ENTERPRISE ARCHITECT

User Guide Series

Simulation and Behavior

Author: Sparx Systems & Stephen Maguire
Date: 10/11/2023
Version: 16.1

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Introduction

Enterprise Architect is a platform that provides a unique set of developer tools integrated into a sophisticated development environment. In addition to the standard facilities available to work with static code, there is a wide range of facilities that support the simulation and automatic code generation from the models.

In an era dominated by digital disruption and the need for organizations to be agile and responsive to dynamic business changes, these facilities provide an effective insurance policy that will ensure an organization becomes a true digital enterprise.

Enterprise Architect holds a unique position and, regardless of whether you use Enterprise Architect's standard code engineering features or another Integrated Development Environment such as Eclipse or Visual Studio, these facilities provide productivity gains not matched in other tools.

Model Simulation brings your behavioral models to life with instant, real-time behavioral model execution. Coupled with tools to manage triggers, events, guards, effects, breakpoints and simulation variables, plus the ability to visually track execution at run-time, the Simulator is a valuable means of 'watching the wheels turn' and verifying the correctness of your behavioral models. With Simulation you can explore and test the dynamic behavior of models. In the Corporate, Unified and Ultimate Editions, you can also use JavaScript as a run-time execution language for evaluating guards, effects and other script-able pieces of behavior.

With extensive support for triggers, trigger sets, nested states, concurrency, dynamic effects and other advanced simulation capabilities, Enterprise Architect provides a remarkable environment in which to build interactive and working models that help explore, test and visually trace complex business, software and system behavior. With JavaScript enabled, it is also possible to create embedded COM objects that will do the work of evaluating guards and executing effects - allowing the simulation to be tied into a much larger set of dependent processes. For example, a COM object evaluating a guard condition on a State Transition might query a locally running process, read and use a set of test data, or even connect to an SOA web service to obtain some current information.

As Enterprise Architect uses a dynamic, script driven Simulation mechanism there is no need to generate code or compile your model before running a simulation. It is even possible to update simulation variables in real time using the Simulation console window. This is useful for testing alternative branches and conditions 'on the fly', either at a set Simulation break point or when the Simulation reaches a point of stability (for example, when the Simulation is 'blocked').

In the Professional Edition of Enterprise Architect, you can manually walk through simulations - although no JavaScript will execute - so all choices are manual decisions. This is useful for testing the flow of a behavioral model and highlighting possible choices and processing paths.

Dynamic Simulations

Model Simulation brings your behavioral models to life with instant, real-time behavioral model execution. Coupled with tools to manage triggers, events, guards, effects, breakpoints and Simulation variables, plus the ability to visually track execution at run-time, the Simulator is a multi-featured means of 'watching the wheels turn' and verifying the correctness of your behavioral models. With Simulation you can explore and test the dynamic behavior of models. In the Corporate, Unified and Ultimate Editions, you can also use JavaScript as a run-time execution language for evaluating guards, effects and other scriptable items of behavior.

Extensive support for triggers, trigger sets, nested states, concurrency, dynamic effects and other advanced Simulation capabilities, provides a remarkable environment in which to build interactive and working models that help explore, test and visually trace complex business, software and system behavior. With JavaScript enabled, it is also possible to create embedded COM objects that will do the work of evaluating guards and executing effects - allowing the Simulation to be tied into a much larger set of dependent processes. For example, a COM object evaluating a guard condition on a State Transition might query a locally running process, read and use a set of test data, or even connect to an SOA web service to obtain some current information.

As Enterprise Architect uses a dynamic, script driven Simulation mechanism that analyzes and works with UML constructs directly, there is no need to generate intermediary code or compile simulation 'executables' before running a Simulation. This results in a very rapid and dynamic Simulation environment in which changes can be made and tested quickly. It is even possible to update Simulation variables in real time using the Simulation Console window. This is useful for testing alternative branches and conditions 'on the fly', either at a set Simulation break point or when the Simulation reaches a point of stability (for example, when the Simulation is 'blocked').

In the Professional Edition of Enterprise Architect, you can manually walk through Simulations - although no JavaScript will execute - so all choices are manual decisions. This is useful for testing the flow of a behavioral model and highlighting possible choices and processing paths. In the Corporate, Unified and Ultimate Editions it is possible to:

- Dynamically execute your behavioral models
- Assess guards and effects written in standard JavaScript
- Define and fire triggers into running Simulations
- Define and use sets of triggers to simulate different event sequences
- Auto-fire trigger sets to simulate complex event histories without user intervention
- Update Simulation variables 'on the fly' to change how Simulations proceed
- Create and call COM objects during a Simulation to extend the Simulation's reach and input/output possibilities
- Inspect Simulation variables at run time
- Set a script 'prologue' for defining variables, constants and functions prior to execution
- Use multiple Analyzer Scripts with differing 'prologues' for running the Simulation under a wide range of conditions

In the Unified and Ultimate Editions it is also possible to simulate BPMN models.

Using the Model Simulator, you can simulate the execution of conceptual model designs containing behavior. When you start a Simulation, the current model Package is analyzed and a dynamic Simulation process is triggered to execute the model.

To get up and running with Simulation, the only steps required are:

- Build a behavioral diagram (State or Activity for manual or dynamic execution, Sequence for manual interaction only)
- Optional: load the 'Simulation Workspace' layout - a fast way of bringing up all the frequently used Simulation windows
- Click on the Simulator Play button

If the diagram contains any external elements (those not in the same Package as the diagram) you will have to create an Import connector from the diagram's Package to the Package containing the external elements. To do this, drag both Packages from the Browser window onto a diagram and then use the Quick Linker arrow to create the connector between them.

Simulation Overview

Aspect
Overview of the Model Simulator
Use of the Simulation Window and Related Windows, and Running a Simulation
Set Up a Simulation and Activate a Simulation Script
Set Up and Use Simulation Breakpoints
Simulate the Use of Objects
The Use of Different Types of Action in Simulation
Perform Dynamic Simulation with JavaScript
The Use of Guards and Effects in Simulations
The Use of Triggers in Simulations
Call Behaviors and Variables
Simulating Activity Returns
Simulating Structured Activity Behavior
Simulating Multi-Threaded Processes
Simulating Sub-Processes in Separate Diagrams
Performing BPMN Simulations
Simulate Win32 Dialog Behavior

Platforms and Available Editions

Platform/Edition	Details
Models and Platforms Supported	The Model Simulator currently supports the execution of UML Activity, Interaction and StateMachine models and BPMN Business Processes on the Simulation platforms: • UML Basic • BPMN
Edition Support	Model Simulation is available at different levels across the range of editions of

	Enterprise Architect: • Professional - Manual Simulation only • Corporate and above - Adds dynamic JavaScript evaluation; currently JavaScript is enabled for StateMachines and Activity graphs; it is not enabled for Interaction diagrams • Unified and Ultimate - Adds BPMN Simulation

How it Looks

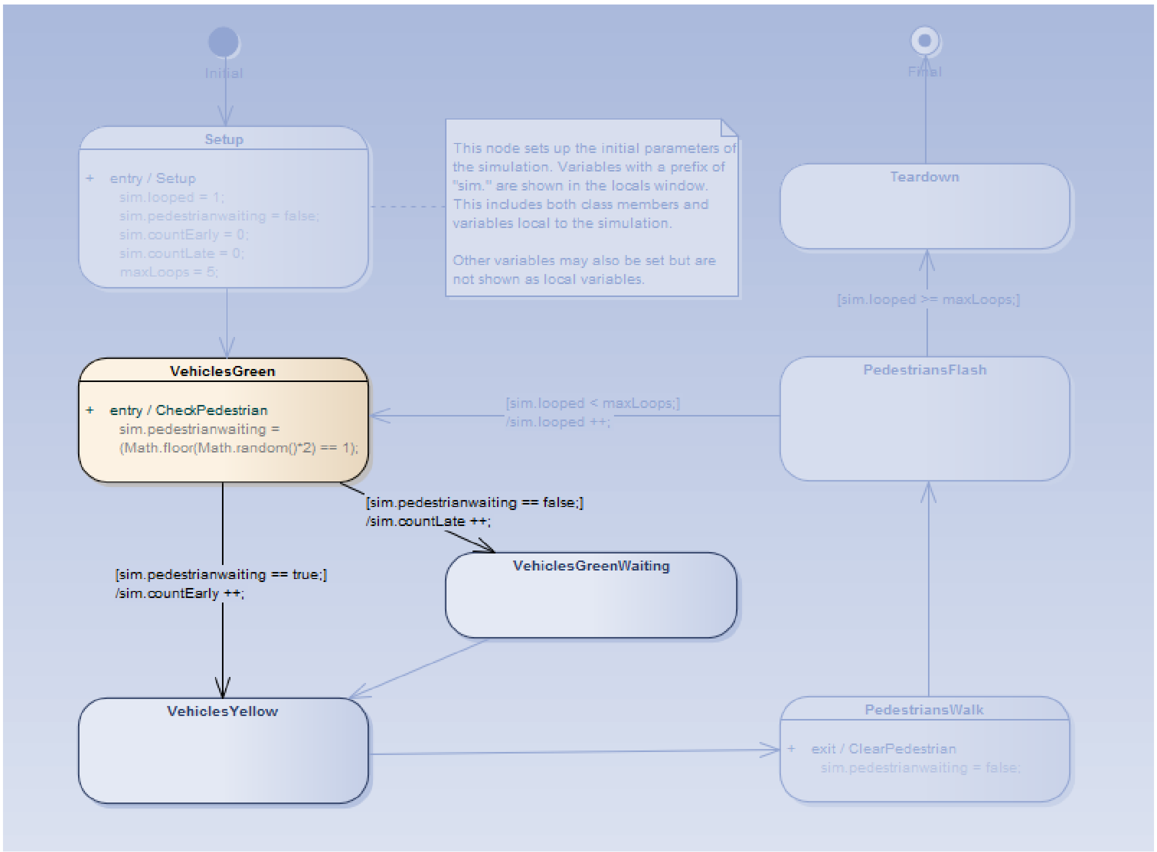
Enterprise Architect has a special way of displaying model information during Simulation. This helps focus attention on the executing or active nodes.

During a Simulation, Enterprise Architect will dynamically track and highlight the active nodes within your model. If a node in another diagram is activated, that diagram will be automatically loaded and the current node highlighted. It is possible to modify the diagram while the Simulation is running; however, the changes made are not recognized until the current Simulation is ended and a new one begun.

Highlighting of the Active Node(s) During Simulation

In this example, the currently active node (VehiclesGreen) is highlighted in normal Enterprise Architect colors, and all possible transitions out of the current node are rendered at full strength.

The elements that are possible targets of the current active node's outgoing transitions are rendered in a semi-faded style so that they are readable and clearly different to the other elements within the diagram. All other elements are rendered in a fully faded style to show they are not targets of the next Simulation step. As the Simulation progresses (especially if automatically run), this highlighting helps focus the attention on the current item and its visual context.



This node sets up the initial parameters of the simulation. Variables with a prefix of "sim." are shown in the locals window. This includes both class members and variables local to the simulation.

Other variables may also be set but are not shown as local variables.

Simulation Windows

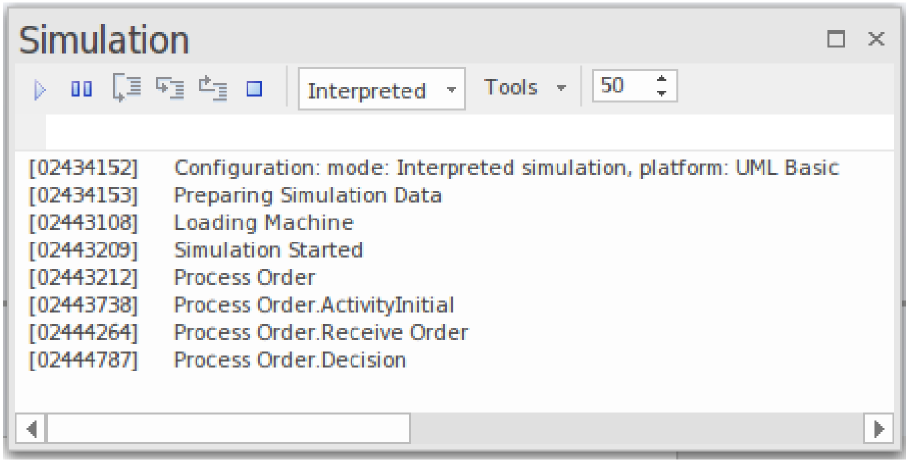
When executing a Simulation in Enterprise Architect it is possible to set break-points, fire triggers, examine variables, record a trace of execution, set Simulation speed, view the Call Stack and visually trace the active nodes as the Simulation proceeds.

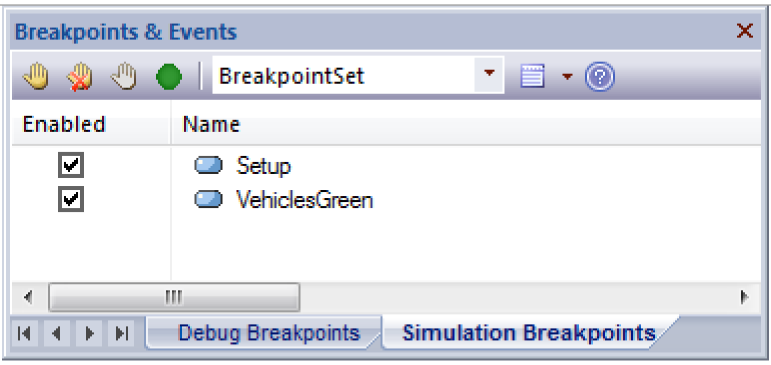
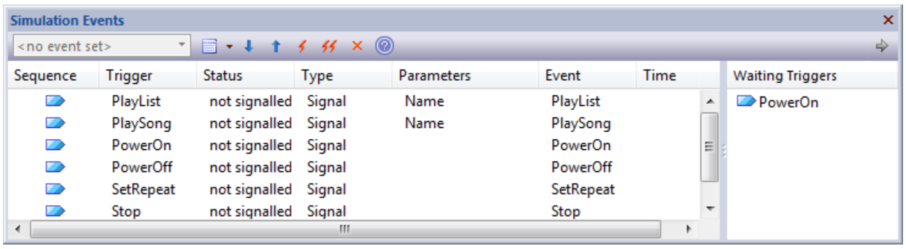
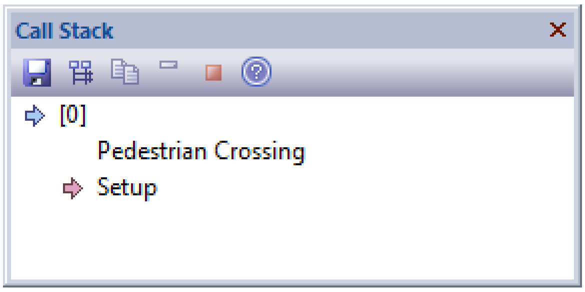
When a Simulation runs, some aspects such as the output and console input are found in the Simulation window itself, while other aspects such as the local variables and Call Stack use the standard Execution Analyzer windows. The topic provides an overview of the main windows used during Simulation.

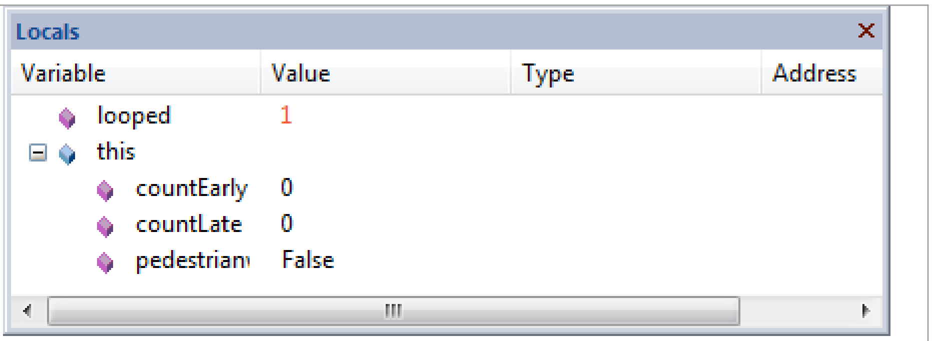
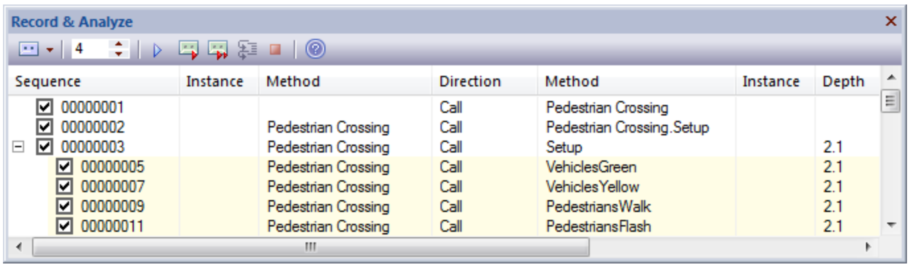
Access

Ribbon	Simulate > Dynamic Simulation > Simulator > Open Simulation Window

Windows

Window	Purpose
Execution and Console	The Simulation window provides the main interface for starting, stopping and stepping through your Simulation. During execution it displays output relating to the currently executing step and other important information. See the *Run Model Simulation* topic for more information on the toolbar commands. Note the text entry box just underneath the toolbar. This is the Console input area - here you can type simple JavaScript commands such as: *this.count = 4;* to dynamically change a simulation variable named 'count' to 4. In this way you can dynamically influence simulation at run-time.  [02434152] Configuration: mode: Interpreted simulation, platform: UML Basic [02434153] Preparing Simulation Data [02443108] Loading Machine [02443209] Simulation Started [02443212] Process Order [02443738] Process Order.ActivityInitial [02444264] Process Order.Receive Order [02444787] Process Order.Decision
Breakpoints & Events Window	The Simulation process also makes use of the 'Simulation Breakpoints' tab of the Breakpoints & Markers window ('Simulate > Dynamic Simulation > Breakpoints'). Here you set execution breakpoints on specific elements and messages in a Simulation. See the *Simulation Breakpoints* topic for more details.

<table>
<tr>
<td></td>
<td>

Breakpoints &amp; Events
BreakpointSet
<table>
<tr><th>Enabled</th><th>Name</th></tr>
<tr><td>☑</td><td>Setup</td></tr>
<tr><td>☑</td><td>VehiclesGreen</td></tr>
</table>
Debug Breakpoints | Simulation Breakpoints
</td>
</tr>
<tr>
<td>Simulation Events Window</td>
<td>
The Simulation Events window ('Simulate > Dynamic Simulation > Events') provides tools to manage and execute triggers. Triggers are used to control the execution of StateMachine transitions.

