleted from the diagram (but not from the Browser window). On deletion, they were replaced on the diagram by text entries in their respective Block compartments. Property 4 was left on the diagram as a Part element; if deleted from the diagram, it would also become

a text entry in a compartment.

Block Element-type

Dragging a Modelica or Simulink Block-type onto a diagram creates a non-specific SysPhS Block. To set this Block to a specific Modelica or Simulink Block, set the 'Name' field under the **SimulinkBlock (SysPhS)** or **ModelicaBlock (SysPhS)** segment in the Properties window.



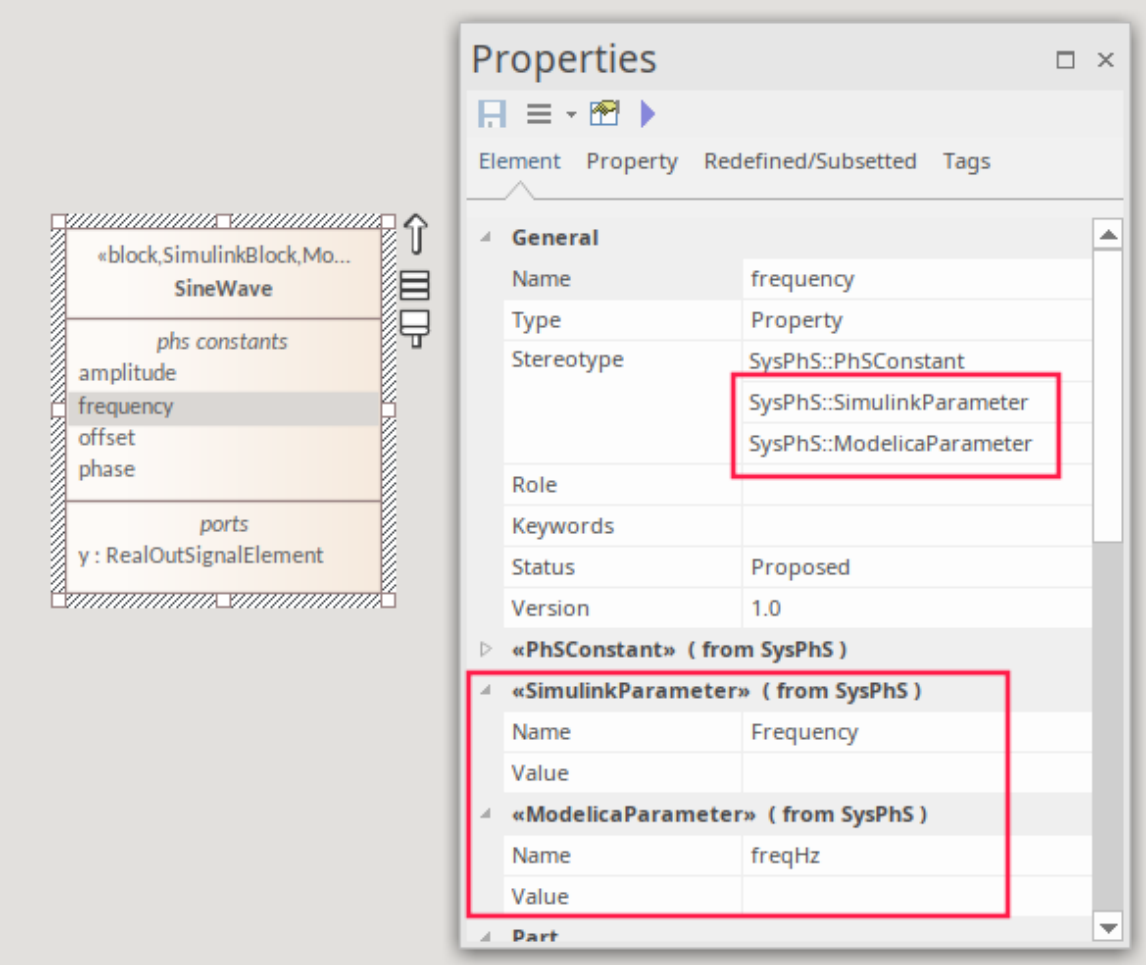
For more details on referencing the Simulink and Modelica Properties in these external tools, see the *Creating Modelica Specific Blocks* and *Creating Simulink Specific Blocks* Help topics.

Note: As shown in the image, the two SysPhS Stereotypes

can both be applied if you want to simulate the model in both external tools. See the *Setting Blocks as Both Modelica and Simulink* Help topic.

Parameter Element-type

The Parameter element-type creates Property elements with SimulinkParameter or ModelicaParameter stereotypes. If you delete the elements from the diagram, they are listed in the *phs constants* compartment of the parent Block element. Here is an example of a SysPhS Parameter set to both Modelica and Simulink, showing a stereotype for each as well as a reference to the respective product's Parameter name in the 'Name' fields.

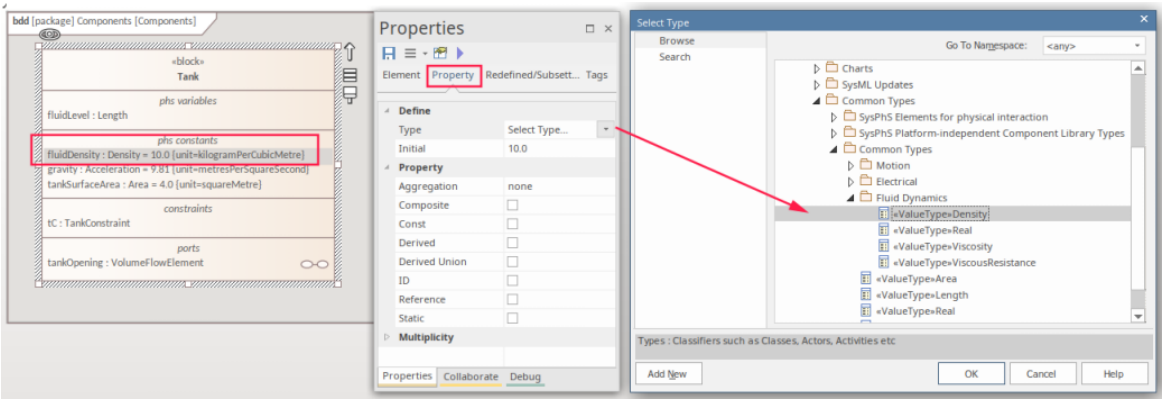


PhsConstant and PhsVariable

To define constant and variable properties of a Block element you can drag the 'PhS Constant' and 'PhS Variable' icons onto an element in the diagram. Again, if you delete the elements from the diagram they are listed in the *phs constants* or *phs variables* compartment.

When setting the value of constants in the model, the values can be set in the Block or in a Part derived from the Block. In the case of, say, gravity as an absolute constant, this is best defined in the Block. The values from the Part or Block can be further altered in the Simulation properties.

The next illustration shows the Fluid Density set for water, which could be overridden in the Part or the simulation to define the density of another fluid (e.g. oil). Where a Block is used repetitively with different values - e.g. one resistor of say 3.3 kohms and another of 5.6 kohms, then the initial value is best defined in the specific Parts, in the IBD, which are derived from a Block that has no initial value.



The 'Properties > Property' tab has two fields:

- Type
- Initial

The type can be set as a standard type or, as in this case, as SysML *ValueType* referenced in the model.

Note that if a PhsConstant or a PhsVariable has an initial value set, it is displayed on the diagram in an *initial value* compartment of the element.

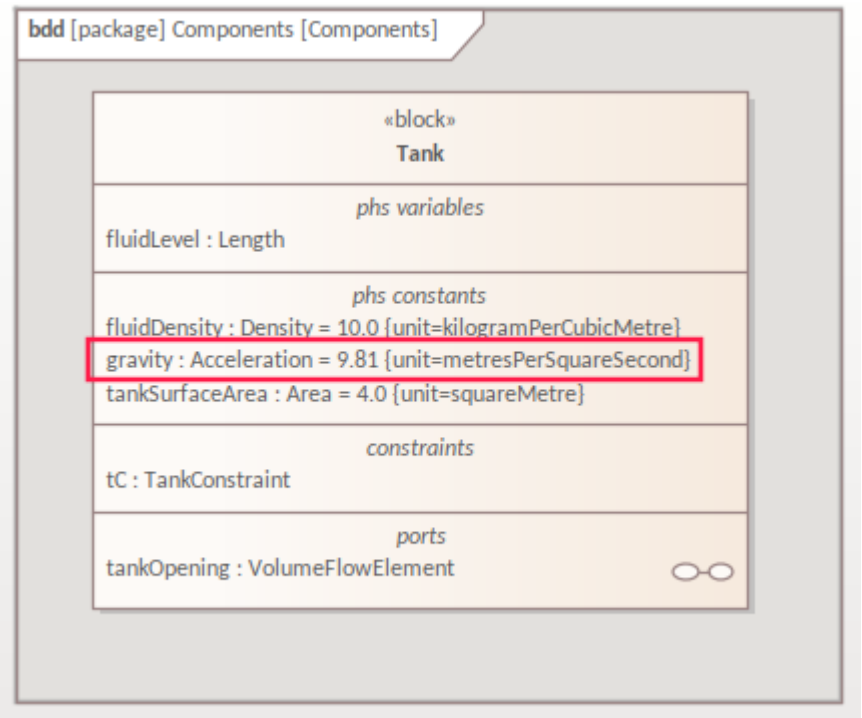
Setting Values

Having set a Block or a Part that references a component, in either Modelica or Simulink, you want to be able to set values to that external component's properties or parameters. The decision to place a value in the Block rather than in a Part derived from that Block is dependant on whether there will be any variation in that value for each Part.

Block Properties

In the example of a tank of water, in which the constant 'Gravity' is defined and is fixed for each tank instance, the value is best placed in the source Block.

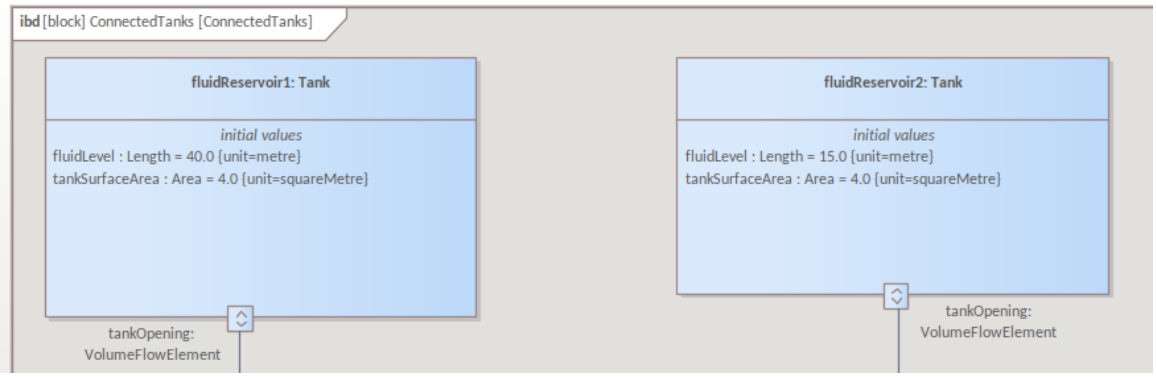
This illustration also shows the Fluid Density set for water (10 kg/m^3), which could be overridden either in a Part or in a simulation to define the density of another fluid, such as oil.



Part Properties

Where a Block is used repetitively with different values - e.g. to represent a resistor of 3.3 kilohms and another of 5.6 kilohms - then the initial value is best defined in the Block as blank, with individual values set in specific Parts derived from the Block.

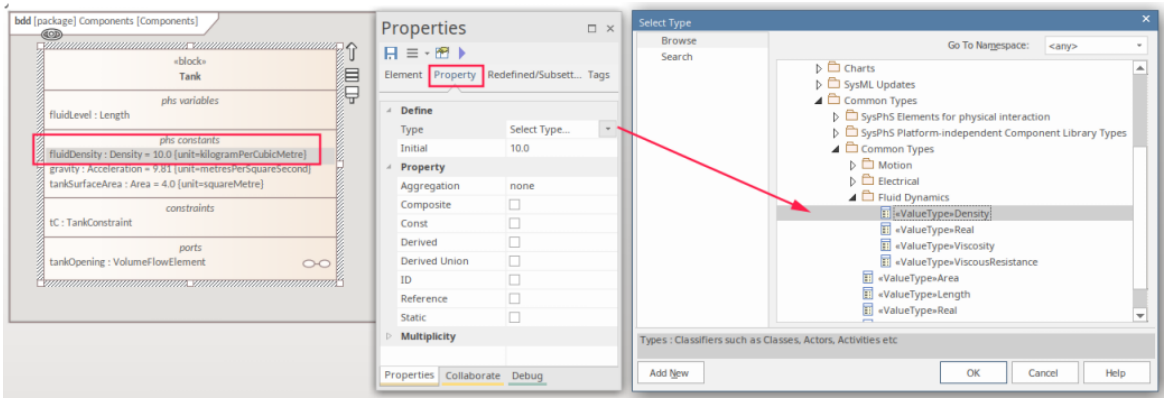
In this example we have two tanks with same radius, but different depths, so the fluidLevel (depth of the tank), is different for the two Tank parts created in this IBD.



Note that the Type is derived from the Block FluidLevel type.

Setting the Type

You can define Properties in a Block to reference a specific Value Type. This is set in the 'Type' field on the Properties > 'Property' tab; it is best to reference a Value Type using the ... button.



For more details see the *Modeling Quantity Using Value Types* Help topic.

Setting Values in a Simulation

In simulations it is often the case that variables must be set at run time rather than in the SysML model. This is where Datasets can be used to lay out a set of values to run though in a series of variations of the simulations. For more details see the *Model Analysis Using Datasets* Help topic.

Creating Modelica-Specific Blocks

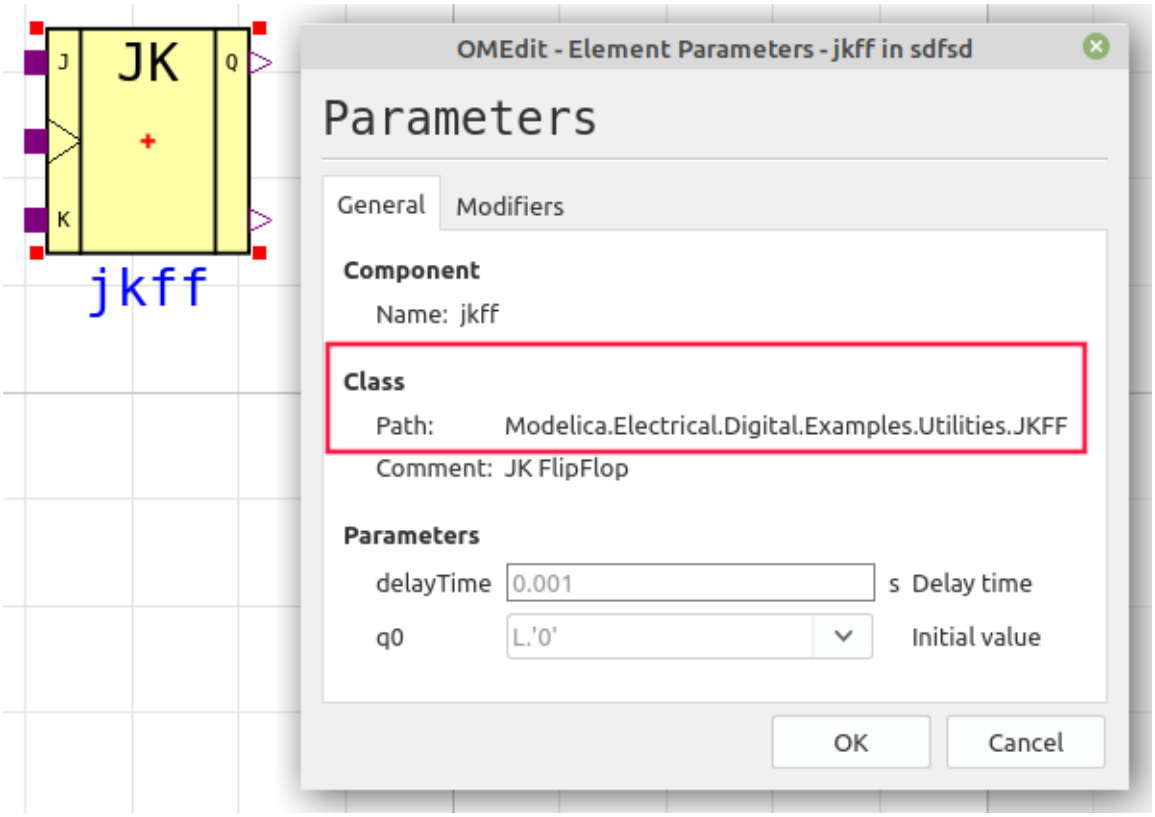
Given the broad range of different part-types that can be used in Modelica, there will be cases where you need to model Modelica components and parts that you cannot derive from the basic Blocks supplied in the SysPhS Component patterns. In such cases you set an Enterprise Architect component to reference the Modelica component.

Using the Modelica Class Path for a Block

The process to reference a Modelica component in an Enterprise Architect SysPhS component is to:

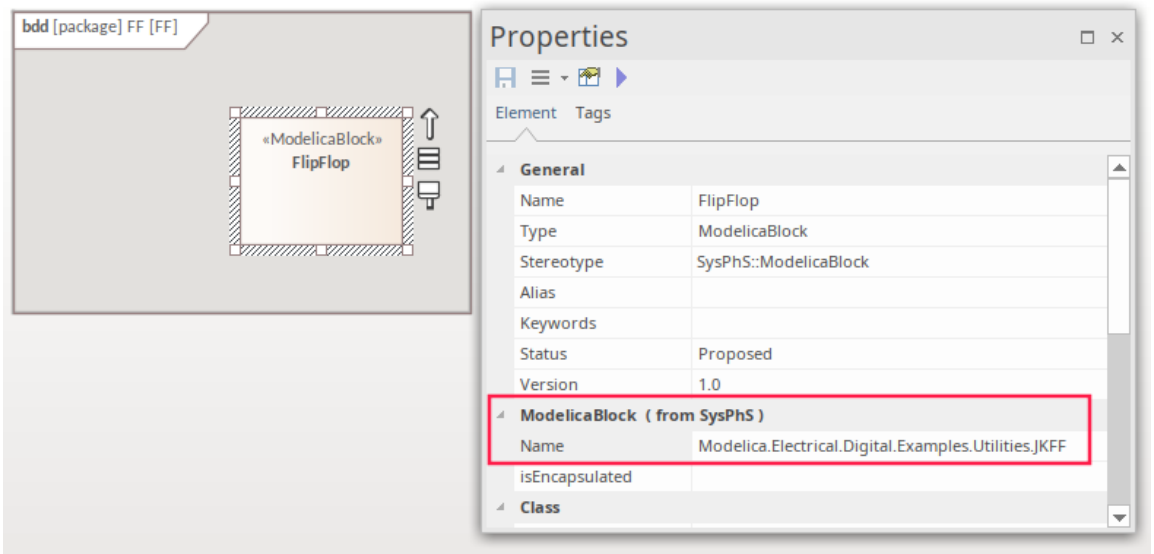
- In Modelica, access the Modelica component's Class Path
- Copy this text
- In Enterprise Architect, paste the copied text into the Modelica Block's 'Name' field

This illustration is an example of a Modelica component (a JK Flip-Flop) and its 'Parameters' dialog, showing the Class-Path.



- Note: The Class-Path can be selected by clicking and dragging the cursor across the path, then pressing Ctrl+C on the block of text.

Place the copied text inside the Properties window for the SysPhS Modelica Block, in the 'Name' field under **ModelicaBlock (from SysPhS):**.



Setting the SysPhS Ports

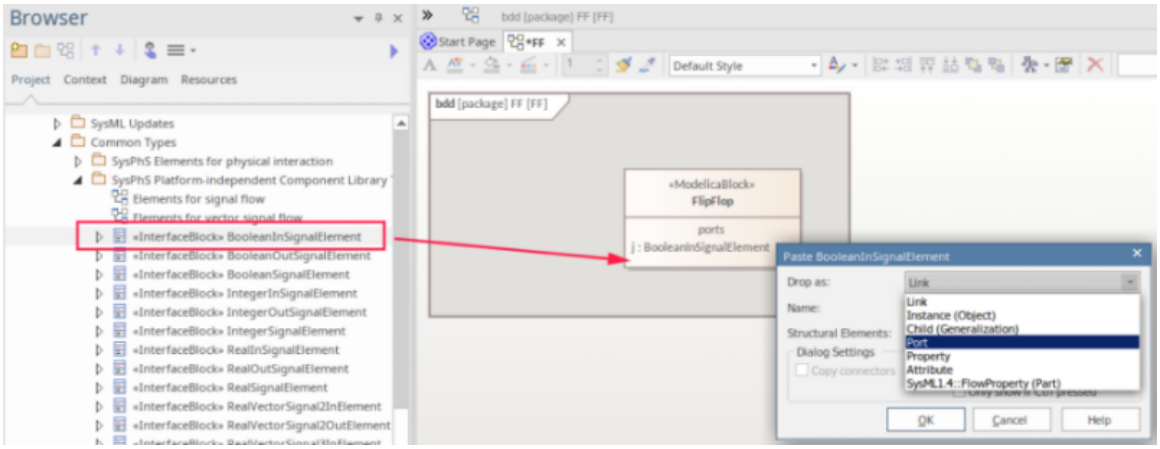
Ports on a Block can be non-typed Ports or predefined Port types.

Non-typed Ports are created by dragging a Modelica Port from the Toolbox onto a diagram.

To reference the SysPhS predefined Port types - such as a *boolean signal input Port* or an *analogue signal output Port*:

- Access the Port types in the *SysPhS Platform Independant Component Library* in the Browser window
- Drag a Port-type from the Library onto a Block on a diagram
- Set it as a Port on the Block

For example, this illustration shows the 'j' and 'k' Ports being created on the Flip-Flop, with the 'j' Port set and the 'k' Port in the process of being defined as a Port.



Creating Simulink and Simscape Specific Blocks

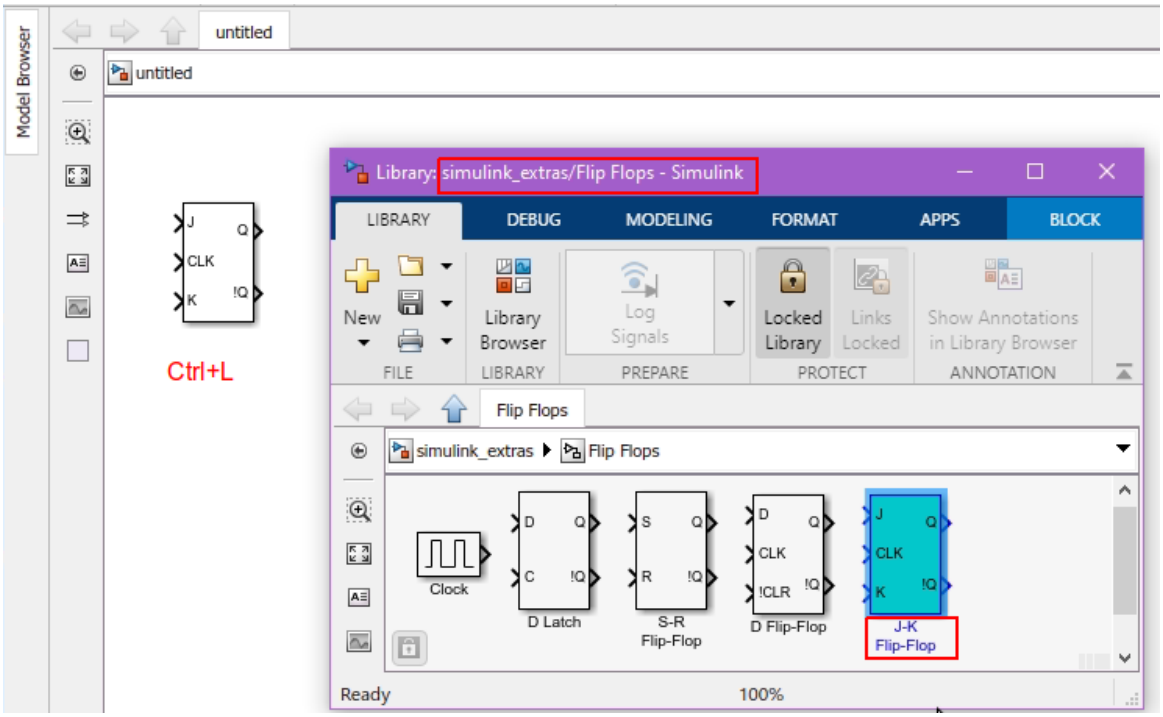
Given the broad range of different part-types that can be used in Simulink, there will be cases where you need to model Simulink and Simscape parts that you cannot derive from the basic Blocks supplied in the SysPhS Component patterns In such cases you can set a reference in a SysPhS Block to that Simulink component-type. This might be for a Block, a Part or an interface to a Part.

Using the Simulink Class Path for a Block

The process to reference a Simulink component in an Enterprise Architect SysPhS component is:

- In a Simulink diagram containing the component, click on the component and press Ctrl+L to access that component-type in the Library
- In the Library window, note the path in the window title, and the Component name under the Component element in the body of the window
- In Enterprise Architect, type the Component path and name into the Properties window, in the 'Name' field under **SimulinkBlock (from SysPhS):**

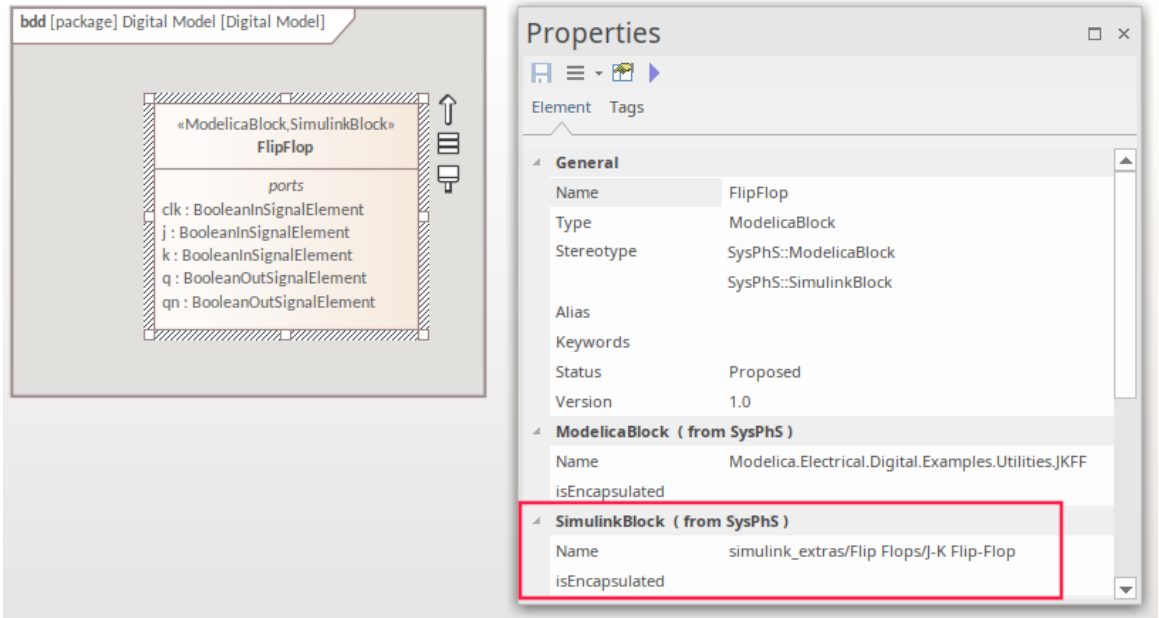
Here is an example of a Simulink JK Flip-Flop and the 'Parameters' dialog, showing the Class-Path.



In this case the path/name is:

- simulink_extras/Flip Flops/J-K Flip-Flop

This illustration shows the text added to the Properties window of the SysPhS SimulinkBlock, in the 'Name' field.



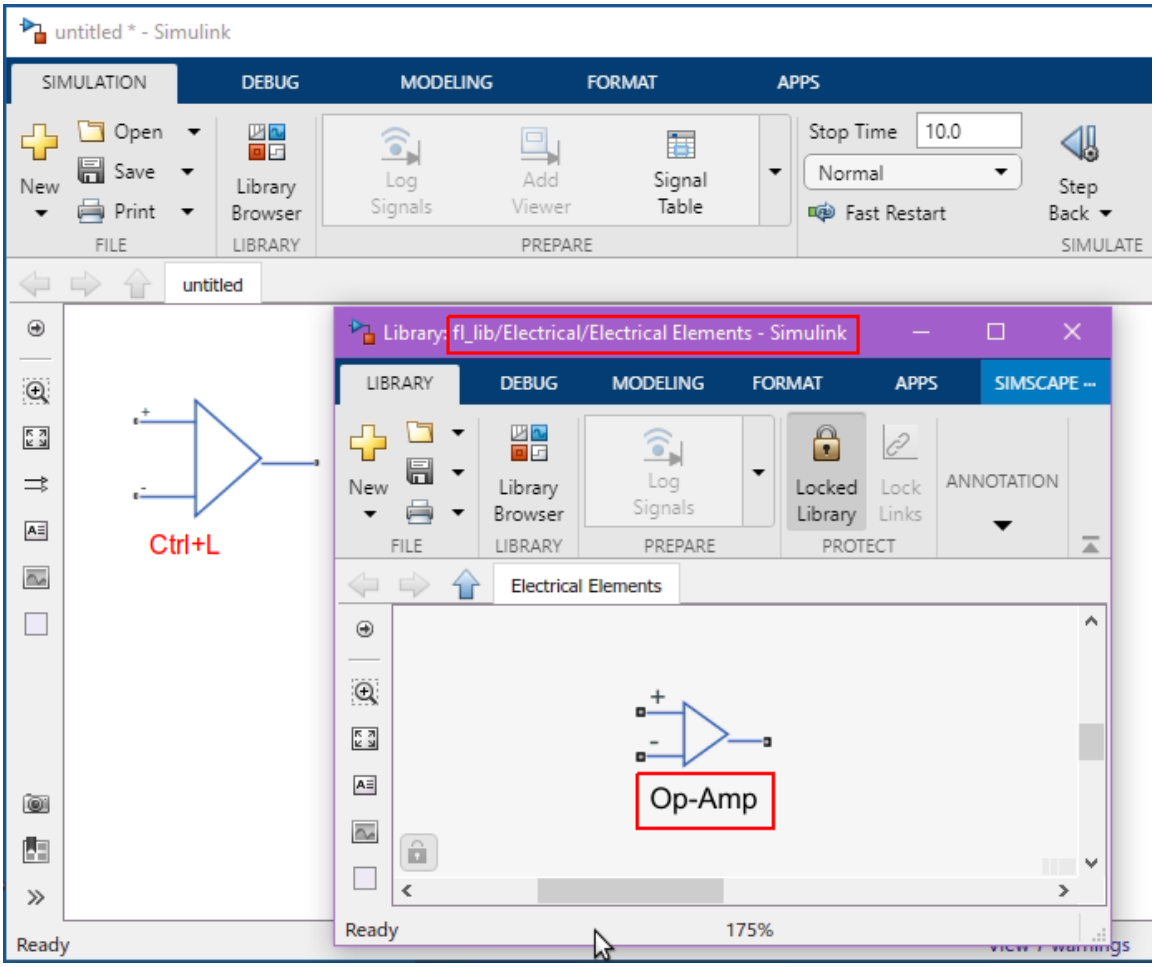
For more detail on setting the Simulink Class Path, see the *Declaring Member Components* 'MathWorks' Help topic.

Using the Simscape Class Path for a Block

The process to reference a Simscape component in an Enterprise Architect SysPhS component is:

- In a Simulink diagram containing the Simscape component, click on the component and press Ctrl+L to access that component-type in the Library
- In the Library window, note the path in the window title, and the Component name under the Component element in the body of the window
- In Enterprise Architect, type the Component path and name into the Properties window, in the 'Name' field under **SimulinkBlock (from SysPhS):**

Here is an example of a Simscape component for an Op-Amp.



Note that the fl_lib is referenced by 'Foundation', so the SysPhS 'Name' is:

- foundation.electrical.elements.op_amp

Setting the SysPhS Ports

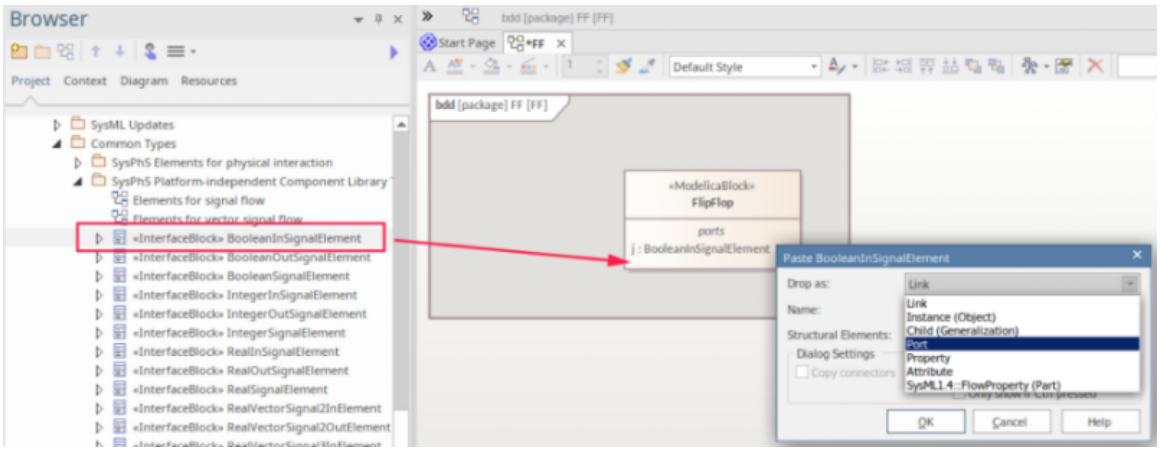
Ports on a Block can be non-typed Ports or predefined Port types.

Non-defined Ports are created by dragging a Simulink Port from the Toolbox onto a diagram.

To reference the SysPhS predefined Port-types, such as a *boolean signal input port* or an *analogue signal output port*, open the *SysPhS Platform Independent Component Library*

in the Browser window. A Port-type can be dragged onto the Block from the Library and set as a Port on the Block.

For example, this is a snapshot of the process for creating the j and k Ports on the Flip-Flop, showing the j Port set and the k Port in the process of being defined as a Port.



Simulink Port Ordering

Simulink Ports are defined using an array (as opposed to the name reference in Modelica), so the order of creation of the Ports is critical. The ordering of the IN Ports is separate from the ordering of the OUT Ports. Port ordering can be viewed in Simulink; the first Port is shown at the top of the Block.

Using the Flip Flop Ports example, the Simulink ordering of the IN Ports would be:

j, clk, k, (see the 'J-K Flip Flop' element in the first image of this topic)

So, these Ports must be created in Enterprise Architect in that order. Note that the Ports are shown in the Block Definition diagram in alphabetic order, not the order of

creation.

A common scenario where ordering has not been applied is when an OUT Port is correctly shown in the SysML diagram, but is incorrectly connected to an IN Port when simulating in Simulink. If this occurs, ensure that the order of creation of the Ports is correctly applied.

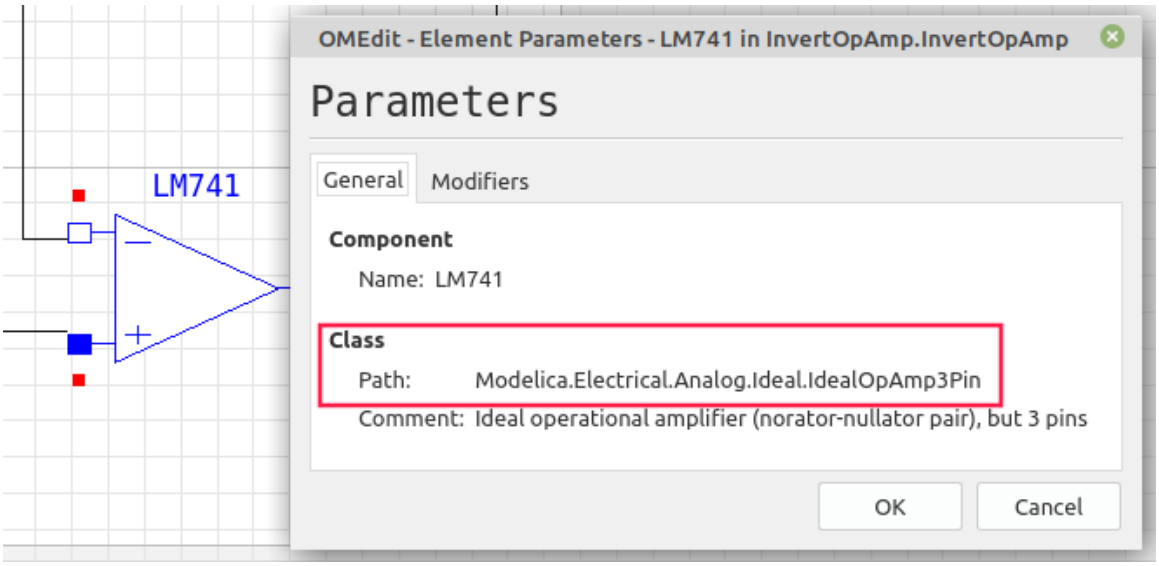
Setting Blocks as Both Modelica and Simulink

For cases where both Modelica and Matlab are used, or for setting generic templates to cover both products, you can set the references to both products' component-type in a SysPhS Block.

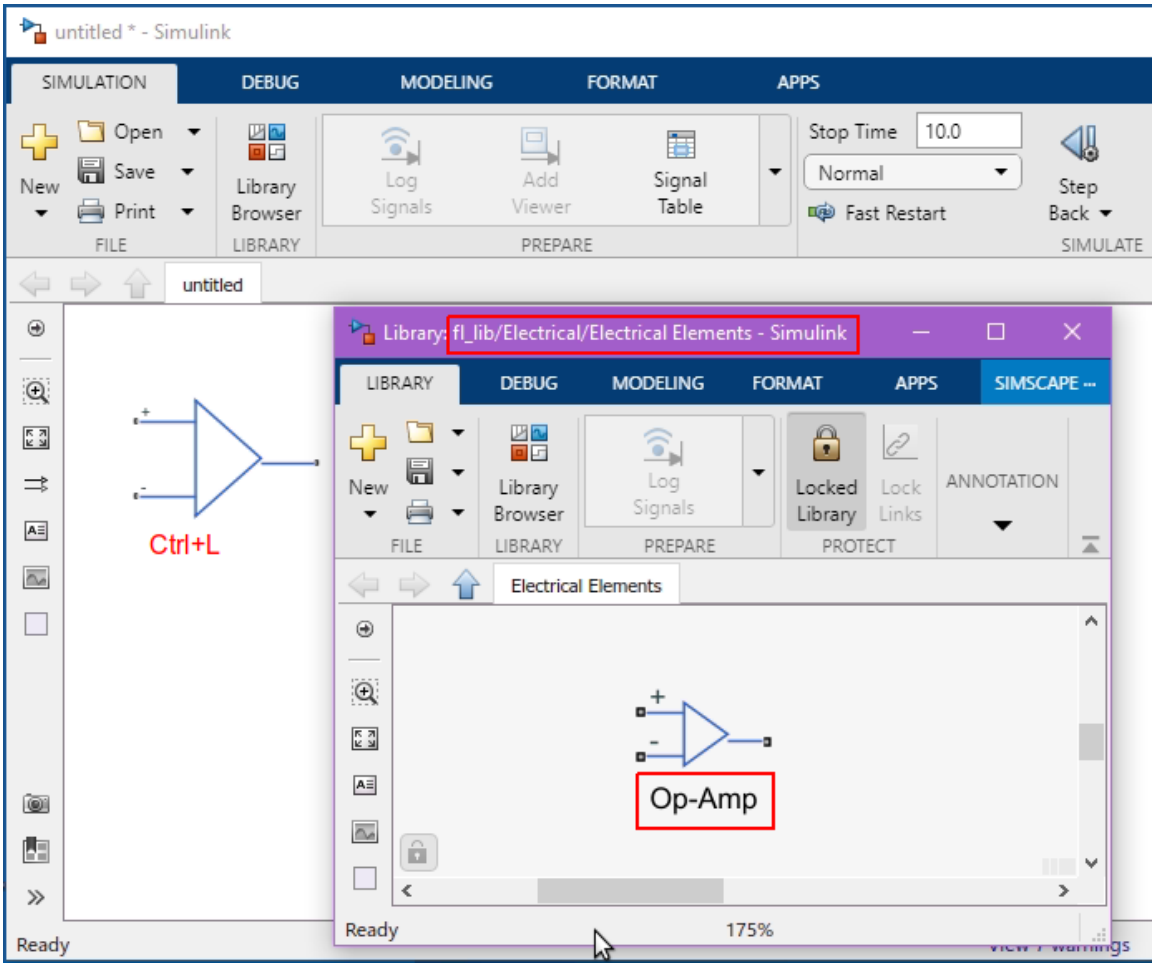
The predefined Blocks that come with the SysPhS Patterns already include both the Modelica and Simulink properties.

Example

Here is an example of a Modelica component for an Op-Amp.