Simulation Events
&lt;no event set&gt;
<table>
<tr><th>Sequence</th><th>Trigger</th><th>Status</th><th>Type</th><th>Parameters</th><th>Event</th><th>Time</th></tr>
<tr><td></td><td>PlayList</td><td>not signalled</td><td>Signal</td><td>Name</td><td>PlayList</td><td></td></tr>
<tr><td></td><td>PlaySong</td><td>not signalled</td><td>Signal</td><td>Name</td><td>PlaySong</td><td></td></tr>
<tr><td></td><td>PowerOn</td><td>not signalled</td><td>Signal</td><td></td><td>PowerOn</td><td></td></tr>
<tr><td></td><td>PowerOff</td><td>not signalled</td><td>Signal</td><td></td><td>PowerOff</td><td></td></tr>
<tr><td></td><td>SetRepeat</td><td>not signalled</td><td>Signal</td><td></td><td>SetRepeat</td><td></td></tr>
<tr><td></td><td>Stop</td><td>not signalled</td><td>Signal</td><td></td><td>Stop</td><td></td></tr>
</table>
Waiting Triggers
PowerOn
</td>
</tr>
<tr>
<td>Call Stack Window</td>
<td>
During the Simulation the Call Stack window ('Simulate > Dynamic Simulation > Call Stack') displays information about the threads and current execution context of the Simulation.
The Simulator supports multi-threaded Simulations and will include a thread entry for every active and paused thread of execution. For each thread, the Call Stack window will show the start or entry context (such as a StateMachine element) plus the current active element within that thread. If the current active element is the entry point of a composite Activity or SubMachine state, the Stack will also include the current active element within that sub-context (and all further nested, active composite, sub-states as well).

Call Stack
[0]
Pedestrian Crossing
Setup
</td>
</tr>
<tr>
<td>Simulation Local Variable Window</td>
<td>The Simulator uses the standard Locals window ('Simulate > Dynamic Simulation > Local Variables') to show all current simulation variables when the simulation is single stepping or paused at a break point. Note that it is possible to dynamically update these variables using the Simulator Console.</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>

Locals
<table>
<tr><th>Variable</th><th>Value</th><th>Type</th><th>Address</th></tr>
<tr><td>looped</td><td>1</td><td></td><td></td></tr>
<tr><td>this</td><td></td><td></td><td></td></tr>
<tr><td>countEarly</td><td>0</td><td></td><td></td></tr>
<tr><td>countLate</td><td>0</td><td></td><td></td></tr>
<tr><td>pedestrian</td><td>False</td><td></td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>Recording</td>
<td>
During execution of your simulation, a recording is kept of all activity and displayed in the Record & Analyze window ('Execute > Tools > Recorder > Open Recorder'). This is similar to how the normal call recording works in the Visual Execution Analyzer.

Record & Analyze
<table>
<tr><th>Sequence</th><th>Instance</th><th>Method</th><th>Direction</th><th>Method</th><th>Instance</th><th>Depth</th></tr>
<tr><td>☑ 00000001</td><td></td><td></td><td>Call</td><td>Pedestrian Crossing</td><td></td><td></td></tr>
<tr><td>☑ 00000002</td><td></td><td>Pedestrian Crossing</td><td>Call</td><td>Pedestrian Crossing.Setup</td><td></td><td></td></tr>
<tr><td>☑ 00000003</td><td></td><td>Pedestrian Crossing</td><td>Call</td><td>Setup</td><td></td><td>2.1</td></tr>
<tr><td>☑ 00000005</td><td></td><td>Pedestrian Crossing</td><td>Call</td><td>VehiclesGreen</td><td></td><td>2.1</td></tr>
<tr><td>☑ 00000007</td><td></td><td>Pedestrian Crossing</td><td>Call</td><td>VehiclesYellow</td><td></td><td>2.1</td></tr>
<tr><td>☑ 00000009</td><td></td><td>Pedestrian Crossing</td><td>Call</td><td>PedestriansWalk</td><td></td><td>2.1</td></tr>
<tr><td>☑ 00000011</td><td></td><td>Pedestrian Crossing</td><td>Call</td><td>PedestriansFlash</td><td></td><td>2.1</td></tr>
</table>
</td>
</tr>
</table>

Set Up Simulation Script

You can use Simulation Scripts to provide fine control over how a Simulation starts. In general, you do not need to set up a Simulation Script unless:

- You want to run an interpreted Simulation that requires variables to be initialized before the Simulation commences; this is useful for setting up global variables and defining functions
- (In the Corporate Edition and above) You do not want to apply the default behavior of interpreting the Guards (that is, you prefer to use a manual execution), or
- You want to have multiple ways of running the same diagram

For most diagrams it is possible to initialize a script for a Simulation simply by setting variables in the first element or connector after the Start element. For State Charts, this is the Transit connector exiting the initial element, and for Activity models this is the first Action element.

As an alternative, you can use Simulation Scripts to initialize settings before a Simulation starts. This is useful for setting up different sets of initial values using multiple Analyzer Scripts, so that you can run your Simulation under a range of pre-set conditions.

To configure a Simulation Script, first select the Package in the Browser window, Package Browser, Diagram List or Model Search. You can then use the Execution Analyzer window to add a new script for that selected Package. You will use the 'Simulation' page of the 'Execution Analyzer' dialog to configure the relevant properties.

Access

Show the Execution Analyzer window using one of the methods outlined here.

On the Execution Analyzer window, either:

- Locate and double-click on the required script and select the 'Simulation' page or
- Click on in the window Toolbar and select the 'Simulation' page

Ribbon	Develop > Source Code > Execution Analyzer > Edit Analyzer Scripts Execute > Tools > Analyzer
Context Menu	Browser window | Right-click on Package | Execution Analyzer
Keyboard Shortcuts	Shift+F12

Configure a Simulation Script

Option	Action
Platform	For UML Activity, Interaction or StateMachine simulation, click on the drop-down arrow and select 'UML Basic'. For BPMN diagrams, click on the drop-down arrow and select 'BPMN'.
Entry Point	Click on the ... button and select the: • Entry point for the Simulation, and • Activity, Interaction or StateMachine to simulate

	If you do not specify an entry point, the Simulator attempts to work through the entire Package.
Evaluate Guards and Effects using JavaScript	(In Corporate and higher editions) Leave the checkbox unselected to perform a manual Simulation, where you select the next State to transition to and the point where a decision must be made. Select the checkbox to execute the code for Effect behavior in the Simulation. The Simulation executes JavaScript code in these places: • State entry/exit/do operations • Transition guard/effect • BPMN Activity Loop Conditions and Sequence Flow Condition Expressions With the exception of the guard, all of these should be one or more valid JavaScript statements, including the semi-colon. The guard must be a valid Boolean expression, also terminated with a semi-colon. Variables that are members of 'sim' or 'this' are listed in the Locals window when a Simulation breakpoint is reached. sim.count = 0;
Input	When JavaScript is enabled, you can type script commands in this field that will execute prior to the Simulation being run.
Post Processing Script	Using a Post Simulation Script, you can run JavaScript after the Simulation ends. Type in the qualified name of a script from the model script control. For example, if you have a script named 'MyScript' in the Script Group 'MyGroup', type in the value 'MyGroup.MyScript'.
OK	Click on this button to save your changes.

Notes

- Usually all Simulation elements and relationships reside within the Package configured for Simulation; however, you can simulate diagrams that include elements from different Packages, by creating Package Import connectors from the configured Package to each 'external' Package (alternatively, for a BPSim model, create a Dependency connector from the configured Package to each external **element**)

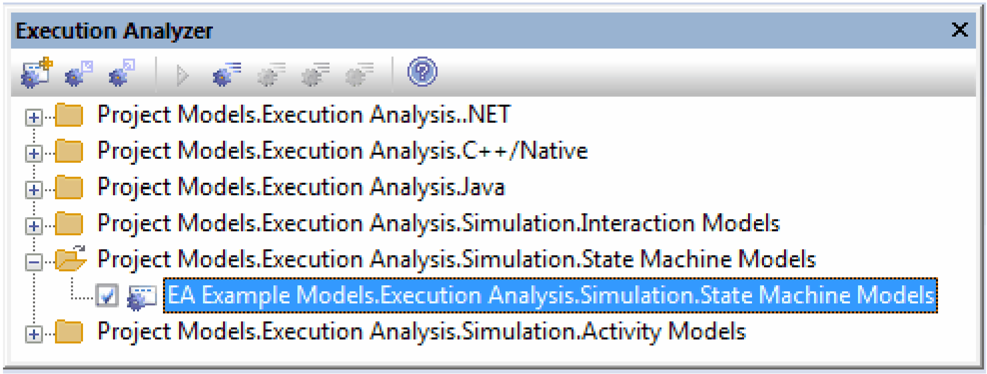
Activate Simulation Script

An Execution Script is configured for a model Package defining the Simulation parameters. The most common reason for activating an Execution Script is when multiple Simulation Scripts are configured against a Package and you want to run a specific one.

Access

Ribbon	Execute > Tools > Analyzer Develop > Source Code > Execution Analyzer
Analyzer Window	Click the Analyzer Script checkbox to make it active
Keyboard Shortcuts	Shift+F12

Activate a Simulation Script for Execution

Step	Action
1	In the Execution Analyzer window, select the required Execution Script. This makes it the current default for your open model, so that clicking on the Simulation Run button will automatically invoke this Simulation Script.  Execution Analyzer Project Models.Execution Analysis..NET Project Models.Execution Analysis.C++/Native Project Models.Execution Analysis.Java Project Models.Execution Analysis.Simulation.Interaction Models Project Models.Execution Analysis.Simulation.State Machine Models ☑ EA Example Models.Execution Analysis.Simulation.State Machine Models Project Models.Execution Analysis.Simulation.Activity Models
2	Click on the checkbox to the left of the script to activate it.
3	Select the 'Simulate > Simulator > Open Simulation Window' ribbon option to execute the simulation.

Run Model Simulation

A Simulation executes the model step-by-step, enabling you to validate the logic of your behavioral model. The current execution step is automatically highlighted in the model's diagram to make it easy to understand the various processes and state changes as they occur during the Simulation.

There are several ways to start a model Simulation:

- When the active diagram can be simulated, the Run button on the main Simulation window will process the current diagram, either by running an existing script or defining a new temporary one
- When the active diagram can not be simulated, the Run button on the main Simulation window will run the Simulation for the active Execution Analyzer script
- By right-clicking on a Simulation script in the Execution Analyzer window and selecting the 'Start Simulation' option
- By right-clicking on a suitable diagram and selecting one of the 'Execute Simulation' options

There are visual cues during execution. When the Simulation is running, Enterprise Architect will actively highlight each active node for each executed step. In addition, all outgoing transitions and control flows will be highlighted, showing the possible paths forward. Elements at the end of possible paths forward will be de-emphasized to half-strength and any other remaining elements will be 90% 'grayed out'. This provides a very dynamic and easy to follow execution that continually refocuses attention on the execution context.

Access

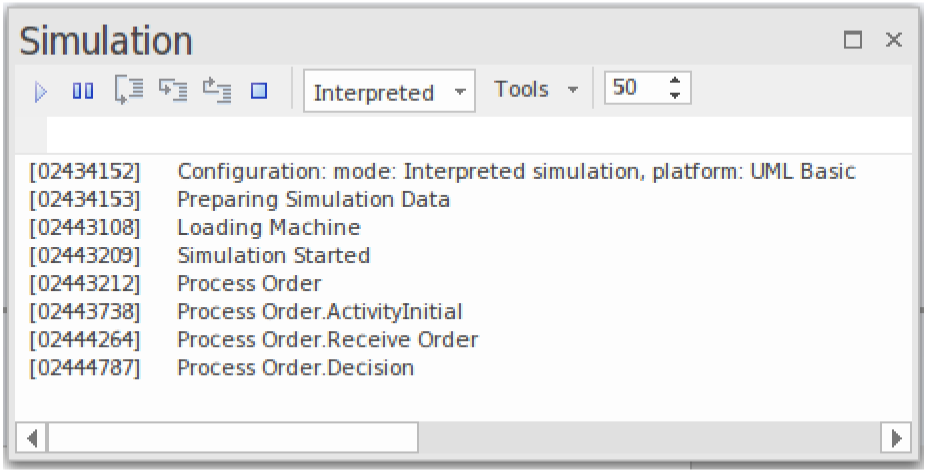
Ribbon	Simulate > Dynamic Simulation > Simulator > Open Simulation Window Simulate > Run Simulation > Start

Edition Specific Details

In the Professional Edition, if a branch is encountered in the execution, the Simulator prompts you to choose the appropriate path to take in your execution.

In the Corporate, Unified and Ultimate Editions, in which JavaScript is enabled, the Simulation will automatically evaluate all guards and effects and dynamically execute the Simulation without user intervention. If the Simulation becomes blocked due to no possible paths forward evaluating to True (or multiple paths evaluating to True) you can modify the Simulation variables on the fly using the console input of the Simulation Execution window.

Run a Simulation Using the Toolbar



```
[02434152]    Configuration: mode: Interpreted simulation, platform: UML Basic
[02434153]    Preparing Simulation Data
[02443108]    Loading Machine
[02443209]    Simulation Started
[02443212]    Process Order
[02443738]    Process Order.ActivityInitial
[02444264]    Process Order.Receive Order
[02444787]    Process Order.Decision
```

Icon	Action
▷	Start the Simulator for the current diagram or, if the current diagram cannot be simulated, run the Simulation using the activated Simulation script.
⏸	Pause the Simulation.
	When the Simulation is paused, step over, step in and step out to control the Simulator's execution at the required step in the model Simulation.
■	Stop the Simulation.
Manual ▾ Manual Interpreted Executable	Click on the drop-down arrow and select the type of Simulation to run: • 'Interpreted' - Perform dynamic execution of a Simulation (Corporate, Unified and Ultimate Editions) • 'Manual' - Step through a Simulation manually (the only option available in the Professional Edition) • 'Executable' - Select when running the Simulation on an Executable StateMachine
Tools ▾	Click on the drop-down arrow and select from a menu of options for performing specific operations on the Simulation script and output, such as Build, Run, Generate and View Breakpoints.
60 ⇕	Vary the execution rate of the Simulation, between 0% and 100%; at: • 100%, the Simulation executes at the fastest possible rate • 0% the Simulator breaks execution at every statement

Notes

- The Simulation tool only becomes active when a valid Simulation Execution Script is activated
- You can set a Simulation script as the current default by setting its checkbox in the Execution Analyzer window

Simulation Breakpoints

The 'Simulation Breakpoints' tab of the Breakpoints & Events window enables you to interrupt and inspect the Simulation process.

When dynamically executing a Simulation (in the Corporate, Unified and Ultimate editions) the process will proceed automatically - if you want to stop execution at some point to examine variables, inspect Call Stacks or otherwise interact with the Simulator, you can set a breakpoint on a model element in much the same way as you would with a line of source code. When the Simulator reaches the breakpoint, execution is halted and control returned to Enterprise Architect.

Access

Ribbon	Simulate > Dynamic Simulation > Breakpoints > Simulation Breakpoints

Breakpoints

The Simulation executes the model step-by-step, enabling you to validate the logic of your behavior model; the Simulation halts when it reaches an element defined as a breakpoint.

The UML elements that can be defined as breakpoints include Actions, Activities, States, and most other behavioral nodes such as Decision, Initial, or Final.

The UML relationships that can be defined as breakpoints include Interaction Messages.

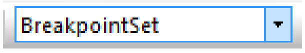
The breakpoints are stored as Breakpoint Sets for a given Enterprise Architect project.

Elements that are included in a Simulation and that have breakpoints are marked by a green circle near the top left corner of the element, whilst the Simulation is in progress. If the Simulation is not running, the green circles are not displayed.

When JavaScript is enabled all Simulation variables will be displayed in the Locals window, and it is possible to modify these Simulation variables using the Simulation window's console input field (underneath the Toolbar).

Toolbar Buttons

Item	Description
✋	Enables all breakpoints defined in the current Breakpoint Set for the Simulation session.
✋	Deletes all breakpoints defined in the current Breakpoint Set for the Simulation session.
✋	Disables all breakpoints defined in the current Breakpoint Set for the Simulation session.
●	Adds a breakpoint for the selected element or Sequence message to the current Breakpoint Set.
	Changes the selected Breakpoint Set for use in the Simulation session.

 BreakpointSet	
	Performs Breakpoint Set commands: • **New Set**: Create a new Breakpoint Set • **Save As Set**: Saves the current Breakpoint Set under a new name • **Delete Selected Set**: Deletes the current Breakpoint Set • **Delete All Sets**: Deletes all Breakpoint Sets saved for the diagram

Objects and Instances in Simulation

As a given business, system or mechanical process executes, the Activities and Actions within it might generate objects of a specific type and perform operations on those objects, perhaps even consuming or destroying them. You can simulate the creation, use and consumption of such objects using a Simulation model that represents the objects and actions with model elements such as Classes, Instance Objects, attributes, operations and Ports (ActionPins and ObjectNodes). The model can also create, act on and destroy several different objects at different stages as part of the same process. Representing model data or objects in Simulation makes the Simulation more accurately reflect the real process.

Object Concepts

Term	Description
SimType	The type of Simulation element, such as Class, Enumeration or Interface. These can be classifiers of objects in a Simulation.
SimObject	An object that is an instance of (is classified by) a SimType element.
Attribute	A property of a SimType element, or of a specified node such as an ActivityNode.
Operation	A behavior of a SimType element, or of a specified node such as an ActivityNode.
Port	A Port of a Class or Object, an ActionPin of an Action, or an ObjectNode of an Activity. Ports of classifiers are a type, whilst a Port of an object is a realization of the type.
Parameter/ Activity Parameter	Parameters of Operations; Activity Parameters are, specifically, parameters of ActivityNodes.
Slot	A realization of an attribute in an object. A Slot has a run time value that can be initialized by the run state value of the Slot. If these values don't exist, the system uses the initial values of the attributes.
Runtime Environment	All objects exist in the JavaScript runtime environment, so you can use JavaScript to create or change simulation objects and simulation variables.
Display Variables	All simulation objects, Simulation variables or events are identified on the Locals window while they are in effect. In some cases, to show the variables you might need to add break points to the model to pause processing while the variable exists. As all objects and variables are shown, global variables that exist outside the simulation but that are significant to it - such as the parent Class and Activity elements within which a process is defined - are automatically also represented as default object variables. So too is the anticipated output of the Activity, as a return variable.

Create Objects in a Simulation

In a Simulation model, you can create Classes and either create instances of them (Global Objects) to represent objects that exist in the process, or define Actions to generate one or more Objects at any point during the process.

You have three options for creating Objects in a Simulation model:

- Manually create the Object
- Dynamically create an Object through a CreateObject Action element
- Use the JavaScript function sim.CreateObject ("name") as the 'Effect' of an Action element, to again create an Object dynamically

Having created an Object dynamically you can also instantiate any inner objects of that Object, such as an Activity on a Class, and act on the properties of that inner object.

Manually Create an Object

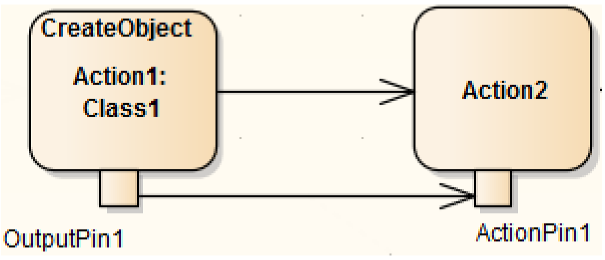
Simply create an Object element on a diagram in the model, either by:

- Dragging an Object element from the 'Object' pages of the Diagram Toolbox and setting its classifier, or
- Dragging a classifier element from the Browser window and pasting it into the diagram as an instance

In the Simulation model you can then set up the Object properties themselves (such as setting run-states to re-set the initial value of an attribute) or the behaviors of Actions to act on the Object (such as passing it along a process flow) and observe what happens to the Object in a Simulation.

Create an Object through a CreateObject Action

If your process generates objects in runtime, you can Simulate this using a CreateObject Action.