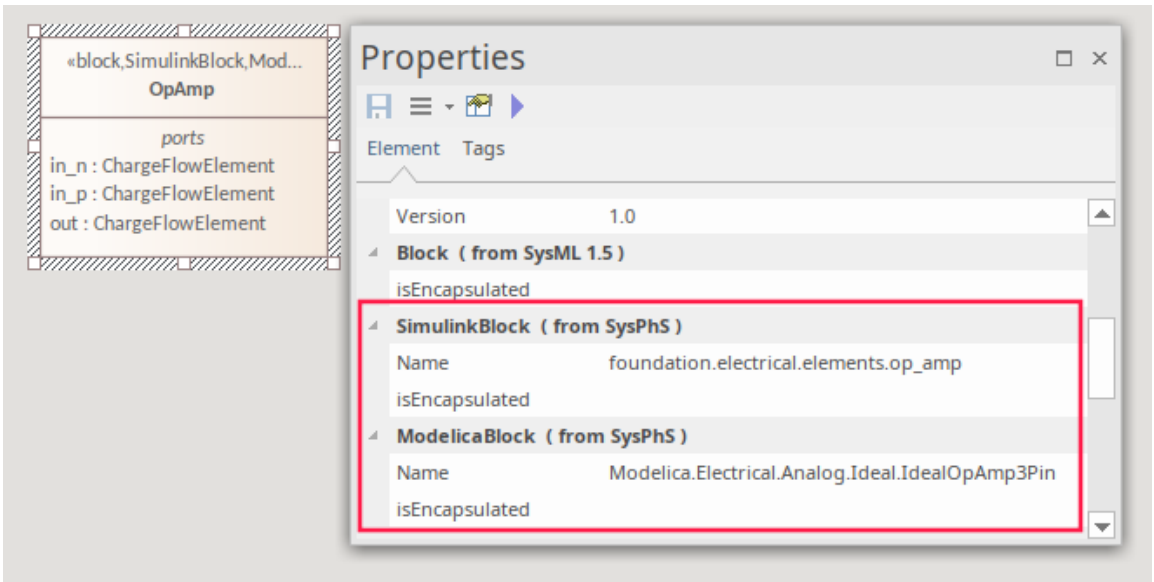


Here is an example of a Simscape component for an Op-Amp.



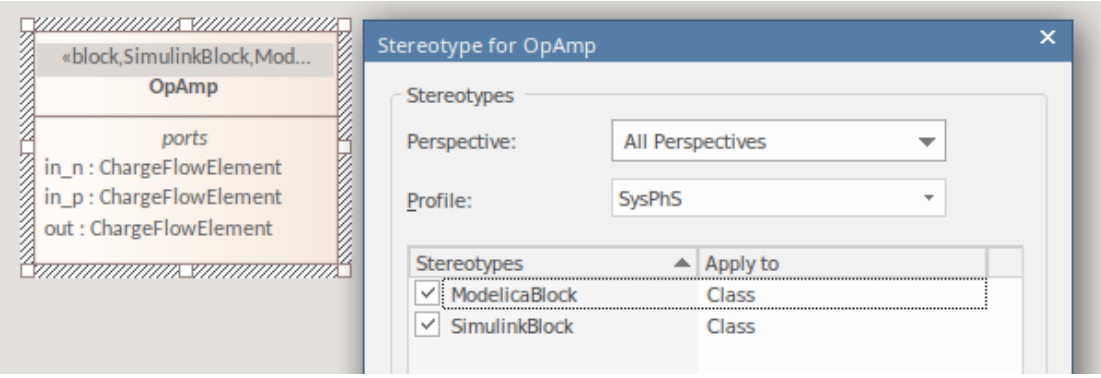
Note that as the fl_lib is referenced by 'Foundation', the SysPhS 'Name' is *foundation.electrical.elements.op_amp*.

Here are the 'Name' settings for both Modelica and Simulink.



To set the properties manually for both products, you must set the Stereotype for both. To do this on a newly-created SysPhs Block element, in the Properties window for that element:

- Select 'Stereotype'
- Click on the [...] button
- Select SysPhS in the Profile
- Tick both the SimulinkBlock and ModelicaBlock stereotypes



Simulation

Enterprise Architect provides integration with OpenModelica and MATLAB's Simulink, to support a 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 export your model to MATLAB Simulink, Simscape and/or Stateflow.

This section describes the process of defining an Internal Block Diagram or Parametric model, annotating the model with additional information to drive a simulation, and running a simulation to generate a graph of the results.

Introduction to SysML Internal Block and Parametric Models

SysPhS 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 Requirement 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.

These topics describe the key options for defining and running Systems Engineering simulations using SysML and SysPhS.

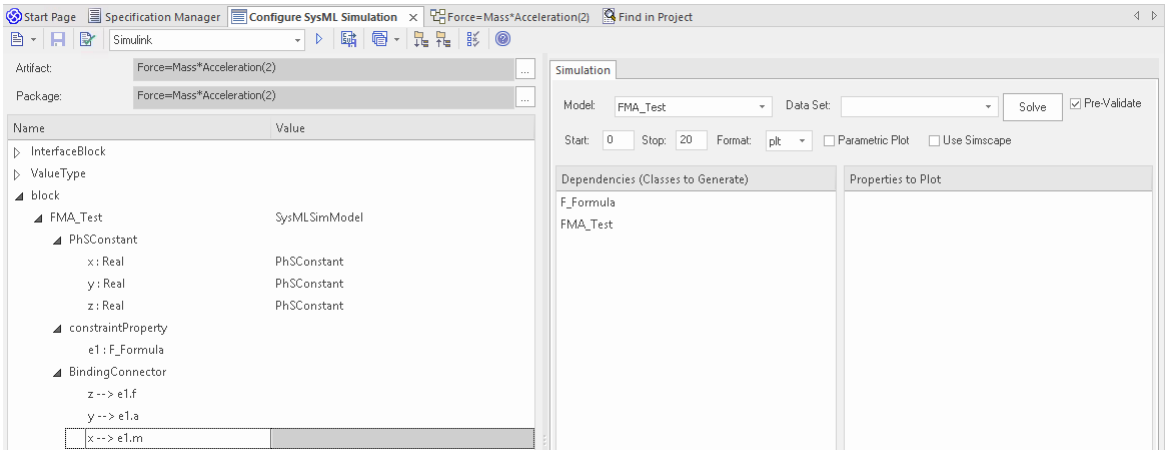
Topic	Description
SysPhS Standard Support	The *SysPhS Standard* is a *SysML Extension for Physical Interaction and Signal Flow Simulation*. It defines a standard method of translating between a SysML model and either a Modelica model or a Simulink/Simscape model, providing a simpler model-based method for sharing simulations.
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* topic.
Viewing the Generated Model	Once a model has been generated to Modelica or Simulink you can directly work with that model in the external application.
SysPhS Debugging Tips	As there can be many factors that can cause an error in a generated script, we provide some tips on isolating the source of an issue in the model.
Model Analysis Using Datasets	When working with a simulation there can be cases where multiple variants are to be tested. To support this, multiple datasets can be defined against Blocks, allowing for repeatable simulation

	variations using the same SysML model.
SysPhS Simulation Examples	To aid your understanding of how to create and simulate a SysPhS 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 *SysPhS 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.

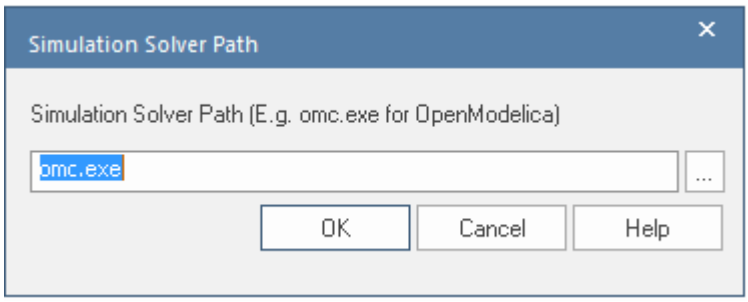


Access

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. For MATLAB/Simulink, the path will be automatically detected so this only needs to be changed if there are issues with the automatic detection, or there are multiple versions of MATLAB installed.

	
	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 arrow 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* Help topic.

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 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 stereotype has 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 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	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.

Viewing the Generated Model

Once a model has been generated out to either Modelica or Simulink there can be cases where you want to directly work with that model in the external application. This includes any cases where the model is not generated correctly and you need to find the issue in the external application.

Run a model generation using the  or Solve button and then view the generated code.

Viewing the Generated Code

To view the generated Modelica or Simulink code in the Enterprise Architect code editor:

- Click on the  icon in the Toolbar
- Select the 'Open Simulation Directory' option
  This opens the directory into which the OpenModelica or Simulink code was generated
- Click on the file to view the script

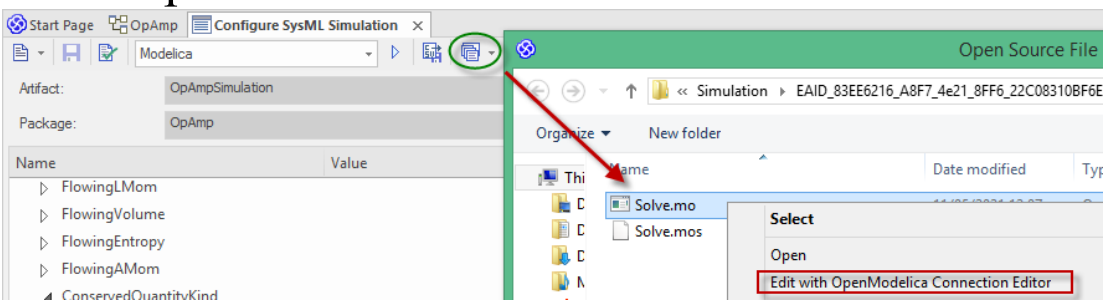
Accessing the Generated Model

To access the Modelica or Simulink model using the generated code:

- Click on the  icon in the Toolbar
- Select the 'Open Simulation Directory' option
  This opens the directory into which the OpenModelica or

Simulink code was generated

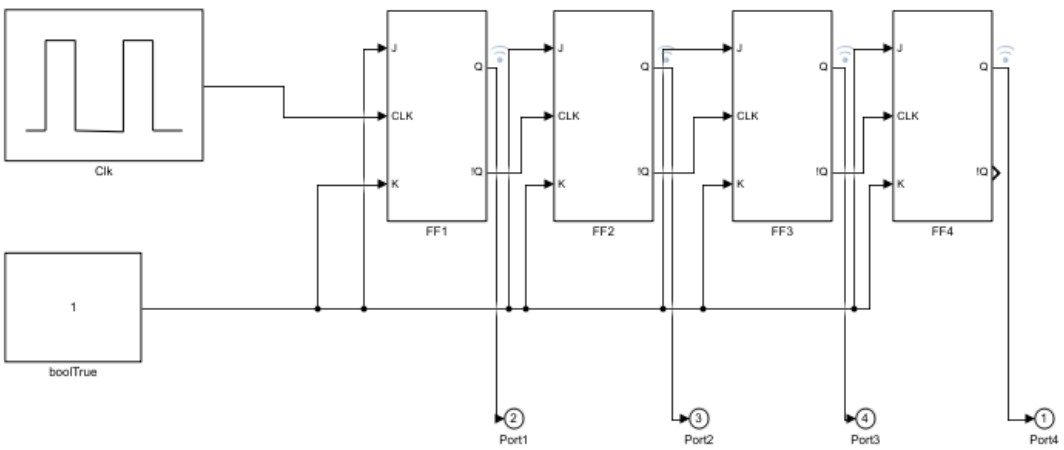
- Right-click on the file
- Select from the context menu the 'Edit with <*external tool*>' option:



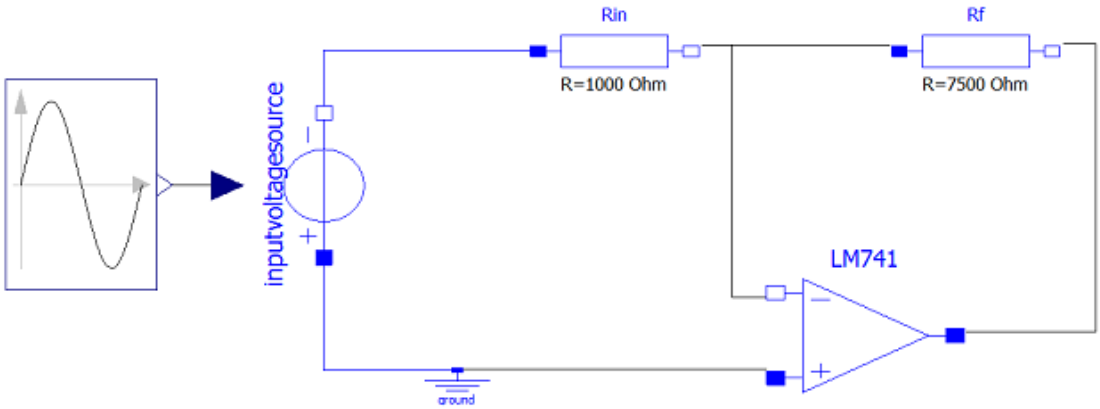
In this case a Modelica .mo file is shown. For a Simulink-generated file the filename is Solve.m.

Double-click on the file to open the external application:

- For Simulink, to view the generated model, you must open the .slx file named after the Block set to SysMLSimModel; here is a view of the Flip-flop example, generated and opened in Simulink:



- For Modelica, in the Libraries browser, the diagram is named as per the SysMLSimModel Block - double-click on it to open it; here is a view of the OpAmp example, generated and opened in Modelica:



See the *Configure SysML Simulation* Help topic for more details.

SysPhS Debugging Tips

As with any code generation from a model, there can be many factors that can cause an error in the generated script used in the external application. Hence the need for options to find the cause of the issue. This topic provides some tips on isolating the source of an issue in the model.

Before checking the generated code, an external script must be generated using the  or Solve button.

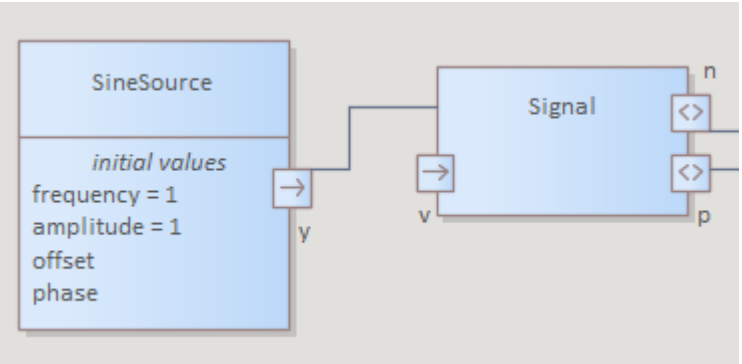
Finding an issue

For finding any issues with the generated script, the key options to use include:

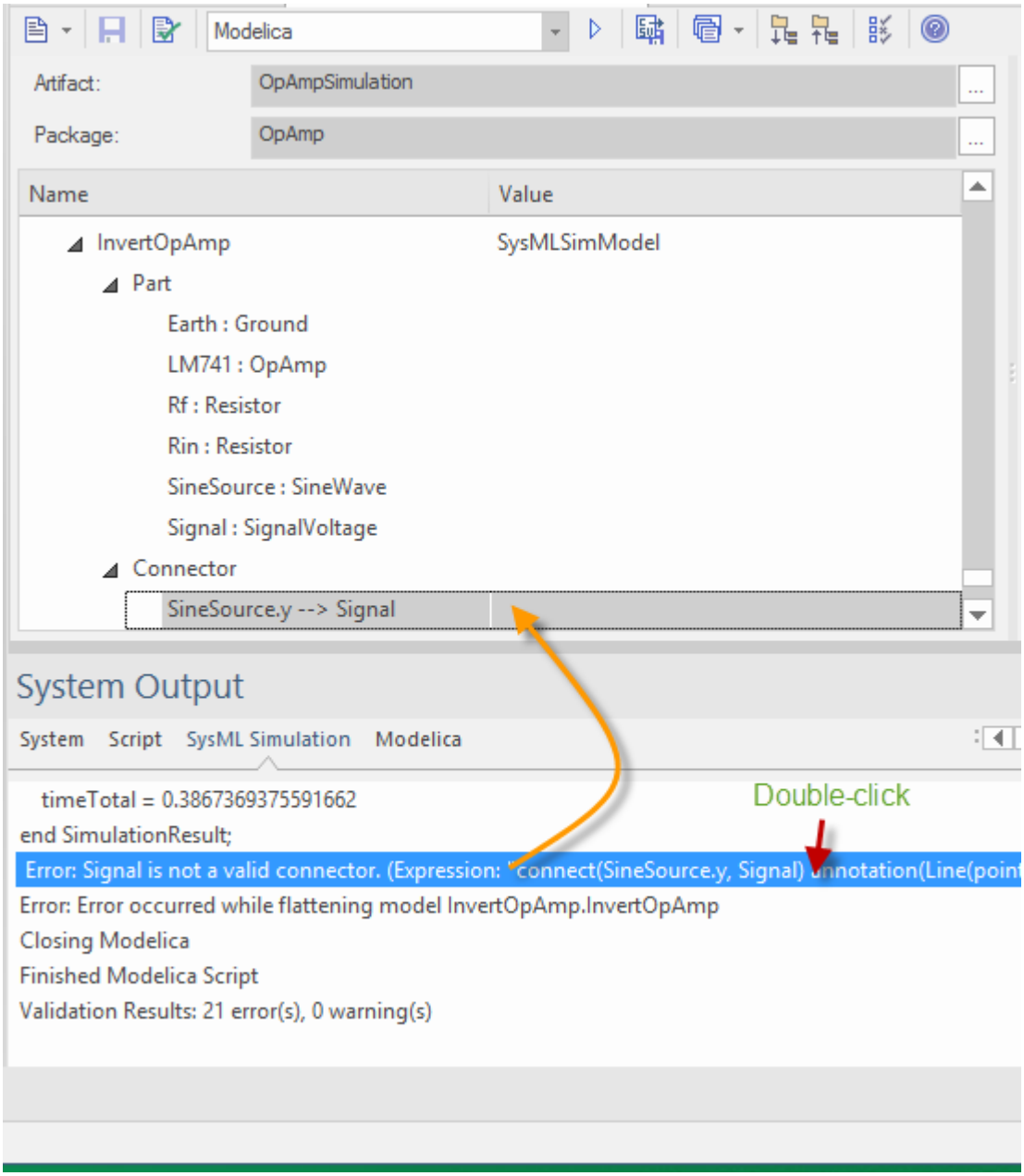
- For the generation process, a listing of the steps run through and any errors generated are logged in the System Output view (Ctrl+Shift+8)
- For any error listed in the System Output window, if it is related to a specific element, double-clicking on that error will locate the associated object in the left pane of the Configure Simulation window
- Where access to the generated script is needed, use the 'Open Simulation Directory' option under the  icon on the Toolbar; see the *Viewing the Generated Model* Help topic

Example

In this example we have a Part in the IBD that is incorrectly linked to another Part. That is, a direct link, not a link through a Port. This causes an error when generated to both Modelica and Simulink.



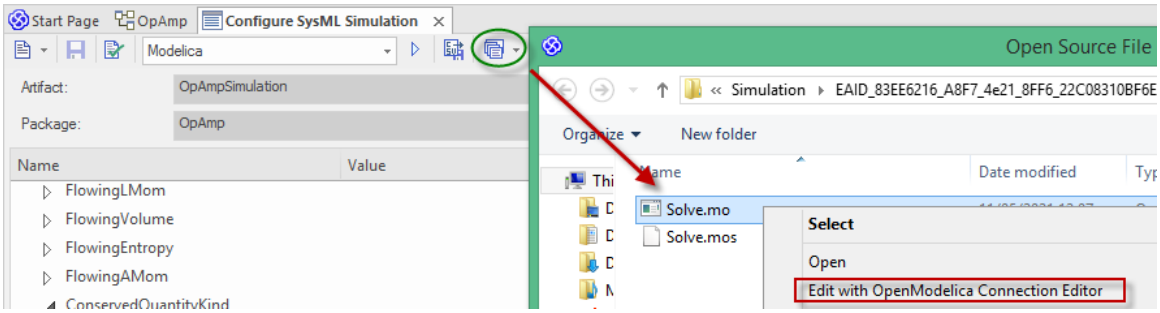
What is generated in the System Output is an error. Double-clicking on that error highlights the related object in the Simulation list.



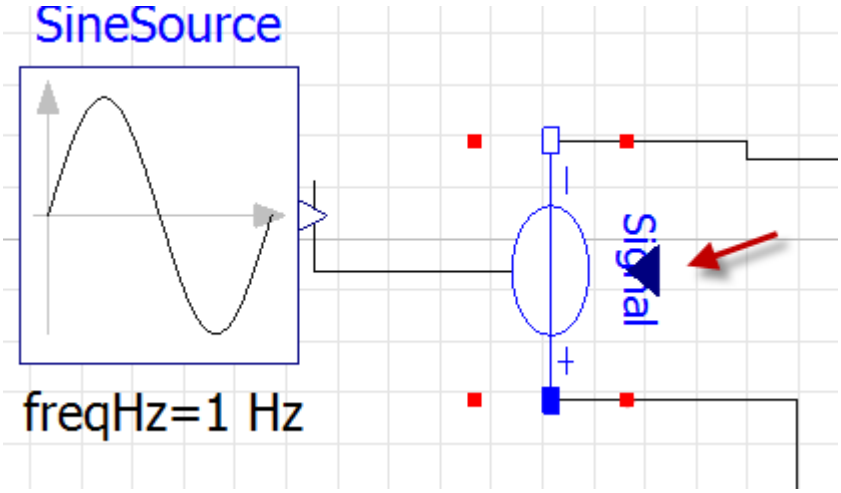
This can be further verified by opening the generated script in the external application and viewing the generated diagram, by:

- Clicking on the icon from the Toolbar
- Selecting the *Open Simulation Directory* option
- Using the context menu to open the file in the external application

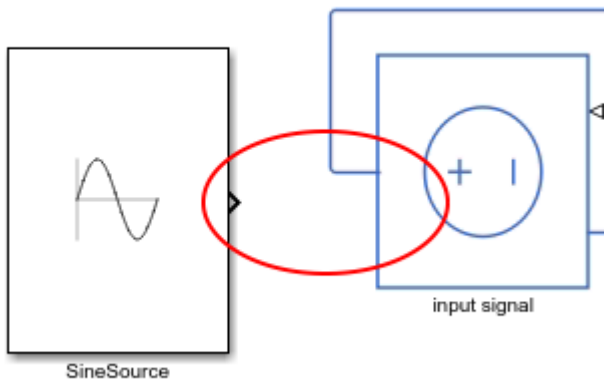
For example, in this case in Modelica:



When opening the generated model in Modelica, using the icon on the Toolbar, this is the view showing that the connector is not linked to the triangle that is the import to the Signal object:



Similarly for Simulink, this is the resulting model showing the lack of a connector from the SineOutput:



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

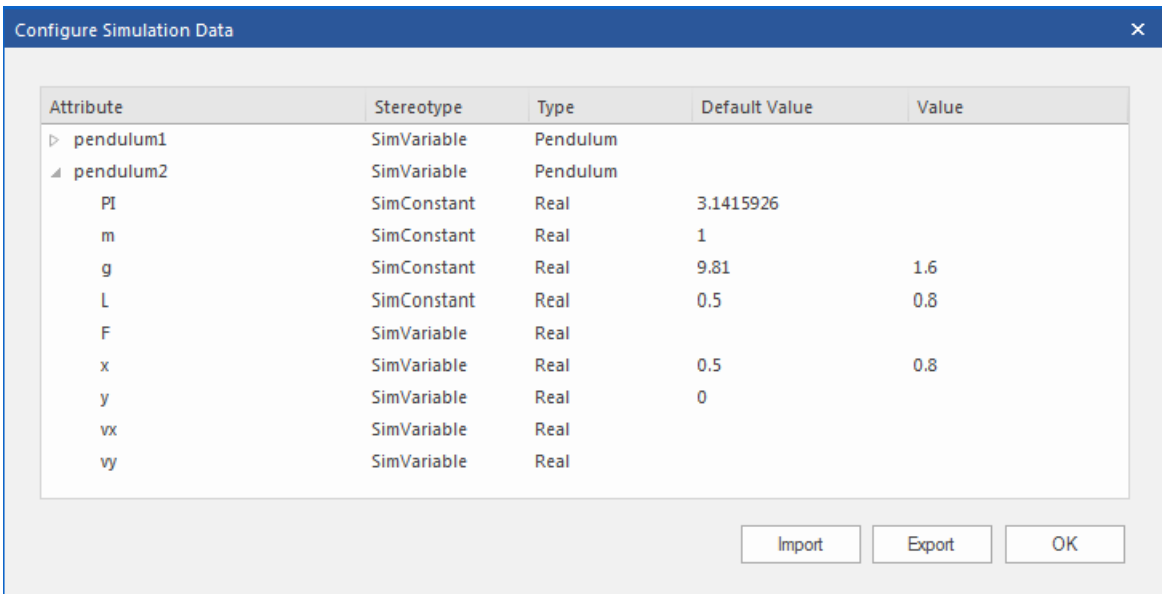
Dataset Management

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, whether it has been configured to be a constant for the duration of the simulation, or a variable for which 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.

SysPhS 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
Analog Electrical Circuit OpAmp	The first example is of the simulation of a simple electronic circuit. The example starts with a Block Definition diagram of Blocks defining the circuit components, and a Block containing an Internal Block Definition diagram that outlines the circuitry. The model is then simulated, and the sine wave voltages at the source and the target terminals of the OpAmp are evaluated and compared to the expected values.
Digital Electronics Simulation Example	The second example shows a standard Flip Flop component that is predefined in both Modelica and Simulink, to create a simple Flip Flop based digital signal frequency divider. This suggests how to

	use the SysPhS notation to reference components that are available in Modelica and Simulink, but not defined in the SysPhS patterns.
Humidifier Simulation Example	The example discusses using a SysML StateMachine diagram for defining the on-off state of a humidifier in operation. This is generated to Modelica and MATLAB's StateFlow diagram for simulation.

Learn More (Videos and WebEA)

- SysML Simulation in Simulink
- SysML Tank Example
- MATLAB Stateflow for StateMachines
- SysPhS OpAmp example
- SysPhS Starter
- Simulating Digital Electronics using Modelica
- WebEA: EA Sim examples

OpAmp Circuit Simulation Example

For this example, we walk through the creation of a SysPhS model for a simple electronic circuit, and then use a simulation to predict and chart the behavior of that circuit.

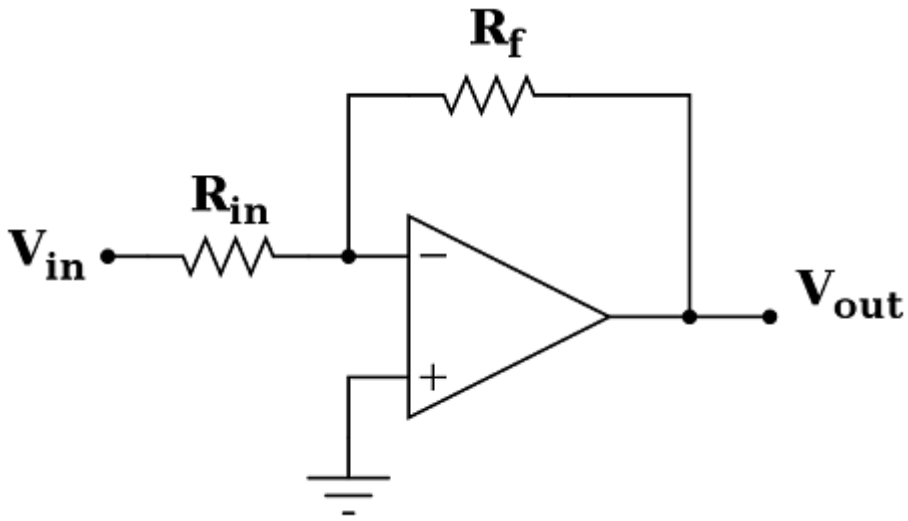
Prerequisites

Running this simulation requires either:

- OpenModelica or
- MATLAB's Simulink and Simscape

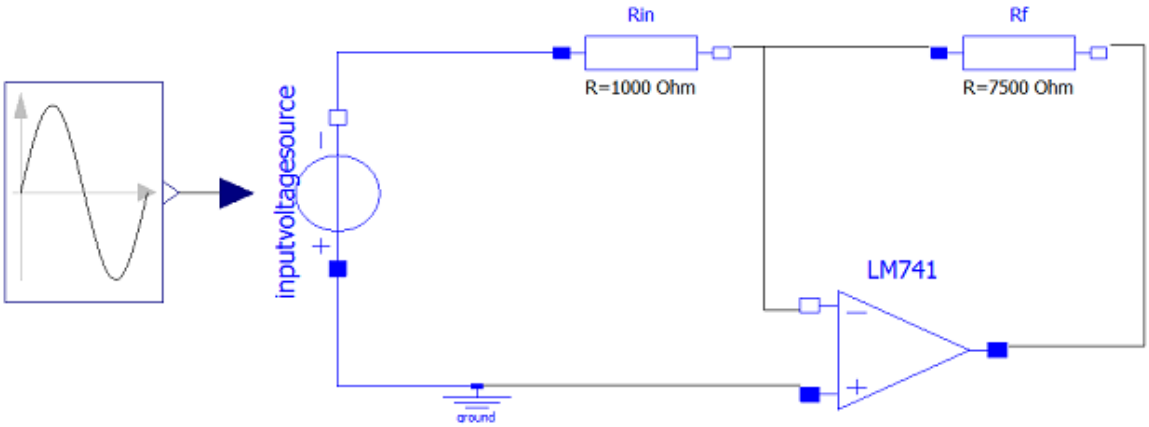
Circuit Diagram

The electronic circuit we are going to model is illustrated in this figure, using standard electronic circuit notation.



The circuit in this example will include a sine wave signal source, an earth, two resistors, and an OpAmp amplifier integrated circuit. The next figure shows the resulting

diagram, as generated from a SysML diagram using SysPhS, into an external application - in this case Modelica.



Create SysML Model

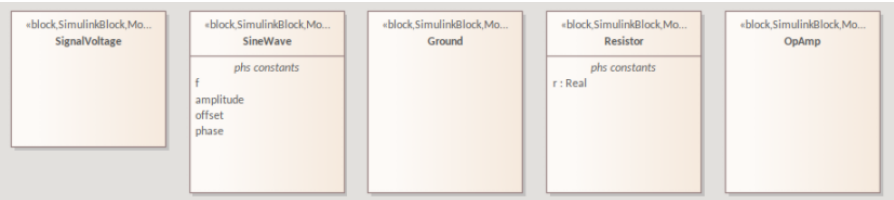
This table shows how we can build a complete SysML model to represent the circuit, starting at the lowest level types and building up the model one step at a time.

Component	Action
Blocks	In SysML, using SysPhS, the circuit and each of the component-types are represented as Blocks. In a Block Definition diagram (BDD), create a circuit component Block. The circuit has five parts: an OpAmp, a signal source, a signal to voltage converter, a ground, and a resistor. These parts are of different types, with different behaviors. The key Blocks, including the OpAmp, Ground and Resistor, are accessible as

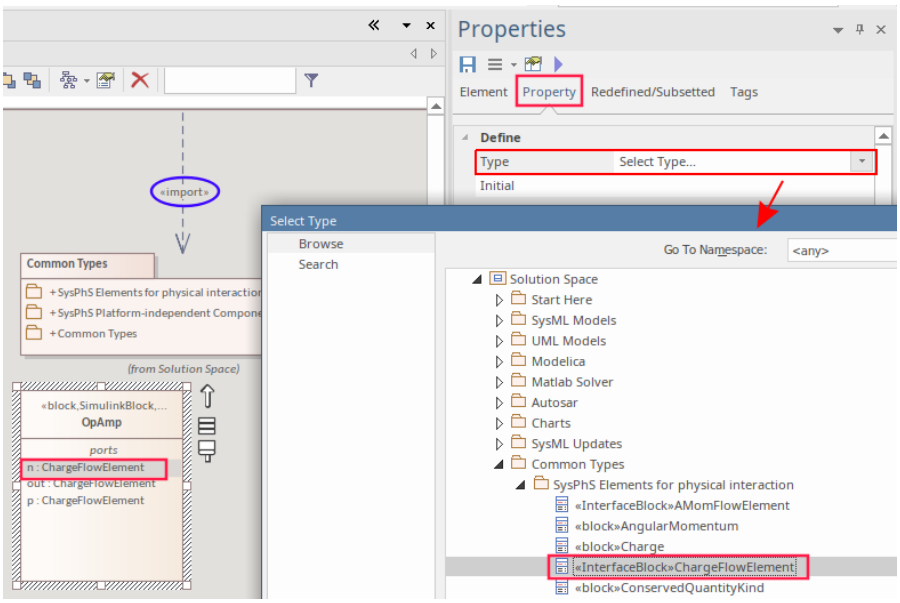
pre-defined Blocks, using the SysPhS Component Patterns. However, for clarity, we will run over their creation from scratch.

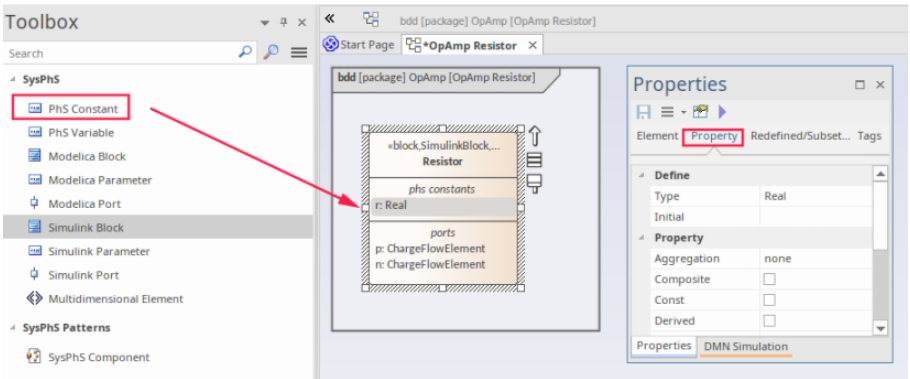
Create a SysPhS Block for each of the Part types. The parts of the Circuit Internal Block Definition (IBD), will be connected through Ports, which represent electrical pins. These are defined in the BDD. The resistor has a positive and a negative pin. The ground has only one pin, which is positive. Electricity (electric charge) is transmitted through the pins. The two Ports are named 'p' (positive) and 'n' (negative), and they are of type ChargeFlowElement.

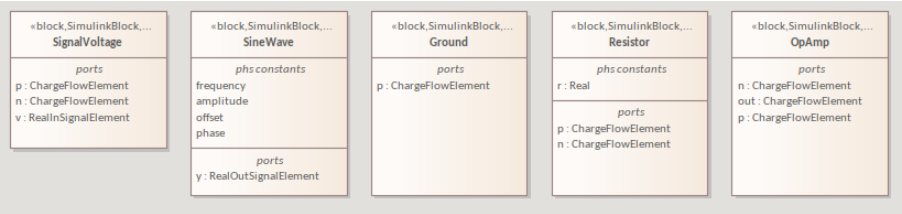
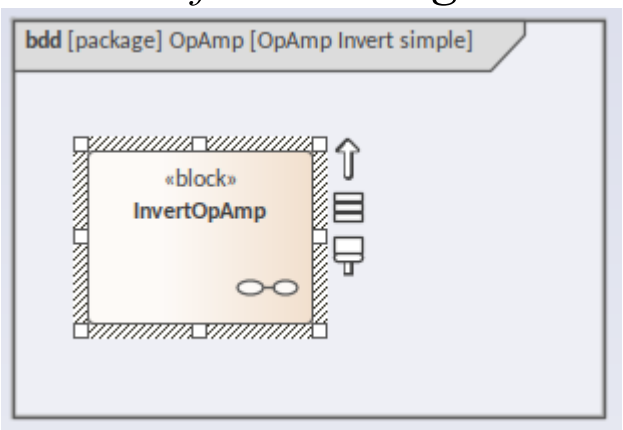
This figure shows the BDD, with Blocks defining the type of components used. This includes a signal source sine wave, a signal voltage converter, a Ground, a Resistor-type and an OpAmp-type.



Note: These Blocks are created from the SysPhS Toolbox using either Modelica Blocks or Simulink Blocks. You can type each Block to both tools. For more information see the *Setting Blocks as*

	Both Modelica and Simulink Help topic.
Phs Ports	The Value Types used for the Ports are pre-defined in the SysPhS Simulation Libraries. The two key quantities used are the ChargeFlowElement and the RealSignalInElement ValueTypes, This figure shows the OpAmp Block in the Block Definition diagram, with the Ports on the OpAmp Block set to the ChargeFlowElement ValueType, and where this is referenced in the Browser window.  The Import connector to the Common Types Package shows how these must be set. For more information on the Package Import (circled), see the *Referencing the SysPhS Simulation Libraries* Help Topic.

Phs Constants	Both the Resistor and the Sine Wave Blocks have properties defined in their respective components in MATLAB and Modelica. Let's use the resistor as an example. For these components, in their respective tools, we need to set a value of the resistance in Ohms. For most circuits there might be multiple resistor-parts that are derived from this Resistor Block, modeled as Parts in the IBD or Parametric diagrams. Hence, their value (in Ohms), will be set in the IBD, Parametric diagram or possibly in the simulation Datasets. To set the property defining the resistance we need to create a Phs Variable 'r'. The initial value is left unset.  **Note**: The SysPhS Patterns include a Block of type Resistor that is already constructed. For setting the resistance values in each individual component (Part), see the

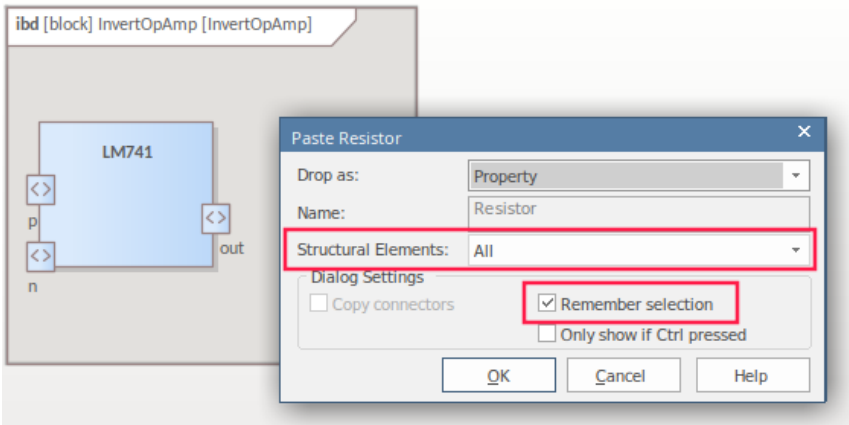
	Initial Values row in this table, which shows the Blocks set with the Ports and Phs Constants. 
Internal Structure	For the internal structure we create a Block with a child IBD diagram. • Create a Block for an inverting OpAmp circuit • Under this create an Internal Block Diagram (IBD) using the context menu option: *Create New Child Diagram | Internal Block Definition diagram*  • Double-click to open the IBD • Set the diagram to only show the Port's name, using: - F5 | Element | Element Appearance

- Unset these two options:
  - Show Stereotypes
  - Show Property Type

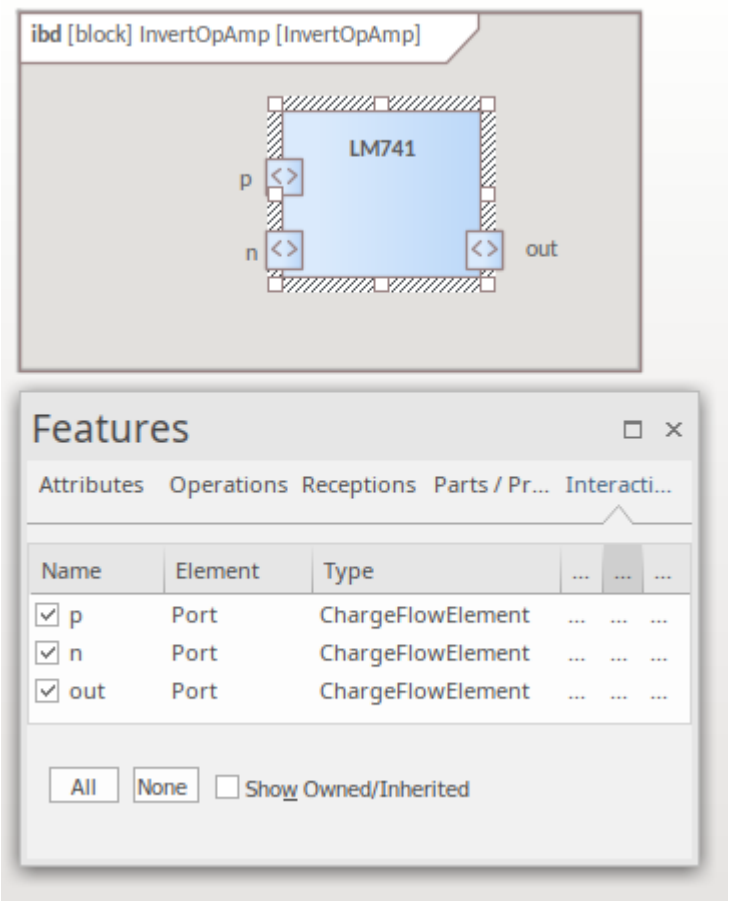
  This will then show only the Port and Type

On the IBD, create Parts and connect them:

- From the Browser, drag the Blocks onto the IBD as Parts (Properties)
- To view the Parts in compartments, delete them from the diagram
  Note: Set the Paste option 'Structural Elements to 'ALL'

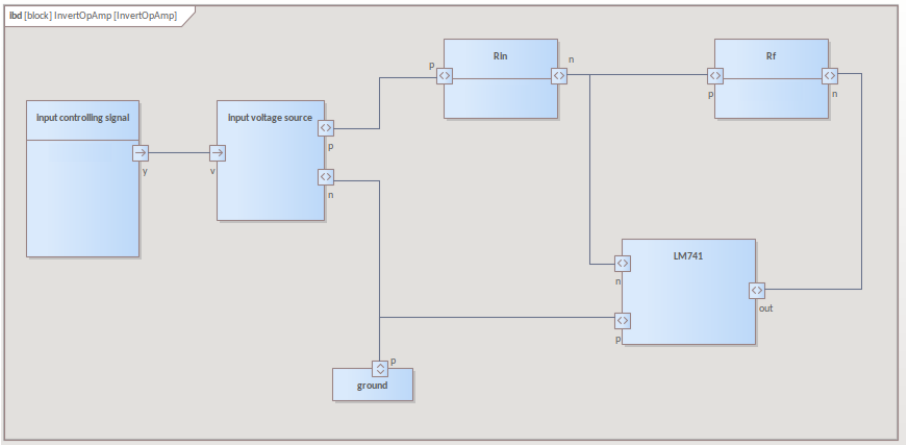


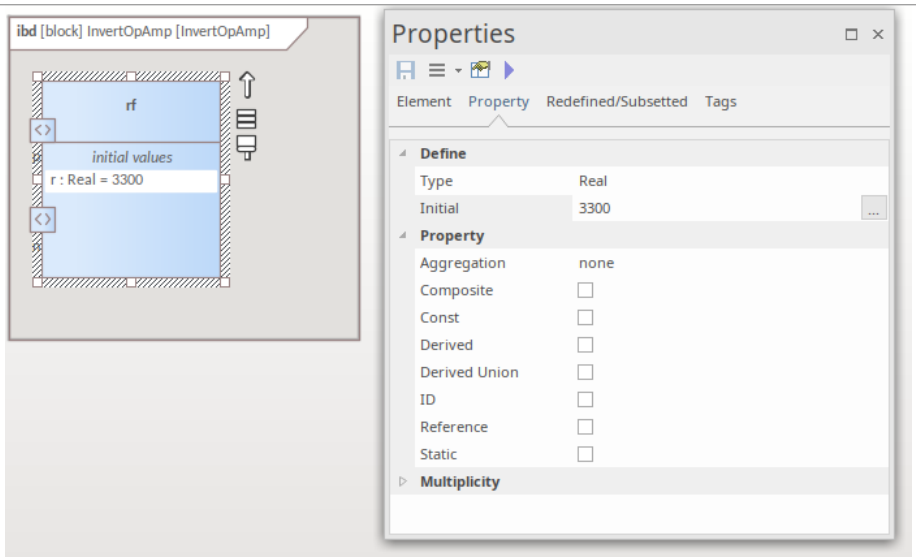
- Otherwise, in the Features view, for the 'Interaction' tab click on the All button, and for the 'Parts / Properties' tab click on the All button.



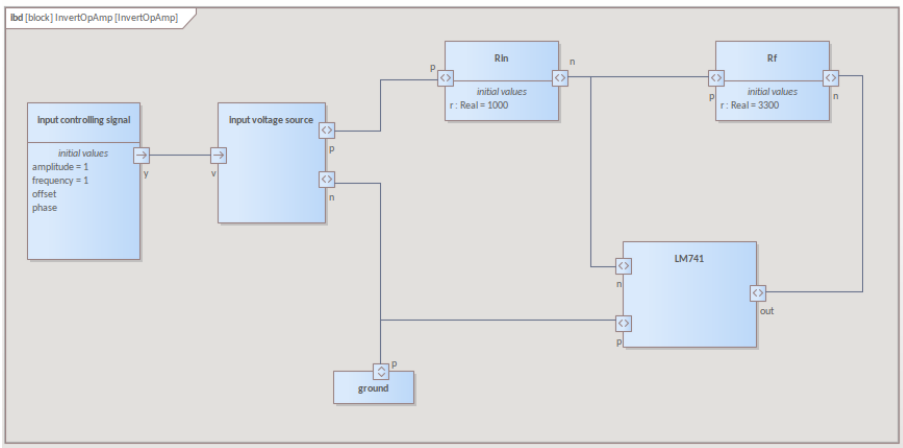
- Add properties for the Source signal, converter, 2 Resistors, OpAmp and Ground, typed by the corresponding Blocks

Bindings	To model the wiring up of these components: • Set the connections between the Ports with connectors of type Connector.

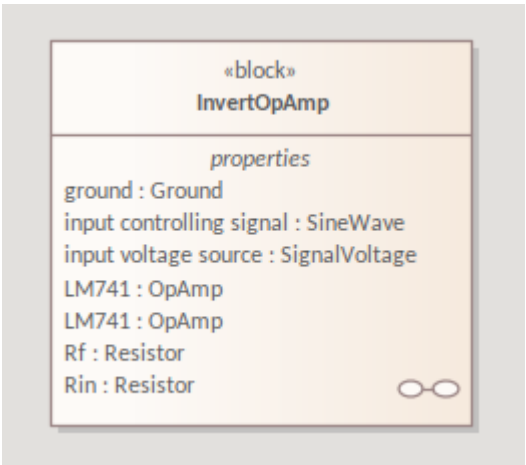
	 Notice that this follows the same structure as the original circuit diagram, but the symbols for each component have been replaced with properties typed by the Blocks we have defined.
Initial Values	As there are two Resistors, the input resistor (RIn) and the feedback resistor (Rf) must be created as two different Properties (Parts) in the IBD, sourced from the Resistor Block, as each of them will have different values. The values must be set in the Properties window, 'Property' tab, 'Initial' field.



For the InputControllingSignal, initial values are required for the amplitude and frequency.

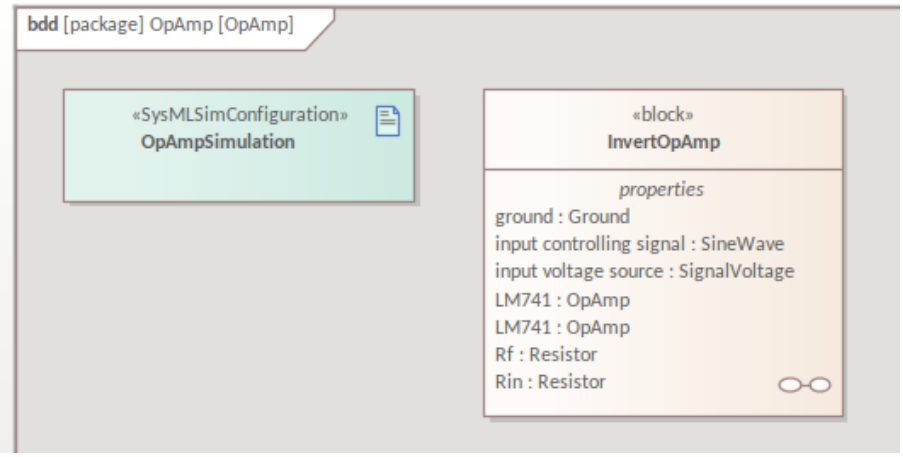


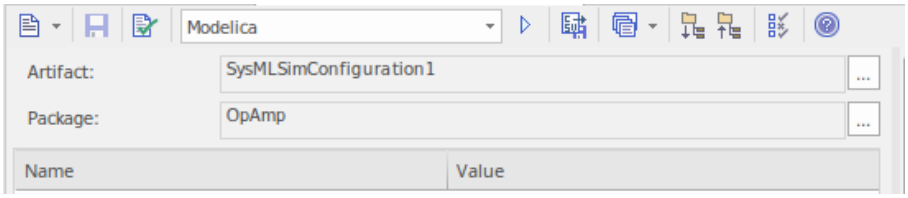
On returning to the BDD you should now have the InvertOpAmp Block shown as:

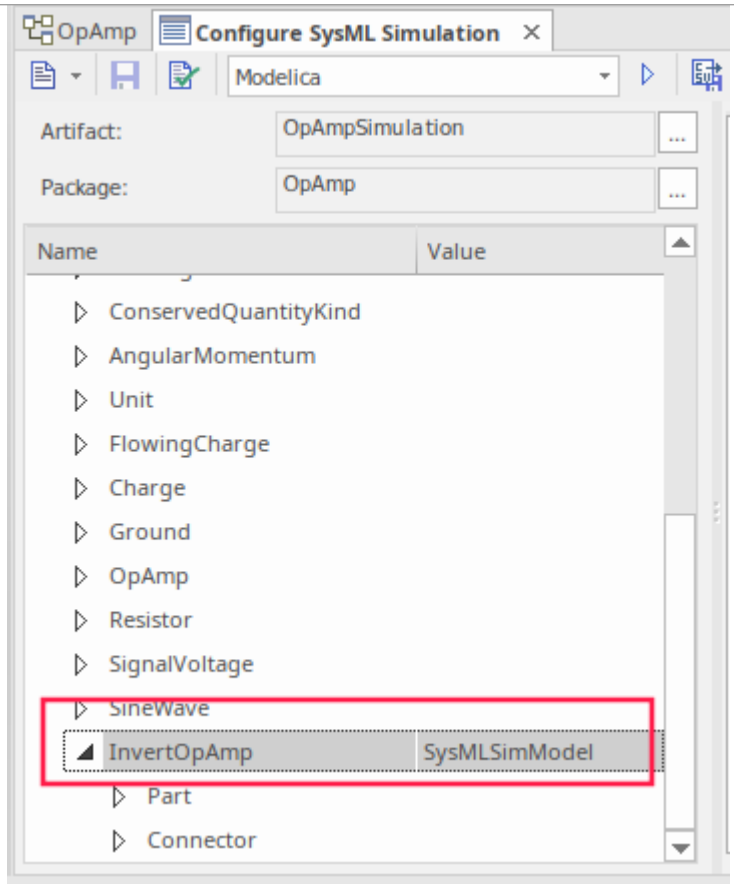


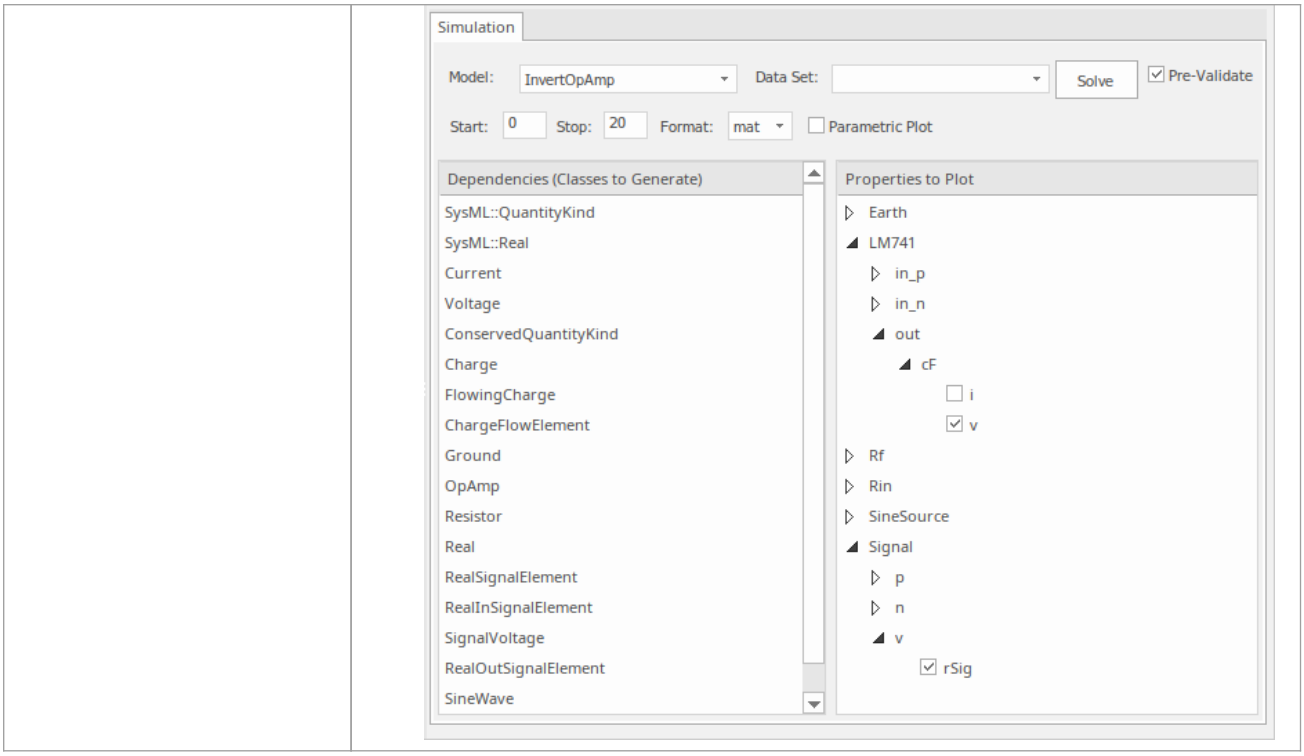
Configure Simulation Behavior

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

Step	Action
Create a SysMLSimConfiguration Artifact	• Open the Block Definition diagram • Click on the open space in the diagram • Press the Spacebar • From the 'Artifacts' sub-menu, select 'SysMLSim Configuration' This creates a new SysMLSim Configuration Artifact: 
Set the Package	• Double-click on the SysMLSim Configuration Artifact This opens the Configure SysML Configuration window

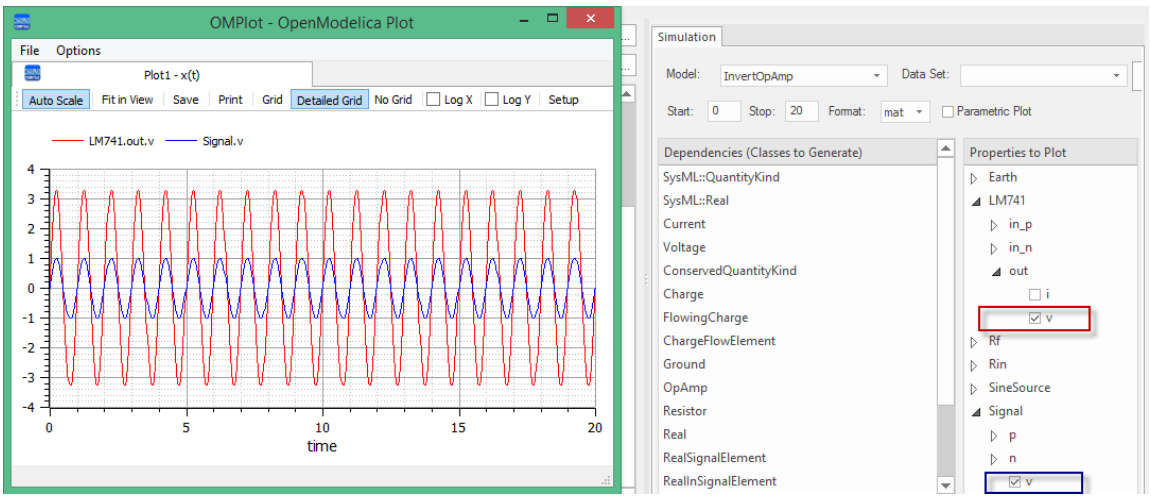
	• In the 'Package' field click on the [...] button and select the Package containing the SysML diagram: 
Set Modelica or Simulink	In the top drop-down you can select which simulation tool to use: • Modelica • Simulink For more details on these settings, see the *Configure SysML Simulation* Help topic.
Set the Block to Simulate	• In the left-hand list, under 'Name', find 'InvertOpAmp' • In the 'Value' column, click on the drop-down and select 'SysMLSimModel'

	
Select Properties to Plot	You can now select the Properties to be plotted: • In the right-hand pane, select the Ports to plot



Run Simulation

On the 'Simulation' tab, click on the Solve button. This example shows a plot generated in Modelica:



In the legend you can see LM741.out.v and 'Signal.v that are plotted, and match them with the two properties selected under *Properties to Plot*.

View the Model in Modelica or Simulink

To view the generated model in the external applications, Modelica or Simulink, see the *View the Model in Modelica or Simulink* Help topic, along with tips for debugging any issues in the generated code.

Digital Electronics Simulation Example

For this example, we walk through the creation of a SysPhS model for a simple digital electronic circuit, and then use a simulation to predict and chart the behavior of that circuit.

This example works with components that are not included with the SysPhS common components, so it runs through the process for creating Blocks to match the external components from scratch. For a webinar demonstrating this, see the link in *Learn More* at the end of the topic.

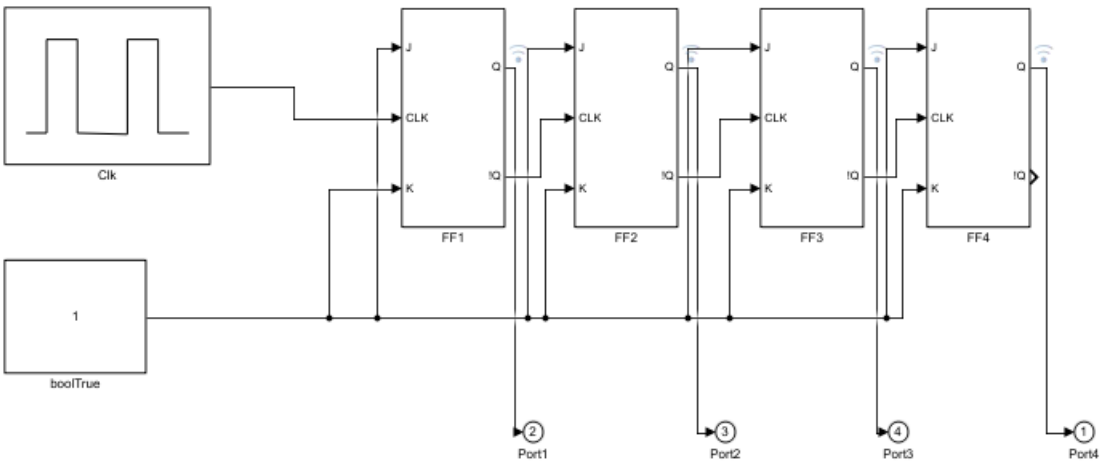
Prerequisites

Running this simulation requires either:

- OpenModelica or
- MATLAB's Simulink

Circuit Diagram - a Digital Frequency Divider

The digital electronic circuit we are going to model is shown in this figure, which uses standard electronic circuit notation.

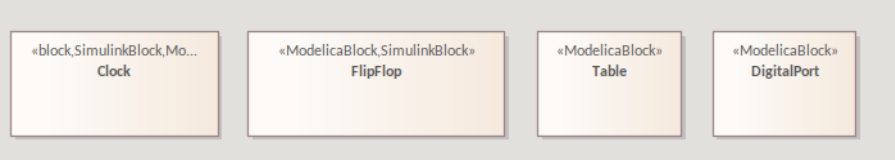


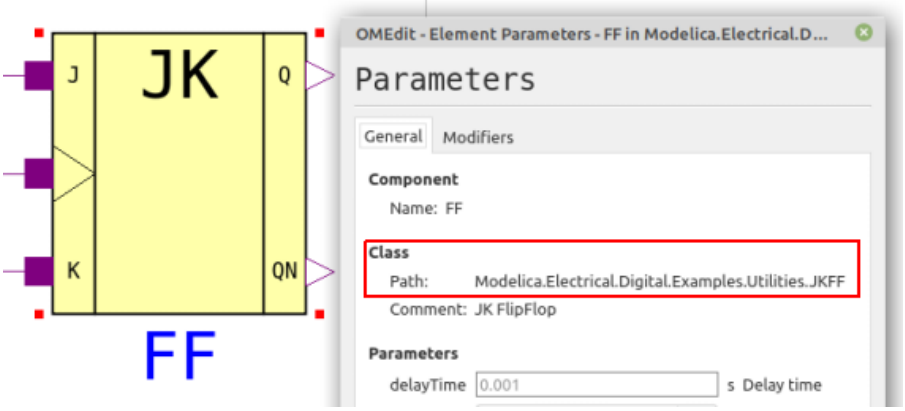
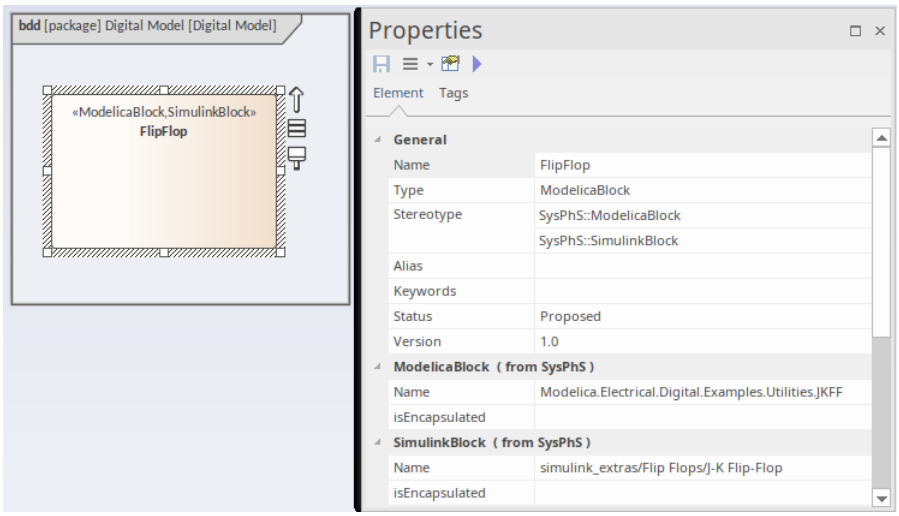
The circuit in this example includes a pulsed digital signal source, four Flip-Flops and a logical Boolean true state to form a simple frequency divider circuit.

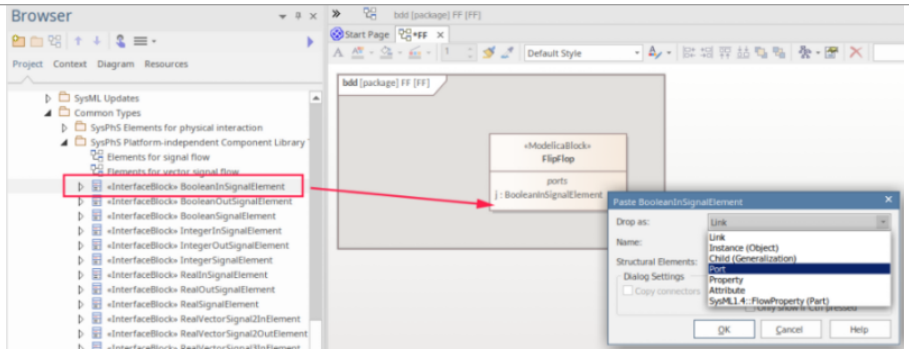
Create SysML Model

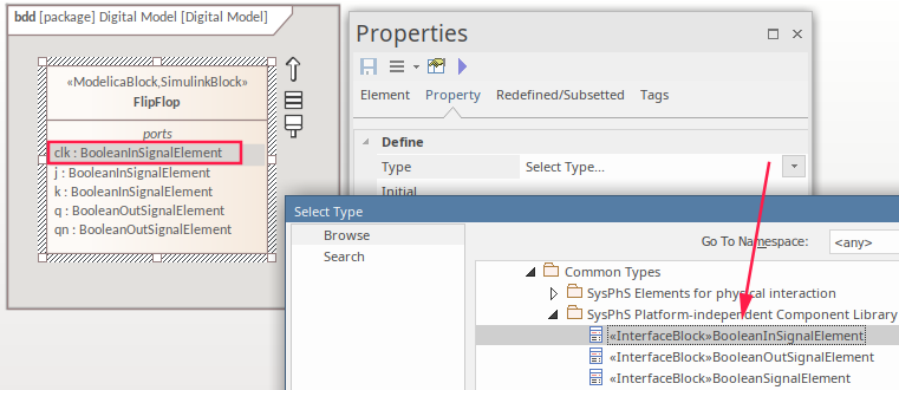
This table shows how we can build a complete SysML model to represent the circuit, starting at the lowest level types and building up the model one step at a time.

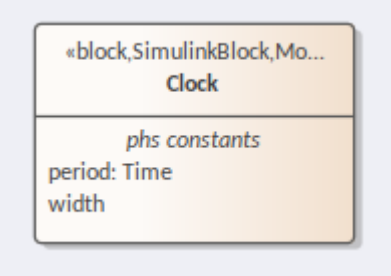
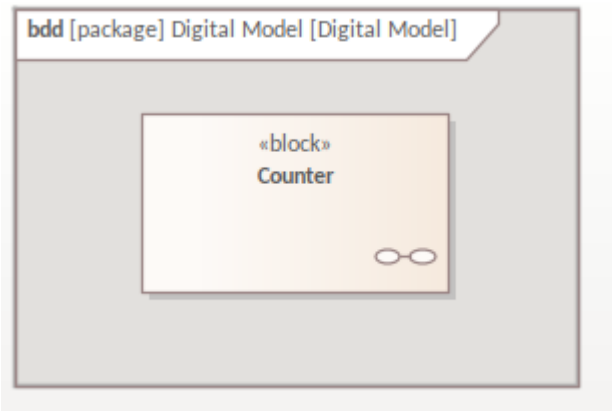
Component	Action
Blocks	In SysML, using SysPhS, the circuit and each of the component-types can be represented using a Block. Firstly create a Block Definition diagram (BDD) under a Package called 'Digital Model'. In the BDD, you will create a set of components for the circuit, as SysML Blocks. The circuit contains

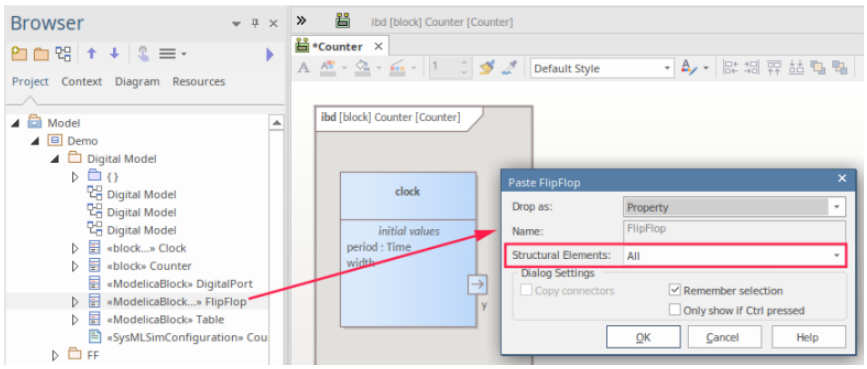
	representations of four Part-types - a pulsed digital signal source, a Flip-Flop, a boolean Port and a logical boolean true state. These Parts are of different types, with different behaviors. Create a SysPhS Block for each of the Part types. The Parts for the circuit Internal Block Definition (IBD), will be connected through Ports, which represent electrical pins. These must be defined in the BDD. This figure shows the BDD, with Blocks defining the types of component used.  Note that these Blocks are created from the SysPhS Toolbox using either Modelica Blocks or Simulink Blocks. You can type these Blocks to both tools. For more information see the *Setting Blocks as Both Modelica and Simulink* Help topic.
Setting Modelica and Simulink Path	In order to define a Block specific to Modelica or Simulink you need to access the Component's Path in the respective application, then set this in the Block's Properties.

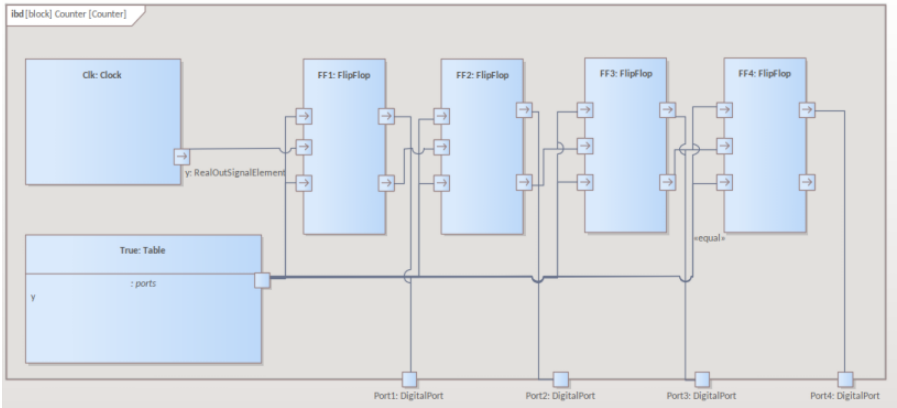
	For example we can find the Flip-Flop component in Modelica.  We then copy that to the Block's properties.  For more details see the *Creating Modelica Specific Blocks* and *Creating Simulink Specific Blocks* Help topics.
PhS Ports	For setting Ports on the Blocks (in this case the Flip-Flop Clock Port) you drag either a Modelica or a Simulink PhS Port onto the Block. This Port must then be typed as a BooleanInSignal.

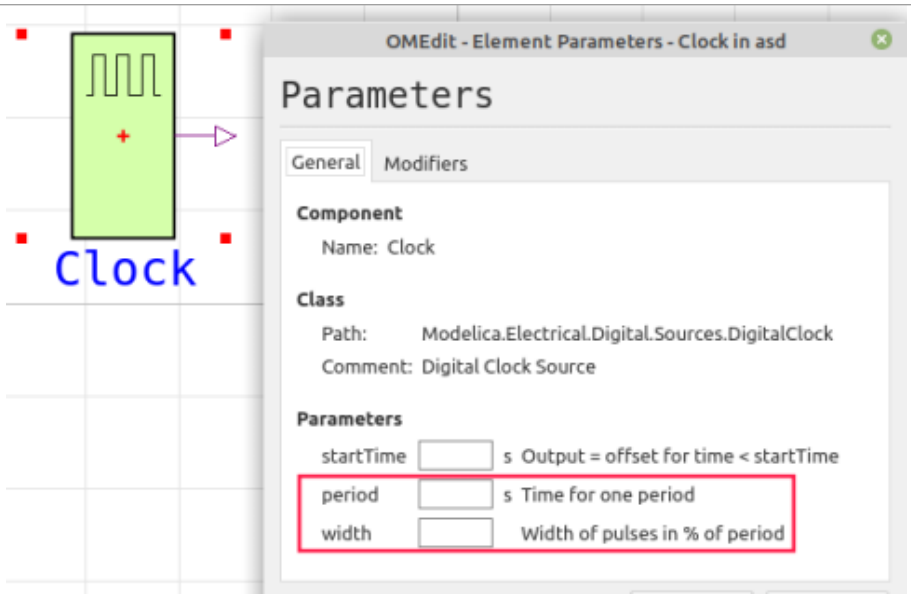
	 **Note:** • For Simulink the ordering on creating the Ports is critical - see *Simulink Port Ordering* in the *Creating Simulink and Simscape Specific Blocks* Help topic • These Ports can be set to both Modelica and Simulink by adding the stereotype for the other Port-type
Common Types	As a starter for all SysPhS models you must ensure that the SysPhS common types are loaded in the repository and referenced in the new model, using the Package Import connector. For more information see the *Referencing the SysPhS Simulation Libraries* Help Topic. The Value Types used for the Ports are pre-defined in the SysPhS Simulation Libraries. The two key types used are the BooleanInSignal and the BooleanOutSignal ValueTypes. This figure shows the Flip-Flop Block in the Block Definition diagram, with the

	Clock Port set to the BooleanInSignal ValueType and this being referenced in the Browser window. 
Phs Constants	The Clock and Boolean-true Blocks both have properties defined in their respective components in MATLAB and Modelica. Let's use the Clock as an example. For this component-type, in both Simulink and Modelica, we need to set a value for the period of each pulse and the width of each pulse. These Properties must be set and typed. The value of the Properties will be set in the IBD, the Parametric diagram or possibly in the simulation Datasets. To set the property defining the period: • Drag a PhsConstant from the SysPhS toolbox onto the Clock • Delete the element from the diagram to show this in its compartment • In the Properties window, Properties

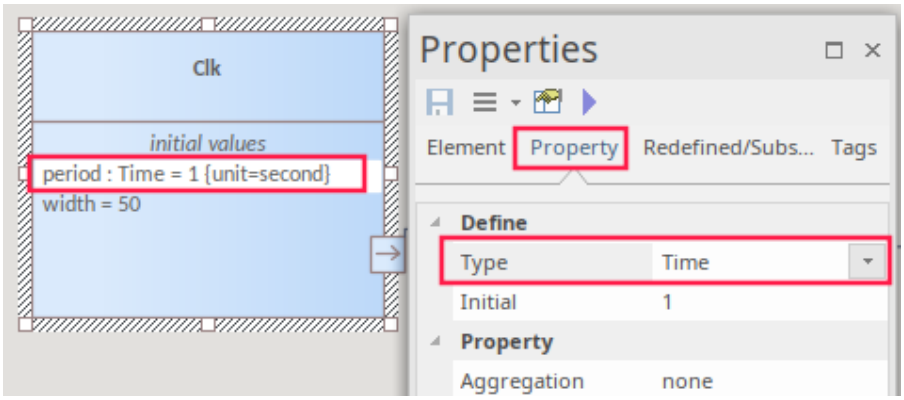
	tab, select the 'Type' field • Click on [...] and select 'Time' as the type  For setting the values in the individual component (Part), see the *Initial Values* row in this table.
Internal Structure - the Circuit	For the internal structure we create a Block with a child IBD diagram. • Create a Block for a Flip-Flop circuit • Under this Block create an Internal Block Definition (IBD) diagram using the context menu option *Create New Child Diagram | Internal Block Definition diagram*  • Double-click to open the IBD • Set the diagram to only show the Port's

	name, using: - F5 | Element | Element Appearance • Then unset these two options: - Show Stereotypes - Show Property Type This will then show only the Port and Type On the IBD you create Parts and connect them: • From the Browser window, drag the Blocks onto the IBD as Parts (Properties) • To view these Parts in compartments, delete them from the diagram Note: Set the Paste option for 'Structural Elements' to 'ALL'  • Add a Clock, four Flip-Flops and a table • Add four digital Ports to the boundary of the Counter IBD
Bindings	To model the wiring up of these

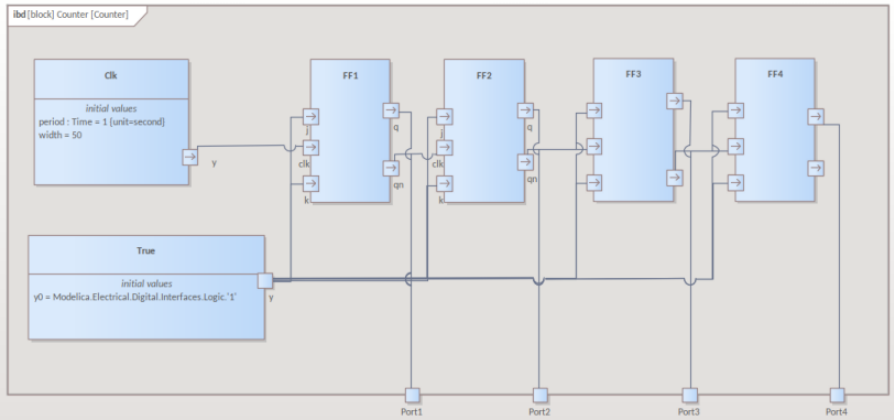
	components: • Create connections between the Ports with connectors of type 'Connector'  Notice that this follows the same structure as the original circuit diagram, but the symbols for each component have been replaced with properties typed by the Blocks we have defined.
Initial Values	The digital pulse source is a DigitalClock component in Modelica and Simulink. This requires two parameters - 'Period' and 'Width', as shown in the Modelica editor.



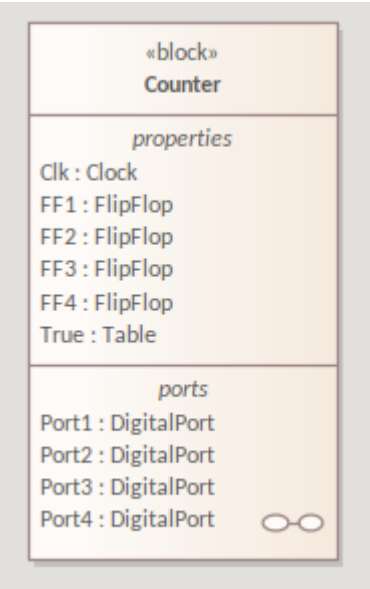
The values of these parameters must be set in the IBD Part 'Clk', in the Properties window 'Property' tab, 'Initial' field



The J & K Ports require a fixed logical 'True' state. This is defined using a Table set to 'true' using an initial value, as shown here for Modelica.