Step	Action
1	On your Activity diagram, drag an 'Action' icon from the Diagram Toolbox, and select the 'Other | CreateObject' context menu option to define it as a CreateObject Action element.
2	Set the classifier of the CreateObject Action to the Class of which the Object will be an instance. This is set in the Properties window > CreateObjectAction > Classifier, using the [...] button.
3	Create an Action Pin on the CreateObject Action, of kind output.
4	Create or select the next Action in the processing sequence, and add an Action Pin of kind input. Connect the two Actions with a Control Flow connector, and the two Action Pins with an Object Flow connector.  CreateObject Action1: Class1 Action2 OutputPin1 ActionPin1
5	Perform a Simulation on the diagram. When the CreateObject Action is executed, it creates an Object having the properties of the classifier, and stores it in its Output Pin. The Object itself is passed through

	the Object Flow connection to the Input Pin of Action 2, where its properties can be listed in the Locals window for the Simulation.

Create Object Using JavaScript

You can also create Simulation objects dynamically using a JavaScript command in the 'Effect' field of the Action element. The command is:

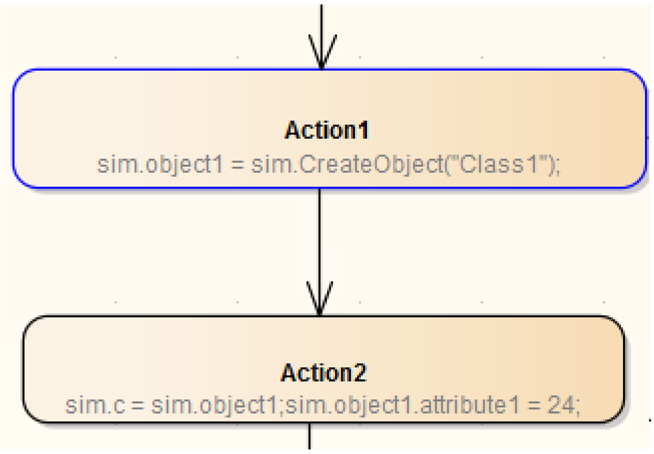
sim.newObject = sim.CreateObject("ClassName");

or

sim.newObject = new SimObject("ClassName"); (natural JavaScript)

That is: 'Simulate the creation of an Object based on Class <name>'. The classifying Class would exist in the same Package as the Action.

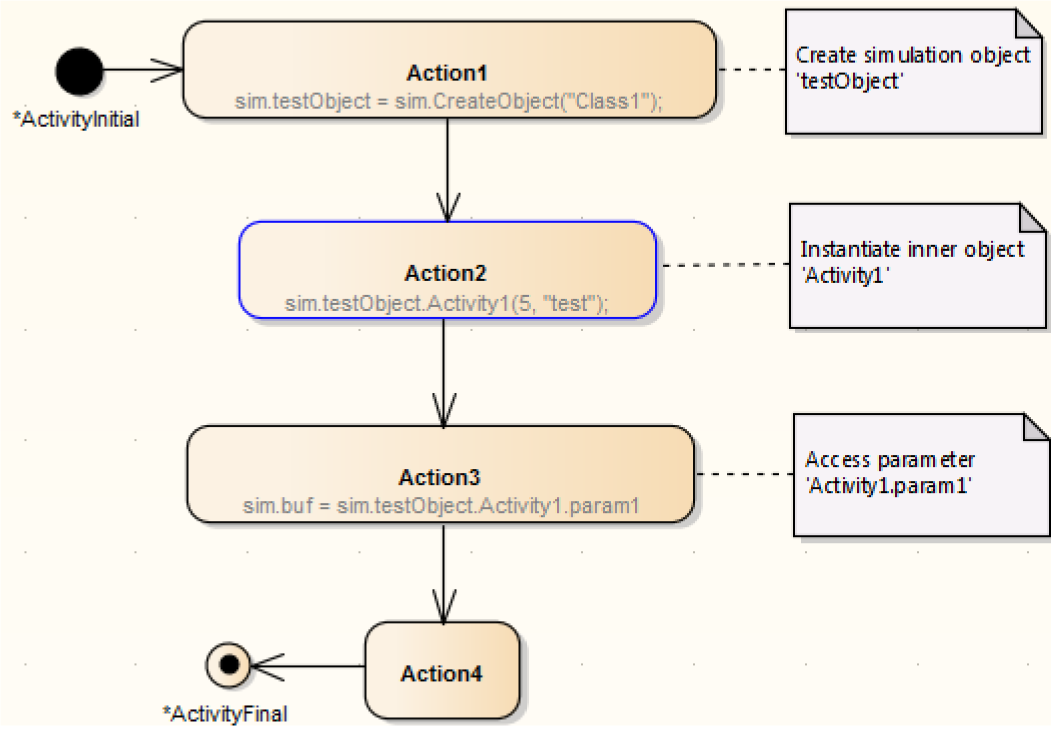
As for the CreateObject Action element, the Object is created during the Simulation and can be passed down to and processed by 'downstream' elements. In this example, the created Object is identified as sim.object1 and in Action 2 it is accessed and one of its attributes given a different value (also by JavaScript as an Effect of the Action).


Action1
sim.object1 = sim.CreateObject("Class1");
Action2
sim.c = sim.object1;sim.object1.attribute1 = 24;

Instantiate Inner Objects

As described earlier, you can create an Object using either JavaScript or a CreateObject Action. Similarly, you can instantiate inner objects using JavaScript or a CallBehavior Action.

In this example, using JavaScript, the Simulation first creates a test object based on Class1. Class 1 has an Activity element and diagram, with an Activity Parameter 1 set to the integer 5 and an Activity Parameter 2 set to the string 'test'. The value of Activity Parameter 1 is captured as a buffer value 'buf'.


*ActivityInitial
Action1
sim.testObject = sim.CreateObject("Class1");
Create sim ulation object 'testObject'
Action2
sim.testObject.Activity1(5, "test");
Instantiate inner object 'Activity1'
Action3
sim.buf = sim.testObject.Activity1.param1
Access param eter 'Activity1.param1'
Action4
*ActivityFinal

Destroy Objects in a Simulation

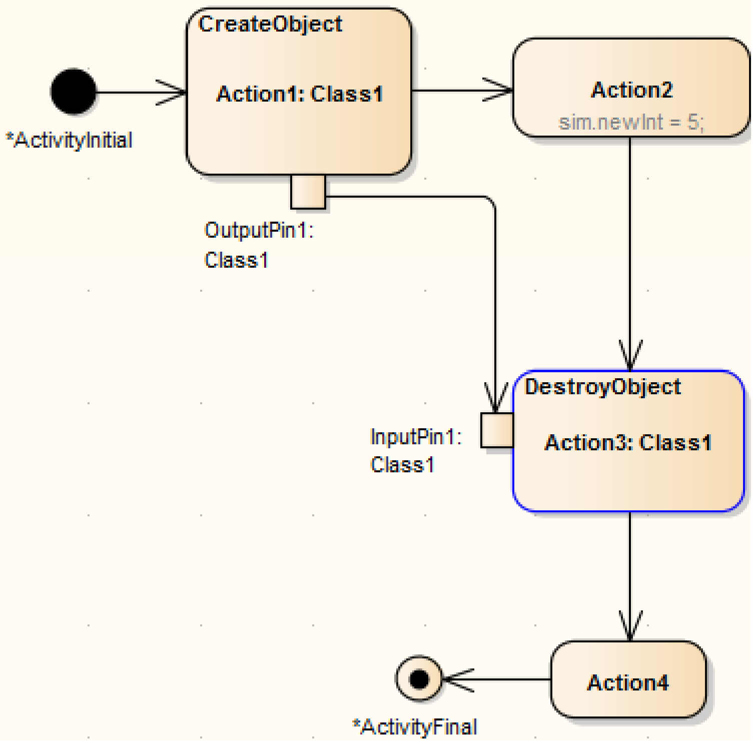
Having created or generated Objects in your Simulation model, you can define Actions to destroy those objects at any point during the process. All Simulation objects are destroyed automatically when the Simulation completes.

You have two options for destroying the Objects in your Simulation model:

- Dynamically destroy the Objects through a DestroyObject Action element
- Dynamically destroy the Objects using JavaScript in an Action element

The result of the deletion can be observed in the change of local variables, on the Local window.

Destroy an Object through a DestroyObject Action

Step	Action
1	On your Activity diagram, drag an 'Action' icon from the Diagram Toolbox, and select the 'Other | DestroyObject' context menu option to define it as a DestroyObject Action element.
2	Set the classifier of the DestroyObject Action to the Class of which the Object is an instance. (Advanced | Set Classifier). Create an Action Pin on the DestroyObject Action, of kind *input*.
3	Connect the Input Action Pin to an Object Flow connector from the last Action that operated on the Object. In this example, the last Action that operated on the Object is the Action that created it.  CreateObject Action1: Class1 *ActivityInitial Action2 sim.newInt = 5; OutputPin1: Class1 DestroyObject Action3: Class1 InputPin1: Class1 Action4 *ActivityFinal
4	Perform a Simulation on the diagram. The process passes the Object name or value into the Input Action Pin as a parameter. When the DestroyObject Action is executed, it deletes the Object having that name or value from the model. In the example, the instance of Class1 is specifically destroyed before Action4 is processed, but the results

	of Action2 are unaffected.

Destroy an Object using JavaScript

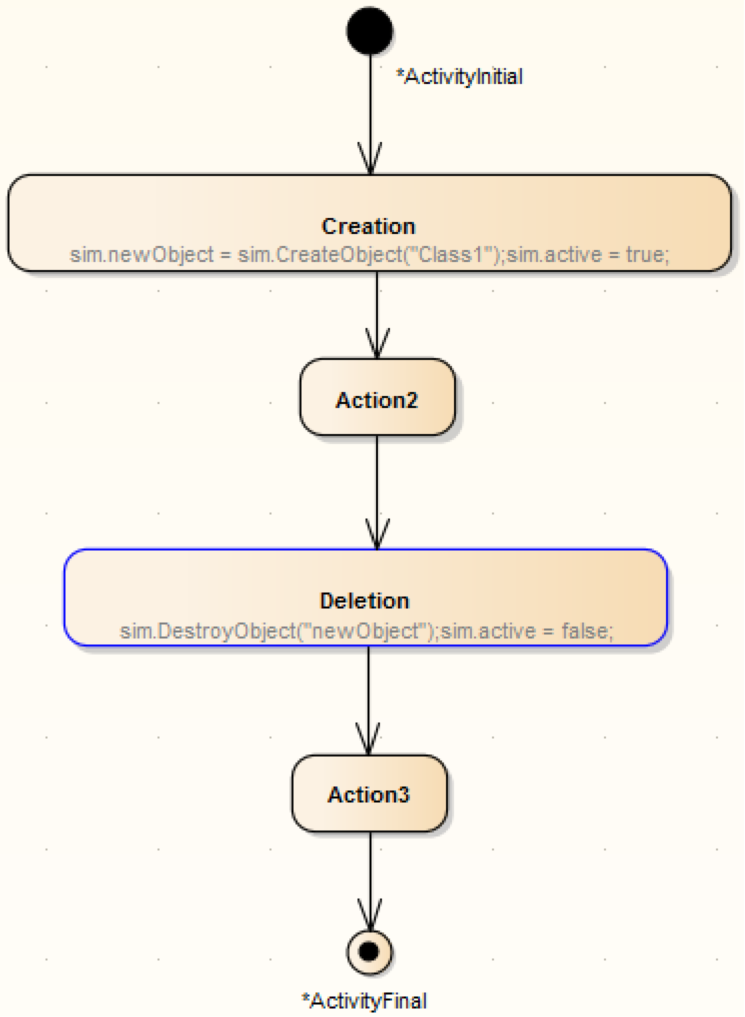
In the 'Properties' dialog of the Action element, in the 'Effect' field on the 'Effect' page, type either:

sim.DestroyObject ("objectname")

or

delete sim.objectFullName

For example:


*ActivityInitial
Creation
sim.newObject = sim.CreateObject("Class1");sim.active = true;
Action2
Deletion
sim.DestroyObject("newObject");sim.active = false;
Action3
*ActivityFinal

Notes

- In either case, you can also destroy a global object (one that is created outside the process flow) by identifying the Object to the Action performing the destruction; in the case of the DestroyObject Action, by passing the Object name from a Port on the Object to the Input Pin on the Action through an Object Flow connector

Dynamic Simulation with JavaScript

The Corporate, Unified and Ultimate editions of Enterprise Architect provide the capability of using JavaScript to evaluate guards, effects and other aspects of behavior within the Simulation context. This provides for a fully automated, intelligent execution of your State or Activity model, with fine control over breakpoints, execution speed and Simulation variables.

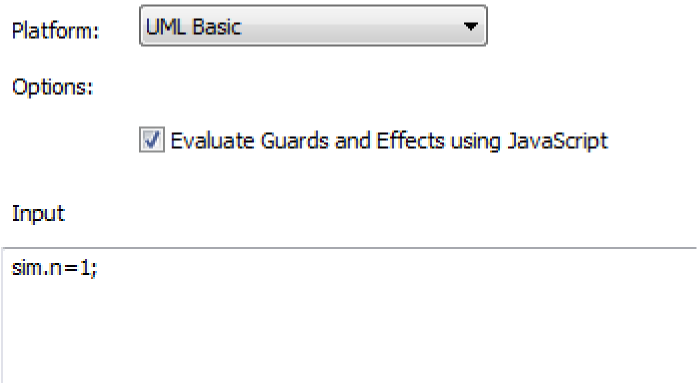
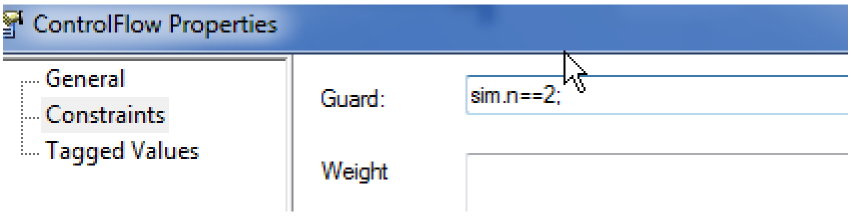
You can write JavaScript that uses any variables. To enable you to display the values of such variables through the user interface, two built-in objects are defined - **sim** and **this** - whose members can be displayed in the Local Variables window (also called the Locals window). Examples of the variables that can be displayed are:

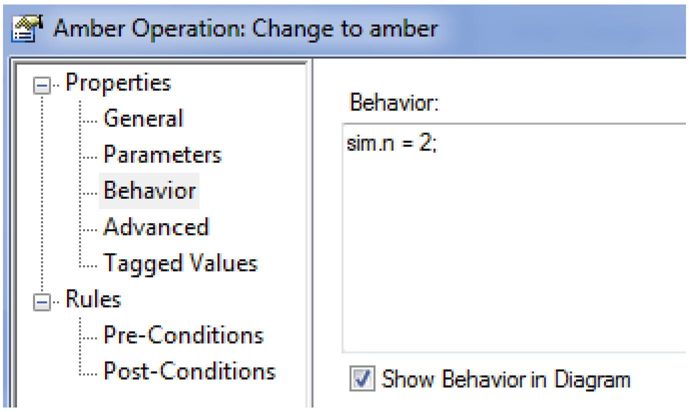
- sim.logger
- sim.Customer.name
- this.count
- this.Account.amount

The recommended convention is to add any global or control variables not declared in the owning Class to the **sim** object. In contrast, it would be normal to add attributes of the owning classifier to the **this** object.

Some examples of where and how you can set Simulation behavior using JavaScript are provided here. Further examples are provided in the EAExample.eap model that comes with Enterprise Architect.

Using JavaScript

Setting	Action
Analyzer Script Input	If you enter JavaScript code into the Execution Analyzer window 'Input' field, this code will be injected into the Simulation and executed before the Simulation starts. This is useful for establishing COM variables, global counters, functions and other initialization.  Platform: UML Basic Options: ☑ Evaluate Guards and Effects using JavaScript Input sim.n=1;
Transition and Control Flow Guards	This is the workhorse of the Simulation functionality. As Enterprise Architect evaluates possible paths forward at each node in a Simulation, it tests the Guards on outgoing transitions and control flows and will only move forward if there is a single true path to follow - otherwise the Simulation is considered 'blocked' and manual intervention is required. You must use the '==' operator to test for equality.  ControlFlow Properties General Constraints Tagged Values Guard: sim.n==2; Weight

Element Behavior	Entry and Exit behavior can be defined for States. Such code will execute at the appropriate time and is useful for updating Simulation variables and making other assignments.  Amber Operation: Change to amber Properties General Parameters Behavior Advanced Tagged Values Rules Pre-Conditions Post-Conditions Behavior: sim.n = 2; ☑ Show Behavior in Diagram You can also simulate the behavior of Classes via their Object Instances, and Activities in your model.

Using COM	One very important feature of the implementation of JavaScript in Enterprise Architect's Simulator is that it supports the creation of COM objects. This provides the ability to connect the running Simulation with almost any other local or remote process and either influence the Simulation based on external data, or potentially change data or behavior in the external world based on the current Simulation state (for example, update a mechanical model or software Simulation external to Enterprise Architect). The syntax for creating COM objects is shown here:
	this.name="Odd Even";
	var logger = new COMObject("MySim.Logger");
	logger.Show();
	logger.Log("Simulation started");

Using Solvers	Anywhere in Enterprise Architect that has JavaScript code, such as in Dynamic Simulation, you can now use a JavaScript construct called 'Solver' (the Solver Class) to integrate with external tools and have direct use of the functionality within each tool to simply and intuitively perform complex math and charting functions. The calls help you to interchange variables between the built in JavaScript engine and each environment easily. Two Math Libraries that are supported are MATLAB and Octave. 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 product documentation. Being part of the JavaScript engine, these Solver Classes are also immediately accessible to Add-In writers creating model based JavaScript Add-Ins. See the *Octave Solver, MATLAB Solver* and *Solvers* Help topics for further details.
Signalled Actions	It is possible to raise a signalled event (trigger) directly using JavaScript. The BroadcastSignal() command is used to raise a named trigger that could influence the current state of a Simulation. For example, this fragment raises the signal (trigger) named "CancelPressed". BroadcastSignal("CancelPressed"); Note that a trigger named CancelPressed must exist within the current Simulation environment and must uniquely have that name. You can also call the signal using its GUID. For example:

	BroadcastSignal("{996EAF52-6843-41f7-8966-BCAA0ABEC41F}");

IS_IN()	The IS_IN(state) operator returns True if the current Simulation has an active state in any thread matching the passed in name. For example, to conditionally control execution it is possible to write code such as this:
	if (IS_IN("WaitingForInput"))
	BroadcastSignal("CancelPressed")
	Note that the name must be unique within all contexts.

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 JavaScript Simulation will truncate strings that exceed the defined maximum length of the Trace message.

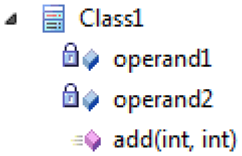
Call Behaviors

In the course of simulating a process, you can enact the behaviors defined in an operation of either a Class (through its Simulation Object) or an Activity in the model. In each case, you use JavaScript to call the behavior.

Invoke the Behavior of a Class

A Class in your model defines a behavior that you want to simulate. This behavior is defined in the Behavior page of an Operation of the Class.