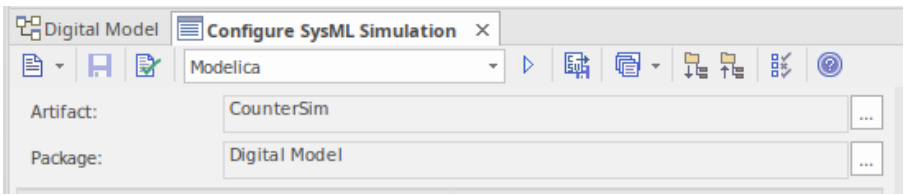
On returning to the BDD you should now have the Counter Block shown as:

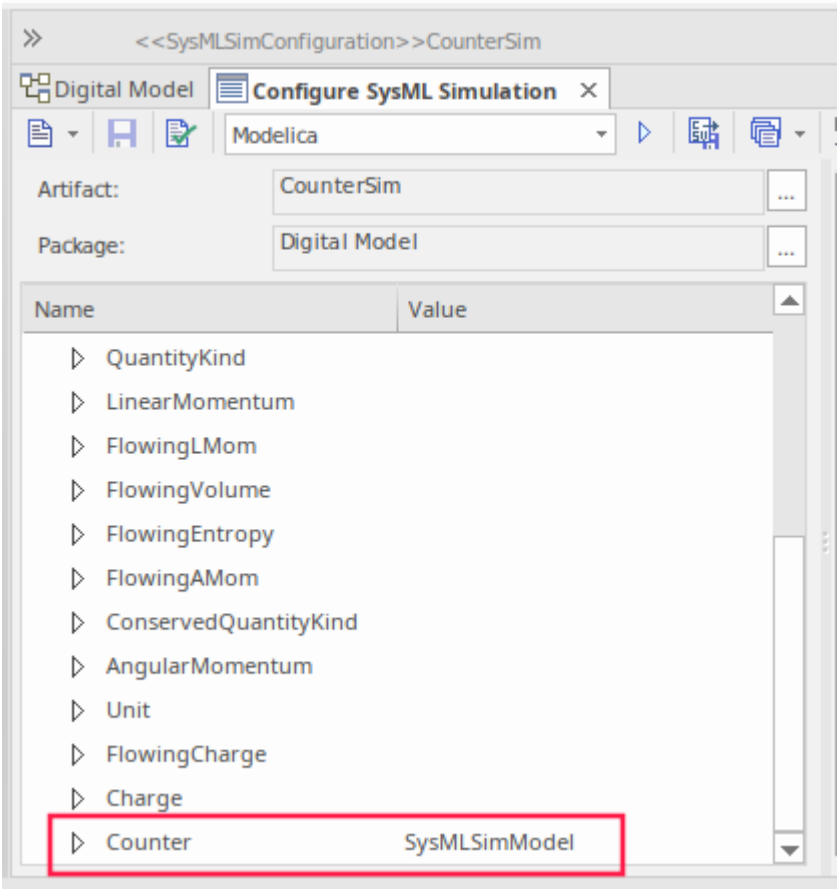


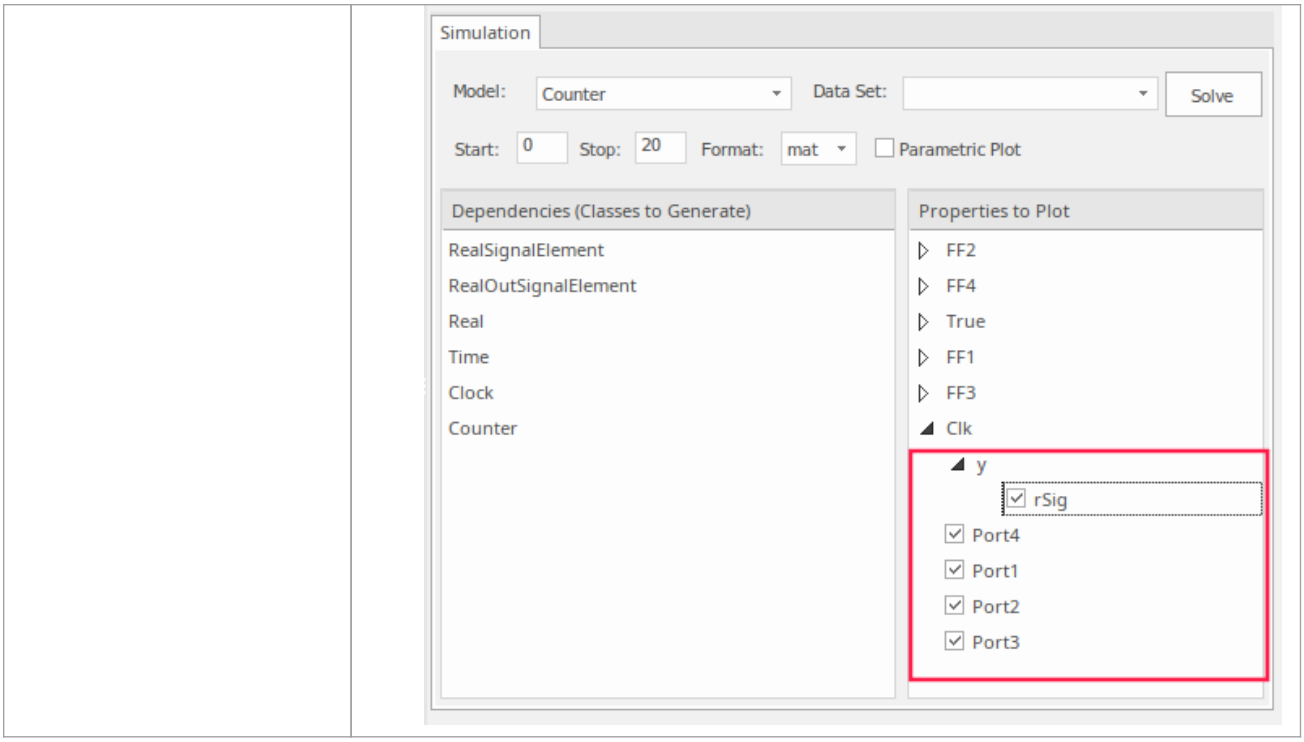
Configure Simulation Behavior

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

Step	Action
Create a SysMLSimC	• Open the Block Definition diagram • Click on the open space in the diagram

onfiguration Artifact	• Press the Spacebar • From the 'Artifacts' sub-menu, select 'SysMLSim Configuration' This creates a new SysMLSim Configuration Artifact
Set the Package	• Double-click on the SysMLSim Configuration Artifact This opens the Configure SysML Configuration window • In the 'Package' field, click on the [...] button and select the Package containing the SysML diagram 
Set Modelica or Simulink	In the top drop-down select which simulation tool to use: • Modelica • Simulink For more details on these settings, see the *Configure SysML Simulation* Help topic.
Set the Block to Simulate	• In the left-hand list, under 'Block', find 'InvertOpAmp' • In the 'Value' column, click on the drop-down and select

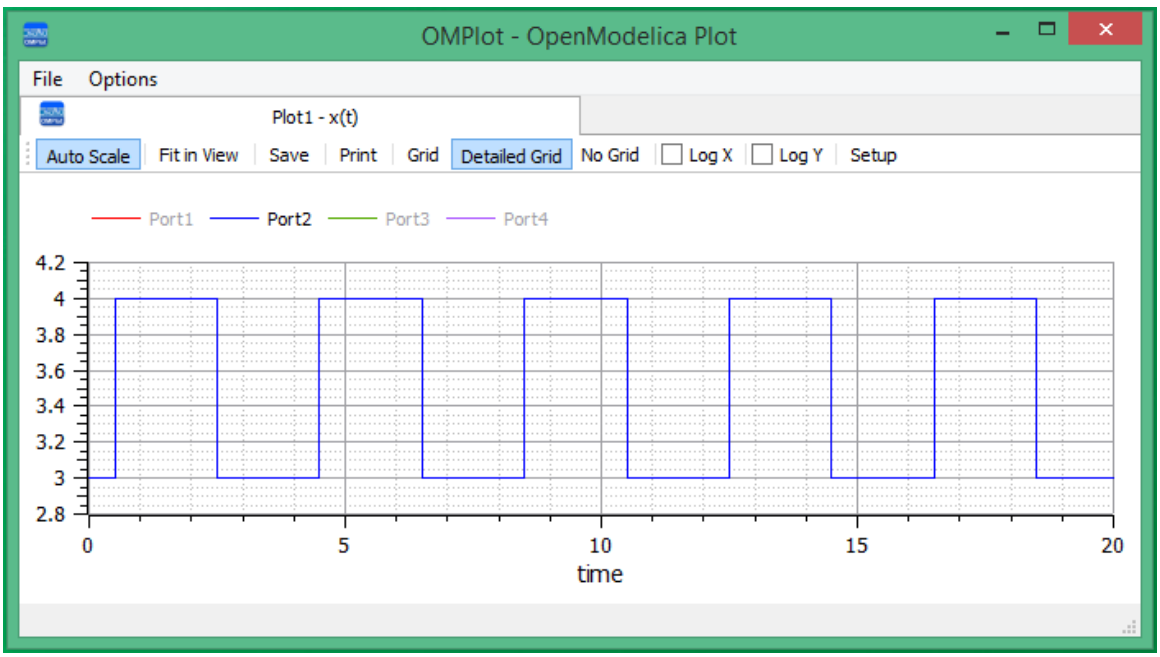
	'SysMLSimModel' 
Select Properties to Plot	You can now select the Properties to be plotted: • In the right-hand pane, select the Ports to plot



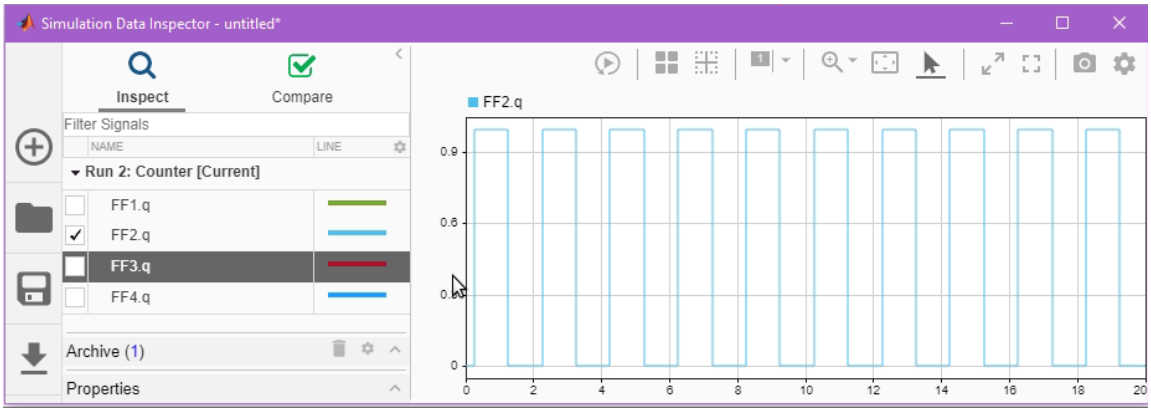
Run Simulation

On the 'Simulation' page, click on the Solve button. This shows an example of the plot generated in:

Modelica



Simulink



In the legend you can see Port 2 is selected, whereas the other Ports have been de-selected to show a simple *Plot*.

View the Model in Modelica or Simulink

To view the generated model in the external applications, Modelica or Simulink, see the *Viewing the Generated Model* Help topic. Also see the tips for debugging any issues in the generated code, in the *SysPhS Debugging Tips* Help topic.

Humidifier Example

For this example we walk through the creation of a SysPhS model of a room humidifier to illustrate the use of signal flows and StateMachines. The signals in this example reflect physical quantities, but differ from the physical interaction in the sense of physical substances with flow rates and potentials.

The complete humidifier example is a complex set of SysML models, but it contains a section that requires the use of MATLAB's Stateflow for simulation. We will use this section of the example to work through the use of the SysML StateMachine and view how this is configured and run in MATLAB using Stateflow, as well as in Modelica.

Prerequisites

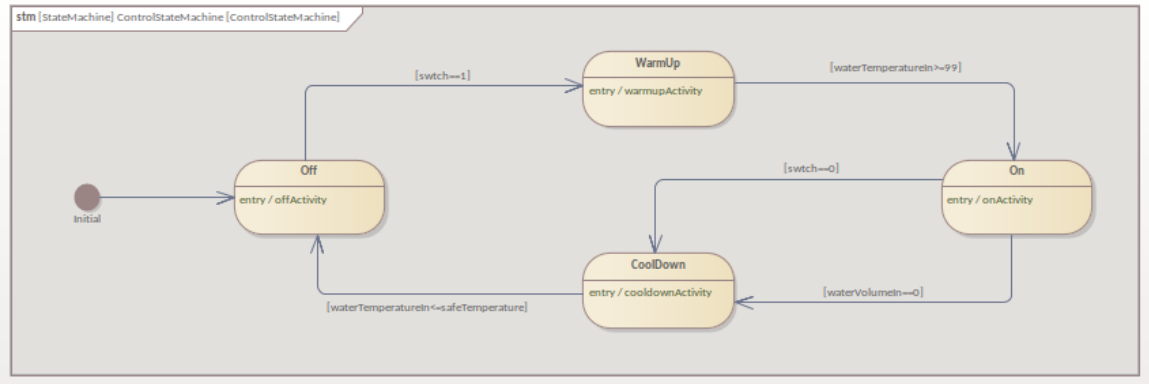
Running this simulation requires either:

- OpenModelica or
- MATLAB's Simulink and StateFlow

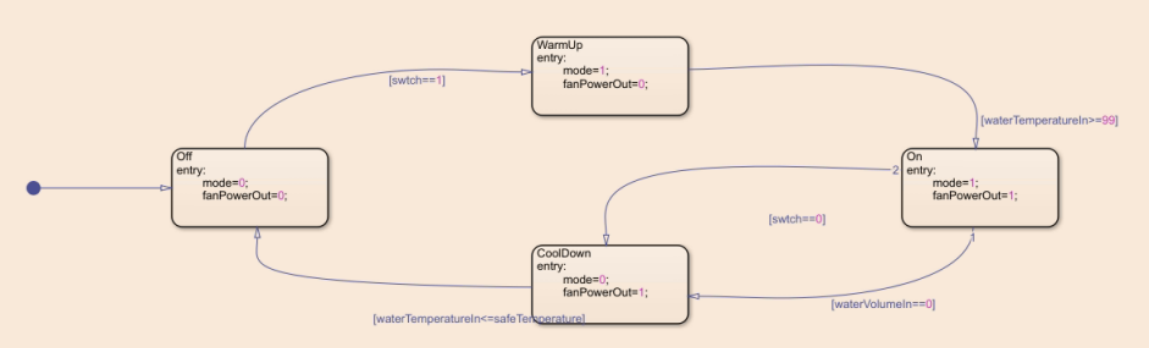
Overview

The complete simulation example is available in the WebEA model. See the links under *Learn More* at the end of this topic.

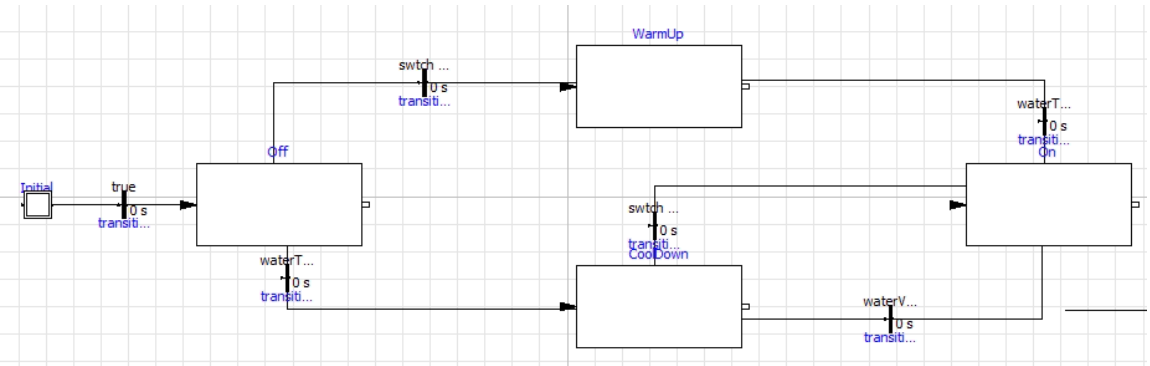
The SysML StateMachine diagram is:



For MATLAB the Stateflow diagram generated consists of:



For Modelica the generated diagram looks like this:

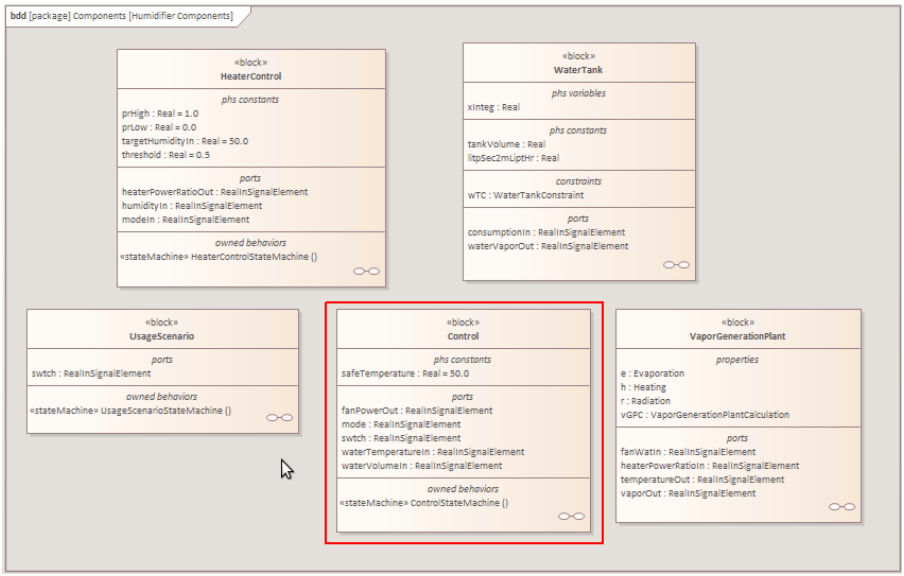


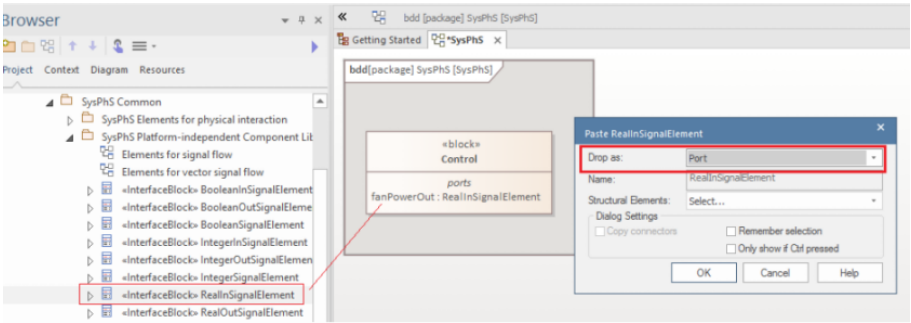
Create SysML StateMachine Model

This table shows how we can build the SysML StateMachine model to represent the switching states.

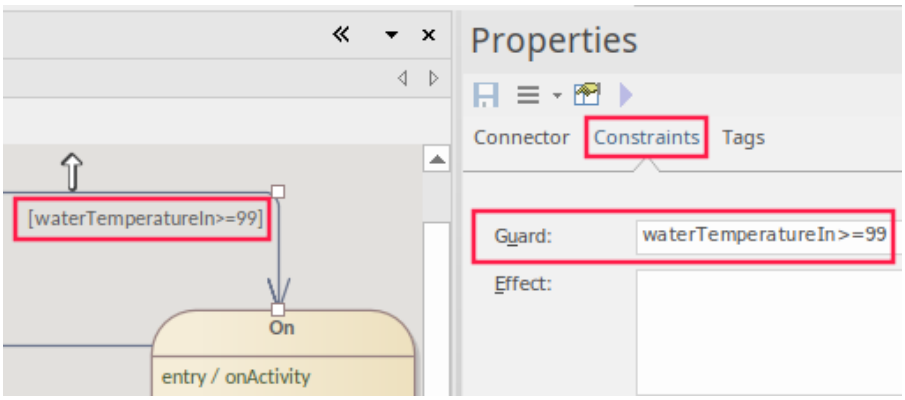
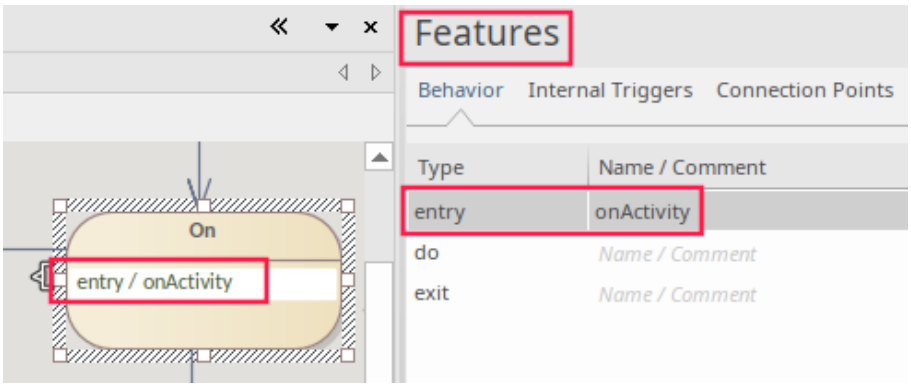
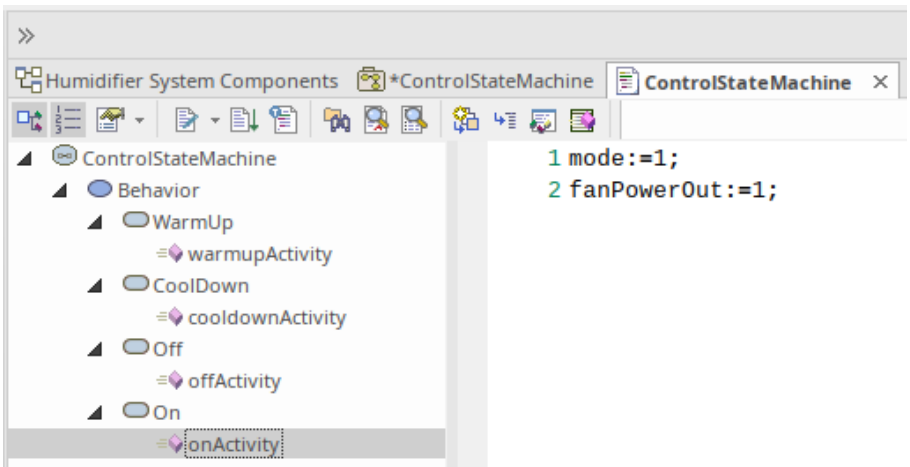
Note: The full SysML StateMachine features are only partially supported in MATLAB's Stateflow and Modelica. Therefore, in order to create State diagrams for simulation in

these products, it is necessary for the connector-types and object-types used in Enterprise Architect's StateMachine model to only contain the corresponding features supported in Stateflow and Modelica.

Step	Description
Blocks	In SysML, using SysPhS, the core humidifier components are represented by Blocks. For this example, where we are creating a StateMachine simulation, the focus is on the 'Control' Block. The Block Definition diagram (BDD) called 'Humidifier Components', which includes the 'Control' Block, is only one in a list of BDDs defining the humidifier.  The 'Control' Block has the StateMachine 'ControlStateMachine' as a Composite diagram under it; this is illustrated in the first diagram in the *Overview* section at the start of this topic.

	The 'Control' Block is where Ports and a PhS constant for the humidifier control are defined. The Ports are of type RealInSignal, which are signal flows.
Common Types	As a starter for all SysPhS models, you need to ensure that the SysPhS common types are loaded in the repository, and referenced in the new model using the Package 'Import' connector. For more information see the *Referencing SysPhS Libraries* Help topic.
Phs Ports	The Value Types used for the Ports are pre-defined in the SysPhS Simulation Libraries. The key type used is the RealInSignal ValueType (RealInSignalElement). This can be dragged onto the Block as a Port and then renamed. As an example of setting a Port on the Block, in this case the *fanPowerOut* Port, you drag a RealInSignalElement onto the Block and define it as a Port. 

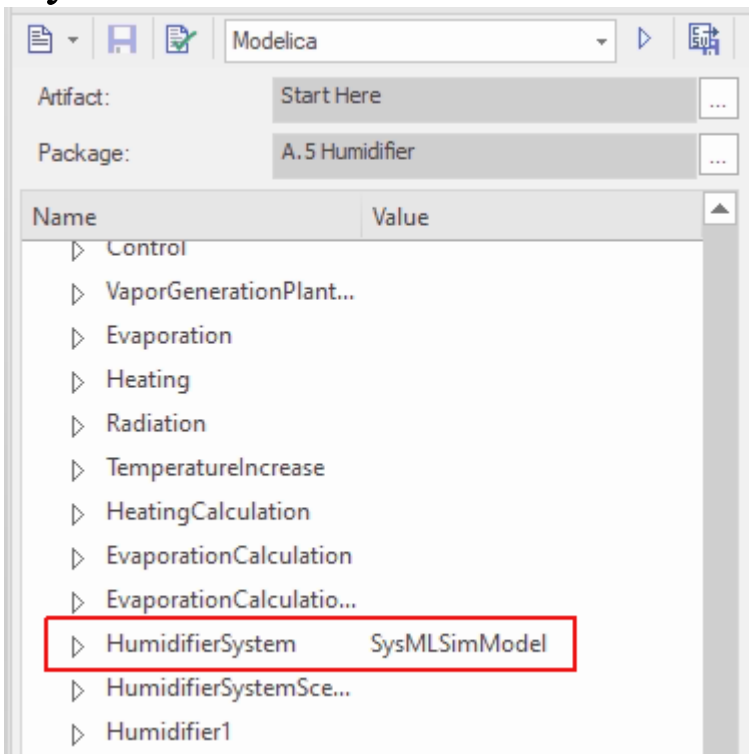
	Note: • These Ports can be set to both Modelica and Simulink by adding the stereotype for the other Port-type
Creating the StateMachine	To define a StateMachine as a Composite child-diagram to the Block: • Select 'New Child Diagram > StateMachine' from the context menu on the Block, then • Select 'New Child Diagram > Select Composite' from the context menu on the Block, and then select the diagram just created
Creating the States and Transitions	In the child StateMachine diagram, you add the Initial element, four States and a Final element. Then name these appropriately. Note: By pressing the Spacebar when the diagram is selected, you can select an Initial object from the context menu, and then use the Quick Linker to add the remaining objects.
Transition Guards	The Transitions between the States contain conditions in their Guards. These conditions are defined in the Properties window 'Constraints' tab.

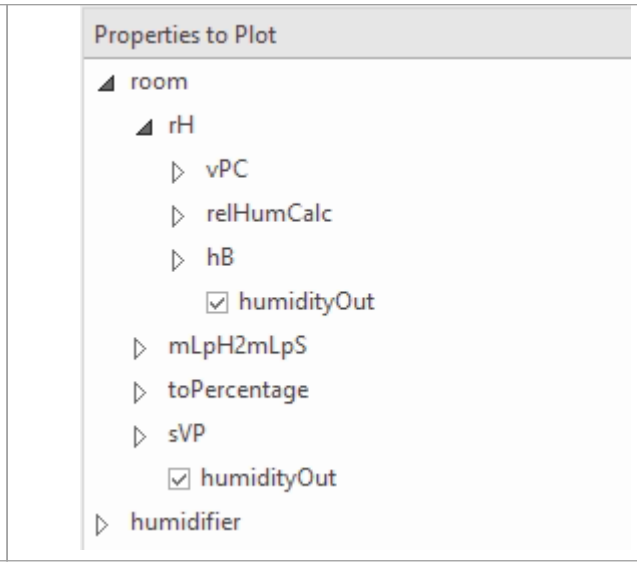
	The variables used are the 'Control' Block's Phs constant and Ports. See the image in the earlier *Blocks* row. 
Behaviors	Each State contains a script defining the mode and status of the fan. These scripts are set in the 'Entry' operation.  To edit the script for this, select the root StateMachine and pressing Alt+7. 

Configure Simulation Behavior

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

Step	Action
Create a SysMLSimConfiguration Artifact	• Open the Block Definition diagram • Click on the open space in the diagram • Press the Spacebar • From the 'Artifacts' sub-menu, select 'SysMLSim Configuration' This creates a new SysMLSim Configuration Artifact
Set the Package	• Double-click on the SysMLSim Configuration Artifact This opens the Configure SysML Simulation window • In the 'Package' field, click on the [...] button and select the Package containing the SysML diagram
Set Modelica or Simulink	In the top drop-down, select which simulation tool to use: • Modelica • Simulink

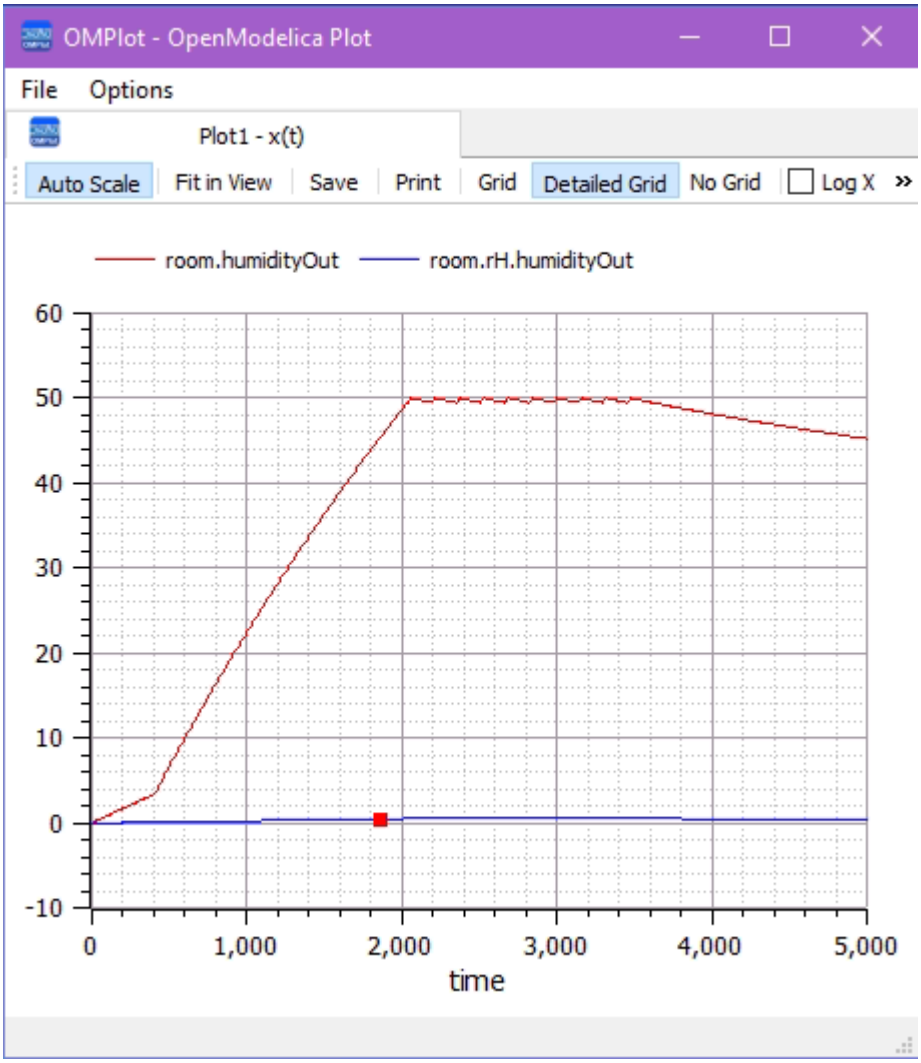
	For more details on these settings, see the *Configure SysML Simulation* Help topic.
Set the Block to Simulate	• In the left-hand list, under 'Block' in the 'Name' column, find 'HumidifierSystem' • In the 'Value' column, click on the drop-down arrow and select 'SysMLSimModel' 
Select Properties to Plot	You can now select the properties to be plotted: • In the right-hand pane, select the Ports to plot



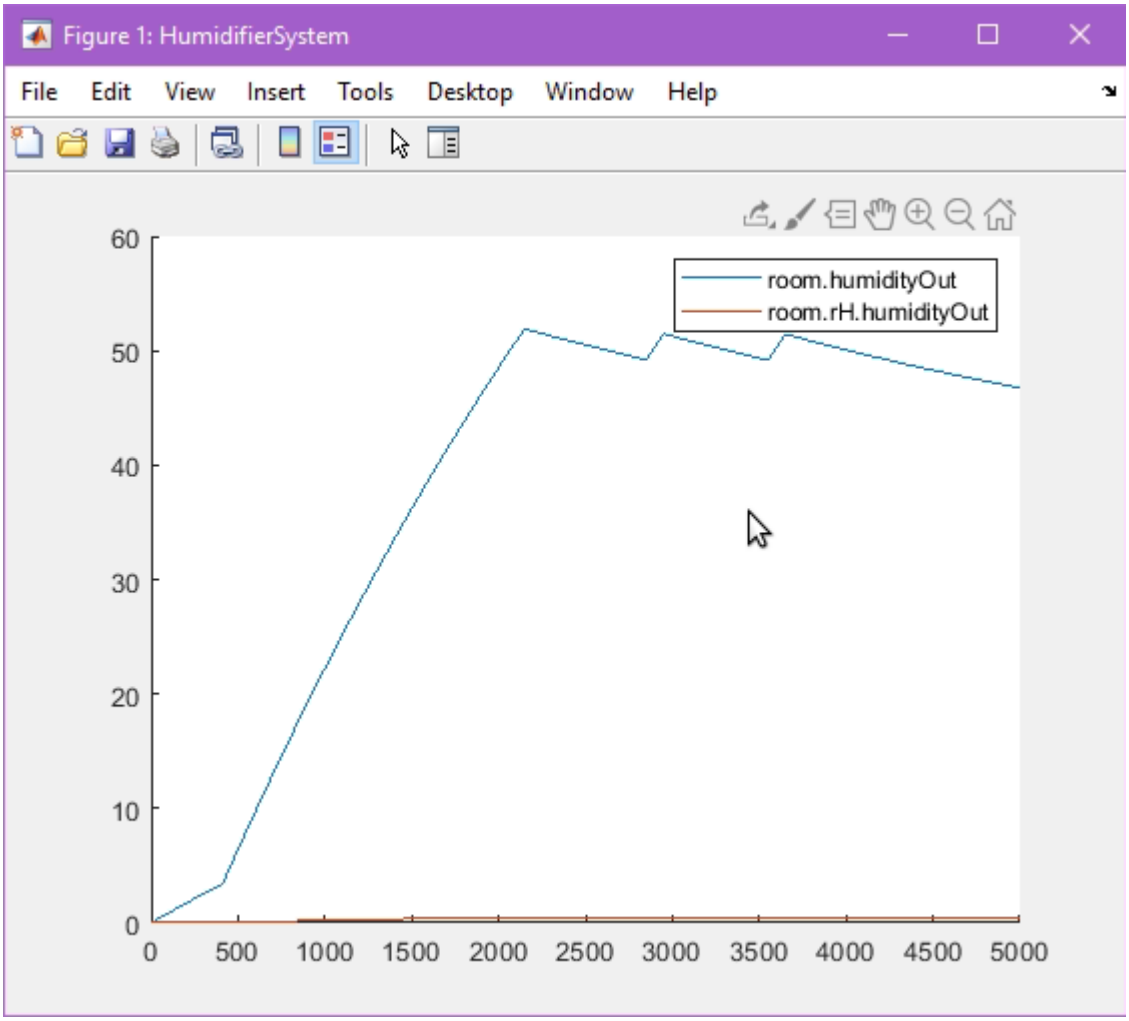
Run Simulation

In the 'Simulation' page, click on the Solve button. This shows an example of the plot generated in:

Modelica


OMPlot - OpenModelica Plot
File
Options
Plot1 - x(t)
Auto Scale
Fit in View
Save
Print
Grid
Detailed Grid
No Grid
Log X
room.humidityOut
room.rH.humidityOut
60
50
40
30
20
10
0
-10
0
1,000
2,000
3,000
4,000
5,000
time

Simulink



Note: The output in Simulink looks different due to the default precision for Simulink models not being sufficient for this specific example. Simulink settings can be modified by opening the generated file directly in Simulink.

View the Model in Modelica or Simulink

To view the generated model in the external applications, Modelica or Simulink, see the *View the Model in Modelica or Simulink* Help topic, which includes tips for debugging any issues in the generated code.

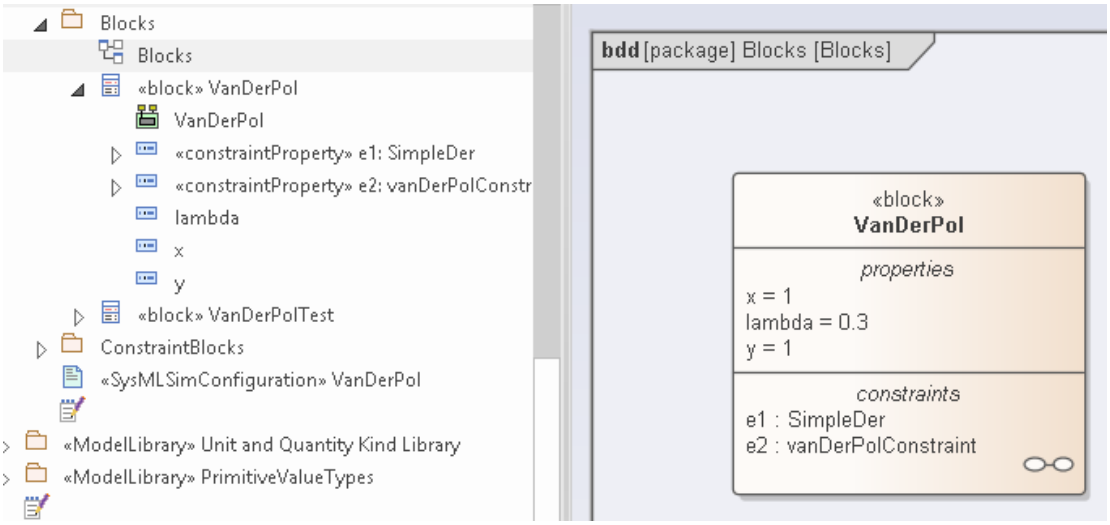
Updating the SysMLSim Configuration

In releases of Enterprise Architect prior to Release 15.2, for each Part/Property the type and options were defined within the Configure SysML Simulation window, opened from the SysMLSim Configuration Artifact. This approach is still valid and available; however, using the latest features of the SysPhS Standard you can now define simulation parameters in the model itself, meaning you can generate to different simulation tools (Simulink and Modelica), from the same model and make adjustments at an elemental level.

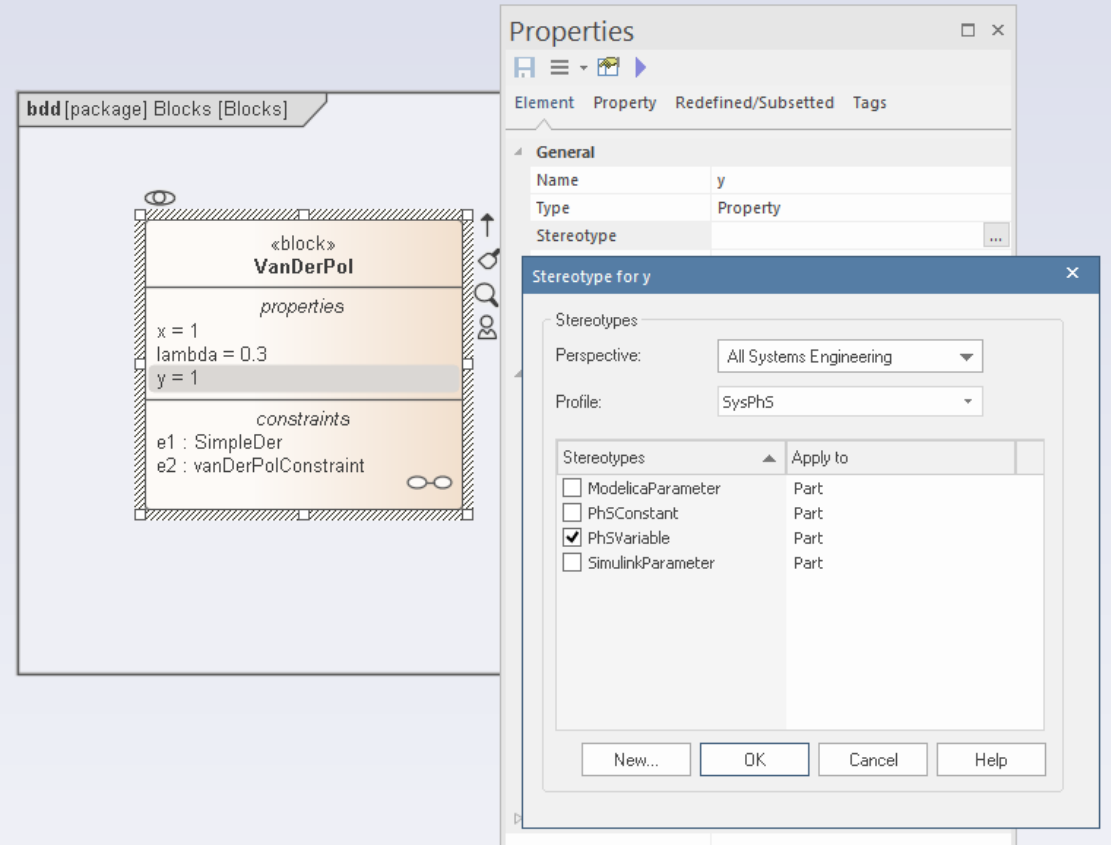
If you prefer to use the SysPhS option, you can update recent simulation configurations to reflect the use of the SysPhS standard, and support simulation through different tools.

To update the existing configurations:

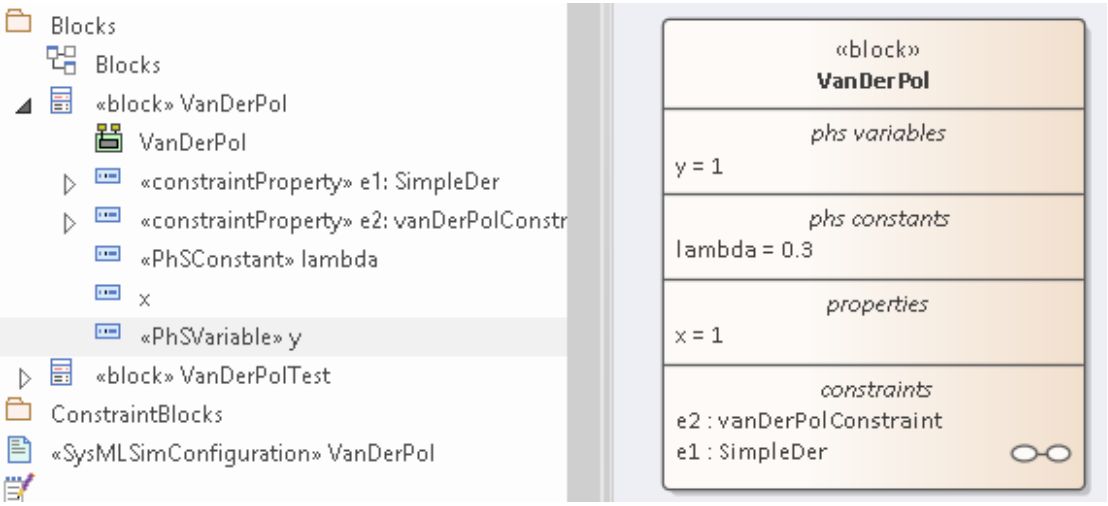
1. Ensure that you have a Package *Import* reference to the SysPhS Simulation Libraries. For more details see the Help topic *Referencing the SysPhS Simulation Libraries*.
2. Open the diagram containing the Block element of the simulation. For example:



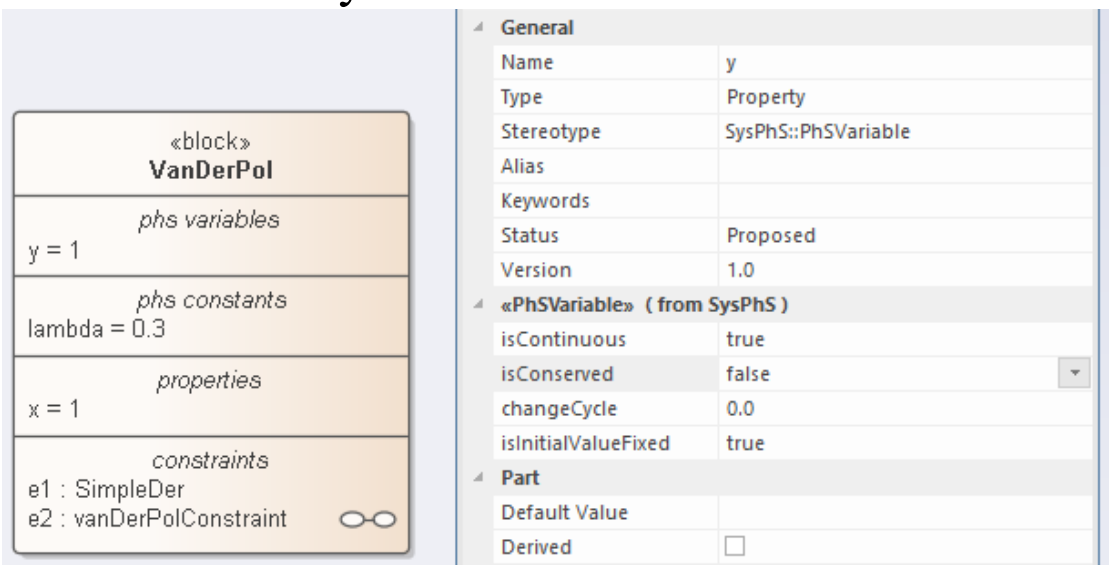
3. Notice the Property elements (in our illustration, *x*, *y* and *lambda*) in the Browser window; neither there, nor in the diagram, nor in the Properties window is it possible to see what type of property each one is.
4. In the element in the diagram, click on a Property and, in the 'Stereotype' field in the Properties window, click on the ... icon to display the 'Stereotype for <propertyname>' dialog. Ensure that the 'Perspective' field is set to 'All Systems Engineering' and the Profile field is set to 'SysPhS'.



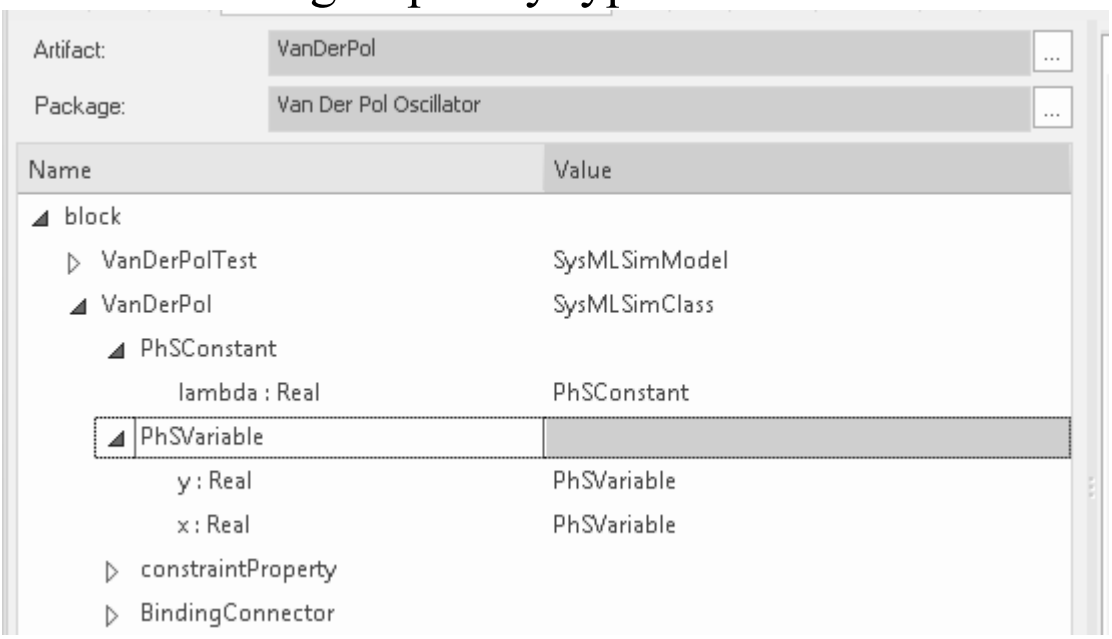
5. Click on the checkbox for the appropriate property type - PhSVariable or PhSConstant - and click on the OK button.
6. Note how the constants and variables are assigned to their own element compartments on the diagram, and how the Property types (as stereotypes) can also be seen in the Browser window.



7. Notice also that, in the Properties window, you can see the Stereotype defining the type of the property, and the options for that property type (as Tagged Values) as defined in the SysPhS standard.



8. Finally, if you open the Configure SysML Simulation window for the SysMLSim Configuration Artifact for this simulation, you will see again that the Properties are identified and grouped by type.



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,

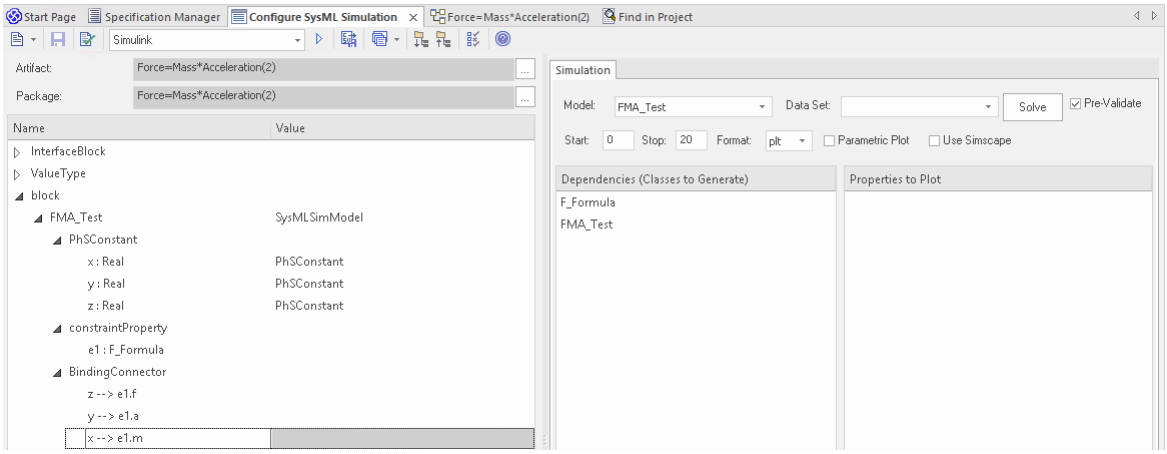
Models	reliability and other physical characteristics. These models combine Requirements models with System Design models, by capturing executable constraints based on complex mathematical relationships. Parametric diagrams are specialized Internal Block diagrams that help you, the modeler, to combine 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.

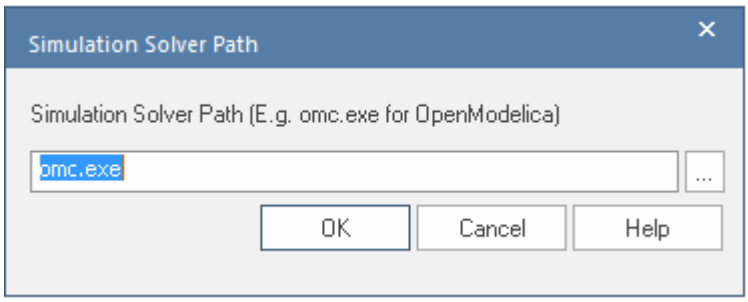


Access

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. For MATLAB/Simulink, the path will be automatically detected so this only needs to be changed if there are issues with the automatic detection, or there are multiple versions of MATLAB installed.

	
	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 arrow 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* Help topic.

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 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 stereotype has 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 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	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

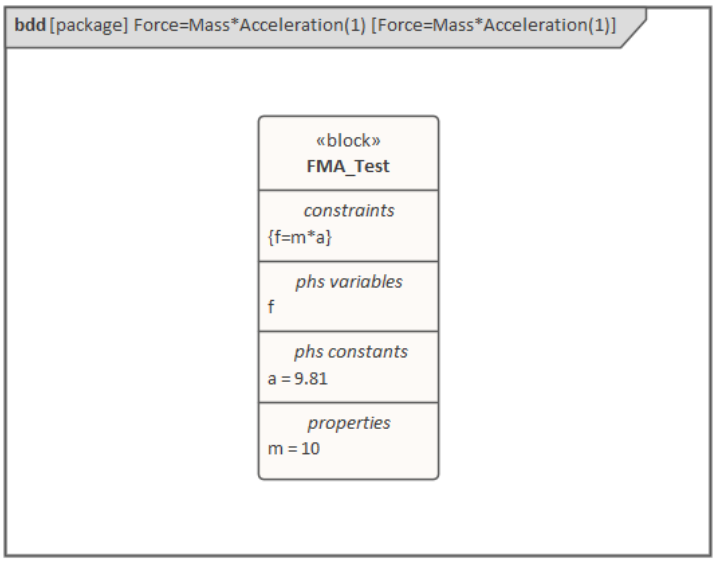
Ribbon	Simulate > System Behavior > Modelica/Simulink > SysMLSim

	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.

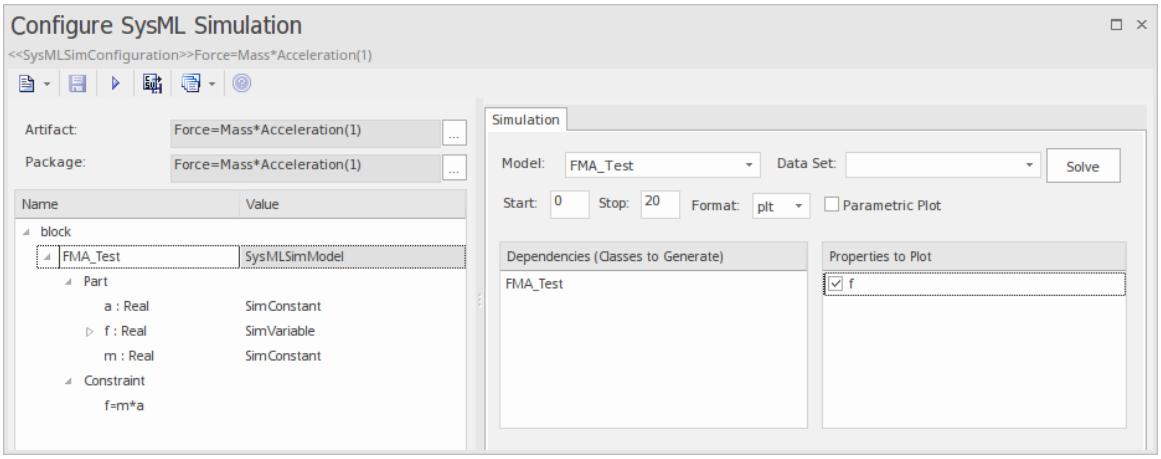


Tip: You can define multiple constraints in one Block.

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.

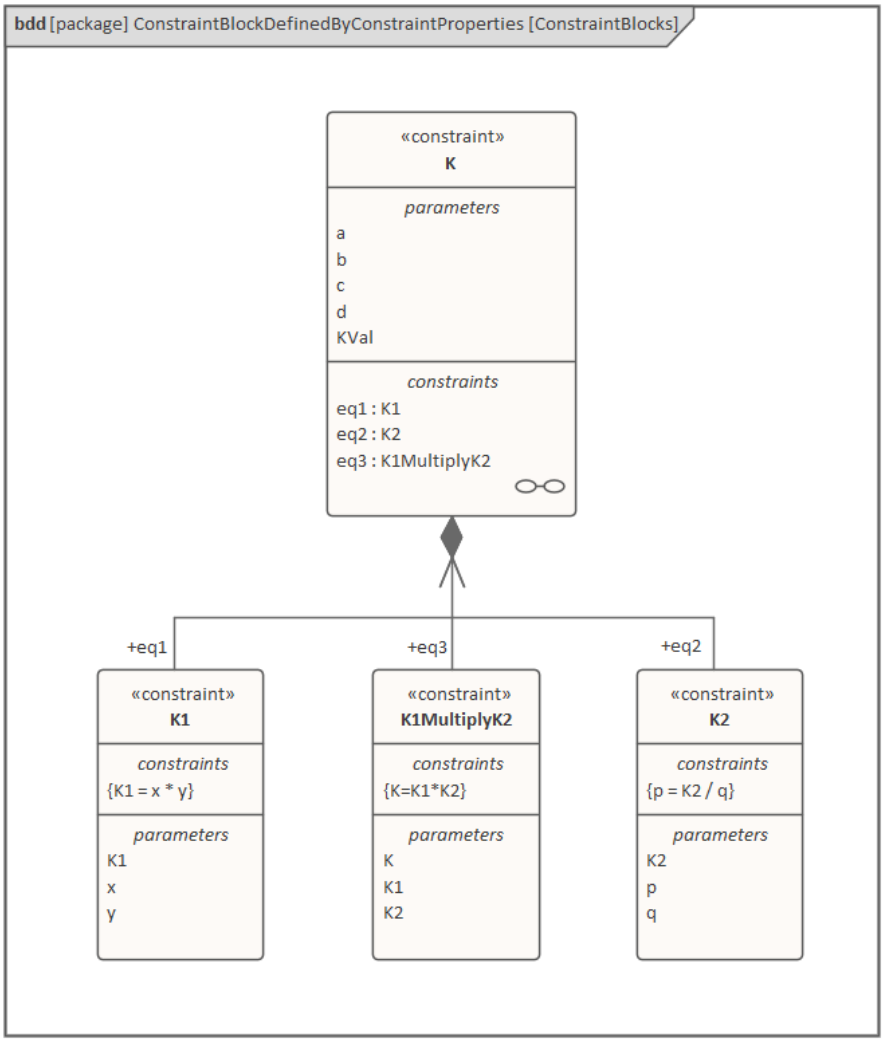


A chart should be plotted with f = 98.1 (which comes from 10 * 9.81).

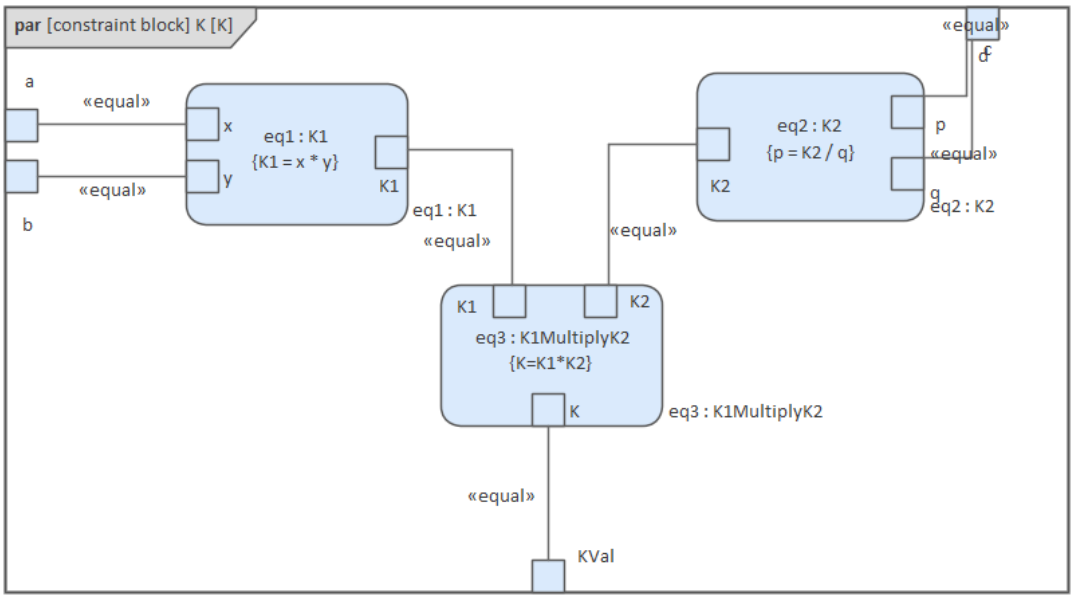
Connected Constraint Properties

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

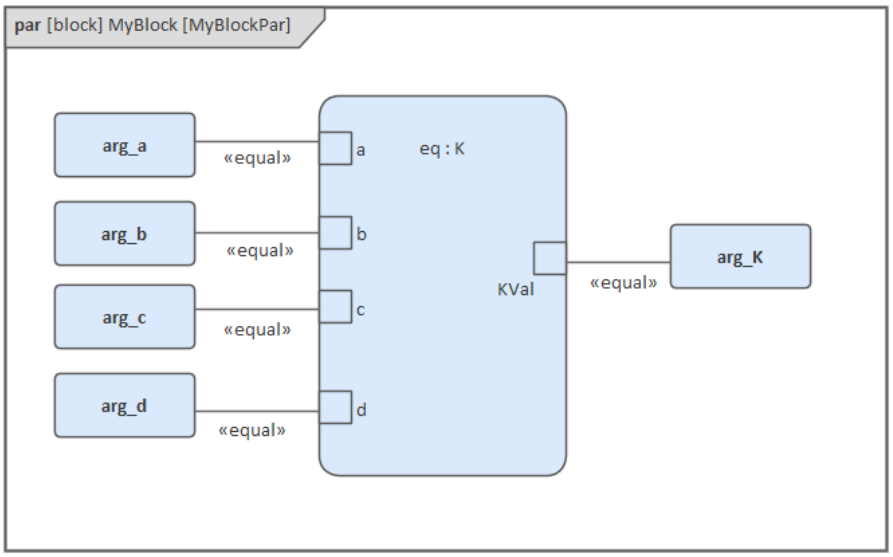
In this figure, ConstraintBlock 'K' defines parameters 'a', 'b', 'c', 'd' and 'KVal', and three constraint properties 'eq1', 'eq2' and 'eq3', typed to 'K1', 'K2' and 'K1MultiplyK2' respectively.



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

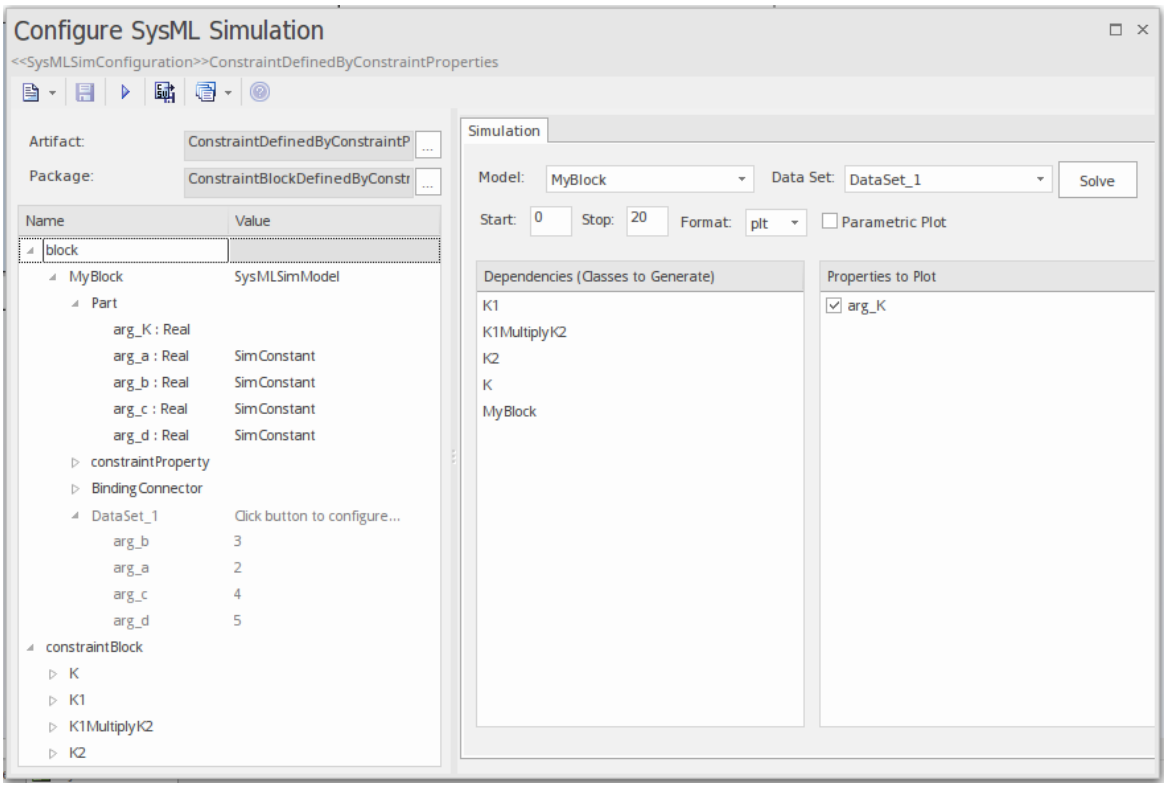


- Create a model MyBlock with five Properties (Parts)
- Create a constraint property 'eq' for MyBlock and show the parameters
- Bind the properties to the parameters



- 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.

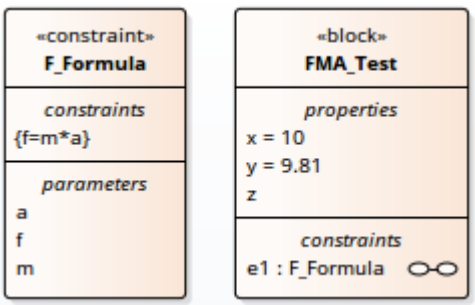
We can easily solve K2 to be p * q in this example, but in some complex examples it is extremely hard to solve a variable from an equation; however, the Enterprise Architect SysMLSim can still get it right.

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:

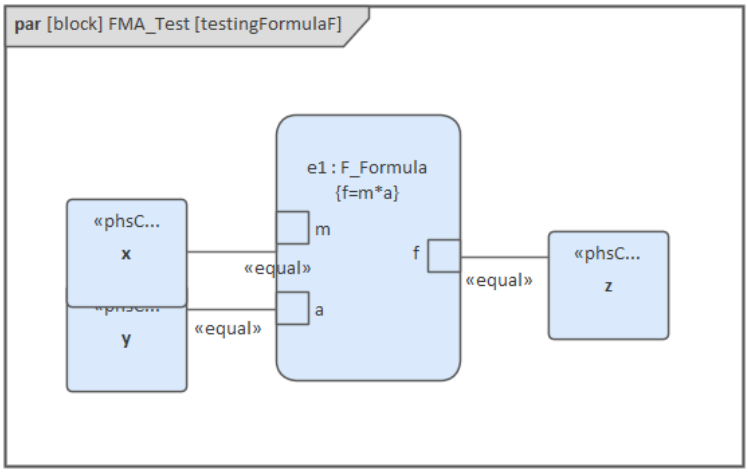
- Create a ConstraintBlock element 'F_Formula' with three parameters 'a', 'm' and 'f', and a constraint 'f = m * a'



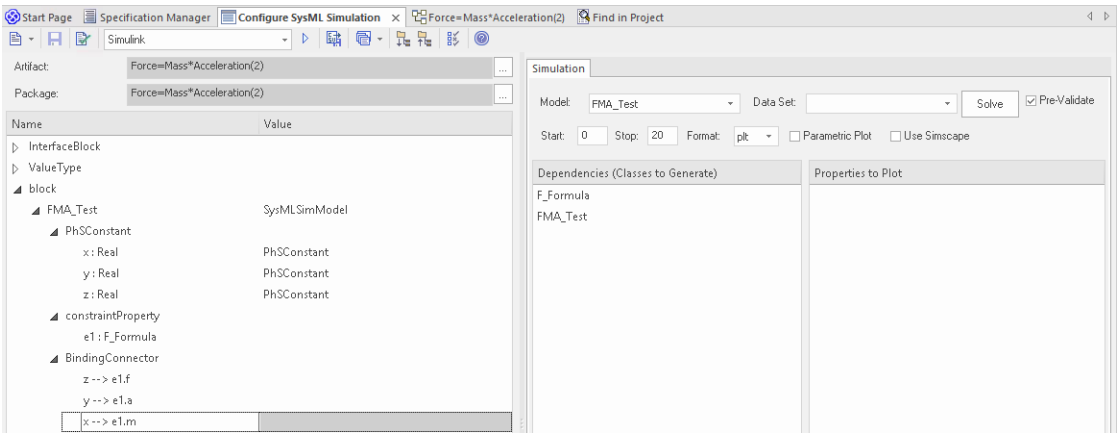
Tip: Primitive type 'Real' will be applied if property types are empty

- 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:

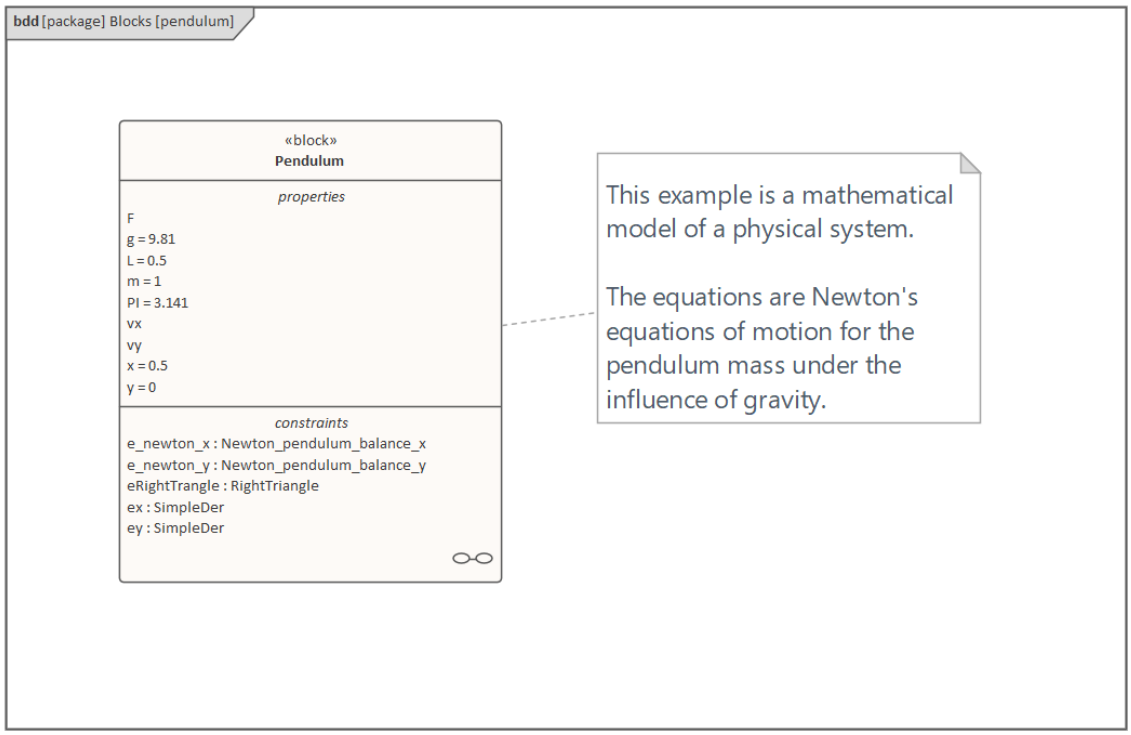
```
source.p.v = resistor.n.v;
source.p.i + resistor.n.i = 0;
```

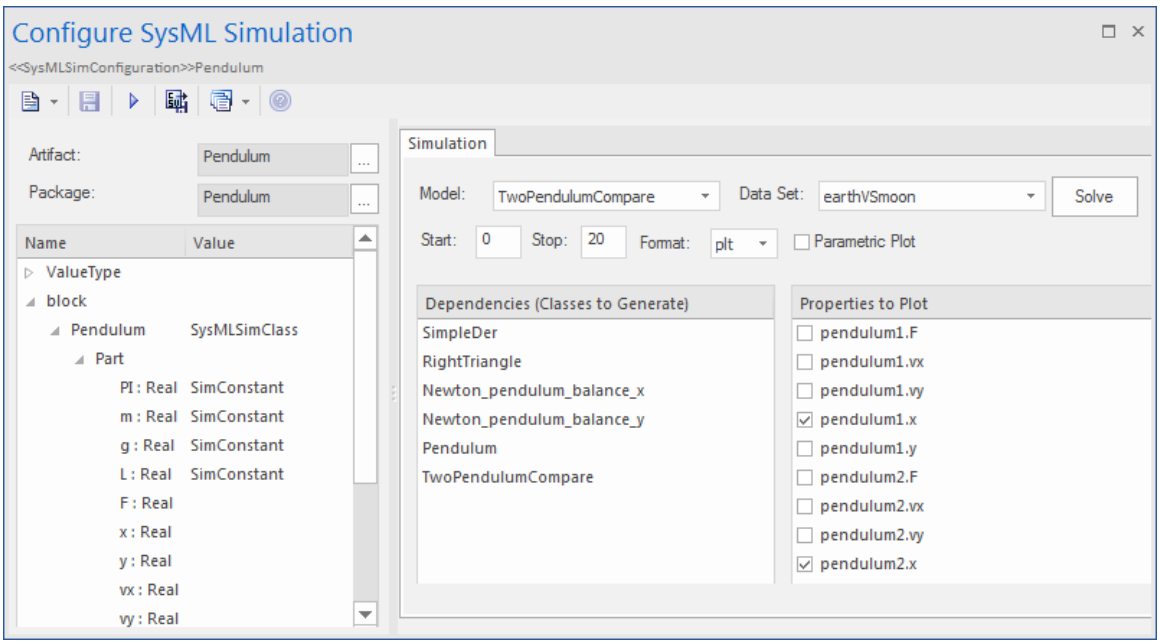
Default Value and Initial Values

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;
```

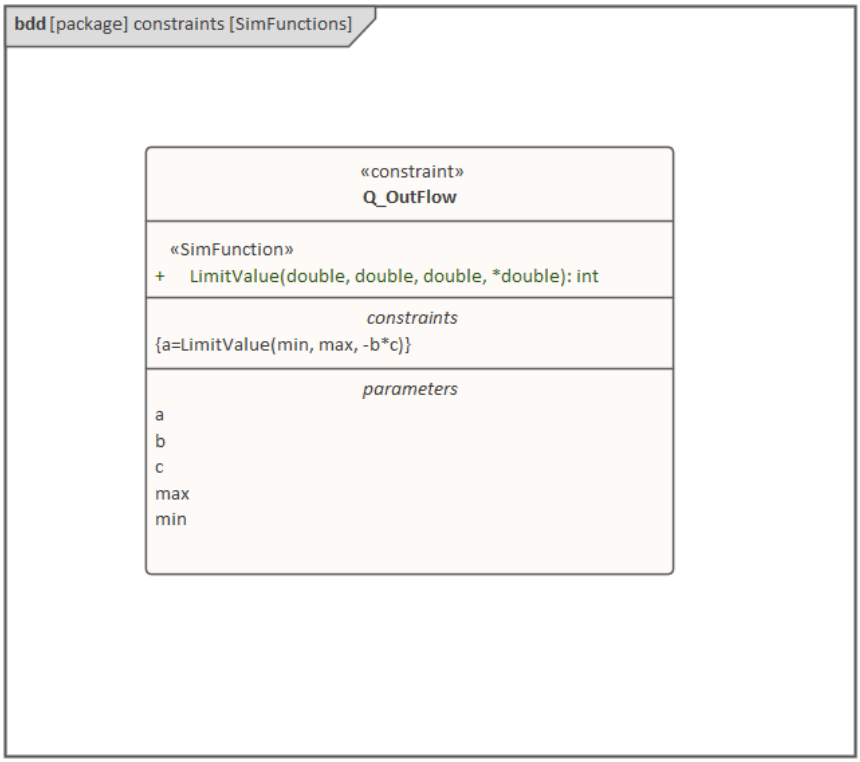
- Properties 'PI', 'm', 'g' and 'L' are constant, and are generated as a declaration equation
- Properties 'x' and 'y' are variable; their starting values are 0.5 and 0 respectively, and the initial values are generated as modifications

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;
```

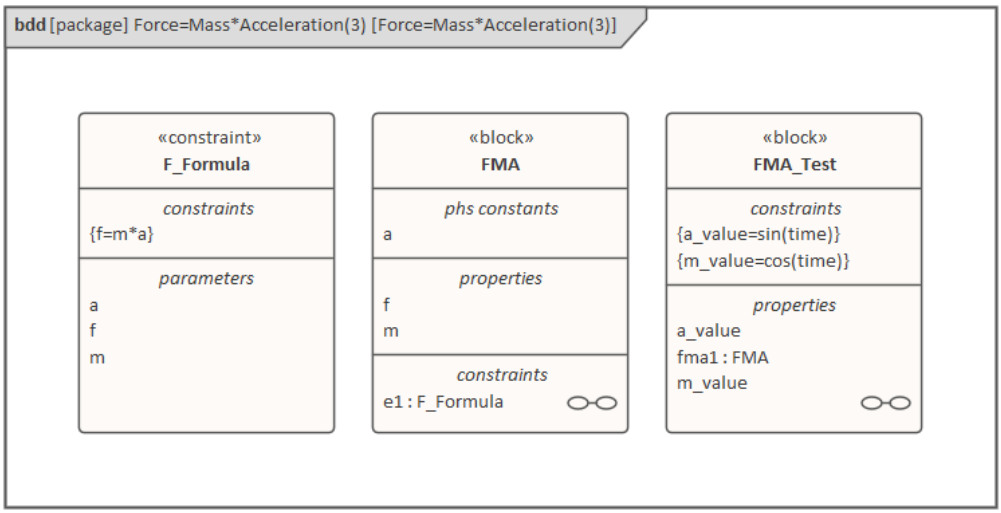
When generating code, Enterprise Architect will collect all the operations stereotyped as 'SimFunction' defined in ConstraintBlocks and Blocks, then generate code resembling this:

```
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;
```

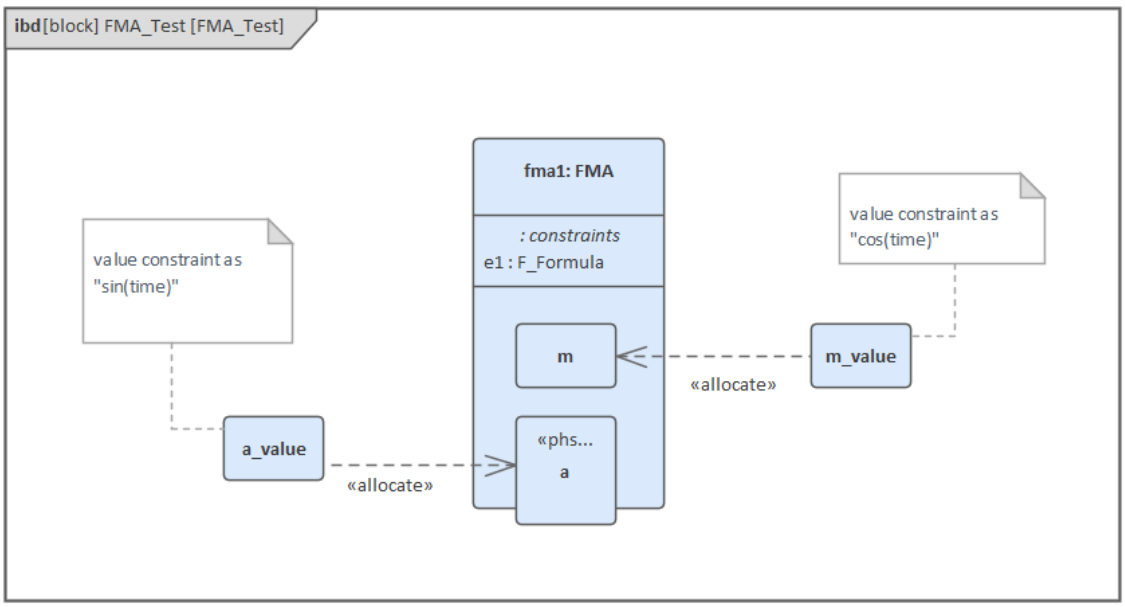
Value Allocation

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

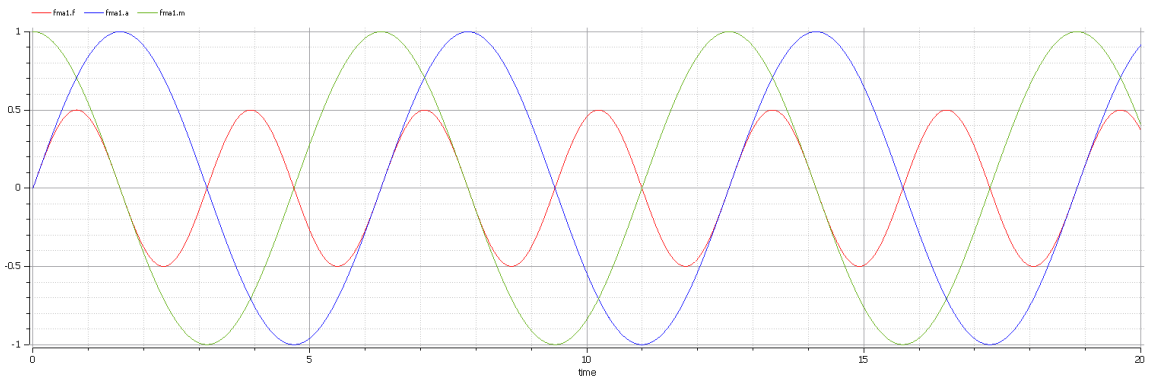


- A Block 'FMA' is modeled with properties 'a', 'f', and 'm' and a constraintProperty 'e1', typed to Constraint Block 'F_Formula'
- The Block 'FMA' does not have any initial value set on its properties, and the properties 'a', 'f' and 'm' are all variable, so their value change depends on the environment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

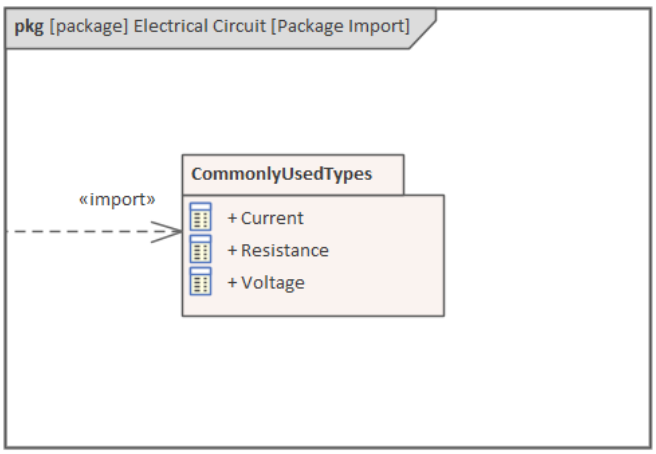


Packages and Imports

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

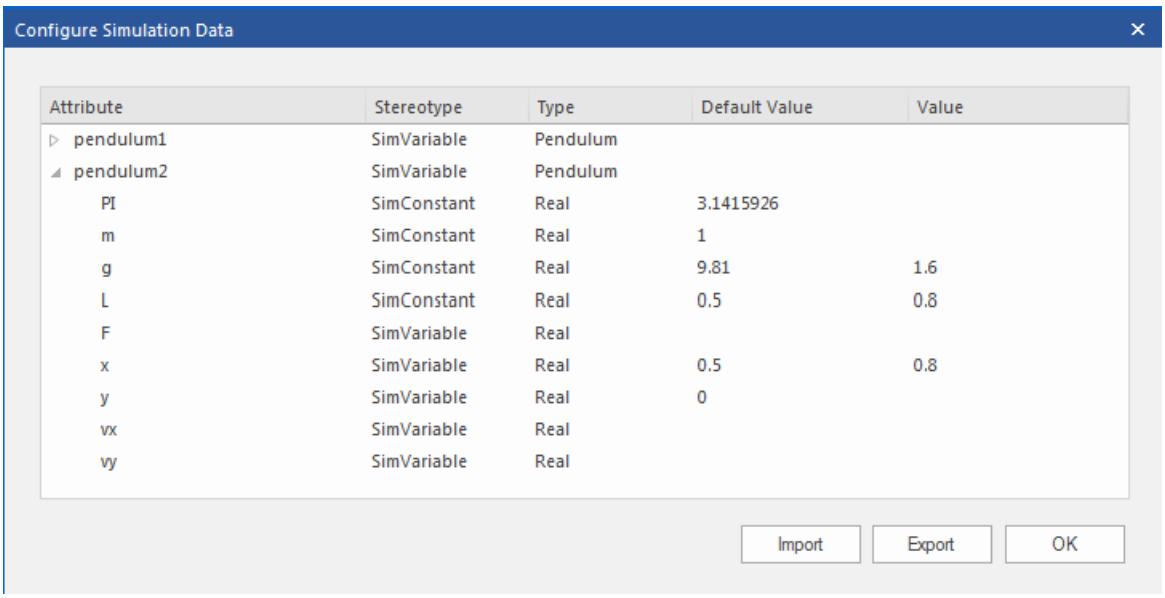
Dataset Management

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, whether it has been configured to be a constant for the duration of the simulation, or a variable for which 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

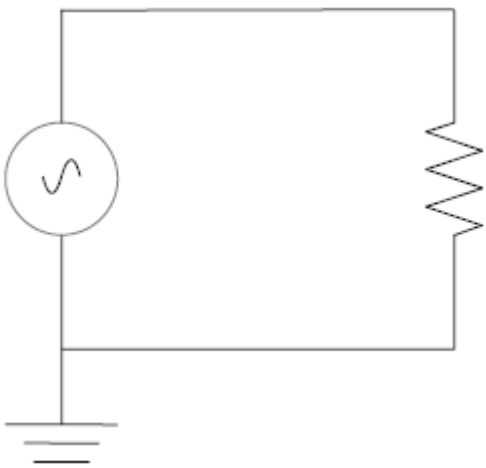
Regulator	being distributed between them. We first simulate a well-balanced system, then we simulate a system where the water will overflow from the second tank.

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.

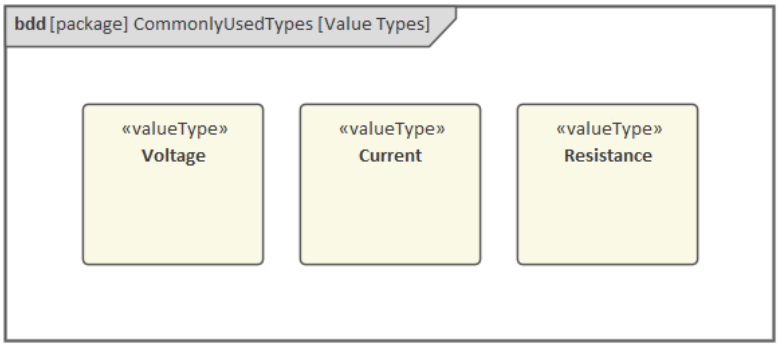


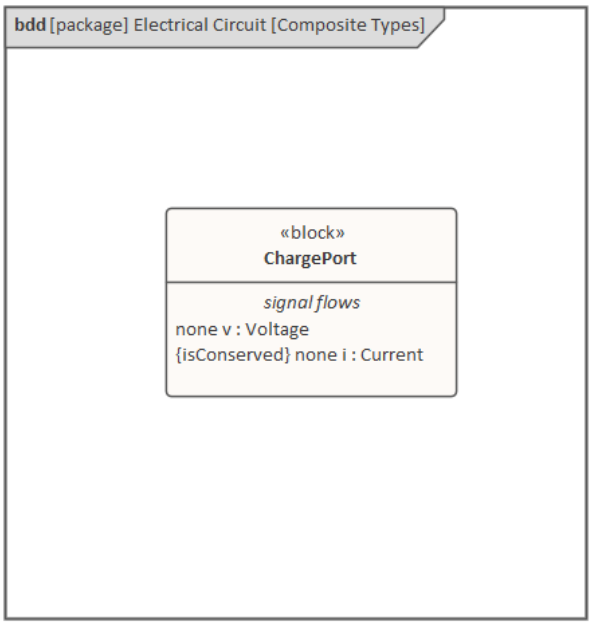
The circuit includes an AC power source, an earth and a resistor, connected to each other by electrical wire.