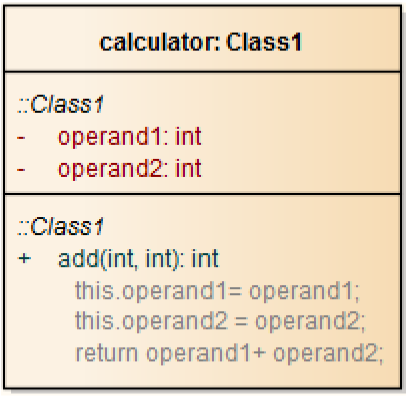
For example, the Class is intended to add two integers, through the Operation **add(int, int)**. The integers in this case are parameters of the operation, defined by attributes of the Class, *operand1* and *operand2*.



Class1
operand1
operand2
add(int, int)

Step	Action

1	In the Properties window for the operation, select the 'Behavior' tab and edit the 'Behavior' field to apply the JavaScript Simulation Objects (*this* or *sim*) to the behavior definition.
	In the example:
	this.operand1=operand1;
	this.operand2=operand2;
	return operand1+operand2

2	Drag the Class onto your Simulation Activity diagram and paste it as an Instance.
	In the example, the Object is called 'calculator'. For clarity, the element shown here is set to display inherited attributes and operations, and the behavior code, on the diagram.
	
	calculator: Class1
	::Class1
	- operand1: int
	- operand2: int
	::Class1
	+ add(int, int): int
	this.operand1= operand1;
	this.operand2 = operand2;
	return operand1+ operand2;

3	On the Simulation diagram, for the appropriate Action element, open the 'Properties' dialog and on the 'Effect' page type in the JavaScript to capture and simulate the Object's behavior. In the example, the JavaScript defines a value that will be provided by simulating the behavior of the operation from the Object, as performed on two provided integers. That is: sim.result=sim.calculator.add(7,9)
4	Run the Simulation, and observe its progress in the Locals window. Ultimately the Class behavior is reflected in the result of the Simulation.

	In the example: result = 16.

Invoke the Behavior of an Activity

An Activity element can have a behavior, defined by an operation in that element. As a simple example, an Activity might have an operation called Get Result, with the behavior return "ON";.

You can simulate this behavior in the Activity's child diagram (that is, internal to the Activity), with a JavaScript statement in the appropriate Action element's 'Effect' field. In the example, this might be:

```
sim.result=this.GetResult();
```

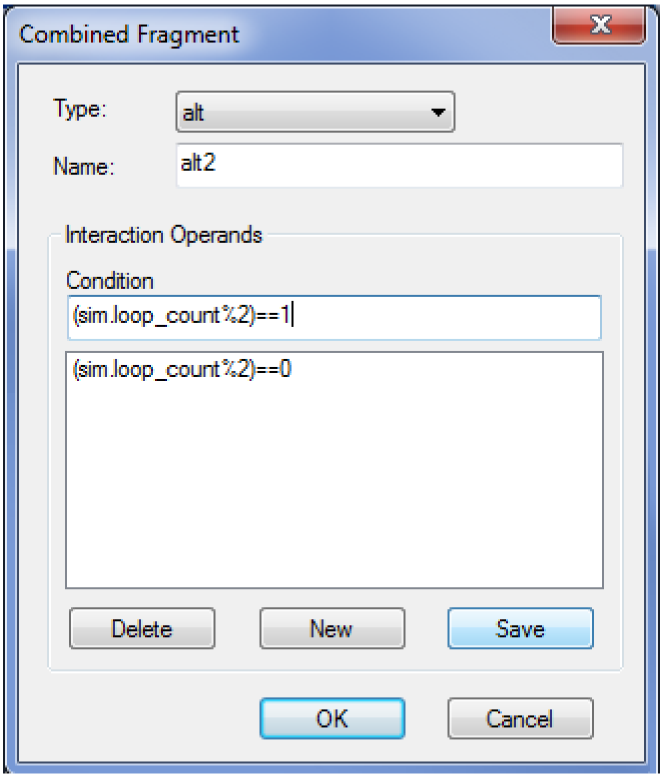
The statement invokes the parent Activity's operation GetResult and assigns the outcome of that operation's behavior to sim.result. You can observe the progress of the Simulation and the result of simulating the behavior in the Locals window, where (in this example) the value result "ON" will ultimately display.

Interaction Operand Condition and Message Behavior

When you simulate the behavior of a Sequence diagram, you can use a Condition for the CombinedFragment Interaction Operand, to control the flow during the course of the Simulation.

A Message in Sequence diagram can link to an Operation, so the behavior of the Operation can also be used during the course of the Simulation.

Interaction Operand Conditions

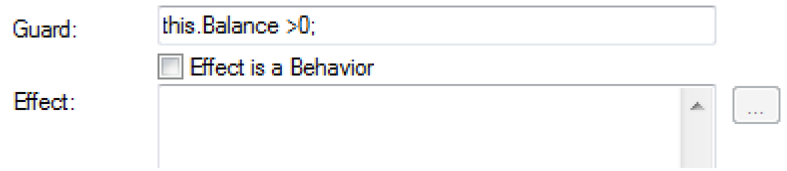
Field/Column	Description
Operand Condition	Interaction Operand Conditions are conditional statements that are evaluated whenever the Simulator has to determine which path to take next. Operand Conditions typically have these characteristics: • Defined in the 'Combined Fragment' dialog • Written in JavaScript • Can refer to variables defined during Simulation
Adding Operand Conditions	To add an Operand Condition: 1. Double-click on the CombinedFragment element to open the 'Combined Fragment' dialog. 2. Click on the New button. 3. In the 'Condition field', type the JavaScript for the condition. 4. Click on the Save button.  Type: alt Name: alt2 Interaction Operands Condition (sim.loop_count%2)==1 (sim.loop_count%2)==0
Evaluation Semantics	During execution the Simulator evaluates any Operand Condition within the CombinedFragment; the CombinedFragment type and the outcome of the

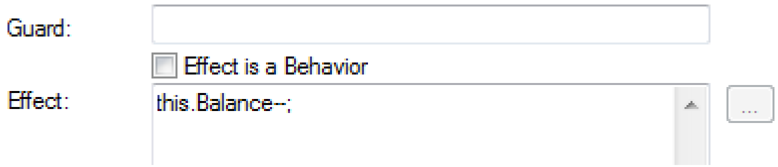
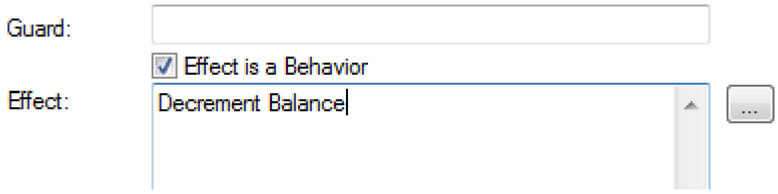
	evaluation can determine the path that the Simulation continues on.

Guards and Effects

Guards and Effects are used to control the flow of the Simulation and to execute additional actions or effects during the course of a Simulation.

Guards and Effects

Concept	Detail
Guards	Guards are conditional statements that are evaluated whenever the Simulator has to determine the path to take next. Guards typically have these characteristics: • Defined on transitions and control flows to govern how a Simulation proceeds • Written in JavaScript • Can refer to variables defined during Simulation
Adding Guards	Guards are defined on the Transition or Control Flow in the 'Properties' dialog for the selected connector. A Guard is typically a piece of JavaScript that will evaluate to either True or False. For example, here is a conditional statement that refers to a current variable (Balance) being greater than zero. Note the use of the prefix '*this'* to indicate that the variable is a member of the current Class context.  Guard: this.Balance >0; □ Effect is a Behavior Effect:
Evaluation Semantics	During execution the Simulator will examine all possible paths forward and evaluate any guard conditions. This evaluation could establish that: • A single valid path forward evaluates to True; the Simulator will follow that path • Two valid paths exist; the Simulator will block, waiting for some manual input via the console window to resolve the deadlock • No valid path exists; the same response as when two paths are found - the Simulator waits for the user to change the execution context using the console window • No paths evaluate to True but a default (unguarded path) exists; the Simulator will take the unguarded path
Effects	Effects are defined behaviors that are executed at special times: • On entry to a state • On exit from a state • When transitioning from one state to another (transition effect) Effects can either be a section of JavaScript code or a call to another Behavior element in the current simulation.
JavaScript Effects	A JavaScript effect might resemble this example, in which the Balance variable is decremented:

	 Guard: □ Effect is a Behavior Effect: this.Balance--;
Call Behavior Effects	In this example the Effect is a call behavior effect. In this case, it calls into a model the Activity named Decrement Balance that is defined elsewhere. The simulation will then enter into that diagram/behavior and continue to execute until returning to the point at which the Effect was invoked.  Guard: ☑ Effect is a Behavior Effect: Decrement Balance
Order of Execution of Effects	In complex simulations that might involve transitioning out of deeply nested states into other deeply nested states in a different parent context, it is important to consider these rules concerning the order of execution: • All exit actions (Effects) encountered leaving a nested context are executed in order of most deeply nested to least deeply nested • All actions (Effects) defined on transitions are executed next • Finally, all entry Effects are executed from the least deeply nested context to the most deeply nested So the basic rule is: all exit actions, followed by all transition actions, and finally all entry actions.
Note on JavaScript Variables	JavaScript variables to be accessed and referred to during Simulation execution belong to either: • sim (for example, sim.pedestrianwaiting) - typically used for global Simulation variables, or • this (for example, this.CustomerNumber) - typically used to refer to owning Class attributes This is important to let the JavaScript engine know you are referring to a Simulation variable and not a simple local variable used during, for example, basic calculations. You can create Simulation variables of arbitrary scope and depth - for example, this.customer.name is a legitimate qualified name.

Triggers

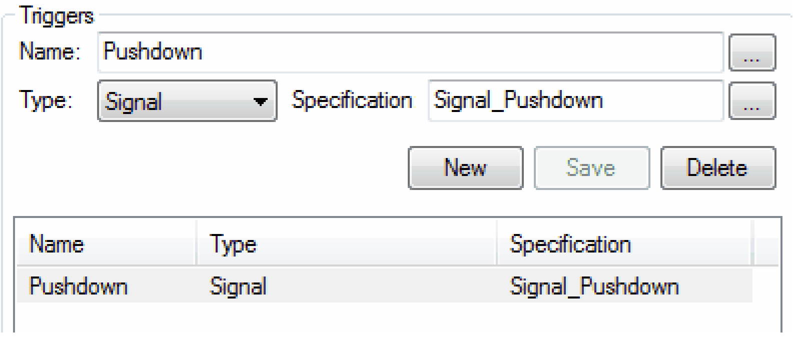
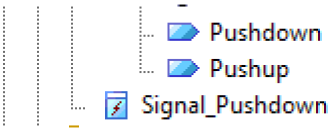
Triggers represent signals and events that can activate transitions leaving the current state(s). A trigger might represent a real world signal or event such as:

- A button being pressed
- A message being received
- A pedal being depressed
- A switch being thrown
- A state in a concurrent region being entered or exited

For a trigger to have an Effect

- Transitions must be defined that will fire when the simulation receives the Signal or Event
- The current Simulation State(s) or its parent(s) must have an outgoing transition that accepts that trigger
- The transition activated must be unguarded or have a guard that will evaluate to True

Managing Triggers

Action	Detail
Creating Triggers	Triggers are either created as an instance of a Signal element or as an anonymous event. Triggers are connected to Transitions in the 'Transition Properties' dialog as shown here. In this example a Trigger named 'Pushdown' has been defined based on the Signal 'Signal_Pushdown'. • Omitting the Type and Specification details results in a simple anonymous Trigger. • If parameters are needed, these are defined on the Signal and must be supplied at the time the event fires  Triggers Name: Pushdown Type: Signal Specification Signal_Pushdown Name / Type / Specification Pushdown / Signal / Signal_Pushdown A trigger will appear in the 'Project' tab of the Browser window, as illustrated here: 
Using Triggers	Triggers are deployed by connecting them to transitions, as in the earlier example, and are used during simulation by 'firing' them into the running simulation as required.

When using triggers these points should be taken into account:

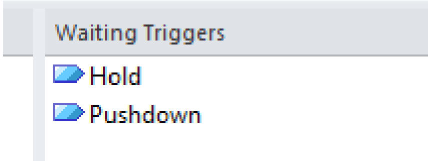
- A 'triggered' transition can not take place until its effective trigger is signalled or fired
- When a trigger is received it will activate all current waiting transitions dependent on that trigger (that is, the trigger is broadcast)
- Triggers are evaluated on all transitions for all parents of a current child state; this allows a parent state to exit all child states if necessary
- Once used in a simulation, a trigger is consumed and must be re-fired if needed again
- Sets of triggers can be saved and either manually or automatically fired to facilitate automated model simulation under different event models

Firing Triggers

Firing triggers means to signal or activate a trigger within the current simulation. This could activate zero, one or many waiting transitions depending on the state and concurrency of the current Simulation.

Firing triggers can be achieved in many ways. The most efficient is the 'Waiting Triggers' list.

During the course of model Simulation, if the Simulator reaches an impasse due to required triggers not being available (fired), the list of all possible candidate triggers is shown in the 'Waiting Triggers' list of the Simulation Events window.



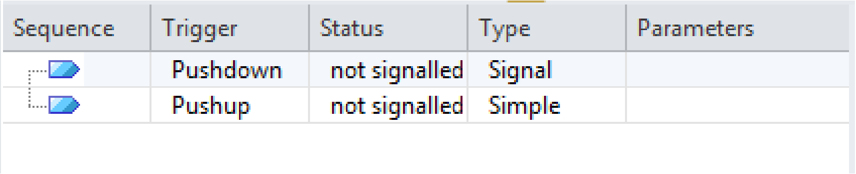
Waiting Triggers

Hold

Pushdown

Double-clicking a trigger in this list will fire it into the Simulation. Other ways to fire a trigger include:

1. Double-click an un-signalled trigger in the Events window.



Sequence	Trigger	Status	Type	Parameters
	Pushdown	not signalled	Signal	
	Pushup	not signalled	Simple	

You can also use the context menu on these events to either signal an un-signalled event, or to re-signal an event that has already been fired previously.

2. Use the context menu of the Transition required to fire and select the 'Signal Trigger in Simulation' menu option.

Action Behavior By Type

You can vary the behavior initiated by an Action element by defining (or even redefining) its type. In Simulation, you can apply and observe a number of different behaviors using the Actions in the types and groups described in this table.

Action Types

Type	Description
Object Actions	Object Actions operate on an object in a specific way, such as creating, destroying or reading the object. They include: • CreateObject • DestroyObject and • Read Self
Variable Actions	Variable Actions have an association variable in the form of the Tagged Value variable with the value of the name of an object in run-time. They provide the variable not only as an object but also as a property (such as an attribute or Port) of an object. They include: • ReadVariable • WriteVariable • ClearVariable • AddVariableValue • RemoveVariable
StructuralFeature Actions	StructuralFeature Actions operate on a structural feature, namely an attribute of an Activity or of the classifier of an object. They include: • ReadStructuralFeature • WriteStructuralFeature • ClearStructuralFeature • AddStructuralFeatureValue • RemoveStructuralFeatureValue
Invocation and Accept Event Actions	Invocation and Accept Event actions define the Triggers and Signals of an event. They include: • SendSignal • BroadcastSignal • AcceptEvent • SendObject • CallBehavior • CallOperation • AcceptCall
Miscellaneous Actions	The ValueSpecification Action evaluates a value; it must have an input value and some evaluating code as its behavior or effect.

Structured Activity Simulation

One of the more complex structures in a behavioral model is a Structured Activity, which models a series of actions either in a nested structure or in a process of assessment and execution. The assessment types of Structured Activity are the Conditional Node and Loop Node, both of which you can simulate quite easily.

Conditional Node

A Conditional Node essentially consists of one or more pairs of Test / Body partitions, each pair being referred to as a Clause. The Test partition is composed of Activity diagram elements that test a condition, and if that condition is met further Activity diagram elements in the Body partition are executed to produce a result.

If there is one Clause, the Conditional Node either outputs the result of the Body partition, or no result. If there is more than one Clause, control flows from one Test to the next until either a condition is met and a Body partition is executed to produce a result, or all Tests fail.

Simulation currently supports use of the 'Is Assured' checkbox setting in the 'Condition' tab of the Properties window. The other two checkbox settings are ignored. If the 'Is Assured' checkbox is:

- Selected, at least one Test must be satisfied, so its Body is executed and a result output; if no Test is satisfied and no result output, the Conditional Node is blocked and processing cannot continue beyond it
- Not selected, a Test can be satisfied and a result output, but if no Test is satisfied and no result output, processing can still continue beyond the Condition Node

You can simulate a range of possible paths and outcomes by typing JavaScript *sim.* statements that define or lead to specific Test results and Body results, in the 'Effect' fields of the Action elements within each partition of each Clause. These *sim.* statements must identify the full path of the Conditional Node, Clause and Output Pin being set. For example, in a test to see if a person qualifies as a senior citizen:

```
if (sim.Person.age >=65)
sim.AgeCondition.Clause1.Decider1=true;
else
sim.AgeCondition.Clause1.Decider1=false;
```

The Condition Node is called *AgeCondition*, the test is in *Clause1* and the OutputPin for that test is *Decider1*.

Loop Node

A Loop Structured Activity Node commonly represents the modeling equivalents of While, Repeat and For loop statements. Each Loop Node has three partitions:

- Setup - which initiates variables to be used in the loop's exit-condition; it is executed once on entry to the loop
- Test - which defines the loop exit-condition
- Body - which is executed repeatedly until the Test produces a False value

You define the Loop Nodes by dragging Activity diagram elements from the Toolbox pages into the Setup, Test and Body partitions. The Body partition can contain quite complex element structures, defining what the Loop Node actually produces in the process.

The Loop Node has a number of Action Pins:

- Loop Variable (Input) - the initial value to be processed through the Loop
- Loop Variable (Output) - the changing variable on which the Test is performed
- Decider - an Output Pin within the Test partition, the value of which is examined after each execution of the Test to determine whether to execute the loop Body
- Body Output - the output value of the processing in the Body partition, which updates the Loop Variable Output Pin

for the next iteration of the loop, and

- Result - the value of the final execution of the Test partition (which is the value passed back from the last execution of the Body partition)

You can simulate the effects of different actions and outputs through the Loop, by typing JavaScript *sim.* statements that define or lead to specific Test results and Body results, in the 'Effect' fields of the Action elements within each partition. These *sim.* statements must identify the path of the Loop Node and Output Pin being set. For example, in an Action in the Test partition:

```
sim.LoopNode1.decider = (sim.LoopNode1.loopVariable>0);
```

Activity Return Value Simulation

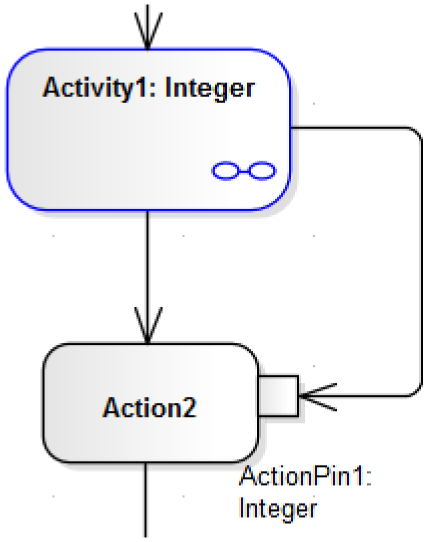
An Activity is likely to produce a return value as the output of the process it represents. You can simulate how that return value is passed on to the next stage in the process, under three scenarios:

- The Activity simply produces a return value, which is passed directly to the next Action
- The Activity has one or more Activity Parameters - represented on a diagram by Activity Nodes - that accept input values to or hold output values from the child Actions of the Activity, and the output Activity Parameter collects and passes on the return value
- The Activity is instantiated by a CallBehavior Action that replicates the behavior of the Activity and passes the return value onwards

Activity Return Value Pass Out

(This method is unique to Enterprise Architect simulation, mimicking the effect of an Activity Parameter without one having to exist.)