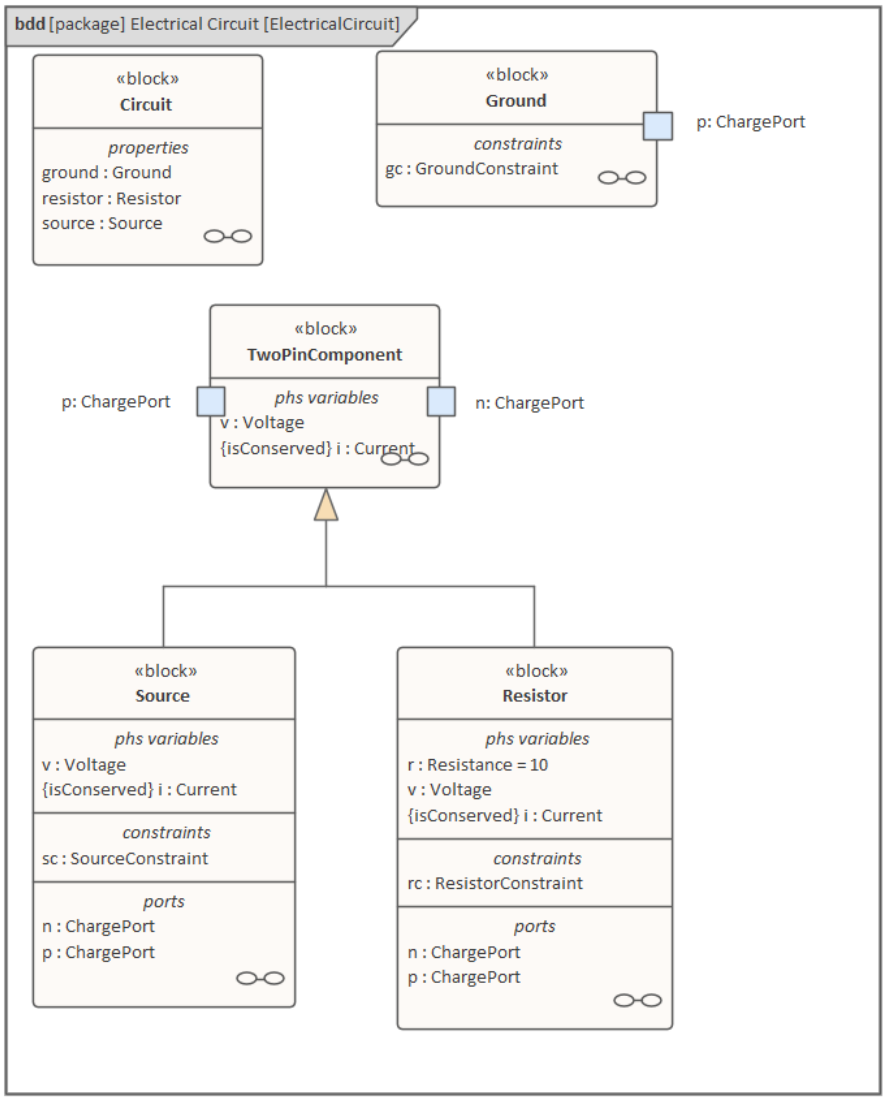
Create SysML Model

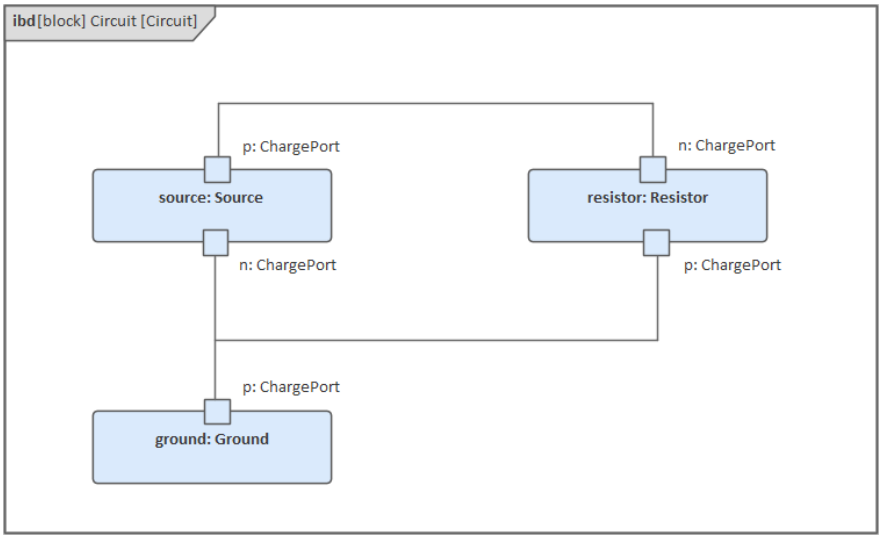
This table shows how we can build up a complete SysML model to represent the circuit, starting at the lowest level types and building up the model one step at a time.

Component	Action

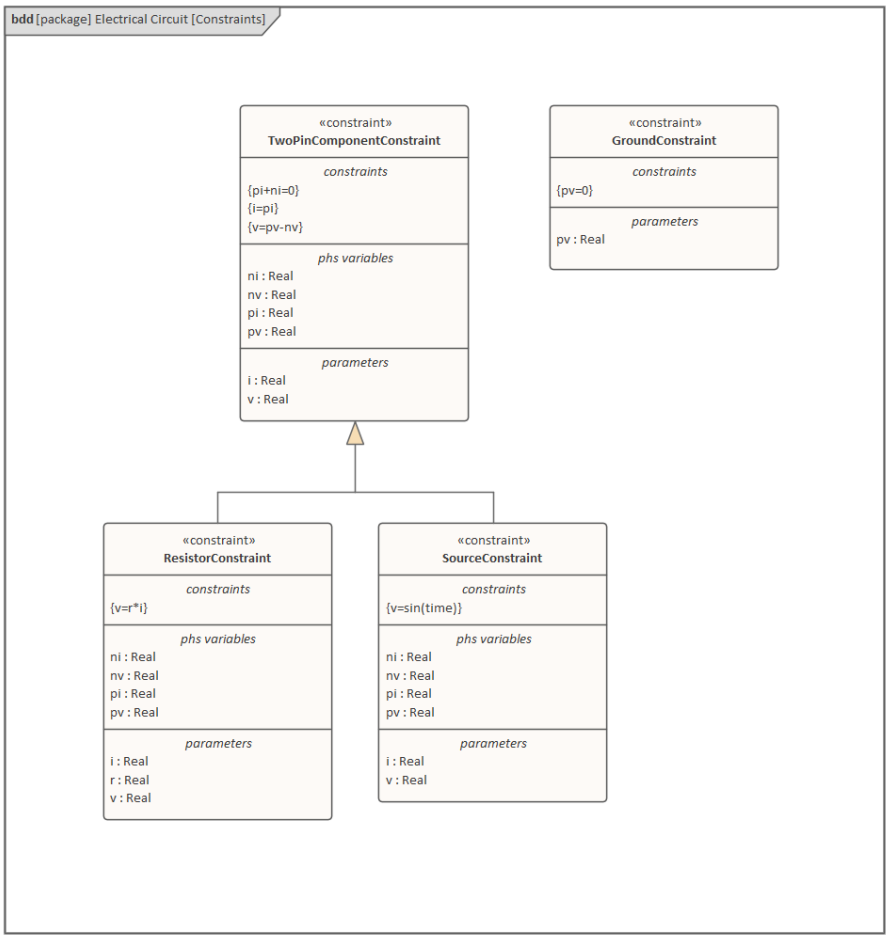
Types	Define Value Types for the Voltage, Current and Resistance. Unit and quantity kind are not important for the purposes of simulation, but would be set if defining a complete SysML model. These types will be generalized from the primitive type 'Real'. In other models, you can choose to map a Value 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

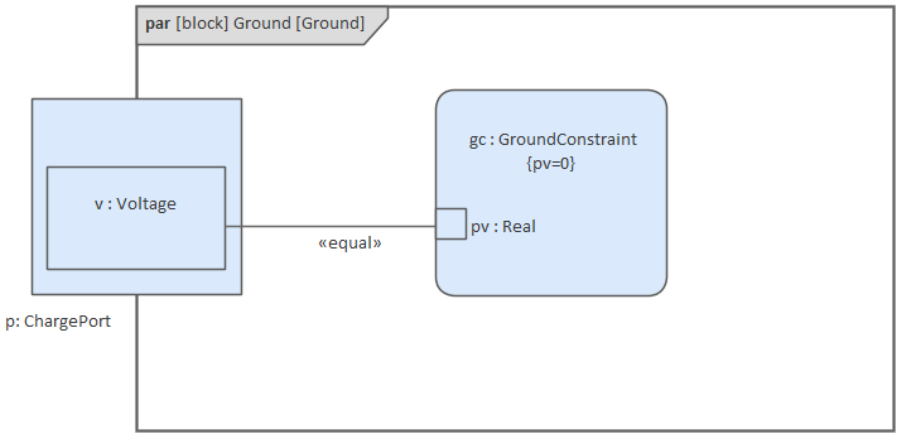
	named 'p' (positive) and 'n' (negative), and they are of type ChargePort. This figure shows the BDD, with the Blocks Circuit, Ground, TwoPinComponent, Source and Resistor. 
Internal Structure	Create an Internal Block Diagram (IBD) for Circuit. Add properties for the Source, Resistor and Ground, typed 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.

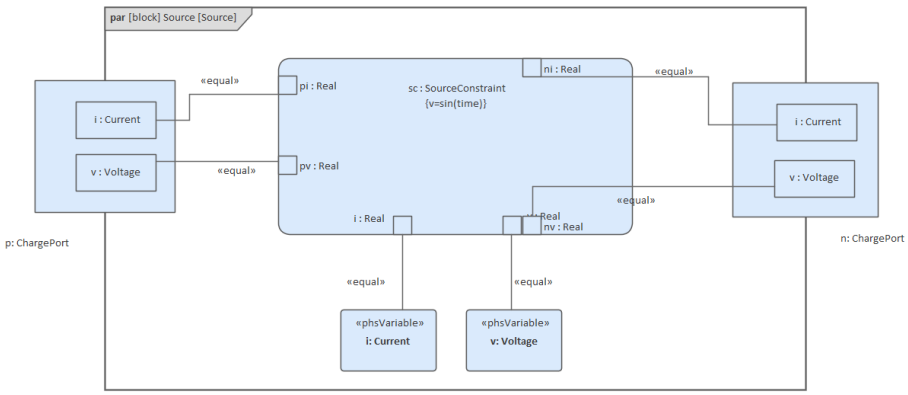


Bindings	The values of Constraint parameters are equated to variable and constant values with binding connectors. Create Constraint properties on each Block (properties typed by ConstraintBlocks) and bind the Block variables and constants to the Constraint parameters to apply the Constraint to the Block.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



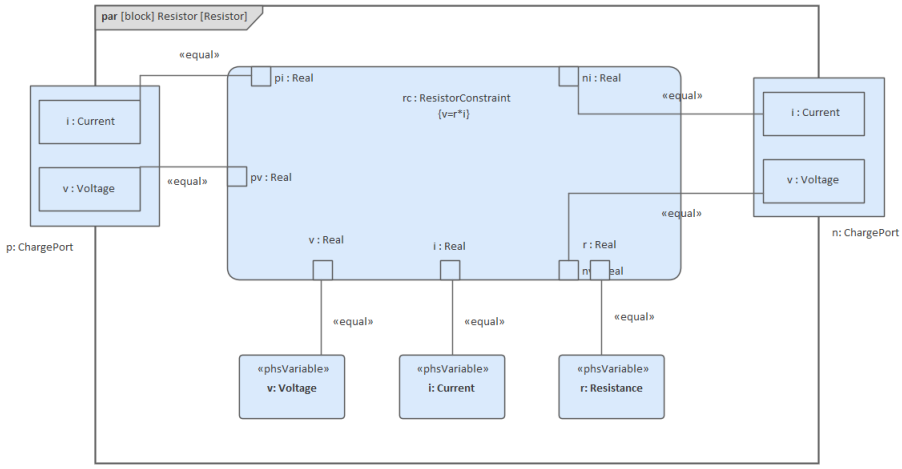
For the Source constraint, bind:

- 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.

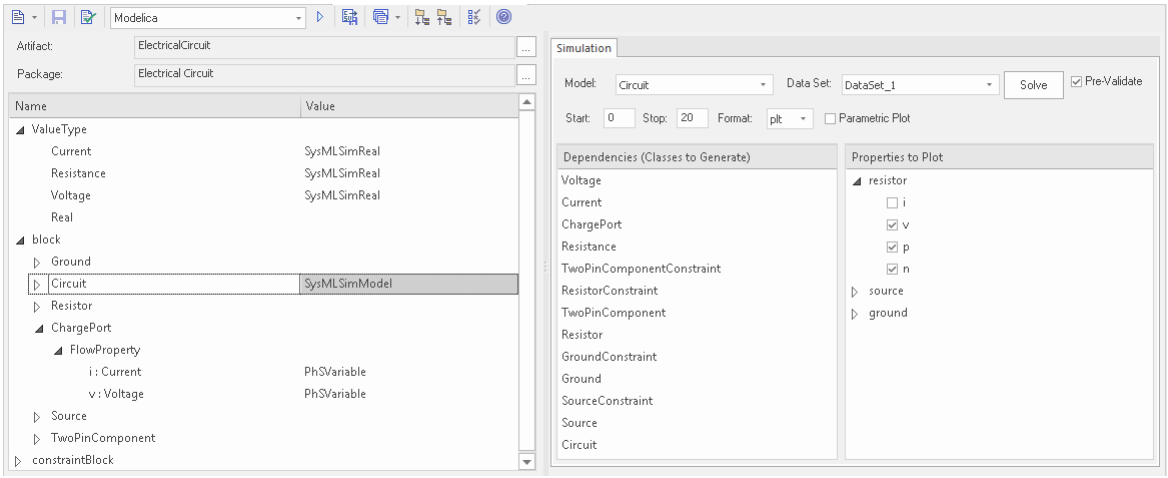
Step	Action
SysMLSimConfiguration Artifact	• Select 'Simulate > System Behavior > Modelica/Simulink > SysMLSim Configuration Manager' • From the first toolbar icon drop-down, select 'Create Artifact' and create the Artifact element • Select the Package that owns this SysML model

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'.

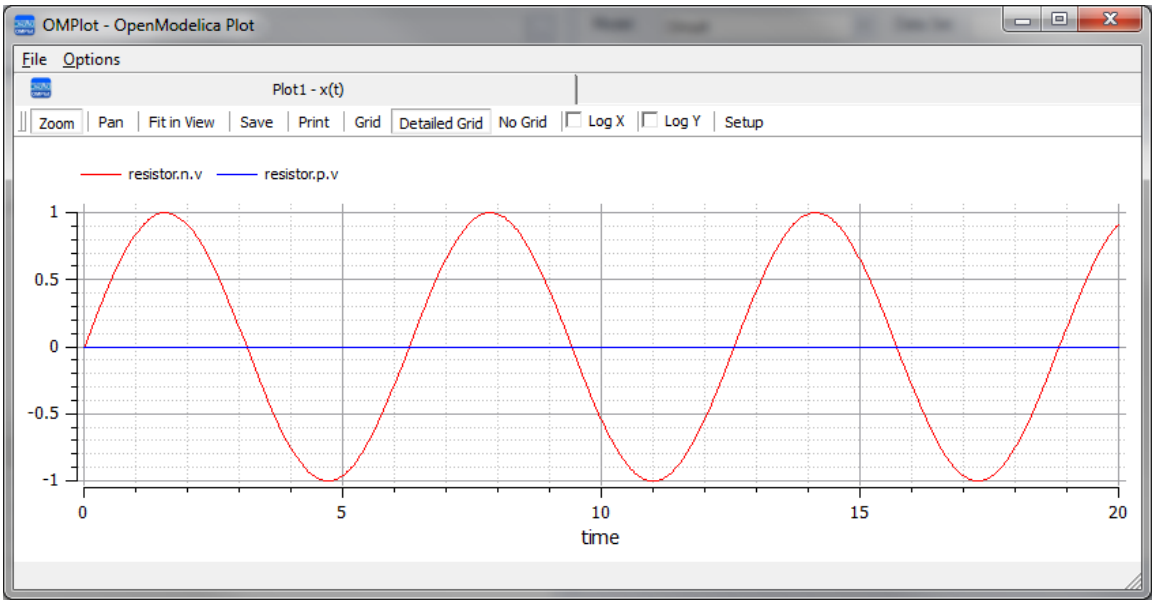
Run Simulation

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.

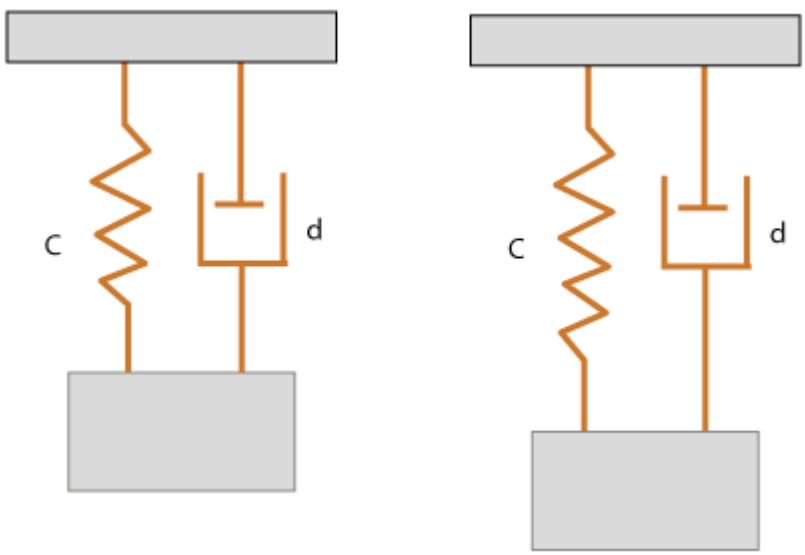


Mass-Spring-Damper Oscillator Simulation Example

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

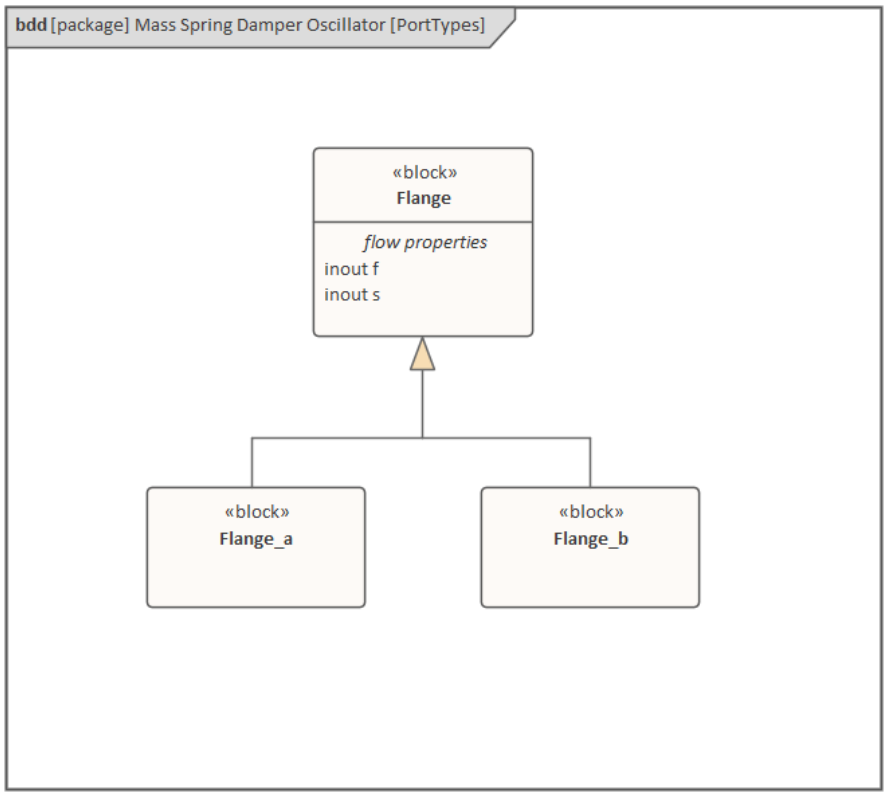
A mass is hanging on a spring and damper. The first state shown here represents the initial point at time=0, just when the mass is released. The second state represents the final point when the body is at rest and the spring forces are in equilibrium with gravity.



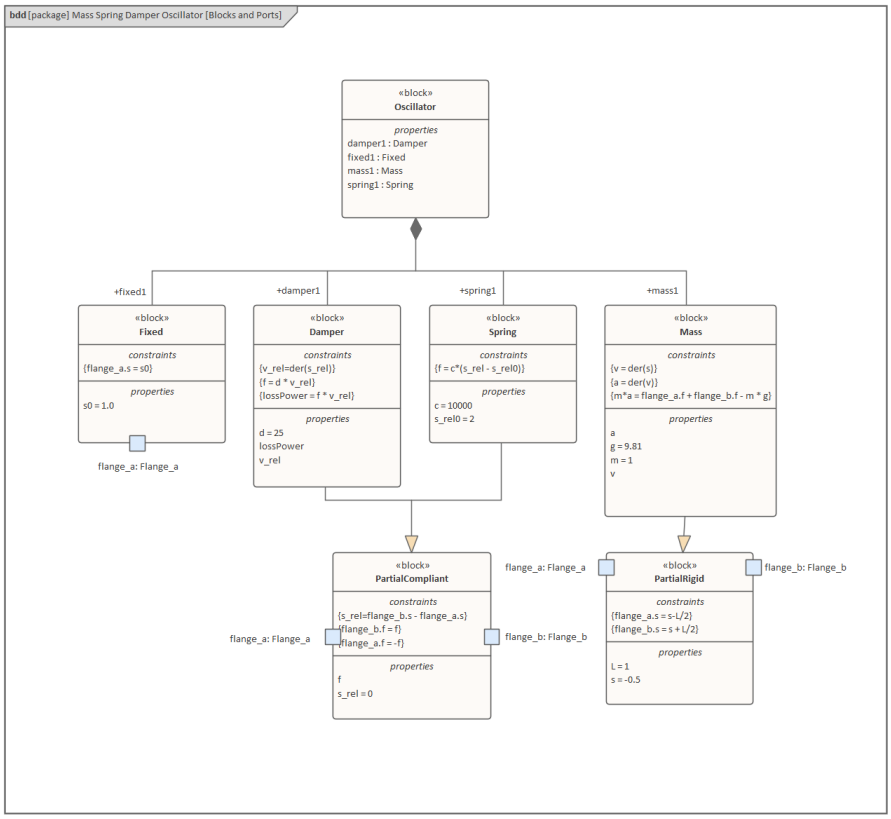
Create SysML Model

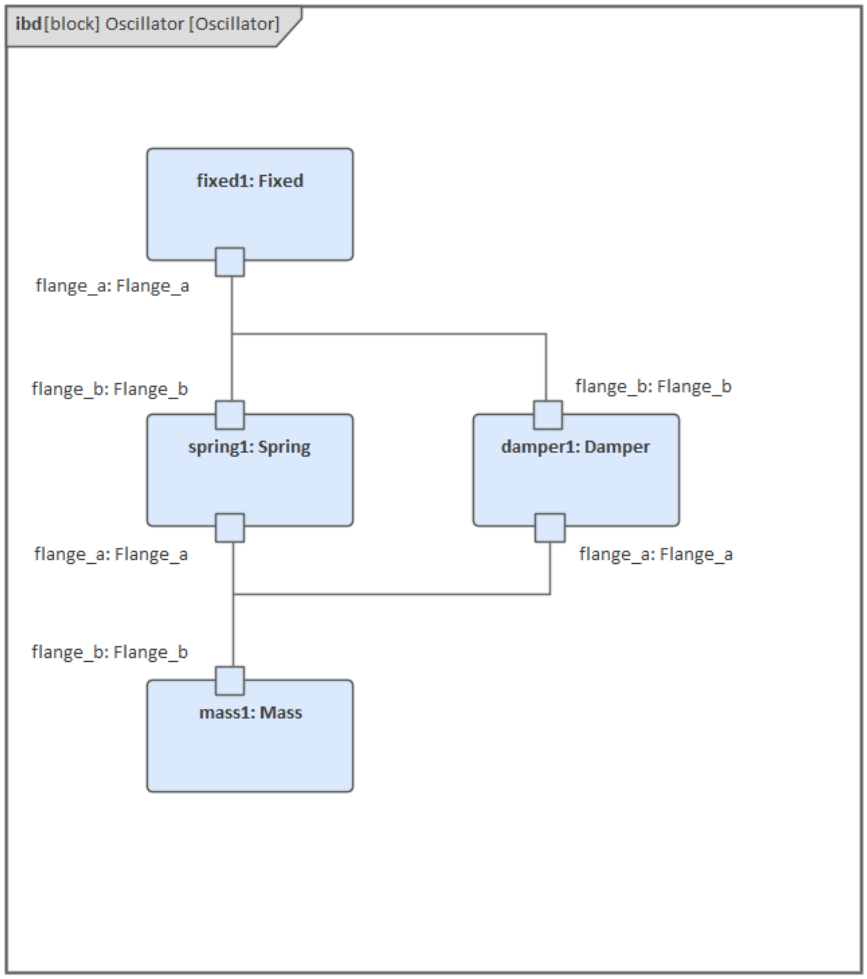
The MassSpringDamperOscillator model in SysML has a main Block, the *Oscillator*. The Oscillator has four parts: a fixed *ceiling*, a *spring*, a *damper* and a *mass body*. Create a Block for each of these parts. The four parts of the Oscillator Block are connected through Ports, which represent mechanical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

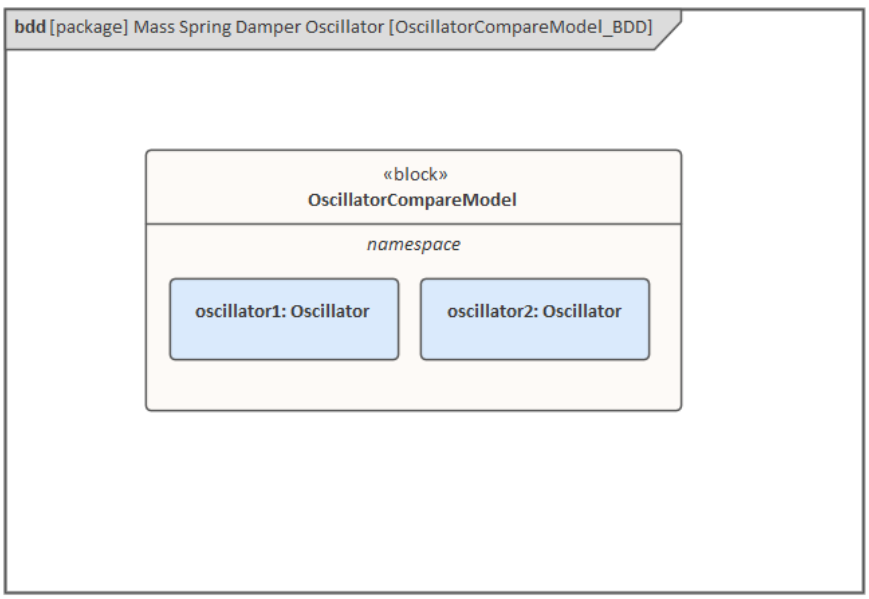
Two Oscillator Compare Plan

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*

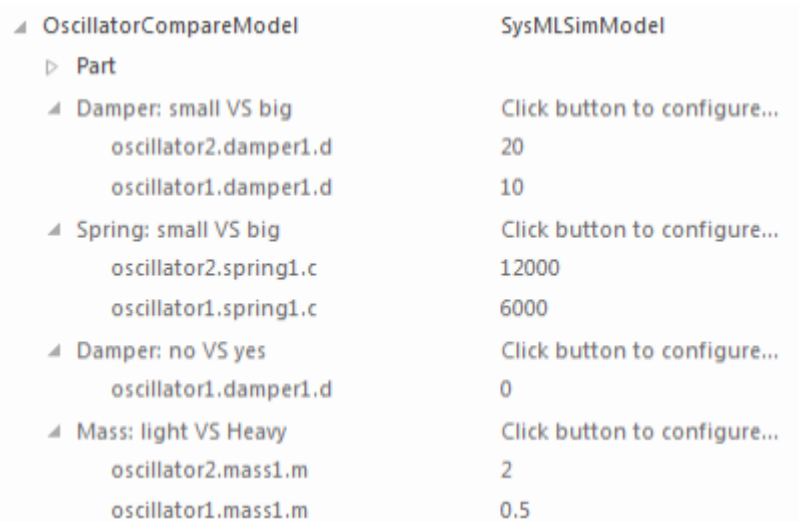


Setup DataSet and Run Simulation

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

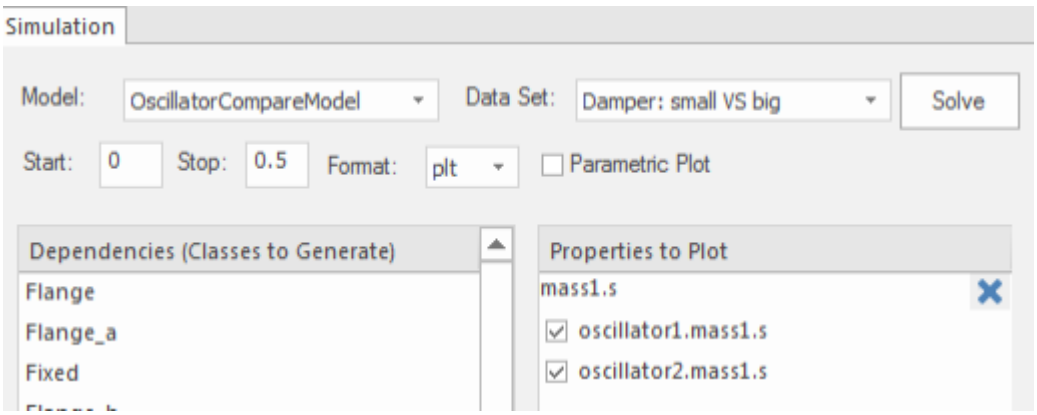
- Damper: small vs big
  provide 'oscillator1.damper1.d' with the value 10 and 'oscillator2.damper1.d' with the larger value 20
- Damper: no vs yes
  provide 'oscillator1.damper1.d with the value 0; ('oscillator2.damper1.d' will use the default value 25)
- Spring: small vs big
  provide 'oscillator1.spring1.c'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



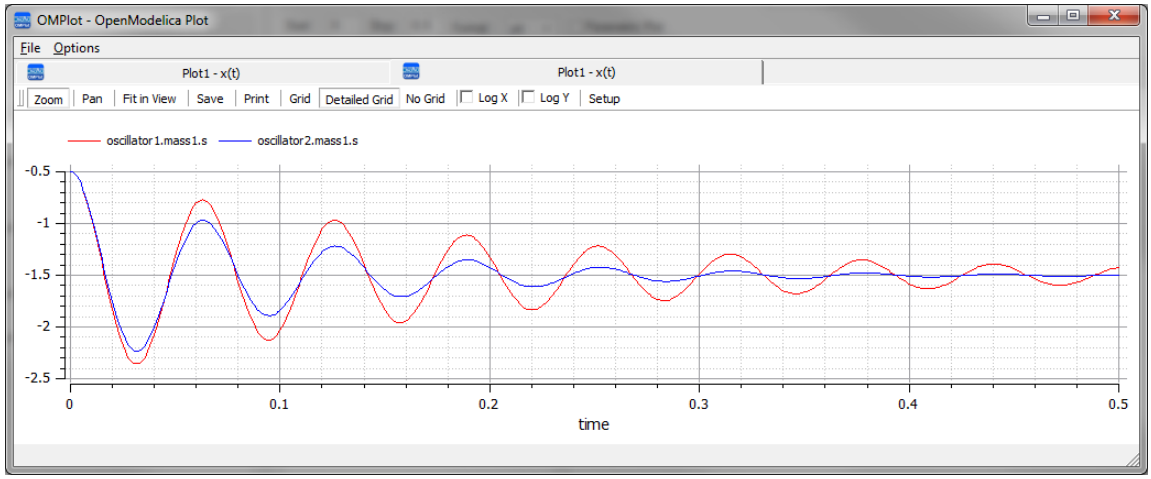
On the 'Simulation' page, select 'OscillatorCompareModel', plot for 'oscillator1.mass1.s' and 'oscillator2.mass1.s', then choose one of the created datasets and run the simulation.

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.

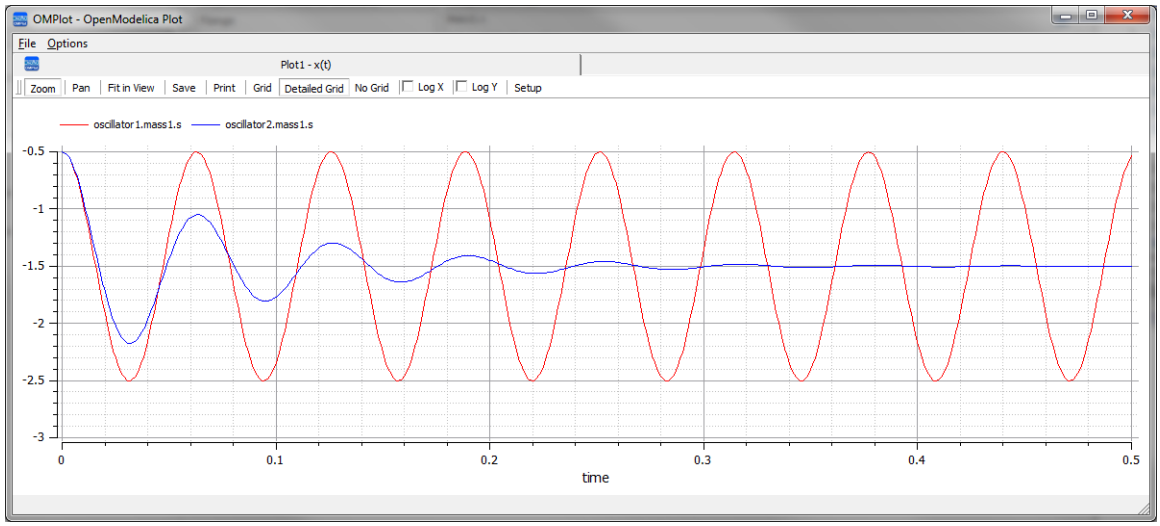


These are the simulation results:

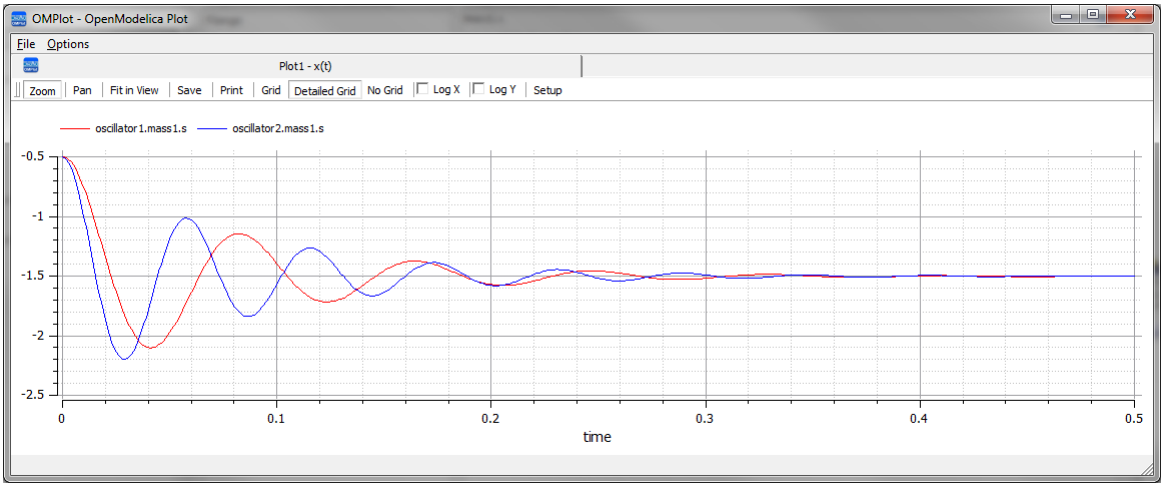
- Damper, small vs big: the smaller damper allows the body to oscillate more



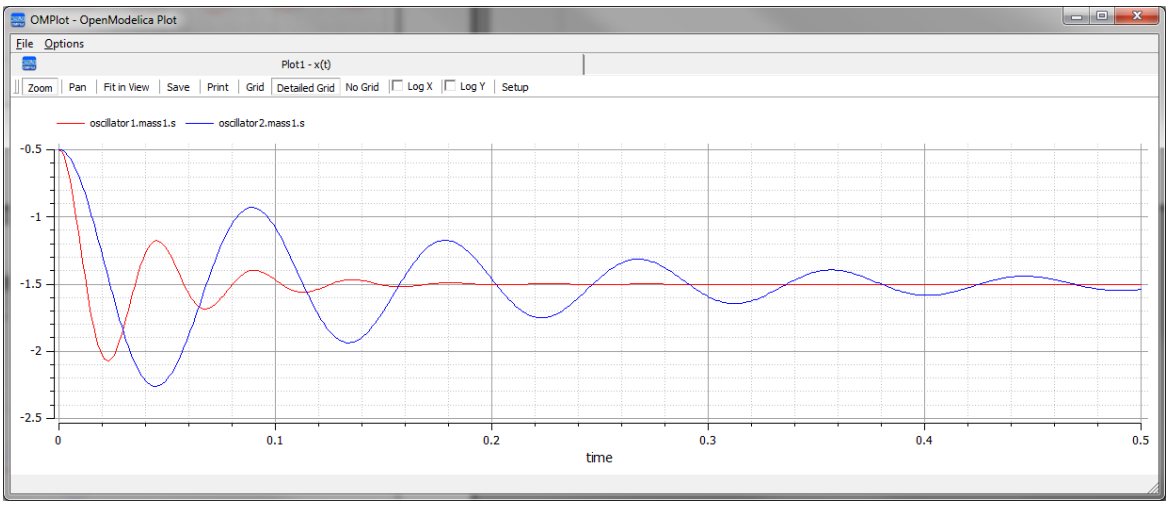
- Damper, no vs yes: the oscillator never stops without a damper



- 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



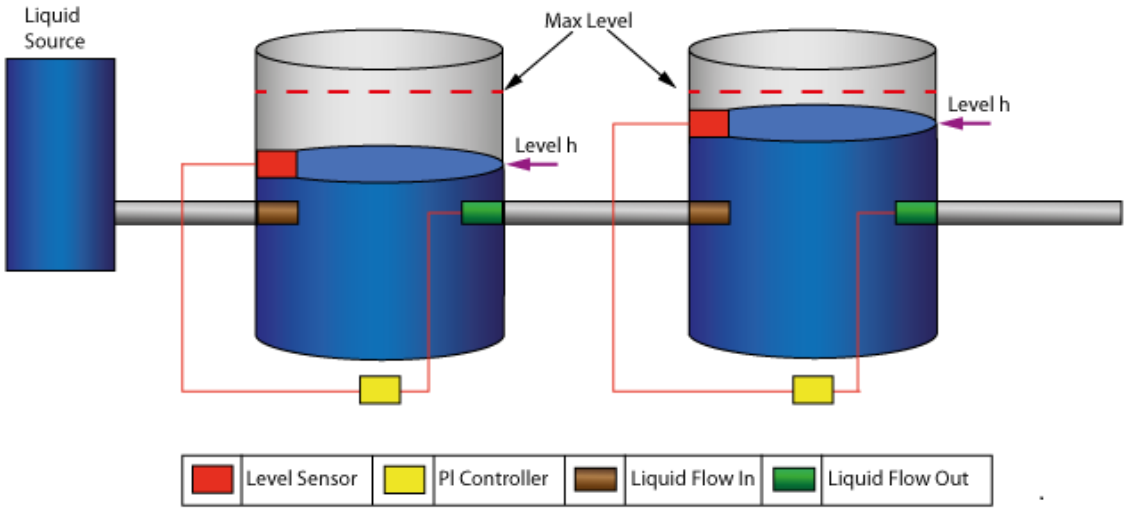
Water Tank Pressure Regulator

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.

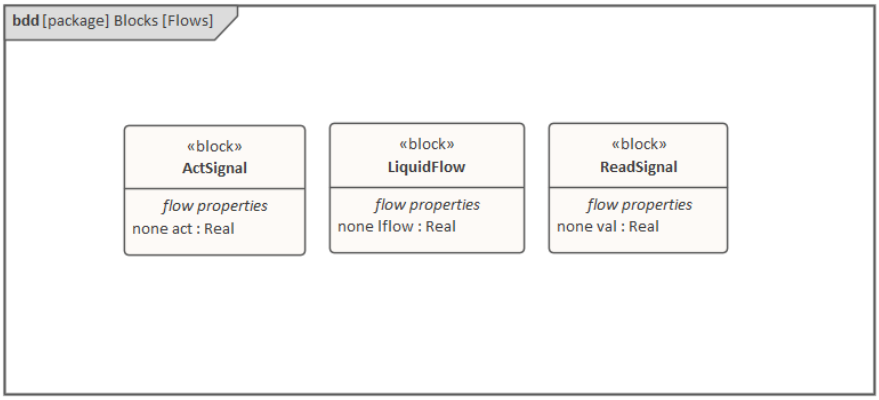
System being modeled

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"



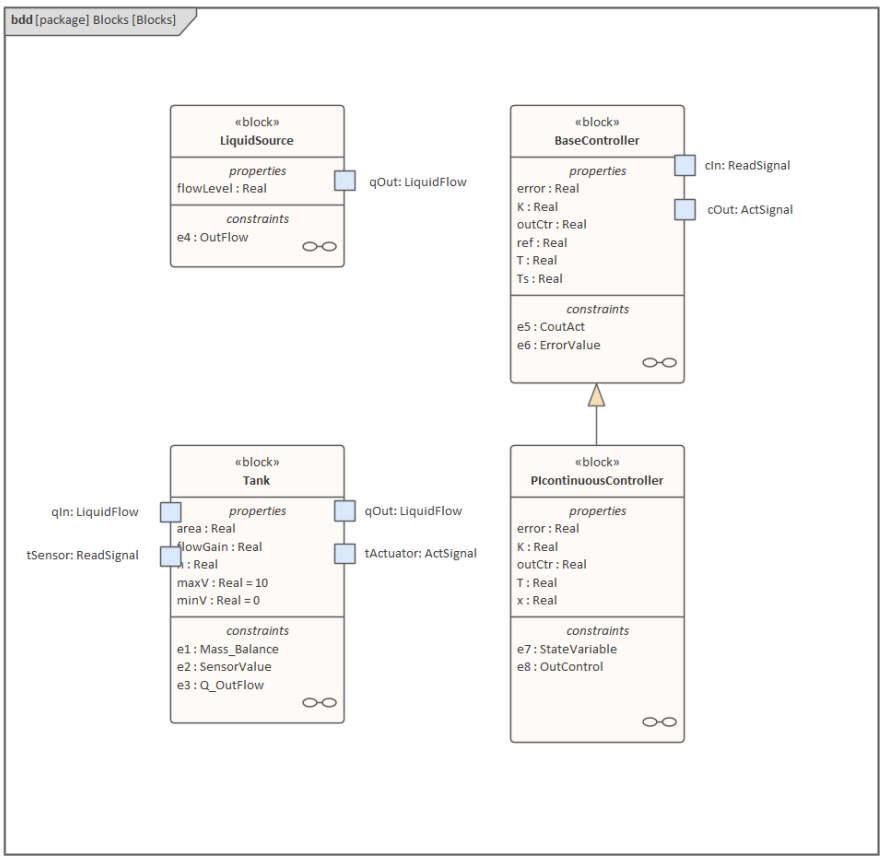
Block Definition Diagram

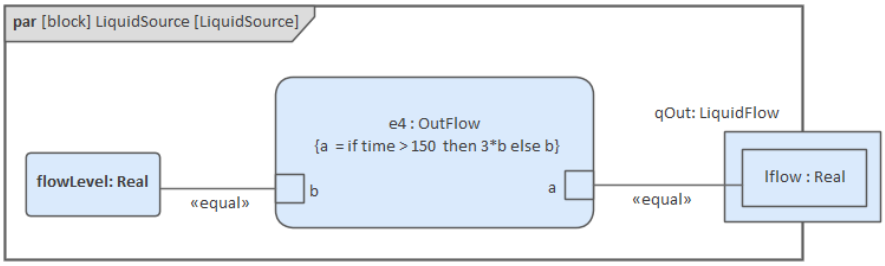
LiquidSource: The water entering the tank must come from somewhere, therefore we have a liquid source component in the tank system, with the property *flowLevel* having a unit of 'm³/s'. A Port 'qOut' is typed to 'LiquidFlow'.

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

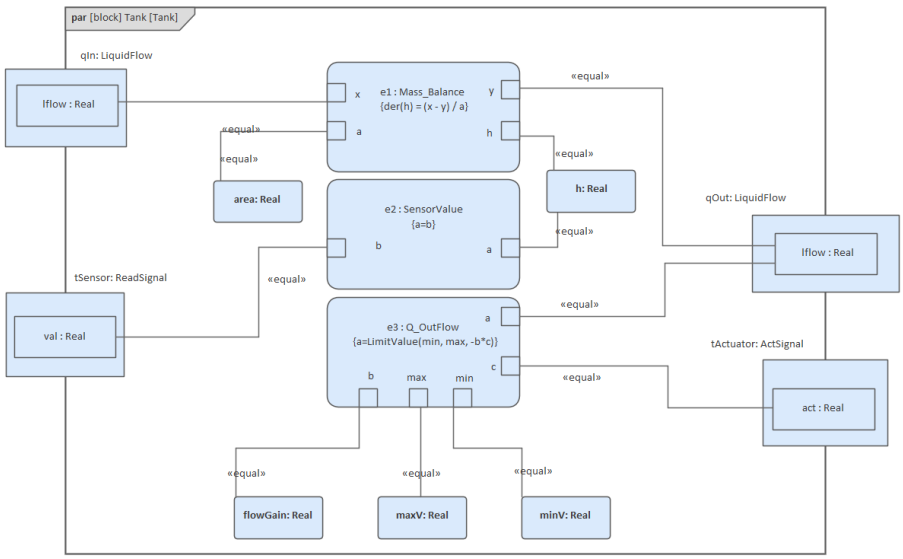
	position PIcontinuousController: specialize from BaseController • Properties: - x: the controller state variable 
ConstraintBlocks	The flow increases sharply at time=150 to a factor of three of the previous flow level, which creates an interesting control problem that the controller of the tank has to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



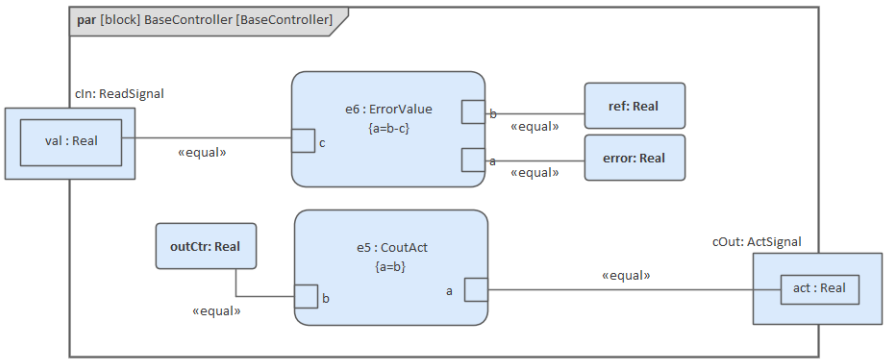
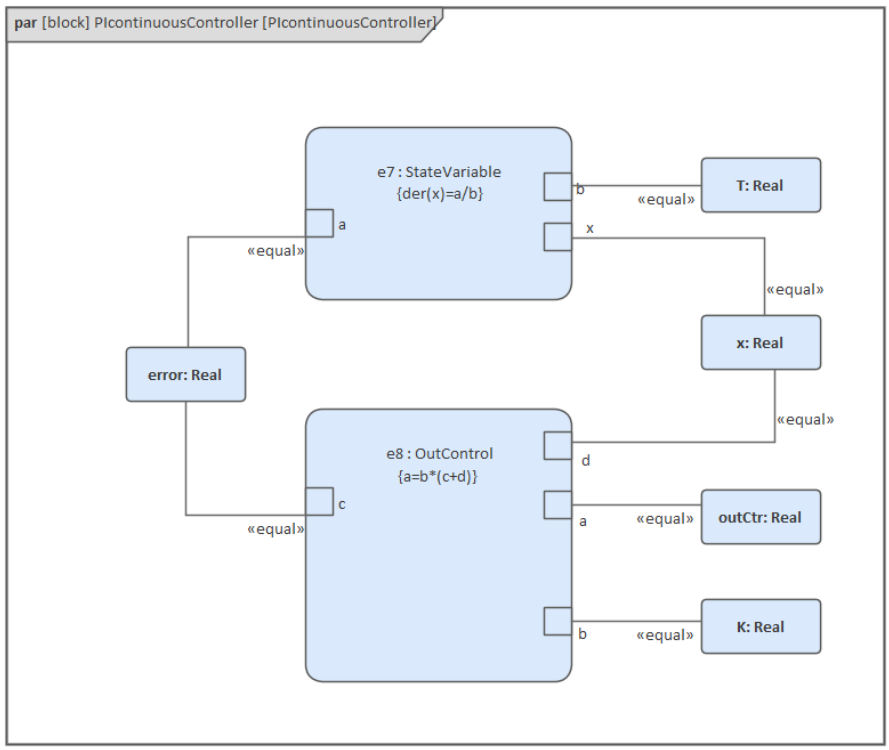
The central equation regulating the behavior of the tank is the *mass balance* equation.

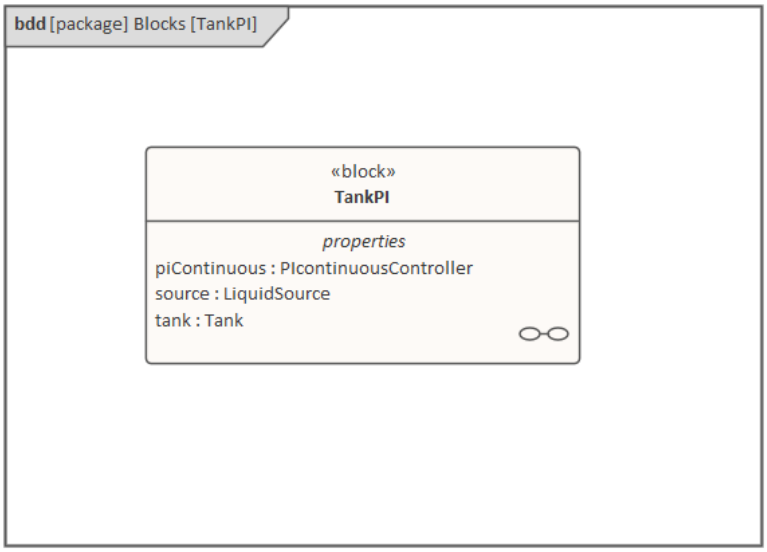
The output flow is related to the valve position by a 'flowGain' parameter.

The sensor simply reads the level of the tank.

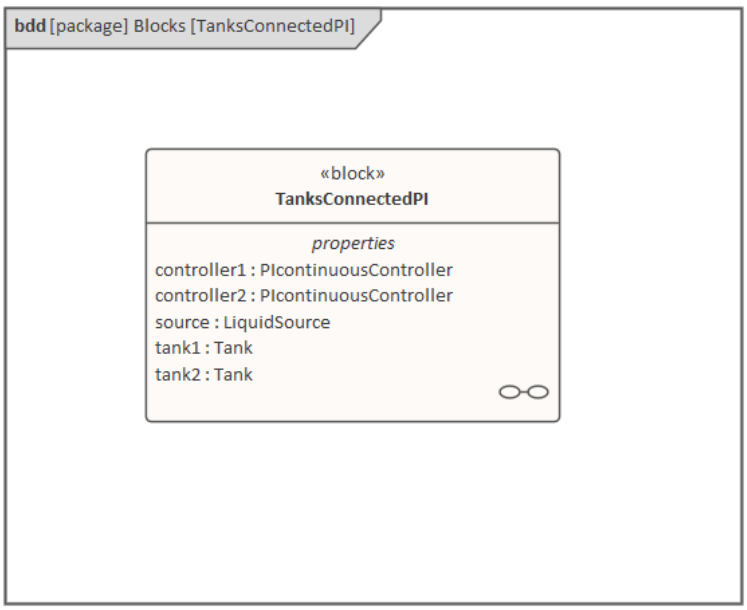


The Constraints defined for 'BaseController' and 'PIcontinuousController' are illustrated in these figures.

	 
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.

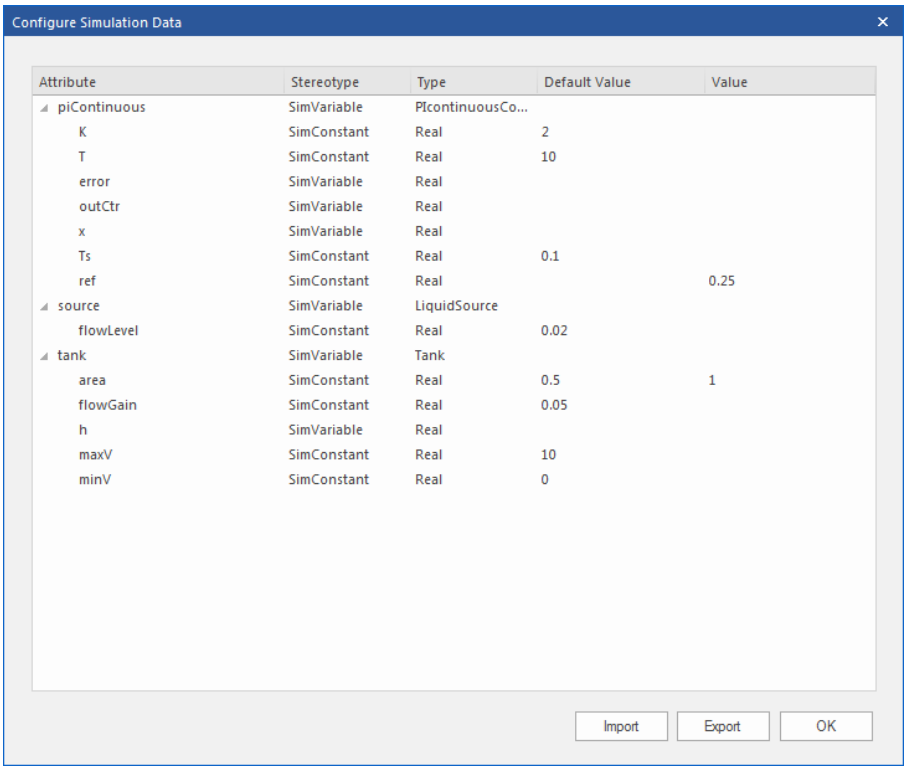
Block	Properties
LiquidSource	Configure as 'SysMLSimClass'. Properties configuration:

	• 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.

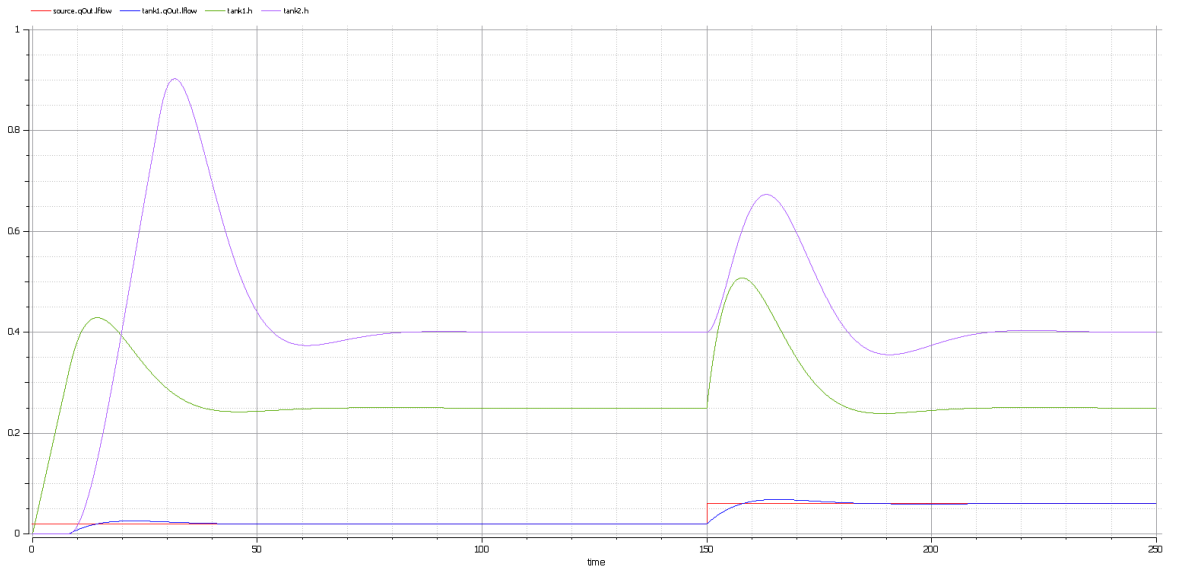
Element	Dataset
LiquidSource	flowLevel: 0.02
Tank	h.start: 0 flowGain: 0.05 area: 0.5 maxV: 10 minV: 0
BaseController	T: 10 K: 2 Ts: 0.1
PIcontinuous Controller	No configuration needed. By default, the specific Block will use the configured values from super Block's default dataSet.
TankPI	What is interesting 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:

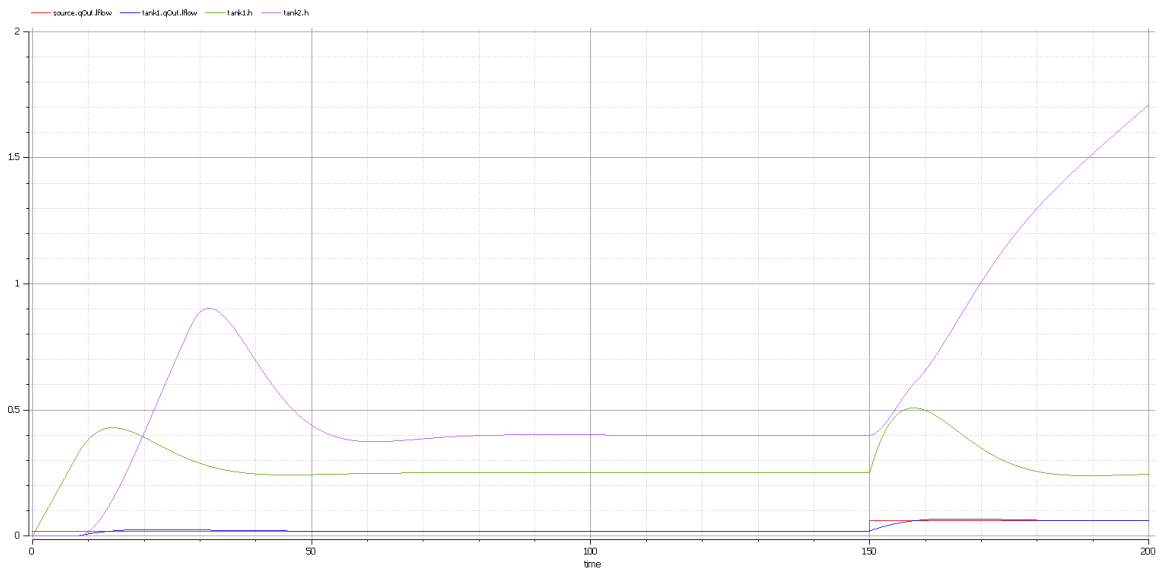
- The liquid flow increases sharply at time=150, to 0.06 m^3/s, a factor of three of the previous flow (0.02 m^3/s)
- Tank1 regulated at height 0.25 and tank2 regulated at height 0.4 as expected (we set the parameter value through the data set)
- Both tank1 and tank2 regulated twice during the simulation; 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.

Simscape Integration

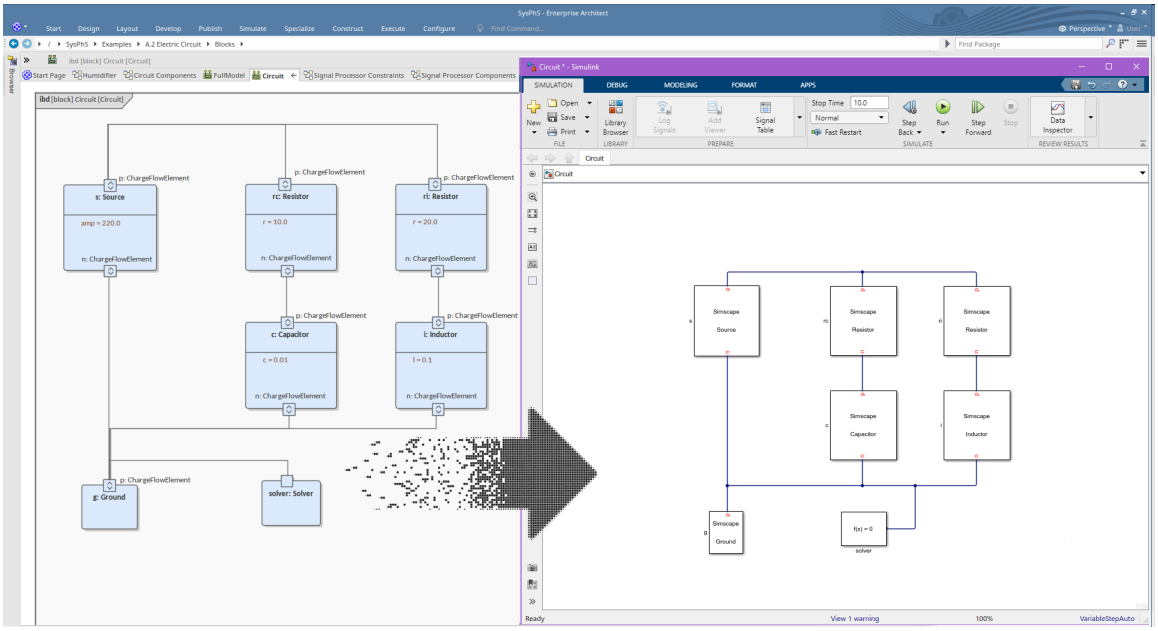
Simscape is used to model physical systems with physical flows, translating SysML Internal Block diagrams into a Simulink model, opening up Simscape's vast array of library blocks across many different physical domains.

Enterprise Architect can translate SysML Internal Block diagrams into MATLAB's Simscape, which is an optional extension to Simulink that allows modeling of physical systems and instructs MATLAB to simulate and plot the requested outputs. The Blocks represent physical objects, and the flows represent the physical flow of a substance or energy (such as liquid, current, gas, flux, force/torque, heat flow and so on; for example, water flowing from one tank to the next, or current flowing through a resistor). You can use the built-in SysPhS patterns to access the vast array of pre-built Simscape library blocks, or create references to your own custom library Blocks using the SimulinkBlock stereotype.

See the link to "*Basic Principles of Modeling Physical Networks"* for details on using Simulink and Simscape.

Physical flow properties must be typed by Block and include one conserved PhSVariable and one non-conserved PhSVariable. Use the Model Builder patterns for SysPhS available under the SysML perspective, which include:

- SysPhS elements for physical interaction
- SysPhS elements for signal flow



Requirements

- The option 'Use Simscape' must be ticked in the Configure SysML Simulation window
- Ports with 'inout' direction (or none) will be assumed to be physical flows
- Any Block with an 'inout' Port will be generated as Simscape – if it also has 'in' or 'out' Ports, these will be set to Simscape 'physical signals'. These can be connected to output 'physical signals' or to Simulink inputs and outputs. The Simulink converter required to do this will be automatically generated and inserted into the model during generation.

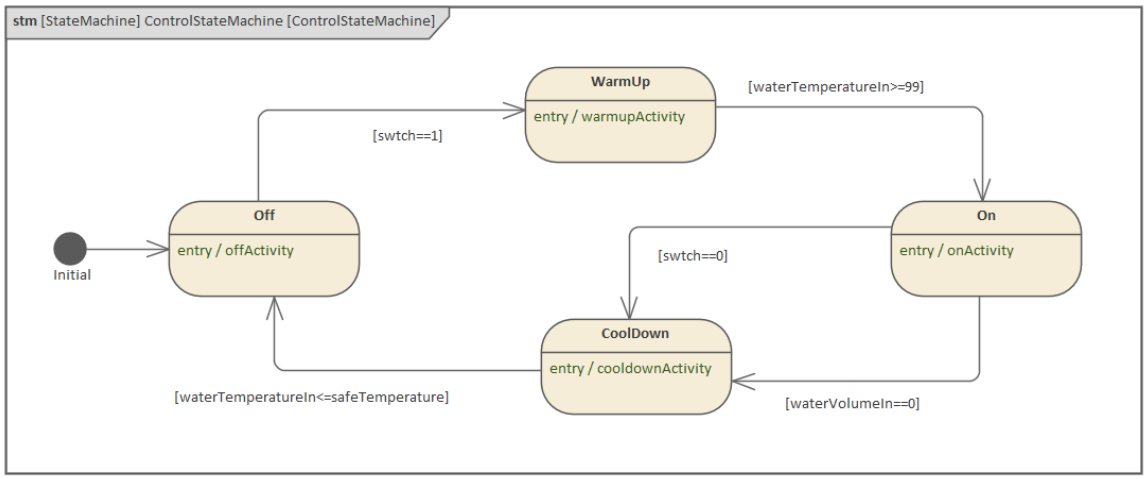
Videos

Sparx Systems provide a YouTube video of generating a SysML Simulation plot using Simscape. See:

- SysML Simulation Plotting Using Simscape

Simulink Integration

Enterprise Architect will translate a SysML model into the Simulink format and automatically run the simulation, plotting the outputs of the selected variables. The generated Simulink file can also be opened directly in Simulink, allowing modification and fine-tuning of the simulation settings and output functionality.

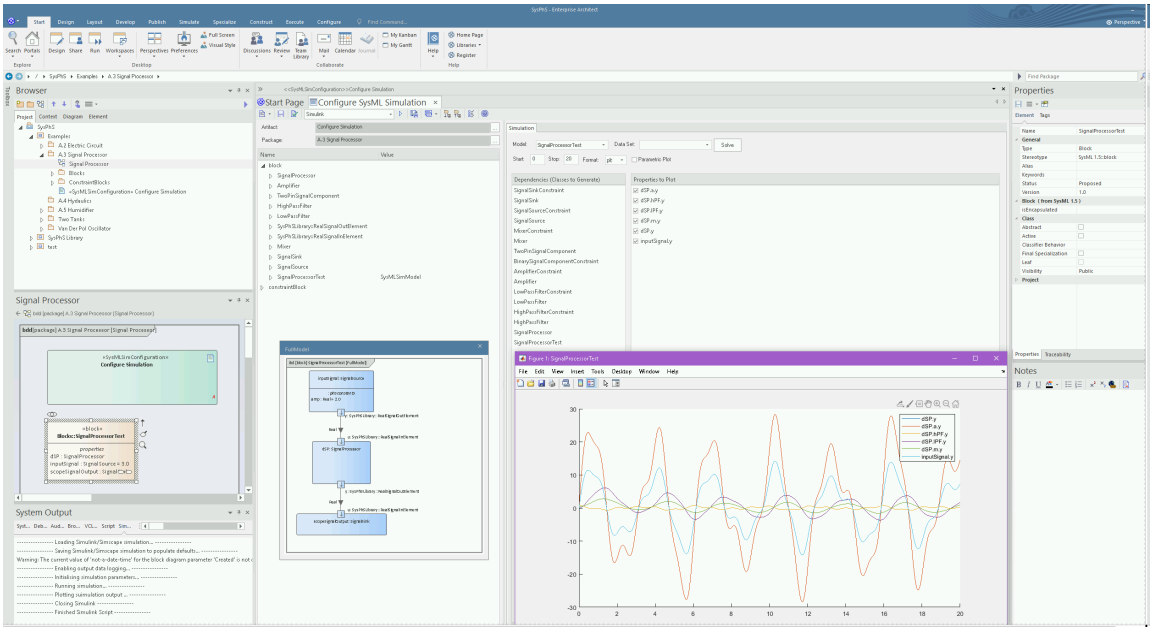


Given Simulink uses a Block diagram approach where the signal flow is unidirectional, what Enterprise Architect supports is the export of models to Simulink for simulating directed messages between Blocks, producing charts of the simulation results.

You have drag-and-drop access to common built-in Simulink library blocks directly through Enterprise Architect patterns or you can reference your own custom built blocks with stereotype parameters under the SysPhS standard.

Simulation with Simulink is an alternative to using OpenModelica, which is also available in Enterprise

Architect.



Modeling for Simulink

For modeling diagrams that can be exported to Simulink you have drag-and-drop access to common built-in Simulink library blocks directly through Enterprise Architect patterns, or you can reference your own custom built blocks with stereotype parameters under the SysPhS standard.

Simulink can be used to model Signal flow; that is, one Block sends a piece of information to another Block. This means, for Simulink modeling, all Ports must have a direction defined as either *in* or *out*. Item Flows between Ports must match the in/out directions. (Note that Modelica supports modeling of either Signals or physical systems.)

Requirements

In order to conform with the unidirectional requirements of Simscape:

- Ports must have direction defined as either 'in' or 'out'
- Ports must be connected consistent with their direction; for example, an 'in' Port cannot be connected to another 'in' Port
- Constraint parameters will be considered to be variables unless specified as stereotype ‘PhSConstant’ (or set to SimConstant in the Configure SysML Simulation window)
- Constraint equations must have a single term on the left-hand-side (lhs) of the equals sign, and this term must

be either an output or a derivative

Videos

Sparx Systems provide a YouTube video of performing a SysML StateMachine Simulation in Simulink. See:

- SysML StateMachine Simulation in Simulink

Notes

- Simulink will only allow the output Ports to be plotted; there could be a future enhancement to allow other variables to be plotted too - note that an input Port is inherently the same as the output Port that connects to it

Learn More

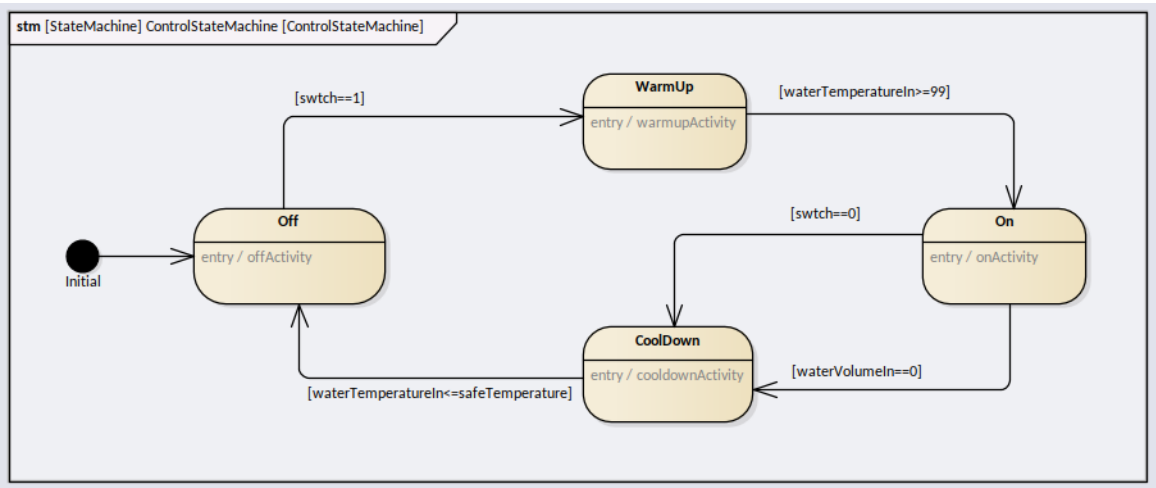
- Creating Simulink and Simscape Specific Blocks

Stateflow Integration

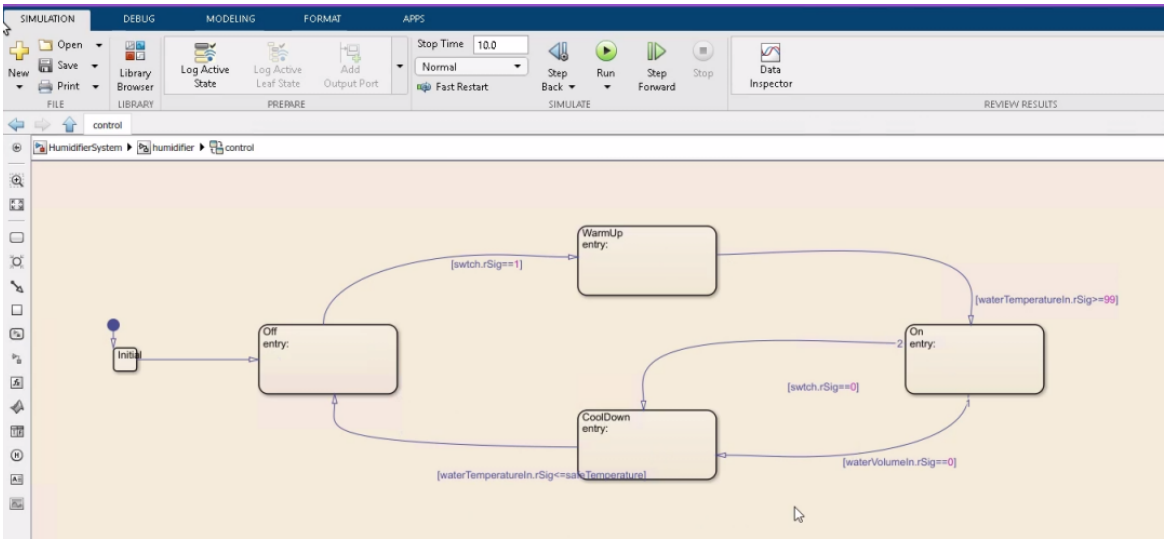
A significant feature when using Enterprise Architect's SysML simulation is the ability to generate MATLAB Stateflow diagrams to be run under Simulink, allowing you to guide your SysML simulations using StateMachines modeled in Enterprise Architect that are translated to Stateflow diagrams.

The Stateflow diagrams use elements and connectors that have close equivalents to the OMG StateMachine standard, such as States and Transitions. Enterprise Architect supports all the Stateflow features; however, Stateflow only uses a subset of the OMG StateMachine standards.

This is an example of a StateMachine diagram in Enterprise Architect:



This is an example of how the code generated by Enterprise Architect, from the diagram above, is rendered as a Stateflow diagram:



Supported Element Types

The SysML element types that can be translated to StateFlow objects are:

- State
- Initial
- Final
- Junction
- Choice (converted to Stateflow Junction)
- History

Supported Connector Types

The key SysML StateMachines connector type does have a direct translation to the Simulink connector type 'Transition'.

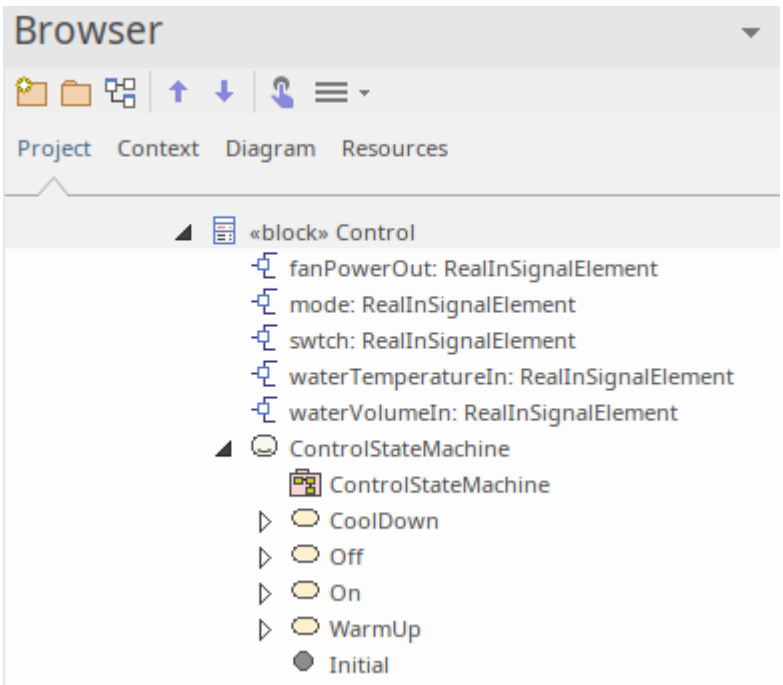
Modeling for Stateflows

You have several options when creating a StateMachine diagram that is intended to be rendered as a Stateflow diagram. These include which elements and connectors can be used, the placement of the StateMachine, where code can be defined in the model, and what format of code can be used.

StateMachine Placement

When creating a model to be generated as a Stateflow diagram, the StateMachine diagram must be placed under a SysML Block and have a child StateMachine element, with a StateMachine diagram under that StateMachine element.

For example, in this case the Block 'Control' has a child StateMachine named 'ControlStateMachine', which has a child StateMachine diagram 'ControlStateMachine':



Note that the Block must only contain a StateMachine diagram; it should not contain other diagrams.

Defining code

There are a number of options for placing script in a StateMachine model. The key considerations for this are: what is the format of the script, and where can it be placed?

Code format

When you write any valid SysML code it will be translated to valid Stateflow code. The SysPhS specification defines the Modelica maths language as the 'standard', which is what Enterprise Architect accepts as code snippets in the StateMachine diagram, and translates these to the MATLAB/Stateflow equivalent. For more details refer to section 10 of the *OMG SysPhS specification*.

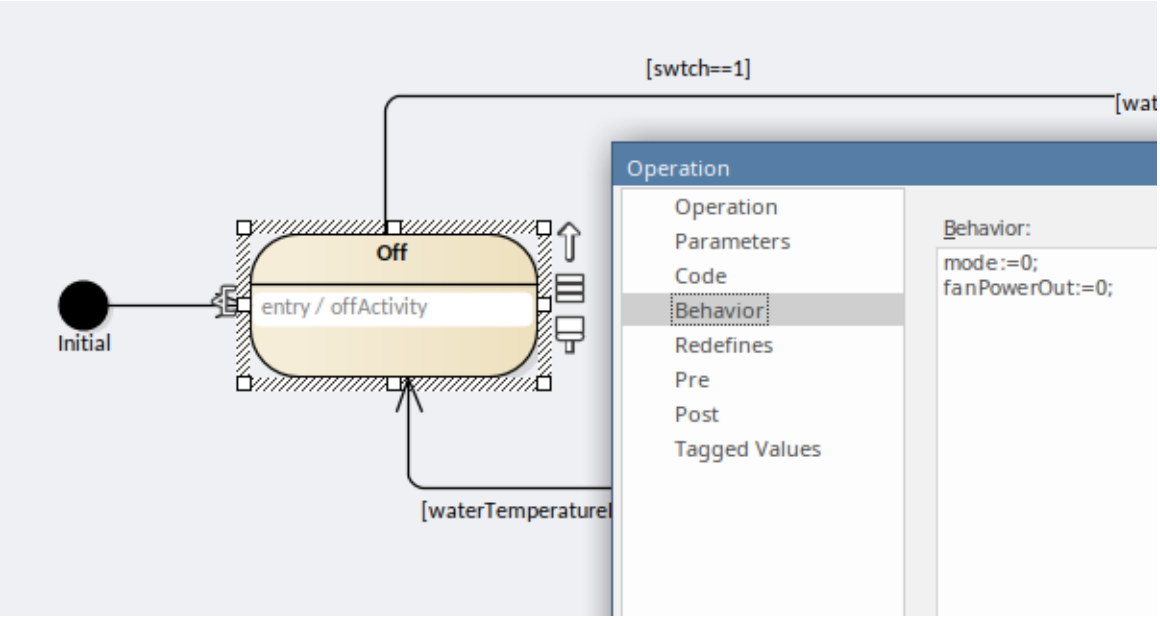
Code placement

Internal to a State and in conformance with the UML specification, Stateflow supports the three standard Operations:

- Entry
- Exit
- Do

Each of these can contain code, which is defined in the

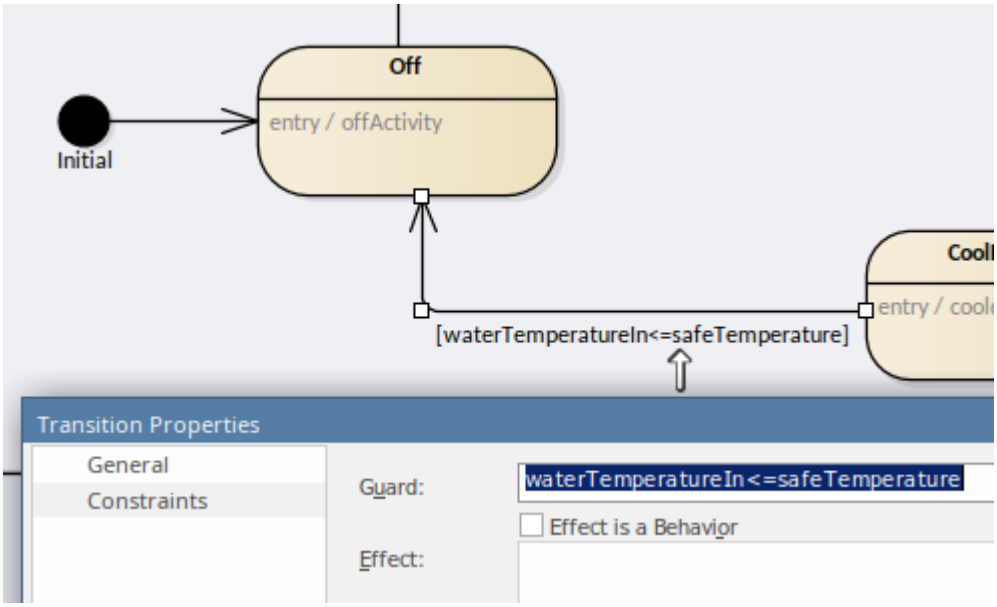
State's Operation under Behavior. For example, this is a code snippet for an Entry:



On Transitions there are three key options for using code:

- Transition Guard
- Transition Effect
- Trigger

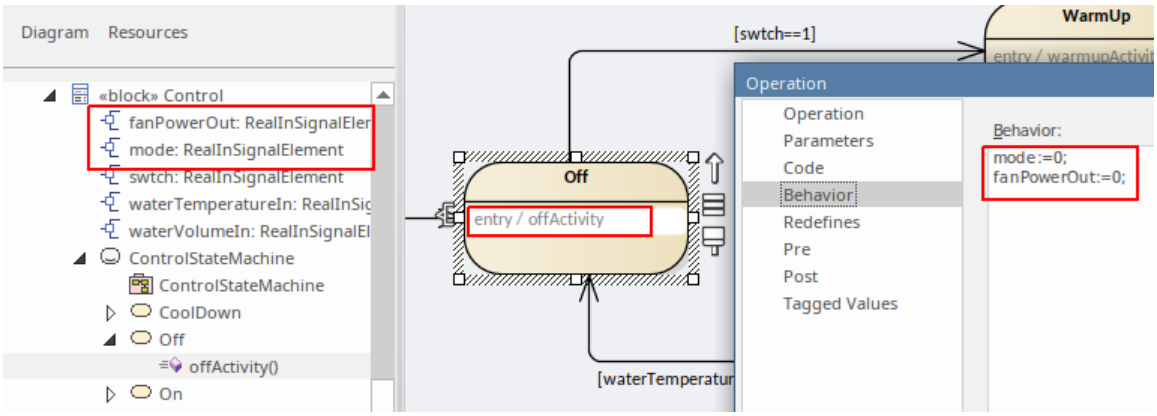
For example, this is a condition statement as defined in a Transition Guard:



Using Block Properties

For the code, the Block Properties such as constants and Ports can be referenced within the script. In the case of the Ports, the details being input into the Block are derived using the input Port name, and similarly values can be assigned to a variable of the same name as the output Port.