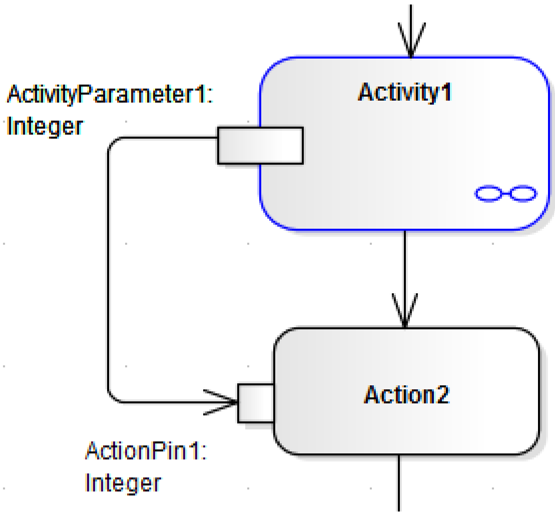
The Activity has a return value, which is transferred from the Activity element to an Action Pin on the next Action in the process via an Object Flow connector.



You can simulate this by setting a simple JavaScript statement to set the return value in the Activity's child element (such as this.return=12;) and, running the simulation, see the value transferred to the Action Pin in the Locals window.

Activity Parameter Pass Out

If the Activity has an Activity Parameter, its value passes to the corresponding Activity Node and then, via an Object Flow connector, to the Input ActionPin of the next Action in the process, as shown:

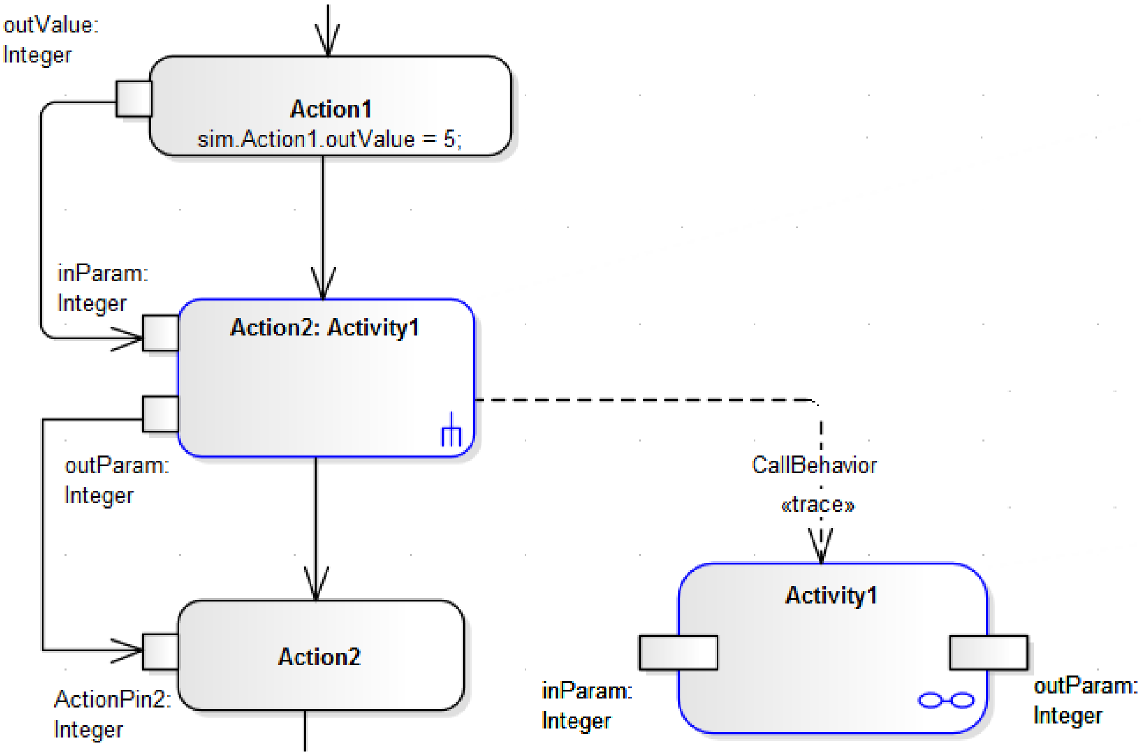


In the Locals window, you can either observe the Parameter's default value pass through to the ActionPin, or you can use JavaScript in the Activity's child Actions to simulate an update of the value within the Activity. For example:

```
this.ActivityParameter1=20;
```

CallBehavior Action

An Activity might be used a number of times in a process, in which case you might want to generate a separate instance of the Activity each time. You can do this using a CallBehavior Action to create an object of the Activity and execute its behavior. The input and output Activity Parameters are bound to corresponding input and output Action Pins (arguments) on the CallBehavior Action.



When you simulate the part of the process containing the Activity, you provide an input value (as in Action 1) that passes into the input Action Pin on the CallBehavior Action, which creates an Object of the Activity. The CallBehavior executes the behavior of the Activity, using the input Action Pin to act as the input Activity Parameter, and the Output Action Pin to receive the return as the output Activity Parameter. The Activity return value is then passed to an Action Pin on the next Action, using an Object Flow connector. You can provide JavaScript statements in the Actions of the

Activity to act on the input value and generate a return value, such as:

sim.buf=this.inParam; and

this.outParam=sim.buf + 11:

Simulation Events Window

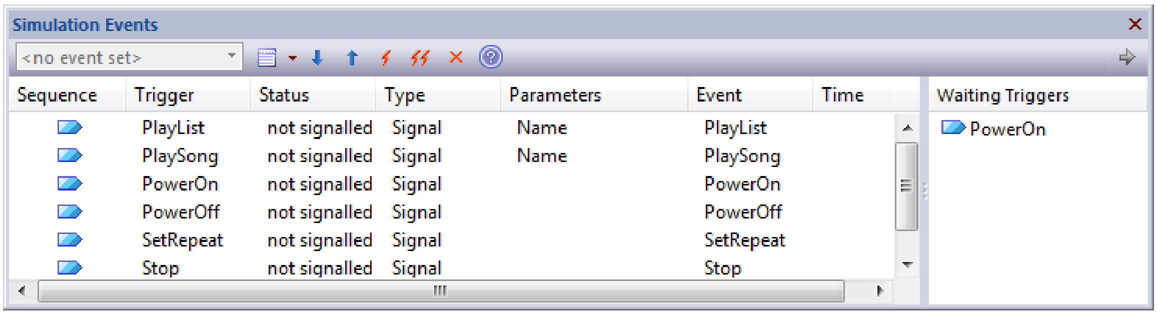
The Simulation Events window is where you manage triggers and sets of events in a simulation. Its main functions are to:

- Add, delete and re-sequence a set of triggers for a simulation
- Display a list of fired, lost and waiting events for the current running simulation
- Provide options to fire any arbitrary trigger into the current simulation
- Provide a convenient 'Waiting Triggers' list of triggers that the simulation is waiting on
- Save trigger sets for later use in both manual and automated simulations
- Accept triggers dragged from the Browser window into the current list
- Enter trigger parameters for a waiting trigger prior to firing

As triggers are consumed in the simulation, their status and position is logged in the main body of the Simulation Events window.

You can save the log of fired triggers as a trigger set or event set to reapply in another Simulation run, which you can execute manually or automatically. See the topic *Trigger Sets and Auto-Firing* for more information on building and using Trigger sets.

This image illustrates the Simulation Events window during execution.



Sequence	Trigger	Status	Type	Parameters	Event	Time
	PlayList	not signalled	Signal	Name	PlayList	
	PlaySong	not signalled	Signal	Name	PlaySong	
	PowerOn	not signalled	Signal		PowerOn	
	PowerOff	not signalled	Signal		PowerOff	
	SetRepeat	not signalled	Signal		SetRepeat	
	Stop	not signalled	Signal		Stop	

Waiting Triggers
PowerOn

Access

Ribbon	Simulate > Dynamic Simulation > Events

Column Details

Field/Column	Action
Sequence	During and after the simulation, indicates the position in the sequence in which a trigger was fired or is expected to be fired. Note that if a trigger is fired out of sequence, it will be moved to the bottom of the signalled events section.
Trigger	The name of the trigger - identifies the Trigger used to initiate the event.
Status	Indicates the status of the Trigger. Values can be: • used - the trigger has been fired and processing has passed on

	- lost - the trigger has been fired in the list, but it had no effect - signalled - a trigger was fired and consumed by one or more transitions - not signalled - the trigger has not yet been fired
Type	Indicates the type of trigger. Currently only supports: - Signal - (no type) an anonymous trigger
Parameters	For a Signal Trigger, initially shows the parameters required for firing by the Signal specification. For example a "Login" signal might include username and password parameters - and each triggered invocation can use different parameters. Each time the simulation fires the trigger, the system will prompt you for values. You can also edit the values directly in the list when the trigger is set to not signalled. Parameters are very useful for testing the conditional logic in your simulation and to simulate a variety of inputs and data coming in from outside the simulation.
Event	For a: - Signal Trigger, identifies the Signal specification - For anonymous Triggers has no value
Time	The simulation time at which the trigger was signalled. Note that this is an absolute (real world) time, and not a relative simulation event time.
Waiting Triggers	Lists the Triggers available for selection from the current state(s), including those where more than one trigger is possible at a single transition. Double-click on a trigger to add and signal it as the next trigger in the current event sequence. You can show and hide this panel by clicking on the gray arrow just above the panel.

Toolbar Items

Option	Action
Set1 ▾	Use this drop list to select and work with previously defined trigger sets. Before running a simulation, select a previously-defined trigger set to use for the next simulation run. You elect to not use a trigger set by selecting the <no event set> option.
▤▾	Click to create and delete trigger sets: - Save Set - Save the current trigger list as a new trigger set; the system prompts you for a name for the new set - Save Set As - Create a copy of the current set under a new set name - Delete Selected Set - Delete the current trigger set - Delete All Sets for Diagram - Delete all saved trigger sets for the current diagram
	Move the selected trigger one line down in the firing sequence of triggers.

↓	This option is not available if there are no signaled triggers below the selected line.
↑	Move the selected trigger entry one line up in the firing sequence of triggers. This option is not available if there are no signaled triggers above the selected line.
⚡	Click to fire the selected trigger. You can also fire the trigger by double-clicking on it.
⚡⚡	Click to toggle auto-firing on and off. Auto-firing will fire the un-signalled triggers in your trigger set sequentially. If your set matches a valid execution path, then the simulation will run automatically. Out of sequence or unused triggers will be 'lost'. A breakpoint pauses the auto-firing and you will need to click on the next trigger to resume auto-firing the simulation.
✕	Delete the selected trigger(s) from the list.

Context Menu Options

Option	Action
Signal Selected	Signal, or fire, the selected not signalled trigger.
Remove Selected	Remove a not signalled trigger from the sequence.
Re-Signal Selected	Fire a used or signalled trigger again.
Set All to Unsignalled	Set all used or signalled triggers to not signalled.
Clear Trigger List	Clear all triggers from the window, regardless of their status.

Waiting Triggers

When a simulation reaches a point where any change of state (for any thread) requires a Trigger to proceed, the simulation is effectively paused and control returns to the system. The simulation is now effectively waiting for some form of event (a real world signal) to proceed. The Waiting Triggers list is useful in helping to determine which Trigger should be manually signaled.

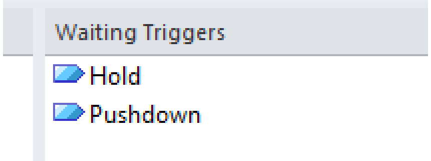
Access

Ribbon	Simulate > Dynamic Simulation > Events The right hand side pane lists available Triggers.

The Waiting Triggers list on the Simulation Events window is:

- Populated on each Simulation cycle with any Triggers that would have an immediate effect if signalled
- Populated with a discrete set (any duplicates are not shown as a Trigger is effectively broadcast to all transitions at once)
- Activated by double-clicking on the Trigger of interest
- Includes all possible triggers - including those activating transitions on parents of currently nested states

This example shows that the current simulation has hit a point where two possible Triggers can influence the flow of execution.



Waiting Triggers

Hold

Pushdown

Due to the nature of Triggers and their effects, the list can refer to each of these example situations equally validly:

- A single state has two outgoing transitions that are respectively waiting for Hold and Pushdown; firing one of these will activate the associated transition in the simulation
- A single state has two or more possible triggers for the same transition, such as a security camera being switched on by a motion detector, sound detector or heat detector
- Two (or more) threads (concurrent regions) each have a state waiting on either Hold or Pushdown; firing one of these triggers will result in the thread(s) waiting on that trigger to proceed while the other thread(s) will remain blocked
- A child state is waiting on one of the triggers while a parent state is waiting on the other; firing a trigger will result in the associated transition being fired and either the child or parent proceeding accordingly
- Any combination of these

Re-Signal Triggers

It is possible to re-signal a Trigger as a shortcut for dragging in additional Trigger instances for signalling.

Access

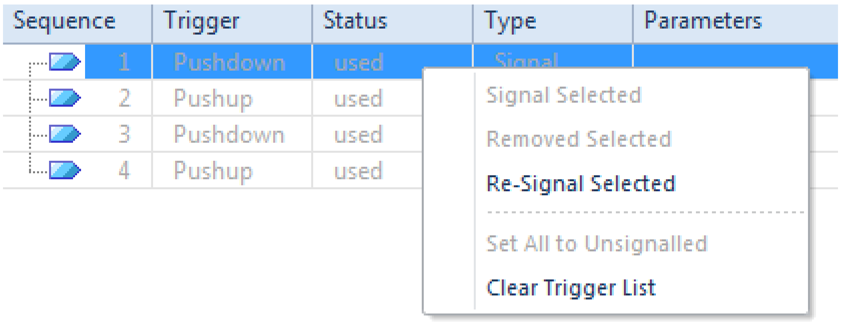
Display the Simulation Events window, then right-click on a Trigger within that window and select the 'Re-Signal Selected' option.

Ribbon	Simulate > Dynamic Simulation > Events > right-click on existing trigger > Re-Signal Selected

Trigger List

The Simulation Events window contains a list of Triggers that have already fired. By right-clicking on a Trigger that you want to signal again, you can use the context menu to cause the re-signal to happen.

This image demonstrates re-signalling in action. When a signal is re-signalled, a new copy is made and placed at the end of the signalled triggers list, where it is automatically fired again.

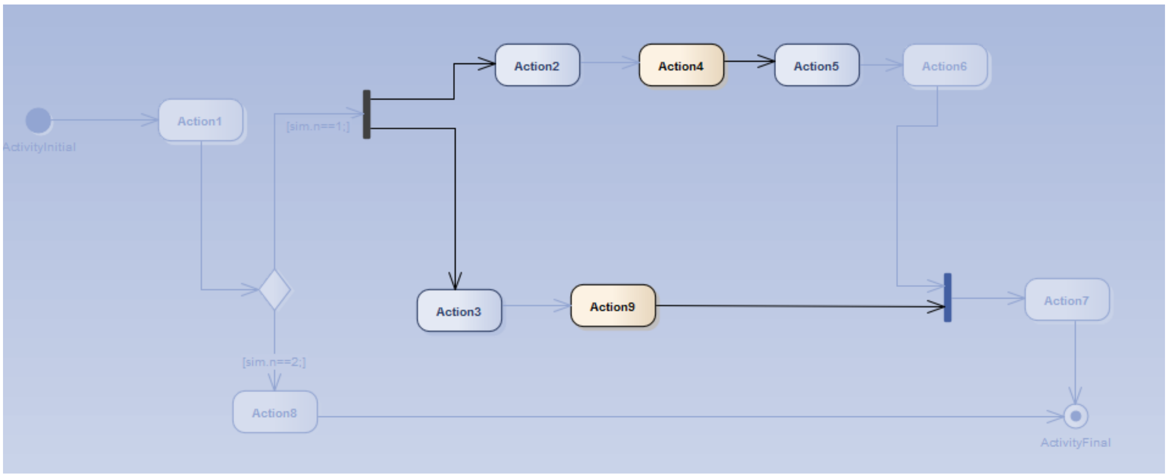


Sequence	Trigger	Status	Type	Parameters
1	Pushdown	used	Signal	
2	Pushup	used		
3	Pushdown	used		
4	Pushup	used		

Multi-threading - Forks and Joins

The Model Simulator provides the ability to handle multi-threaded simulations using Fork and Join nodes.

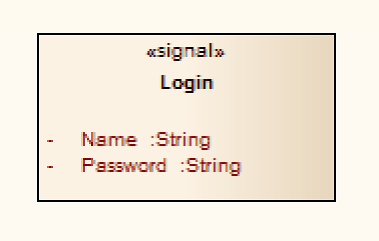
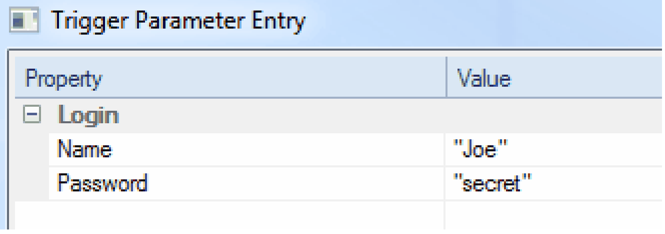
- In the example, the current execution point has forked into two threads, each with its own active node
- As this example progresses, the lower branch will wait at the Join node until the top branch has completed all its Actions
- Once the two threads merge back into one, the Simulation will continue as a single thread until completion
- When automatically stepping, each thread will be seen to execute a single step during one simulation 'cycle' - although when single stepping or at a breakpoint, the behavior is to alternate stepping between threads as each thread receives processing time
- Note that the Call Stack window will show two active threads and one 'paused' thread in the example; once the threads merge there will be a return to single threaded execution
- Also note that the Local variables are shared (global) between all threads; if you want to Simulate private variables on a thread you must create new Simulation variables at the start of each thread - pre-loading such variables with existing global data

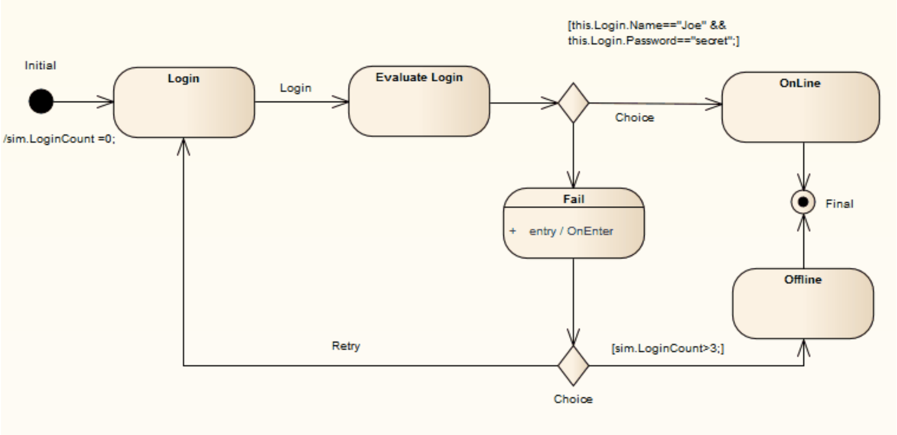


Trigger Parameters

Trigger parameters are arguments passed into the simulation along with a trigger when it is fired. They allow for complex behavior to be specified decision to be made based on variables and data passed into a simulation at run-time by a fired trigger (event).

Parameters

Parameter	Detail
Introduction	To use trigger parameters you: - First create a Signal element with the appropriate attributes that will become your parameters at run time - On a suitable transition in your diagram, create a trigger that is based on the signal created earlier - At run-time, will be prompted to enter suitable parameters - they are then passed in along with the trigger
Signals	A Signal element is a template or specification from which actual triggers can be built. This example has two arguments, a Name and a Password. These will be filled in at execution time either manually or as part of a pre-defined trigger set. 
Trigger Parameters	The Trigger parameters 'prompt' that asks for suitable values for each parameter. Note that you need to enclose strings in double quotes, otherwise the interpreter will think you are referring to other variables.  Trigger Parameter Entry Property | Value Login Name | "Joe" Password | "secret"
Example Diagram	This is an example diagram that makes use of trigger parameters. At the Evaluate Login state, the simulation examines the variables passed in as trigger parameters and makes a decision to either accept the credentials or deny them.

[this.Login.Name=="Joe" &&
this.Login.Password=="secret";]
Initial
Login
Login
Evaluate Login
OnLine
Choice
/sim.LoginCount =0;
Fail
+ entry / OnEnter
Final
Offline
Retry
[sim.LoginCount>3;]
Choice

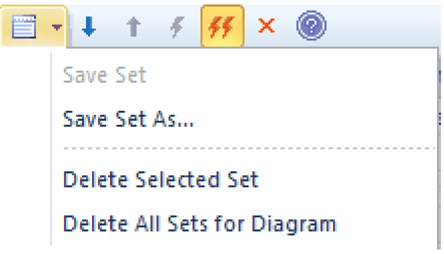
Trigger Sets and Auto-Firing

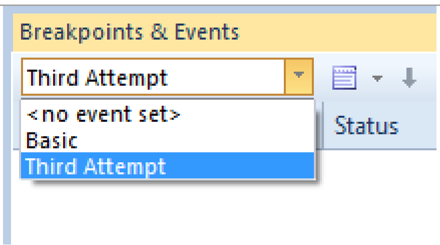
Trigger Sets are an effective means of automating and streamlining the execution, testing and validation of simulation models. By re-using sets of triggers (with or without parameters) it is possible to quickly and efficiently walk through many simulation scenarios, either manually or automatically using the 'auto-firing' tool.

Access

Ribbon	Simulate > Dynamic Simulation > Events

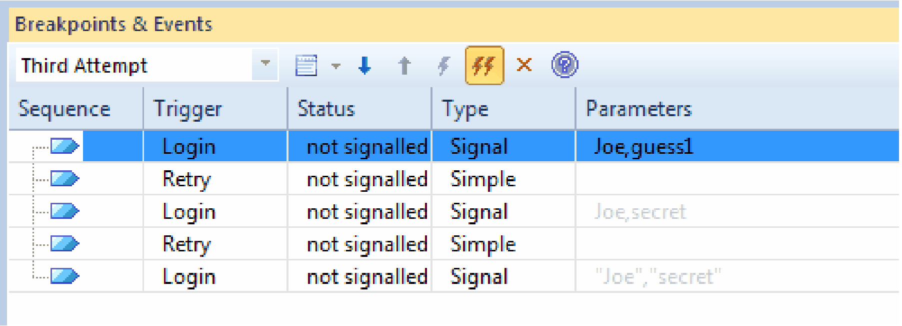
About Trigger Sets

Aspect	Details
Trigger Sets	- Stored with an associated diagram - Made up of a list of Triggers in a set sequence - Can include Trigger parameters where necessary - Can be used manually by double-clicking Triggers to fire as required - Can be used as part of the 'auto-fire' behavior to automate execution - Managed from the Simulation Events Window
Managing Sets	Trigger sets can be created by manually dragging triggers into the active triggers list and then using the 'Manage Trigger Sets' drop menu to save a new set. It is also possible to save a set of triggers built up during a single simulation setting as a new set. This is convenient for creating multiple test paths through a simulation, based on saving the manually fired triggers for each test case.  Save Set Save Set As... Delete Selected Set Delete All Sets for Diagram You can also delete a set and delete all sets for the current diagram. It is also possible to load a set, modify parameters and/or order of firing and save the set with a new name. This is a convenient method for rapidly creating a suite of simulation test scripts.
Using Sets	To use a trigger set you first select it by name from the trigger set drop list as in this example image. Once selected it loads the Trigger List window with the defined trigger set. Note that the special item <*no event set*> means no set is currently selected. At the start of each simulation, if a set is selected, it will be loaded afresh for the next run. If <*no event set*> is selected, the trigger list will be cleared.

<table>
<tr>
<td></td>
<td>


Once you have selected a trigger set and the list of triggers loaded you have two options:

- Fire the triggers as required manually
- Use the auto-fire feature to fully automate the simulation



<table>
<tr><th>Sequence</th><th>Trigger</th><th>Status</th><th>Type</th><th>Parameters</th></tr>
<tr><td></td><td>Login</td><td>not signalled</td><td>Signal</td><td>Joe,guess1</td></tr>
<tr><td></td><td>Retry</td><td>not signalled</td><td>Simple</td><td></td></tr>
<tr><td></td><td>Login</td><td>not signalled</td><td>Signal</td><td>Joe,secret</td></tr>
<tr><td></td><td>Retry</td><td>not signalled</td><td>Simple</td><td></td></tr>
<tr><td></td><td>Login</td><td>not signalled</td><td>Signal</td><td>"Joe","secret"</td></tr>
</table>
</td>
</tr>
<tr>
<td>Auto-Firing</td>
<td>
Auto-firing is a convenient way of streamlining your simulations. Once you have loaded a trigger set, if you select the Auto-fire button then Enterprise Architect will automatically pick up waiting triggers when it reaches an impasse in the simulation. In practice, this means that trigger sets matching exactly a path through the simulation will automatically run without your intervention.

As you can save any number of trigger sets with different pathways and trigger parameters, you can effectively and quickly test and work with many different scenarios.
</td>
</tr>
<tr>
<td>Auto-Firing Rules</td>
<td>
When a simulation runs with auto-firing enabled, Enterprise Architect will wait until a point is reached where the simulation is 'blocked' or stable, waiting on one or more triggers to advance the simulation. At that time, the first unfired trigger in the list will be picked up and fired into the simulation. The outcome depends on the relevance and perhaps on the parameters of the trigger.

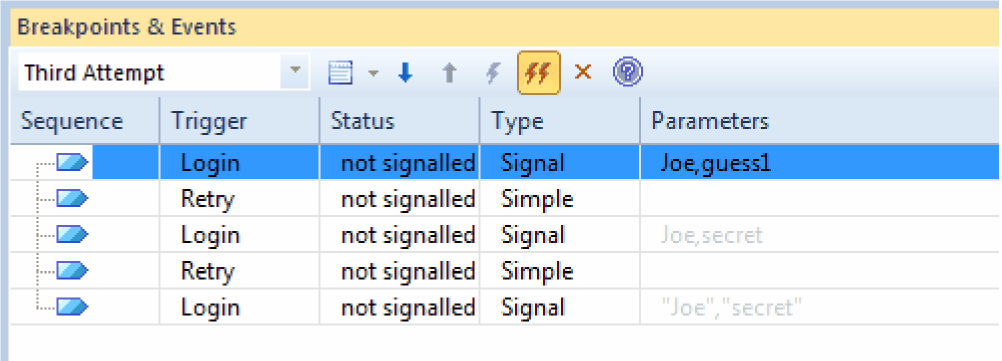
- If the trigger matches a 'waiting' trigger it is immediately consumed and the simulation advances
- If the trigger matches no 'waiting' trigger or possible parent transition, then the trigger is 'lost' and the simulation remains in the current state; this corresponds to a scenario such as a user pressing an 'on' button several times in succession - there is no effect other than that occasioned by the first press
</td>
</tr>
</table>

Using Trigger Sets to Simulate an Event Sequence

As a simple example of how useful trigger sets are, consider this example trigger set and associated diagram.

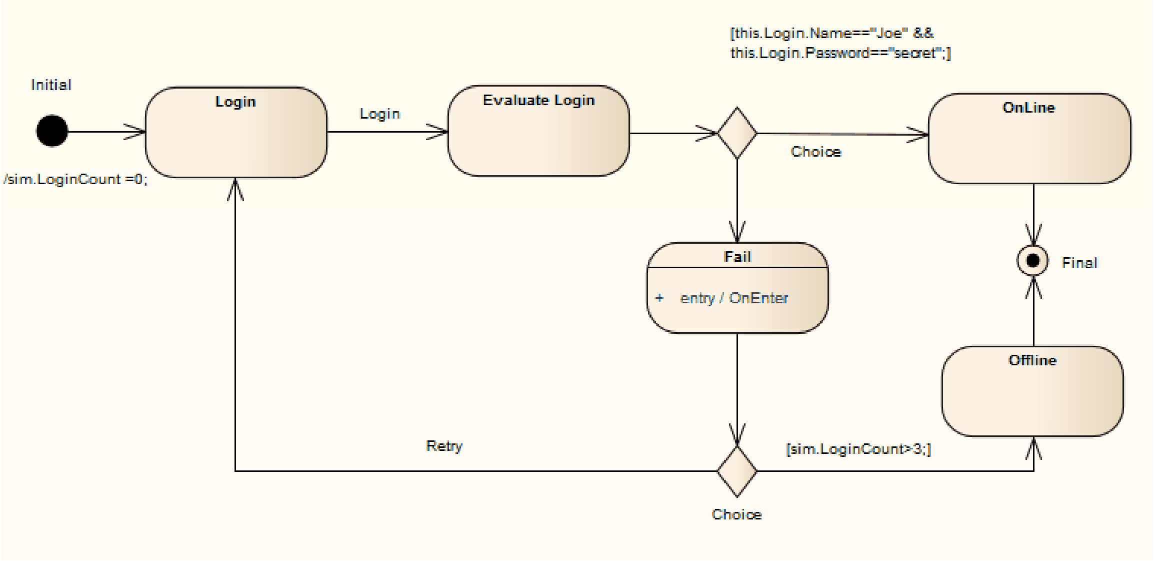
In this example, using a user name and password, we simulate a simple 'three strikes and you are out' login process. The success path is waiting for the name "Joe" and the password "secret". (Note - it is very important that parameters referencing strings are enclosed in quotes, otherwise the interpreter thinks the name refers to another variable within the simulation.)

- Pass 1 tries *Joe* and *guess1* - which fails
- Pass 2 tries *Joe* and *secret*, but as they are referring to variables, not strings - this fails as well
- Pass 3 shows the correct way of referencing trigger parameters, and the simulation will succeed



Sequence	Trigger	Status	Type	Parameters
	Login	not signalled	Signal	Joe,guess1
	Retry	not signalled	Simple	
	Login	not signalled	Signal	Joe,secret
	Retry	not signalled	Simple	
	Login	not signalled	Signal	"Joe","secret"

Here is a simple diagram simulating a login process requiring a username and password pair.



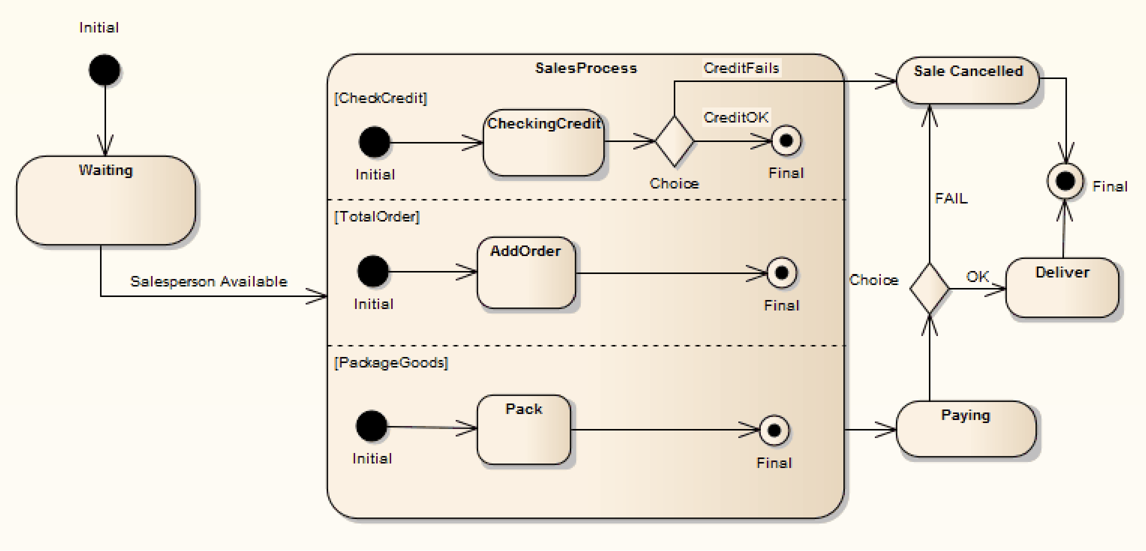
Multi-threading - Concurrent State Regions

Concurrent regions within a State represent state changes and processing that occurs in parallel inside one overall parent State. This is especially useful when one region raises events or modifies simulation variables that another region is dependent on. For example, one region could contain a simulated timer which raised events on set intervals that invoked state changes in the States within other regions.

Concurrent regions are essentially the same as Forks and Joins with similar logic and processing rules.

In the example:

- When the transition to SalesProcess is taken, each region is concurrently activated
- Credit is checked, the order totaled and the goods required packed up
- However, in the event that the Credit Check fails, this triggers the transition to the Sale Cancelled state; note that when this occurs, the entire parent state and all owned regions are immediately exited, regardless of their processing state
- If the Credit Check succeeds, the region moves to the final state and once the other regions have all reached their own final state, the parent state can then be exited

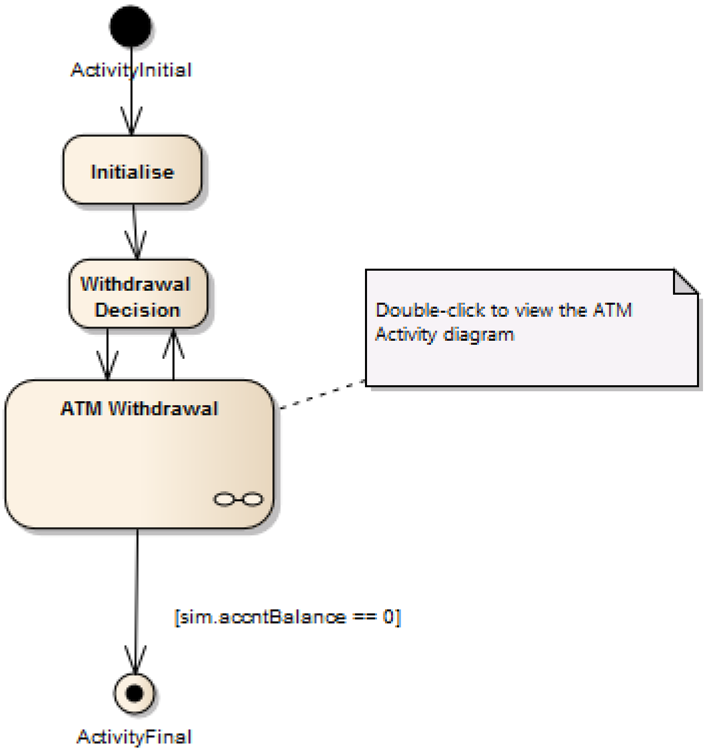


Using Composite Diagrams

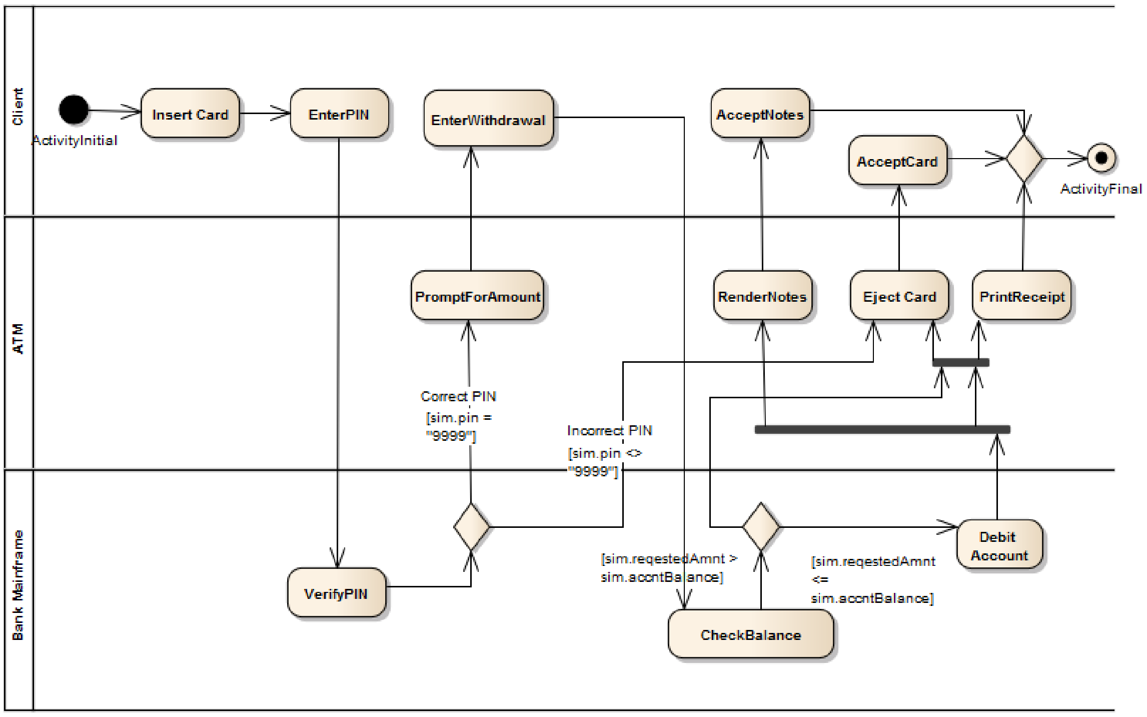
If you want to simulate processing that includes a branch represented on a different diagram (for example, to reduce complexity on the main diagram, or to hide areas of processing that are only actioned under an exception), you can use a Composite element to represent and access the branch on its child Composite diagram. When you run the simulation and it reaches the Composite element, it opens the child diagram and processes it before returning (if appropriate) to the main processing path. This is an excellent method of following the processing path in a complex process, representing sections of the process with Composite Activity elements that expand the actual processing in their respective child diagrams. You can have several Composite elements representing different stages or branches of the process.

One aspect to watch for (and that would be revealed by a failure in the simulation) is to have multiple threads that process simultaneously on separate diagrams. The simulation cannot pass to a new diagram if it is also following another thread on the current diagram.

This diagram provides an overview of an ATM cash withdrawal process:



The ATM Withdrawal Activity is a Composite element. If you double-click on it, you open and display the child diagram, which is a more detailed breakdown of the withdrawal process. Similarly, a simulation will open and process the child diagram.