For example:



Notes

These SysML StateMachine features are not available in Stateflow:

- Deep History state
- History states that have out-going transitions (that is, initial States) to be used on all default entries
- Fork/Join
- Sync
- Junction (only Choice - Junction will be replaced with Choice where possible)
- Entry/Exit points
- References to other StateMachines; Stateflow can do child StateMachines but they can't be referenced again
- Only one StateMachine per SysML Block

OpenModelica Integration

OpenModelica is a free and open-source environment based on the Modelica modeling language for modeling, simulating, optimizing and analyzing complex dynamic systems. Enterprise Architect is integrated with OpenModelica, and supports its use under the SysPhS Standard (*SysML Extension for Physical Interaction and Signal Flow Simulation*) for defining constants and variables within the SysML Blocks rather than in the Simulation configuration. This provides a simpler model-based method for defining and sharing simulations.

You can also display the SysML Block diagrams from your models in Enterprise Architect in the OpenModelica Connection Editor, OMEdit, which displays the Blocks' aliases and notes.

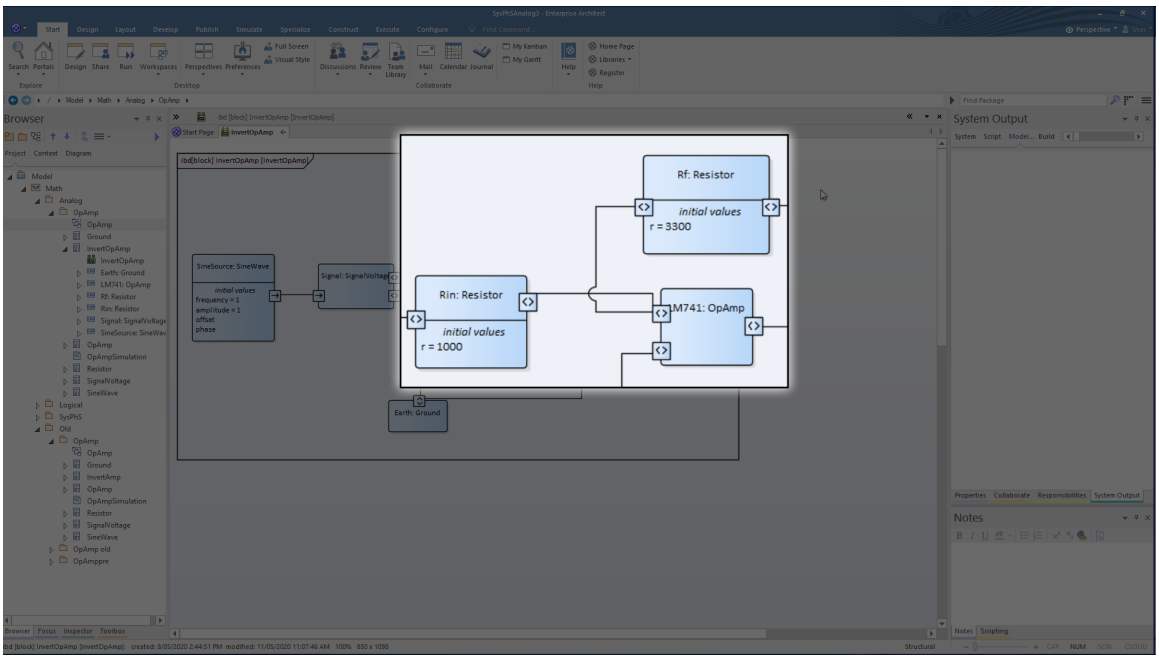
You can create Blocks on the fly using the new SysPhS patterns ready to be simulated in OpenModelica, referencing existing OpenModelica library Blocks or custom user-defined Blocks. With the latest OpenModelica code generation you can view your SysML components in compatible OpenModelica clients such as OMEdit, as well as simulate plots.

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

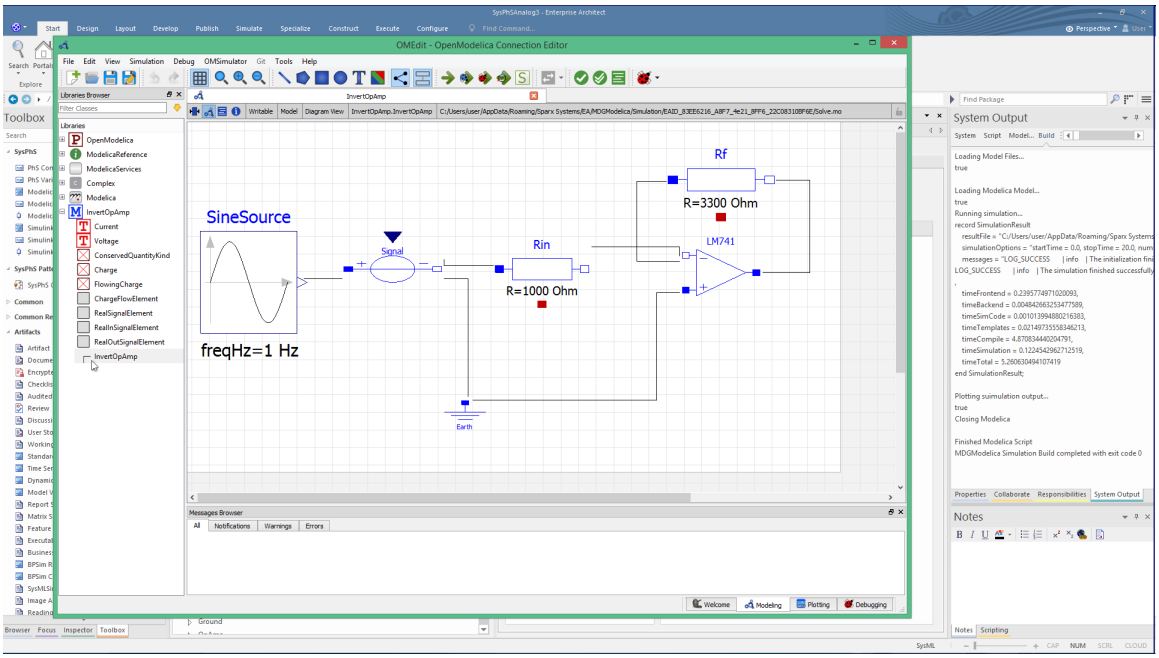
Using OpenModelica is an alternative to using MATLAB Simulink to perform simulation of Parametric models in

Enterprise Architect. You configure your models in either case using the SysPhS Standard, which defines how to translate between a SysML model and either an OpenModelica model or a Simulink/Simscape model.

This is an example of components defined using SysPhS-specific SysML Parts:



The components are as shown in this OpenModelica diagram generated from the SysPhS model:



Installation

Platform	Detail
Windows	If Enterprise Architect is installed on a Windows platform, see the *OpenModelica on Windows* Help Topic.
Linux	If Enterprise Architect is installed on a Linux platform, see the *OpenModelica on Linux* Help Topic.

Application

Using OpenModelica Library	Details on referencing resources available in the OpenModelica Libraries.
Model Analysis using Datasets	Using the Simulation configuration, a Block can be set to have multiple datasets defined against it. This allows for repeatable simulation variations using the same SysML model.

Troubleshooting Simulation	This topic describes possible problems that can arise when using OpenModelica (or MATLAB Simulink) for simulation.

OpenModelica on Windows

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

Install OpenModelica

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

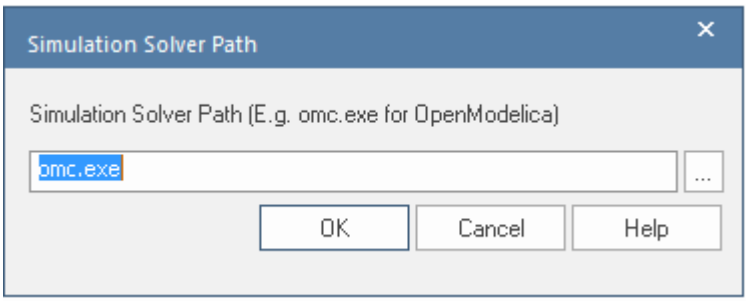
Access

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

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

Configure the Solver

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



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

OpenModelica on Linux

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

This Help topic provides guidance on:

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

Linux Debian / Ubuntu

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

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

This provides the instructions for Debian / Ubuntu Packages.

Run these scripts on a terminal.

Step	Action
1	To add OpenModelica to your additional repository list: for deb in deb deb-src; do echo "$deb http://build.openmodelica.org/apt

	`lsb_release -cs` nightly"; done | sudo tee /etc/apt/sources.list.d/openmodelica.list
2	Import the GPG key used to sign the releases: wget -q http://build.openmodelica.org/apt/openmodelica.asc -O- | sudo apt-key add -
3	Update and install OpenModelica: sudo apt-get update sudo apt-get install openmodelica sudo apt-get install omlib-.* # Installs optional Modelica libraries (most have not been tested with OpenModelica)
4	To check this installation, ensure that you can find the file /usr/bin/omc by, for example, executing this command on the terminal: • ~ $ /usr/bin/omc --version Your installation is successful if the command returns a string resembling this: • OpenModelica 1.13.0~dev-1322-g53a43cf

Linux Mint

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

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

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

Click here for a full list of Linux Mint History and the mappings with Ubuntu.

Step	Action
1	Run this script in a terminal: for deb in deb deb-src; do echo "$deb http://build.openmodelica.org/apt `lsb_release -cs` nightly"; done | sudo tee

	/etc/apt/sources.list.d/openmodelica.list
2	To change the repository URL in Linux Mint: • On the Linux Mint main screen select: 'Menu | Search Bar | Software Sources (type in password) | Additional repositories | Select 'Openmodelica' | Edit URL' • Change the Linux Mint name (for example, *rosa)* to the corresponding Ubuntu name (for example, *trusty*) as in the list at the top of this table; that is: deb http://build.openmodelica.org/apt **rosa** nightly deb http://build.openmodelica.org/apt **trusty** nightly • Click on the OK button
3	• Select 'Openmodelica(Sources)'| Edit URL • Change the Linux Mint name according to the list at the top of the table For example, change the Linux Mint name *rosa* to the corresponding Ubuntu name *trusty*

	• Click on the OK button
4	To update and install OpenModelica, run these scripts in a terminal: sudo apt-get update sudo apt-get install openmodelica sudo apt-get install omlib-.* # Installs optional Modelica libraries (most have not been tested with OpenModelica)

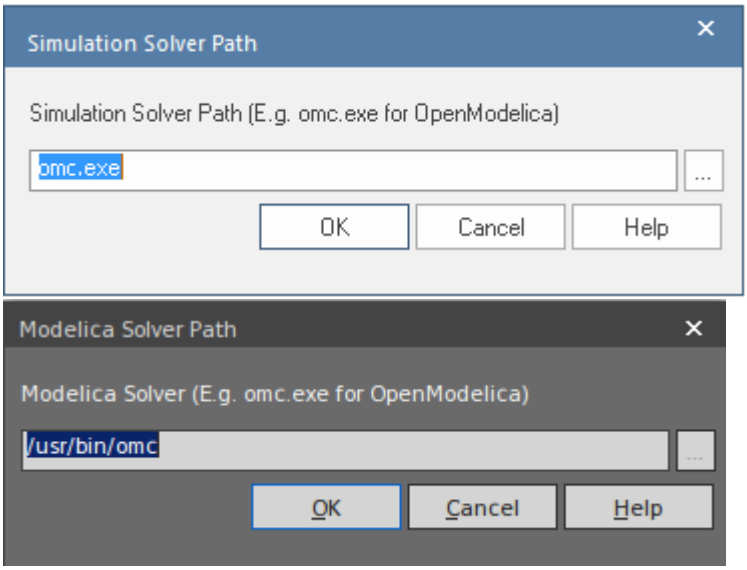
Access

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

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

Configure the Solver

The 'Simulation Solver Path' dialog resembles this:

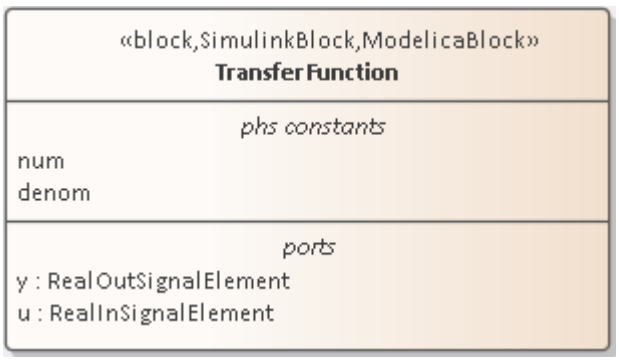


Type in or browse for the path to the solver to use.

SysPhS Simulation

The *SysML Extension for Physical Interaction and Signal Flow Simulation* specification (SysPhS), is an Object Management Group (OMG) specification that extends SysML to provide a common modeling platform for defining consistent models. These models can be translated out to either of two key simulation platforms, Modelica and MATLAB's Simulink/Simscape.

The OMG SysPhS stereotypes help you to define the characteristics of the model simulation within the model itself, rather than in a simulation configuration specification. They provide greater visibility of the type of object or property in the Browser window and Properties window, and in the diagram with specific element compartments for the Property types and for initial values.



The standard is represented in Enterprise Architect by the OMG SysPhS Profile, along with:

- SysPhS Libraries of elements for signal flow and for physical interaction (necessary to perform simulations under the SysPhS standard)
- A dedicated Toolbox page

- A wide range of component element patterns from which to generate common simulation elements such as electronic, logic, and fluid components; the patterns reference the library Blocks in either the OpenModelica or Simulink standard libraries
- Features for Simulating plots using Modelica or MATLAB's Simulink, Simscape and Stateflow.

SysPhS Features

Feature	Description
Referencing SysPhS Libraries	Key resources for working with SysPhS are the SysPhS Simulation Libraries, which include reusable resources that you must reference within your model.
SysPhS Toolbox	The SysPhS pages of the Diagram Toolbox contain basic SysML elements for both OpenModelica and MATLAB Simulink.
SysPhS Patterns	The SysPhS Patterns provide pre-defined SysPhS Blocks that reference equivalent MATLAB and Modelica components. These simple Blocks can be used as starters when working with SysPhS models.

SysPhs Components	SysPhS components enable you to set references to both Modelica and Simulink components.
Simulation	You can define IBD or Parametric models with additional information to drive a simulation, then use the simulation configuration to generate the model in Modelica, Simulink or Simscape, to produce a graph of the results.
SysPhS Examples	There are several examples of using SysPhS for setting up simulations.
Updating SysMLSim for SysPhys	You can update older simulation configurations (pre Enterprise Architect 15.2), to reflect the use of the SysPhS standard.

Options

Extra options for variables and constants, such as isContinuous and isConserved, are automatically set as Tagged Values, again avoiding the need to define them in the configuration specification. These options are also visible on the Block itself and in the docked Properties window.

Videos

- SysPhS Patterns for the Simulation of an Electrical Circuit
- Simulating Digital Electronics using SysPhS and Modelica

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.

Modeling and Simulation with OpenModelica Library

This feature is available from Enterprise Architect release 14.1 onwards.

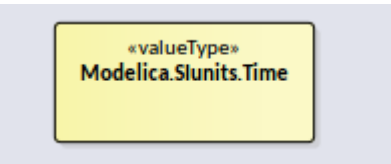
The OpenModelica Libraries are comprehensive resources that provide many useful types, functions and models. When creating SysML models in Enterprise Architect, you can reference resources available in the OpenModelica Libraries.

Referencing a Type Defined in OpenModelica Library

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

Create an element for a Referenced OpenModelica Type

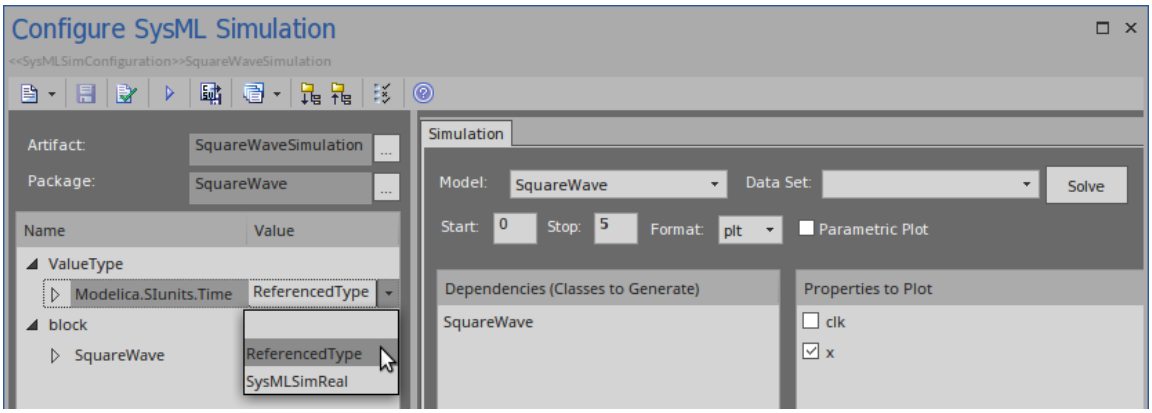
- Create a ValueType element with the full name of the OpenModelica Library path



Configure the ValueType element as 'ReferencedType':

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

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



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

Set the type for a Property to the ValueType element

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

Option 1:

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

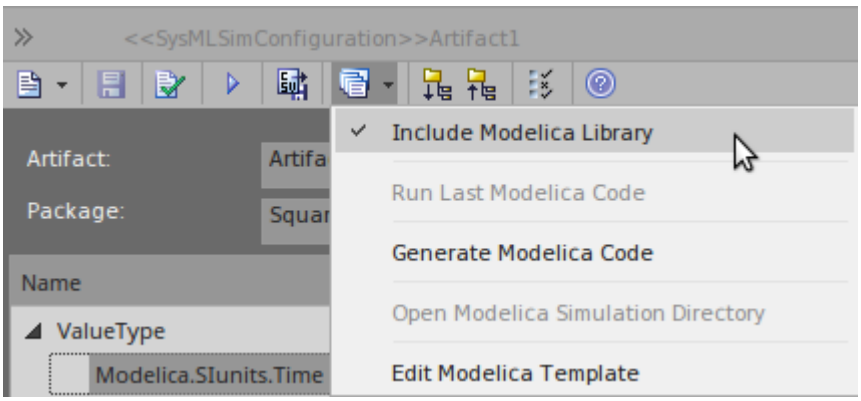
Option 2:

- Select the Property (Part or Port)
- Press Ctrl+L on the Property
- Browse to the ValueType element you created

Including an OpenModelica Library in a Simulation

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

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



- If this option is ticked, this function will be generated to 'Solve.mos' by default:
  loadModel(Modelica);

Click here for a detailed description of the loadModel() scripting function.

Customize the OpenModelica Script Template

You can modify the OpenModelica script template to add extra libraries required by the model and simulation. Select the ribbon option:

Develop > Source Code > Options > Edit Code

Templates

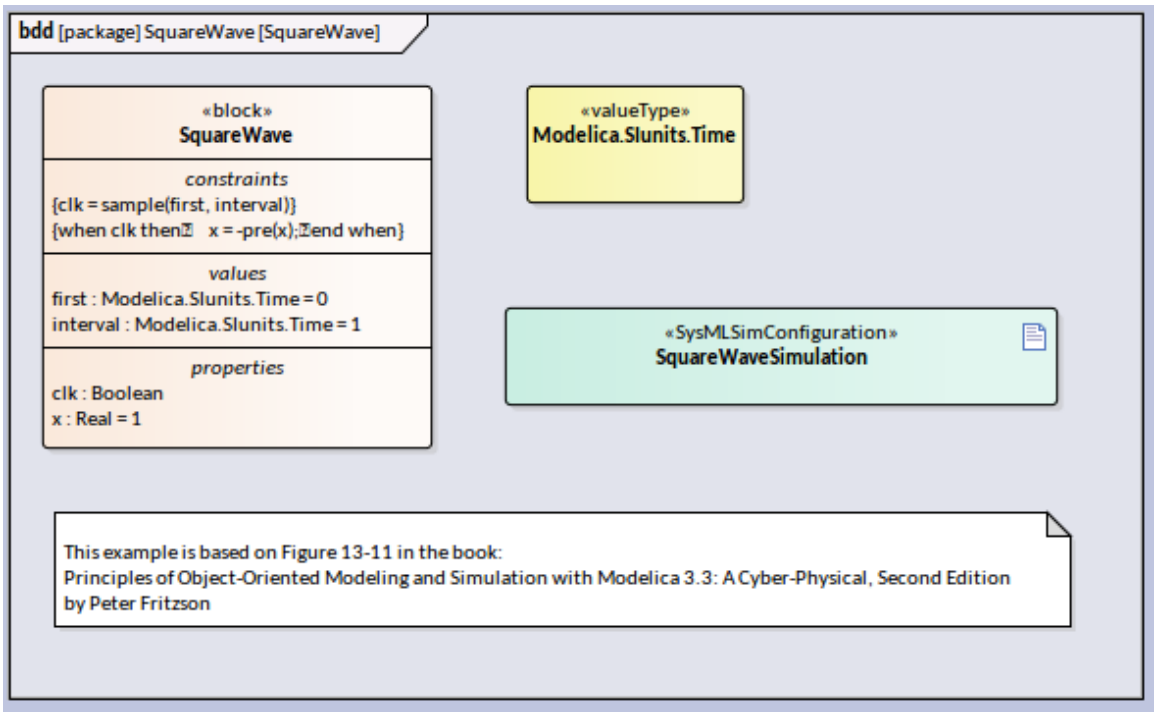
In the 'Language' field select 'Modelica', and in the 'Scripts' list select 'SysMLSim Script'.

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

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

SquareWave Example

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

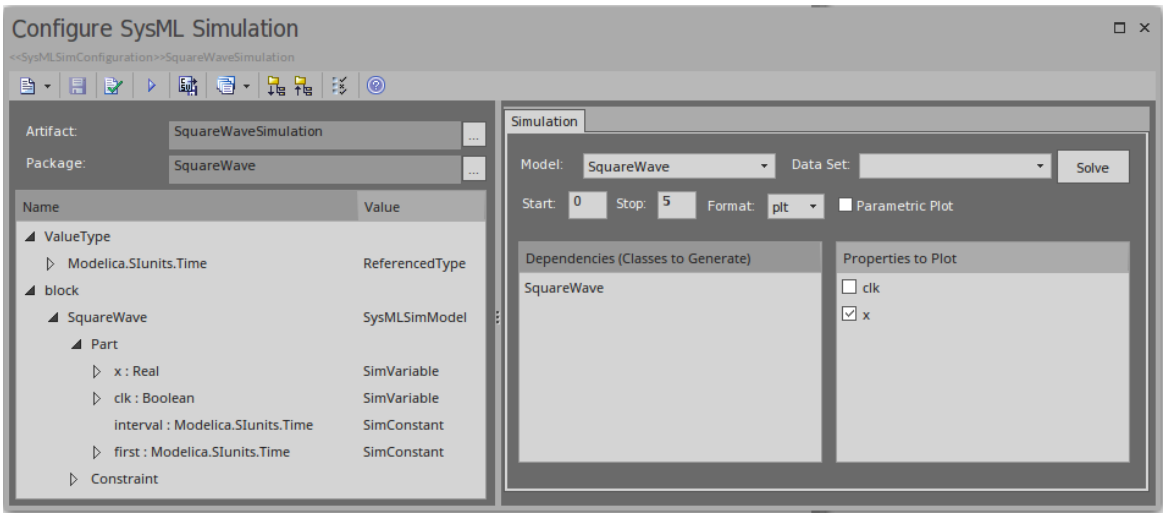


In this example:

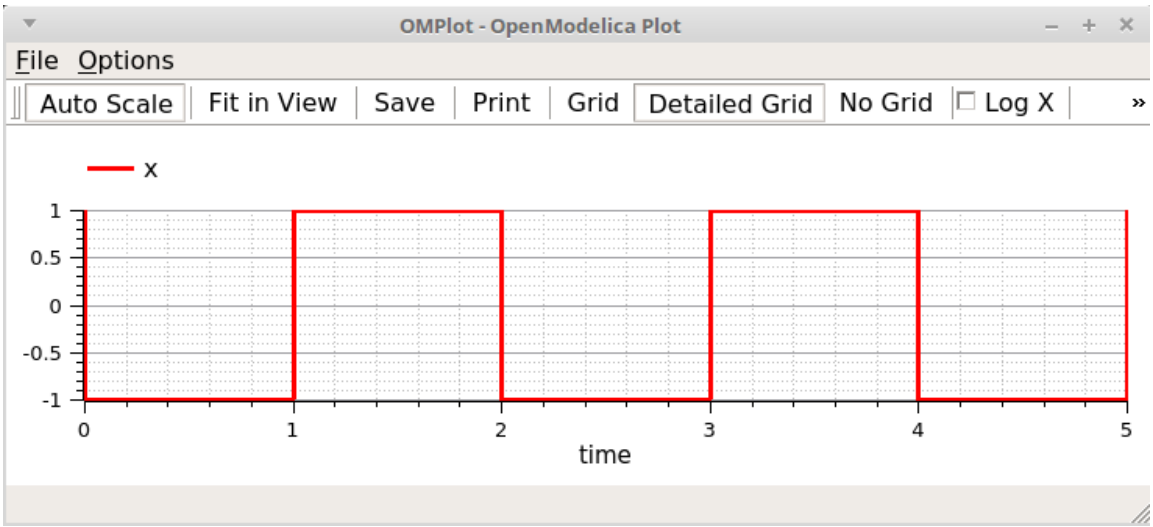
- We create a ValueType *Modelica.SIunits.Time*, which is

used for the properties *first* and *interval* of the Block *SquareWave*

- ValueType *Modelica.SIunits.Time* is configured as 'ReferencedType' in the SysML Simulation window
- Select the menu item 'Include Modelica Library'



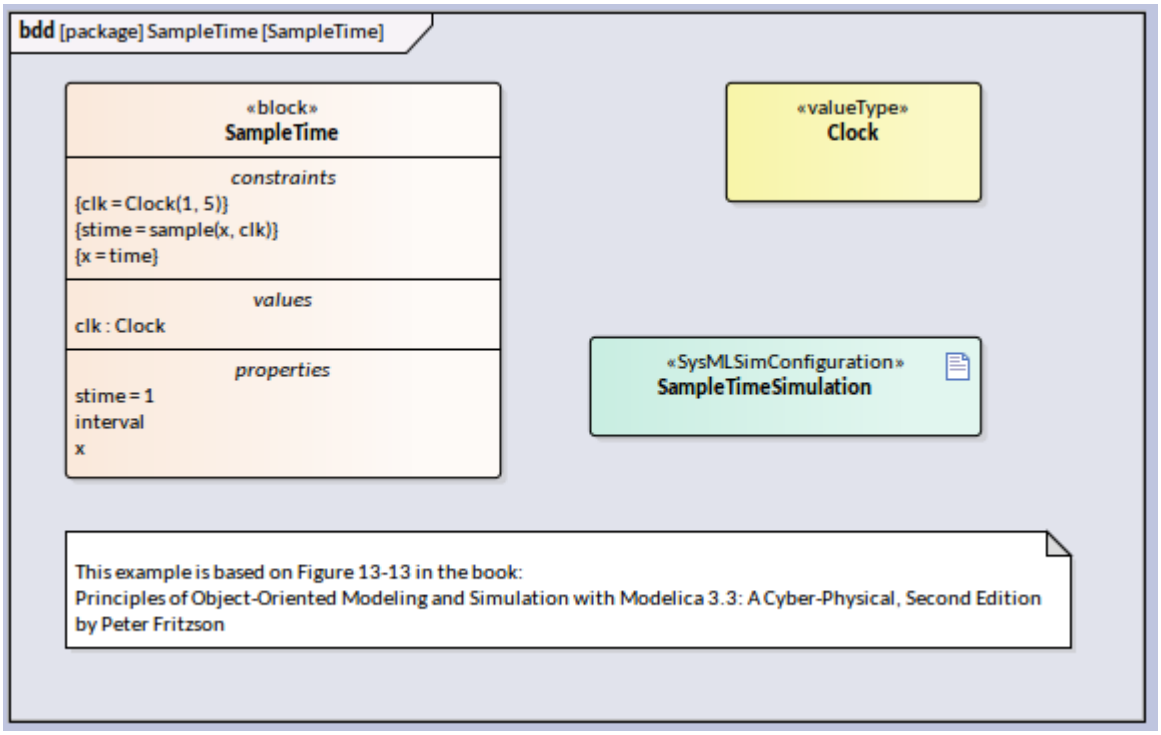
Run the simulation; the variable *x* is plotted like this:



SampleTime Example

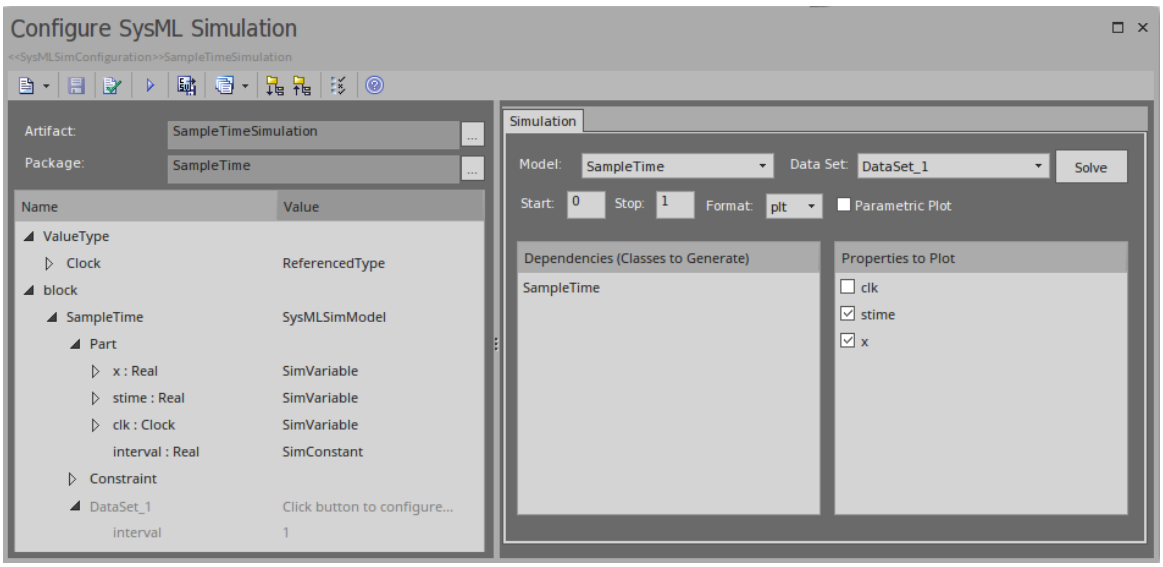
This example is based on Figure 13-13 in: *Principles of*

Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical, Second Edition, by Peter Fritzson.

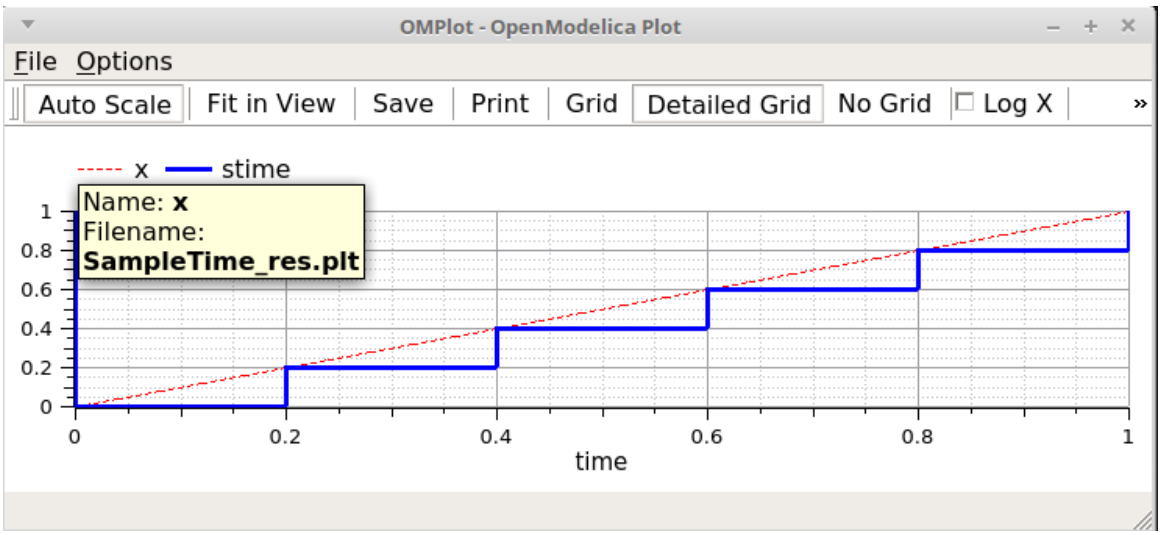


In this example:

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



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



Troubleshooting OpenModelica Simulation

Whilst this topic describes possible problems that can arise when using OpenModelica, many of the points are equally applicable to performing a simulation using MATLAB Simulink.

Common Simulation Issues

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

Issue
The number of equations is less than the number of variables. You might have forgotten to set some properties to 'PhSConstant', which means the value doesn't change during simulation. You might have to provide the 'PhSConstant' property values before the simulation is started. (Set the values through a Simulation Data Set.)
The Blocks that are typing to Ports might not contain

conserved properties. For example, a Block 'ChargePort' contains two parts — 'v : Voltage' and 'i: Current'. The property 'i : Current' should be defined as PhSVariable with the attribute 'isConserved' set to 'True'.
PhSConstants might not have default values — they should be provided with them.
A PhSVariable might not have an initial value to start with — one should be provided.
The properties might be typed by elements (Blocks or Value Type) external to the configured Package; use a Package Import connector to fix this.

SysML Simulation Configuration Filters

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

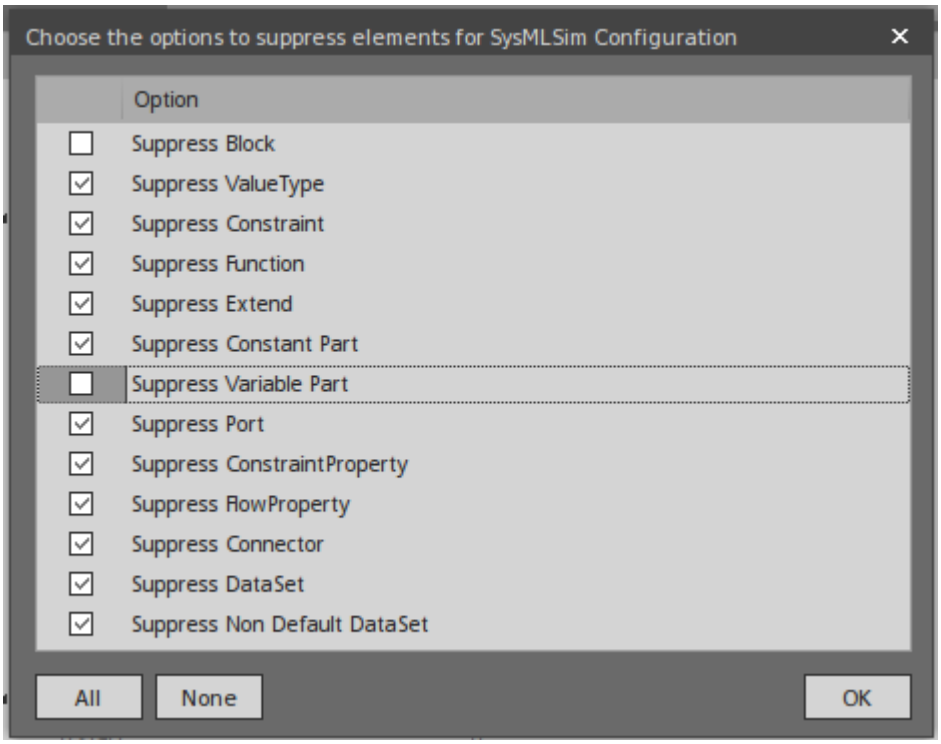
In the TwoTanks example, if we clear the Tank.area property 'PhSConstant' and then do a validation, we will find this error:

Error: Too few equations, under-determined system. The

model has 11 equation(s) and 13 variable(s).

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

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



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

Model Validation Examples

Message	Discussion

Variable Not Defined in Constraint	In the TwoTanks example, when we browse to 'constraintBlock.Outcontrol.Constraint', suppose we find a typing error: we typed 'v' instead of 'b' in the constraint. So, instead of: *a=b*(c+d)* We typed: *a=v*(c+d)* Click on the Validate button on the toolbar. These error messages will appear in the 'Modelica' tab: *Validating model...* *Error: Variable v not found in scope OutControl. (Expression: "* *a=v*(c+d);")* *Error: Error occurred while flattening model TanksConnectedPI* *Number of Errors and Warnings found: 2* Double-click on the error line; the configuration list displays with the constraint highlighted. Change 'v' back to 'b' and click on the Validate button again. No errors are found and the issue is fixed.

	Tips: Using the SysML Simulation Configuration view is a shortcut way of changing the constraints for a Block or Constraint Block. You can: • Change a constraint in place • Delete using the context menu of a constraint • Add a new constraint using the context menu of a Block or Constraint Block
Duplicate Variable Names	In the TwoTanks example, browse to *block.tank.constraintProperty.e1*. Assume that we gave two properties the same name: • Right-click on *e1*, select the 'Find in Project Browser' option, and change the name to *e2;* reload the 'SysML Simulation Configuration' dialog Click on the Validate button on the toolbar; these error messages appear in the 'Modelica' tab: *Validating model...* *Error: Duplicate elements (due to inherited elements) not identical: (Expression: "SensorValue e2;")* *Error: Error occurred while flattening model TanksConnectedPI* *Number of Errors and Warnings*

	found: 2 Double-click on the error line; the configuration list displays with the constraint properties highlighted. Change the name of one of them from *e2* back to *e1* and click on the Validate button again; no errors are found and the issue is fixed.
Properties Defined in ConstraintBlocks Not Used	In the TwoTanks example, in the Browser window, we browse to the element 'Example Model.Systems Engineering.ModelicaExamples.TwoTanks.constraints.OutFlow'. Assume that we add a property '*c*' and potentially a new constraint, but we forget to synchronize for the instances - the ConstraintProperties. This will cause a *Too few equations, under-determined system* error if we don't run validation. Reload the Package in the 'SysML Simulation Configuration' dialog and click on the Validate button on the toolbar. These error messages will appear in the 'Modelica' tab: *Validating model...* *Error: ConstraintProperty 'e4' is missing parameters defined in the typing ConstraintBlock 'OutFlow'. (Missing: c)*

	Error: Too few equations, under-determined system. The model has 11 equation(s) and 12 variable(s). *Number of Errors and Warnings found: 2* Double-click on the error line; the configuration list displays with the ConstraintProperty highlighted. The ConstraintProperty is typed to *outFlow* and the new parameter '*c*' is missing. Right-click on the ConstraintProperty in the configuration list, select the 'Find in all Diagrams' option and right-click on the 'Constraint' property on the diagram; select 'Features | Parts / Properties' and select the 'Show Owned / Inherited' checkbox, then click on 'c'. Reload the model in the 'SysML Simulation Configuration' dialog and click on the Validate button. These error messages will appear in the 'Modelica' tab: *Validating model...* *Error: ConstraintProperty 'e4' does not have any incoming or outgoing binding connectors for parameter 'c'.* *Error: Too few equations,*

	under-determined system. The model has 11 equation(s) and 12 variable(s). *Number of Errors and Warnings found: 2* In order to fix this issue, we can do either of two things based on the real logic: 1. If the property 'c' is necessary in the ConstraintBlock and a constraint is defined by using 'c', then we need to add a property in the context of the ConstraintProperty and bind to the parameter 'c'. 2. If the property 'c' is not required, then we can click on this property in the Constraint Block and press the Ctrl+D keys. (The corresponding ConstraintProperties will have 'c' deleted automatically.)