Client
ATM
Bank Mainframe
ActivityInitial
Insert Card
EnterPIN
EnterWithdrawal
AcceptNotes
AcceptCard
ActivityFinal
PromptForAmount
RenderNotes
Eject Card
PrintReceipt
Correct PIN
[sim.pin = "9999"]
Incorrect PIN
[sim.pin <> "9999"]
VerifyPIN
[sim.reqestedAmnt > sim.accntBalance]
[sim.reqestedAmnt <= sim.accntBalance]
Debit Account
CheckBalance

Win32 User Interface Simulation

Enterprise Architect supports the simulation of dialogs and screens created with the Win32® User Interface profile, to integrate user interface design with defined system behavior. Dialogs can be programmatically referenced and invoked using JavaScript commands within a behavioral model such as a StateMachine, providing a fully customizable and fully interactive execution of your behavioral model.

Button controls can be used to broadcast signals, firing a trigger when the button is clicked. Signal arguments can be filled from the dialog input fields; for example, to capture and send a username and password for evaluation.

Dialogs designed using the Win32 User Interface profile (and existing within the same Package branch as the behavioral model being executed) will be created as new windows in the background at the beginning of simulation. Various properties that can affect the appearance and behavior of each dialog and control can be customized at design time via Tagged Values provided by the Win32 User Interface profile.

To interact with a dialog via JavaScript during simulation, the 'dialog' simulation-level keyword is used, followed by a period and the name of the dialog. Properties and methods can then be accessed; for example, to show the dialog, or to set the text value of an 'Edit Control':

```
dialog.Login.Show=true;
dialog.Login.Username.Text="admin";
```

Examples

To view an example of the Win32 User Interface Simulation, open the EAExample model and locate the diagram:

Example Model > Model Simulation > StateMachine Models > Customer Login > Customer > Customer Login

Common Properties

These common properties and methods are available on most supported Win32 UI Control types.

Property/Method	Description
Enable	Boolean Enables or disables user interaction.
Move To (x,y,width,height)	Move the window to the specified coordinates and set the window height and width.
Show	Boolean Show or hide the dialog. When this property is set to False, the dialog is moved off-screen.
Text	String Set the title of the dialog or window.

JavaScript Functions

Function	Description
BroadcastSignal (string Signal)	Sends a signal to the simulation event queue. Parameters: • Signal: String – the name of the Signal to be broadcast
UIBroadcastSignal (string Signal, array Parameters)	Sends a signal with additional parameters to the simulation event queue. Parameters: • Signal: String – the name of the Signal to be broadcast • Parameters: Array – additional parameters to be supplied for this Signal Example: UIBroadcastSignal("Login",{Name: dialog.Login.Username.Text, Password: dialog.Login.Password.Text});
ShowInterface (string InterfaceName, boolean Show)	Deprecated. See the **Show** property on the 'Dialog' control. For example: dialog.HelloWorld.Show = true;
InterfaceOperation (string InterfaceName, string ControlName, string OperationName,[string arg1],[string arg2])	Deprecated. Operations can be referenced directly from the control. For example: dialog.HelloWorld.ListControl.InsertItem("Test", 2);
GetInterfaceValue (string InterfaceName, string ControlName, string OperationName,[string arg1],[string arg2])	Deprecated. Properties can be referenced directly from the control. For example: dialog.HelloWorld.EditControl.Text;

Notes

- Controls must be within a dialog; any controls outside a dialog will not be interpreted
- Dialogs and controls must be on a Win32 User Interface diagram
- Simple UI controls and Basic UI controls can also be used in a simulation, but are limited in functionality compared to Win32 UI controls
- Dialog names and Control names must be unique; if multiple controls of the same name exist, the simulation will not be able to differentiate between them
- Spaces in dialog names and Control names are treated as underscores
- Dialog names and Control names are case sensitive

Supported Win32 UI Controls

This table identifies all of the Win32 UI Controls available in Enterprise Architect for user interface design and simulation.

Access

Ribbon	Design > Diagram > Toolbox : 🔍 > Specify 'User Interface - Win32' in the 'Find Toolbox Item' dialog
Keyboard Shortcuts	Ctrl+Shift+3 : 🔍 > Specify 'User Interface - Win32' in the 'Find Toolbox Item' dialog

Win32 UI Controls

Control	Description
Button	Button controls are a common way to allow user interaction during runtime; for example, an OK button in a login screen. A Button can respond to a click event, defined by adding an 'OnClick' Tagged Value. In response to a click event, a button can be used to, for example, send a signal, causing a trigger to fire during runtime. Customizable design properties: • Client Edge • Default Button • Disabled • Flat • Horizontal Alignment • Modal Frame • Multiline • Right Align Text • Right To Left Reading Order • Static Edge • Tabstop • Transparent • Vertical Alignment • Visible Tagged Values: • OnClick – specifies a JavaScript command to be executed in response to a click event on this Button Properties: • Enable

	• Show • Text Operations: • MoveTo
Check Box	Customizable design properties: • Auto • Client Edge • Disabled • Flat • Horizontal Alignment • Left Text • Modal Frame • Multiline • Right Align Text • Right To Left Reading Order • Static Edge • Tabstop • Vertical Alignment • Visible Tagged Values: • OnCheck – specifies a JavaScript command to be executed in response to a change in the value of this checkbox Properties: • Checker – integer value [0|1] • Enable • Show • Text
Combo Box	Customizable design properties: • Auto • Client Edge • Data – semi-colon delimited string of values to populate the combo box at runtime; for example, "yes;no;maybe" • Disabled • Has Strings • Lowercase • Modal Frame • Right Align Text • Right To Left Reading Order • Sort • Static Edge • Tabstop • Type • Uppercase • Vertical Scroll

	- Visible Operations - AddString (string) - DeleteAll () - DeleteItem (number) – delete item at specified index - DeleteString (string) – deletes all items matching string - GetCount () - GetString (number) - InsertItem (number, string) - InsertString (number, string) - SetString (number, string) Properties: - Enable - Selection – index of the currently selected item - Show
Dialog	Customizable design properties: - Absolute Align - Application Window - Border - Resizing or Dialog Frame only - Center - Client Edge - Center Mouse - Clip Siblings - Disabled - Horizontal Scrollbar - Left Scrollbar - Local Edit - Maximize Box - Minimise Box - No Activate - Overlapped Window - Palette Window - Right Align Text - Right To Left Reading Order - Set Foreground - System Menu - System Modal - Title Bar - Tool Window - Topmost - Transparent - Vertical Scrollbar - Visible - Window Edge

	Properties: • Enable • Show • Text Operations: • MoveTo
Edit Control / Rich Edit Control	Customizable design properties • Align Text • Auto HScroll • Auto VScroll • Border • Client Edge • Disabled • Lowercase (Edit Control only) • Modal Frame • Multiline • Number • Password • Read Only • Right Align Text • Right To Left Reading Order • Static Edge • Tabstop • Transparent • Uppercase (Edit Control only) • Visible • Want Return Properties: • Enable • Show • Text
Group Box	Customizable design properties: • Client Edge • Disabled • Flat • Horizontal Alignment • Modal Frame • Right Align Text • Static Edge • Tabstop • Visible Properties: • Enable

	• Show • Text
List Box	Customizable design properties: • Border • Client Edge • Disable No Scroll • Disabled • Left Scrollbar • Modal Frame • Right Align Text • Selection • Sort • Static Edge • Tabstop • Vertical Scroll • Visible Operations: • AddString (string) • DeleteAll () • DeleteItem (number) – delete item at specified index • DeleteString (string) – deletes all items matching string • GetCount () • GetString (number) • InsertItem (number, string) • InsertString (number, string) • SetString (number, string) Properties: • Enable • Selection – index of the currently selected item • Show
List Control	Customizable design properties: • Alignment • Always Show Selection • Border • Client Edge • Disabled • Edit Labels • Left Scrollbar • Modal Frame • No Column Header • No Scroll • Single Selection • Sort

	• Static Edge • Tabstop • View • Visible Tagged Values: • Columns – string to initialize column names and sizes for this List Control, separated by semi-colons: for example, "Column1;100;Column2;150;" Operations: • AddString (string) • DeleteAll () • DeleteItem (number) – delete item at specified index • DeleteString (string) – deletes all items matching the string • GetCount () • GetString (number, number) • InsertItem (number, string) • InsertString (number, string) • SetString (number, number, string) Properties: • Enable • Selection – index of the currently selected item • Show
Picture Control	The initial Picture Control image can be set using the Tagged Value 'Image'. Set the value to a filename accessible by the simulation. The image can be modified at runtime using the ChangeImageFile method in JavaScript. This takes a single string parameter of the filename to be loaded. Set the 'Image Type' property to the correct type for the file (either Bitmap, Enhanced Metafile or Icon). This setting cannot be modified at runtime. Customizable design properties: • Border • Center Image • Client Edge • Color (frame color) • Disabled • Image Type • Modal Frame • Real Size Image • Static Edge • Tabstop • View • Visible Operations: • ChangeImageFile (string) - filename Properties: • Show

Progress Control	Customizable design properties: • Border • Client Edge • Disabled • Marquee • Modal Frame • Smooth • Static Edge • Tabstop • Vertical • Visible Tagged Values: • Range – string specifying minimum and maximum values for this control, separated by a semi-colon: for example, "1;100" Properties: • Enable • Pos • Range • Show • Step
Radio Button	Customizable design properties: • Auto • Client Edge • Disabled • Flat • Group • Horizontal Alignment • Left Text • Modal Frame • Multiline • Static Edge • Tabstop • Vertical Alignment • Visible Tagged Values: • OnChangeSelection – specifies a JavaScript command to be executed in response to a change in selection of this radio button Properties: • Checker – integer value [0|1] • Enable • Selection – integer value • Show
Slider Control	Customizable design properties: • Auto Tick

	• Border • Client Edge • Disabled • Enable Selection Range • Modal Frame • Orientation • Point • Static Edge • Tabstop • Tick Marks • Transparent • Transparent Background • Tooltips • Visible Tagged Values: • Range – string specifying minimum and maximum values for this control, separated by a semi-colon: for example, "1;100" Properties: • Enable • PageSize • Pos • Range • Show
Spin Control	Customizable design properties: • Alignment • Arrow Keys • Auto Buddy • Client Edge • Disabled • Modal Frame • No Thousands • Orientation • Set Buddy Integer • Static Edge • Tabstop • Visible • Wrap Tagged Values: • Range – string specifying minimum and maximum values for this control, separated by a semi-colon: for example, "1;100" Properties: • Enable • Pos • Range

	• Show
Static Text / Label	Customizable design properties: • Align Text • Border • Client Edge • Disabled • End Ellipsis • Modal Frame • Path Ellipsis • No Wrap • Notify • Path Ellipsis • Right Align Text • Simple • Static Edge • Sunken • Tabstop • Visible • Word Ellipsis Properties: • Enable • Show • Text
Tab Control	Customizable design properties: • Border • Bottom • Buttons • Client Edge • Disabled • Flat Buttons • Focus • Hot Track • Model Frame • Multiline • Right Align Text • Static Edge • Style • Tabstop • Tooltips • Visible Tagged Values: • Tabs – string specifying names of each tab for this control, separated by a semi-colon: for example, "Tab 1;Tab 2;Tab 3;"

	Properties: • Enable • Show
Tree Control	Customizable design properties: • Always Show Selection • Border • Check Boxes • Client Edge • Disable Drag Drop • Disabled • Edit Labels • Full Row Select • Has Buttons • Has Lines • Horizontal Scroll • Left Scrollbar • Lines At Root • Modal Frame • Right Align Text • Right To Left Reading Order • Scroll • Single Expand • Static Edge • Tabstop • Tooltips • Track Select • Visible Operations: • Delete () - delete the specified TreeItem • InsertItem (string) - dotted path of the new tree item to be inserted; any parent items in this dotted path that do not yet exist will be created automatically • InsertString (string) - See *InsertItem* • TreeItem (string) - dotted path of the tree item to be accessed; use the *Text* property to set text for this tree item, or use the *Delete* operation to delete this item from the tree Properties: • Enable • Selection – string containing dotted path of the selected tree item • Show • Text – get or set text for a specified TreeItem Examples: dialog.MyDialog.MyTreeControl.InsertItem("Root.Parent.Child"); dialog.MyDialog.MyTreeControl.TreeItem("Root.Parent.Child").Text = "Modified"; dialog.MyDialog.MyTreeControl.Selection = "Root.Parent";

	dialog.MyDialog.MyTreeControl.TreeItem("Root.Parent.Modified").Delete();

Win32 Control Tagged Values

Various properties that can affect the appearance and behavior of each Win32 dialog and control can be customized at design time via Tagged Values provided by the Win32 User Interface profile.

Tagged Values

Some control types support the addition of special Tagged Values that modify their behavior.

Controls such as Buttons, Check Boxes and Radio Buttons can react to GUI events and execute a JavaScript command. To allow a control to respond to an event, create a new Tagged Value with an appropriate name; for example, 'OnClick', then type the JavaScript command into the value.

Tab Controls can use a 'Tabs' Tagged Value to define the tabs that will appear within this control when it is simulated.

Slider Controls, Spin Controls and Progress Controls can use a 'Range' Tagged Value to define the default minimum and maximum values accepted by the control during simulation.

Tag	Description
Columns	**Applies to:** List Control **Use:** Initializes column names and widths for a List Control. Each column name and width is separated by a semi-colon; for example, "Column1;100;Column2;150;".
OnClick	**Applies to:** Button **Use:** Identifies the JavaScript command to be executed in response to a click event on a Button control.
OnCheck	**Applies to:** Check Box **Use:** Identifies the JavaScript command to be executed in response to a change in the value of a Check Box control.
OnChangeSelection	**Applies to:** Radio Button **Use:** Identifies the JavaScript command to be executed in response to a change in the value of a Radio Button control.
Range	**Applies to:** Slider Control, Spin Control, Progress Control **Use:** Specifies the default minimum and maximum values for the control, separated by a semi-colon: for example, "1;100".
Tabs	**Applies to**: Tab Control **Use:** Specifies the name of each tab to be created for the Tab Control, separated by a semi-colon: for example, "Tab 1;Tab 2;Tab 3;".

BPMN Simulation

BPMN simulation is a method for visualizing and validating the behavior of your BPMN Business Process diagrams. With visual indications of all currently executing activities and the possible activities that can be executed next, you will easily be able to identify and resolve potential issues with the process you have modeled.

Simulating BPMN models is similar to simulating standard UML Behavioral models, except that BPMN:

- Uses some different element types (such as Gateway instead of Decision) and
- Operates on scripting placed, generally, in the appropriate 'Tagged Value' field associated with the connectors and elements, instead of in the 'Properties' fields (and, if you prefer, rather than in the 'Execution Analyzer Build Scripts' dialog); the scripting is written in JavaScript

Working with BPMN Simulation

Activity	Detail
Create a BPMN Simulation Model	When you create a BPMN model suitable for simulation, you take into consideration how you represent the start point, the flow and the conditions to be tested.
Compare UML Activities to BPMN Processes	The simulation of BPMN Business Process models has a number of differences to the simulation of UML Activity diagrams.

Notes

- BPMN simulation is available in the Unified and Ultimate Editions of Enterprise Architect

Create a BPMN Simulation Model

As part of the process of developing a simulation model, consider which of the three options for performing the simulation you prefer to apply:

- Execute a simulation script to initialize the variables for the diagram - select 'BPMN' as the Platform, execute the simulation as 'As Script' and select the script; you would then define the conditions and decisions as JavaScript declarations within the Tagged Values of the elements and connectors on the diagram, either before you start the simulation or during the simulation
- Do not use a script, but initialize the variables within the first Activity and, again, modify the conditions and decisions within the Tagged Values of the elements and connectors, then execute the simulation as 'Interpreted'; you can then re-initialize the variables during simulation, as well as the conditions
- Execute the simulation as 'Manual' and manage the flow and conditions manually at each step

Create a BPMN diagram suitable for simulation

Step	Action
1	Create a Business Process or BPEL diagram from the BPMN 2.0 technology. If you create a BPEL diagram Enterprise Architect displays specialized dialogs to streamline the creation of compliant models.
2	We recommend that you create a Start Event to clearly show where your simulation starts. You have several choices for the Event Type; the choice does not influence the simulation of your model. If no Start Events are defined, the simulation will start from an Activity that has no incoming Sequence Flows.
3	Add all of the Activities that are involved in the Process being modeled. You have several choices for the Task Type; the choice does not influence the simulation of your model. The behavior of Activities can be further decomposed by specifying an Activity Type of Sub-Process and selecting Embedded or CallActivity. Standard Loops are also supported.
4	Add Sequence Flows between your activities. In the 'BPEL properties' dialog you can enter the condition that must be satisfied (True) before the Sequence Flow will be followed. You can also set the conditionType to 'Default' to ensure that this flow will be taken if all other branches fail the condition specified. If you are not working with a BPEL diagram, you use the conditionExpression and conditionType Tagged Values.
5	Add End Events for any conditions that will cause the process or active execution path to end. You have several choices for the Event Type; of these only the Terminate type will influence the execution. In simulations with multiple active nodes, it causes the entire process to terminate instead of just the thread that reaches that node.

Notes

- To include Activities that are in Packages external to the Package being simulated, either draw a:
  - Package Import connector from the Package containing the diagram
    being simulated to each external Package, or
  - Dependency connector from the Package containing the diagram
    being simulated to each Activity in the external Packages

Initialize Variables and Conditions

For a BPMN simulation model, you can initialize your variables in an Execution Analyzer script. You can also initialize these variables in the Tagged Values of the first Activity element of the process, which gives you greater flexibility in adding and changing variables as the simulation proceeds. Similarly, you can define the conditions and values to apply at the various decision points (Gateways) in the process, in the Tagged Values of the Sequence Flow connectors.

If you want to incorporate a user-interface into your simulation process, using Win32, you again use Tagged Values to identify the dialog or prompt to display, in the Activity element just prior to the point at which the value or decision is processed.

For the simulation of UML diagrams, variables inside the 'sim' object and 'this' object are displayed in the Local Variables window.

Access

Display the 'Tags' tab of the Properties window, using one of the methods outlined here.

Ribbon	Explore > Portals > Windows > Properties > Properties > Tags
Keyboard Shortcuts	Ctrl+2 > 'Tags' tab of the Properties window

Initialize Variables

1. On the diagram, click on the first Activity element in the process.
2. In the 'Tags' tab of the Properties window, click on the drop-down arrow of the taskType 'value' field, and select 'Script'.
3. In the script 'value' field, type in the appropriate JavaScript code, such as:

   sim.loan=true; sim.status="undefined';

Define Conditions

1. On the diagram, click on a Sequence Flow connector that issues from a Gateway element.
2. In the 'Tags' tab of the Properties window, click on the drop-down arrow of the conditionType 'Value' field, and select 'Expression'.
3. In the conditionExpression 'Value' field (<memo>*) click on the ... button to display the Tagged Value Note window. Type in the appropriate JavaScript code, such as:
   sim.status=="Hold"
4. Click on the OK button. The statement text displays as a label of the connector.

Incorporate Win32 User Interface

1. On the diagram, click on the Activity element that represents where the decision is made.
2. In the 'Tags' tab of the Properties window, click on the drop-down arrow of the 'taskType value' field, and select

'Script'.

3. In the 'script value' field, type in the appropriate JavaScript code, such as:

   ```
   dialog.Screen1.Show=True;
   ```

   (This statement displays the dialog Screen1. You can temporarily hide the dialog by changing 'Show' to False.)

Comparison of UML Activities and BPMN Processes

The execution and simulation of BPMN models have a number of differences from the execution and simulation of UML Activity diagrams. The mapping of similar concepts, and the differences between the two methods of expressing the behavior of a system, are presented here.

Comparison of UML Activities and BPMN Processes

UML Activity	BPMN Business Process
The starting point is defined by an Initial Node. No method of specifying why the Activity was started is available.	The starting point is defined by a Start Event. This implies a specific cause for the Activity to start, although it could be unspecified.
The basic behavior unit in an Activity is the Action element. UML provides many different forms of Actions, although the simulation makes use of a small subset of these.	The basic behavior unit in an Activity is the Activity element. A number of different Task Types are available. These typically describe different methods of execution (for example Manual) as opposed to what happens.
A Control Flow is used to connect the elements on an Activity diagram. A distinguishing feature is that only a single Control Flow can be followed from any node, except for an explicit Fork Node. To restrict flow on a Control Flow, add a Guard.	A Sequence Flow is used to connect the elements on a Business Process diagram. These differ from UML Activity diagrams in that all valid sequence flows are taken by default. To restrict flow on a Sequence Flow set the conditionType Tagged Value to 'Expression' and create the script in the conditionExpression Tagged Value.
A Decision node is used to explicitly model a decision being made. A Merge node, which uses the same syntax is used when the potential flows are combined back into one.	A Gateway node set to 'Exclusive' is used when a single path must be selected. It is also used to combine the potential flows again. A direction can be specified as 'Converging' or 'Diverging' to explicitly select between the two modes.
A Fork node is used to concurrently execute multiple nodes, while a Join node, using the same syntax is used to wait for all incoming flows to become available and leave with a single flow.	A Gateway node set to 'Parallel' is used to explicitly model concurrent execution of multiple nodes. It is also used to wait for all incoming flows to become available and leave with a single flow. A direction can be specified as 'Converging' or 'Diverging' to explicitly select between the two modes.
There is no allowance for concurrently executing	A Gateway node set to Inclusive is used to explicitly model the situation where all outgoing flows with a true condition are executed concurrently.

only some outputs from a node for UML Activities. If you needed this you add later Control Flows with the appropriate Guards.	
A Call Behavior Action is used when behavior needs to be further decomposed by referring to an external activity.	Activity elements are set as a CallActivity Sub-Process when behavior needs to be further decomposed by referring to an external activity.
Activity Action Call Behavior Action.	Activity elements are set as an Embedded Sub-Process when behavior needs to be further decomposed without referring to an external activity.

BPSim Business Simulations

The open BPSim specification provides a rich set of materials on how to configure and assign resources to activities or tasks, how to raise events, decision making and other real-world capabilities. Once configured according to the BPSim specification, a business process model (constructed in BPMN) can be passed to a suitable BPSim simulation engine and run according to the process defined in the BPMN model, using the configuration data attached in the BPSim information.

The BPSim specification is very detailed and offers the interested modeler and business strategist an unprecedented flexibility in assigning operating information to a model and then assessing the quality of the solution based on information received back from the Simulation engine. This section describes in detail the various screens and options available when configuring a model for BPSim execution.

Sparx Systems provide a BPSim-capable Simulator - the **BPSim Execution Engine.** This Add-In integrates with the BPSim and BPMN models defined in Enterprise Architect, providing the capability to run and store the results from multiple simulations and to perform convenient comparisons across each configuration's result set.

The **BPSim Execution Engine** is a pre-requisite for accessing and using the BPSim configuration facilities. The Execution Engine is integrated with the Unified and Ultimate editions of Enterprise Architect; for use in the Corporate edition, it can be purchased and installed under separate licence.

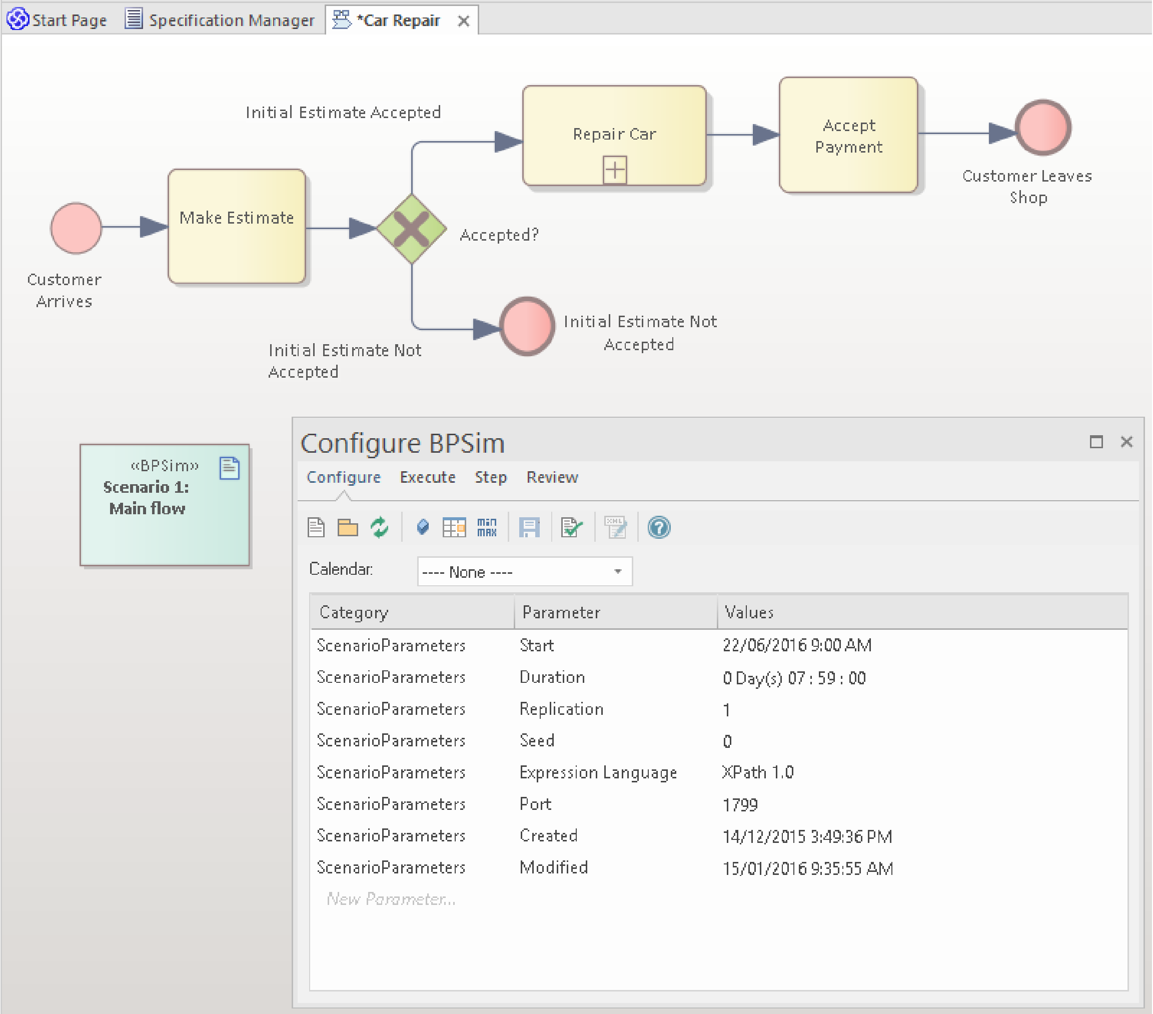
Once you have set up a BPSim configuration, the simulation execution process exports the BPMN model with its BPSim data in a standard form. This ensures that changes to the model are always incorporated into the simulation. Similarly, the model export process captures the BPMN model with its BPSim data in a form that can be imported into another model and consumed by the Sparx Systems BPSim Execution Engine or by any other standards-compliant BPSim engines.

Install BPSim

Whilst BPSim is integrated with the Unified and Ultimate editions of Enterprise Architect, it is separate from the Corporate edition and - after purchase - has to be installed on your system.

For all three editions, you must ensure that the right versions of Java Runtime Environment (JRE) and Java Development Kit (JDK) are also installed on your system.

BPMN Model With BPMN Simulation



Calendar: ---- None ----

Category	Parameter	Values
ScenarioParameters	Start	22/06/2016 9:00 AM
ScenarioParameters	Duration	0 Day(s) 07 : 59 : 00
ScenarioParameters	Replication	1
ScenarioParameters	Seed	0
ScenarioParameters	Expression Language	XPath 1.0
ScenarioParameters	Port	1799
ScenarioParameters	Created	14/12/2015 3:49:36 PM
ScenarioParameters	Modified	15/01/2016 9:35:55 AM
New Parameter...		

The Configure BPSim window helps you to define several categories of Simulation parameter, each category focusing on one aspect of the Simulation Configuration. For example, you would define:

- ScenarioParameters, which define how the Simulation itself should proceed
- Control Parameters, which examine how activity flows through the business process, moderated by the likelihood of a sequence of events and the priorities of certain events
- Time (Temporal) Parameters, which examine how the duration of one or more phases in the processing of an Activity influences the business process
- Resource Parameters, which examine the involvement of types and roles of workers and other resources, their required numbers, their costs and their availability

You can also maintain multiple versions of a configuration (as separate BPSim Artifacts) and easily compare the differences between versions to see how each configuration will vary the flow of the proposed Simulation or process execution. You might, for example, establish a baseline configuration and then create multiple 'what-if?' configurations that vary one or more parameters. Once you have run these configurations through a Simulation Engine, you can examine each result and decide on the relative merits of each configuration. One useful principle to apply here is the simple inheritance of common, unchanged data in one configuration by another configuration that contains only the data being varied - you can therefore run the simulation on a current set of variables, which draws on the standard data configuration at the same time.

Users can combine the BPSim and Charting facilities to quickly vary, simulate and compare aspects of a Business Process model, and show the differences between the Simulations in one of the many Chart formats.

If you are working across multiple projects, you can export and import the BPSim configurations between them. The configuration automatically carries with it the BPMN 2.0 model on which it is based.

The Enterprise Architect Business Process Simulation configuration tool is based on the BPSim Framework developed by the Workflow Management Coalition (WfMC).

Notes

- If you click on a business process element or connector in a diagram or in the Browser window, it is highlighted and selected in the Configure BPSim window
- The Business Process that you simulate can contain elements from more than one Package; to include the external elements in the simulation, you must create a Package diagram containing the 'parent' Package and either the 'external' Packages containing the external elements, or the external elements themselves; create a:
  - Package Import connector from the parent Package to each external Package, or
  - Dependency connector from the parent Package to each external element

Installing BPSim

Whilst BPSim is integrated with the Unified and Ultimate editions of Enterprise Architect, it has to be installed on your system. For these editions, you must ensure that the right versions of Java Runtime Environment (JRE) and Java Development Kit (JDK) are also installed on your system.

Install JDE and JDK

To use the Sparx Systems BPSim Execution Engine, you must have on your system Java Runtime Environment (JRE) version 1.7 or higher and, if your BPSim Configuration contains any property parameters, you must also have Java Development Kit (JDK) version 1.7 or higher.

You do not need to do any further configuration for the engine unless you have multiple versions of JRE/JDK on your system and you want to specify which version the execution engine should use. In this case, apply these environment variables as shown:

1. Click on the Windows 'Start' icon and select the 'Computer' option.
2. From the banner menu, select the 'System properties' option.
3. From the side panel, select the 'Advanced system settings' option.
4. On the 'Advanced' tab of the 'System Properties' dialog, click on the Environment Variables button.
5. On the 'Environment Variables' dialog, in the 'System variables' panel, click on the New button.
6. On the 'New System Variable' dialog, complete the fields with the values shown:

   For JRE: Variable name: MDG_BPSIM_JRE_HOME
   Variable value: C:\Program Files\Java\jre7

   For JDK: Variable name: MDG_BPSIM_JDK_HOME
   Variable value: C:\Program Files\Java\jdk1.7.0_51

7. Click on the OK button.
8. You must re-start your machine for the new variables to take effect.

BPSim Configuration

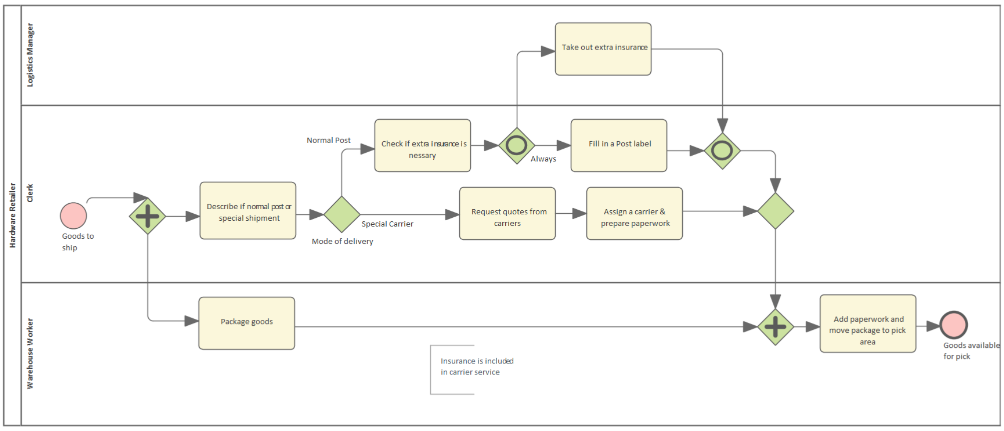
A Business Process Simulation (BPSim) configuration is represented and contained in a Business Process Simulation Artifact element, which you can create on a diagram in any Package in the same project as the BPMN model that you are working with.

Create a Business Process Model

Each BPSim configuration is created specifically for and from an existing Business Process, defined in BPMN. Therefore you will need to create or import the BPMN model on which the configuration is to be based, before you use the Business Process Simulation Artifact.

This example diagram can be found and worked on in the EAExample model, in:

Analysis and Business Modeling > BPMN 2.0 Examples > Process Diagrams > Shipment Process of a hardware retailer


Hardware Retailer
Logistics Manager
Clerk
Warehouse Worker
Take out extra insurance
Normal Post
Check if extra insurance is nessary
Always
Fill in a Post label
Goods to ship
Describe if normal post or special shipment
Special Carrier
Mode of delivery
Request quotes from carriers
Assign a carrier & prepare paperwork
Package goods
Insurance is included in carrier service
Add paperwork and move package to pick area
Goods available for pick

Create a Business Process Simulation Artifact

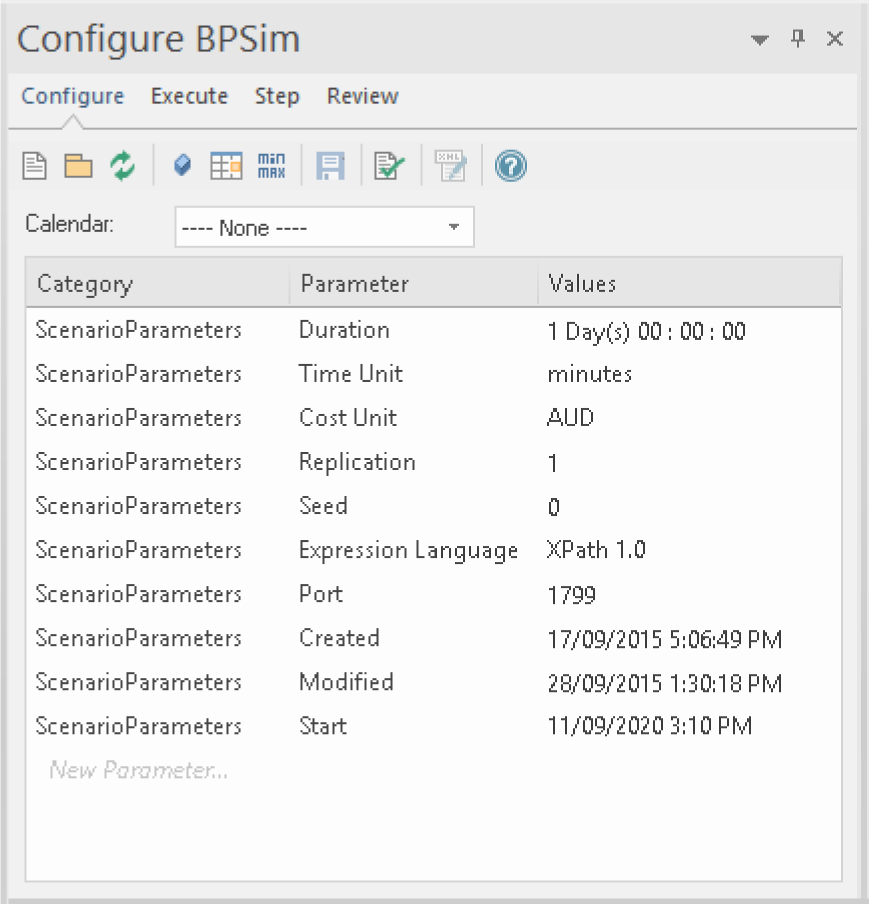
Open a diagram in which to create the Artifact, and display the Diagram Toolbox (press Ctrl+Shift+3). Expand the common 'Simulation' page, and drag the 'Business Process Simulation' icon onto the diagram.

When you set up the Artifact, consider whether you might create one Artifact to define the base configuration, and other Artifacts to define variations or additions in certain aspects of the simulation. You would use Generalization connectors between the 'base' and 'variation' Artifacts so that the variations inherit the data you have defined in the 'base' Artifact. This way, you do not have to keep re-defining the whole configuration in every Artifact you create.

Double-click on the element and give it an appropriate name, such as (for the example) 'Base BPSim Configuration'.

Configure BPSim Window Overview

Right-click on the Artifact element (either in the diagram or in the Browser window) and select the 'Configure BPSim' option. The Configure BPSim window displays for the Artifact.



Calendar: ---- None ----

Category	Parameter	Values
ScenarioParameters	Duration	1 Day(s) 00 : 00 : 00
ScenarioParameters	Time Unit	minutes
ScenarioParameters	Cost Unit	AUD
ScenarioParameters	Replication	1
ScenarioParameters	Seed	0
ScenarioParameters	Expression Language	XPath 1.0
ScenarioParameters	Port	1799
ScenarioParameters	Created	17/09/2015 5:06:49 PM
ScenarioParameters	Modified	28/09/2015 1:30:18 PM
ScenarioParameters	Start	11/09/2020 3:10 PM
New Parameter...		

This window contains four tabs: Configure, Execute, Step and Review.

- Configure: configure BPSim parameters for each BPMN element; define Property parameters, Calendars and Scenario parameters
- Execute: execute the BPMN model with a BPSim configuration
- Step: step over / step in to provide an insight into the execution process, including token status, property values and resource allocations per time/step
- Review: review / compare configuration artifact(s), generate standard or customized simulation result report(s)

BPSim - Configure Page

The BPSim Artifact will be configured to a Package. All BPMN elements under this Package or its sub-Packages will be loaded. By default, the Package containing this Artifact will be configured when loaded into this window.

This window is context sensitive. When an element is selected on a diagram or the Browser window, the list will show current configurations for the element; also, the drop-down lists of values will only show available parameters for the element.

When the BPSim Artifact is the context element, the list will show the ScenarioParameters.

Access

Ribbon	Simulate > Process Analysis > BPSIM > Open BPSIM Manager > Configure

Toolbar Options

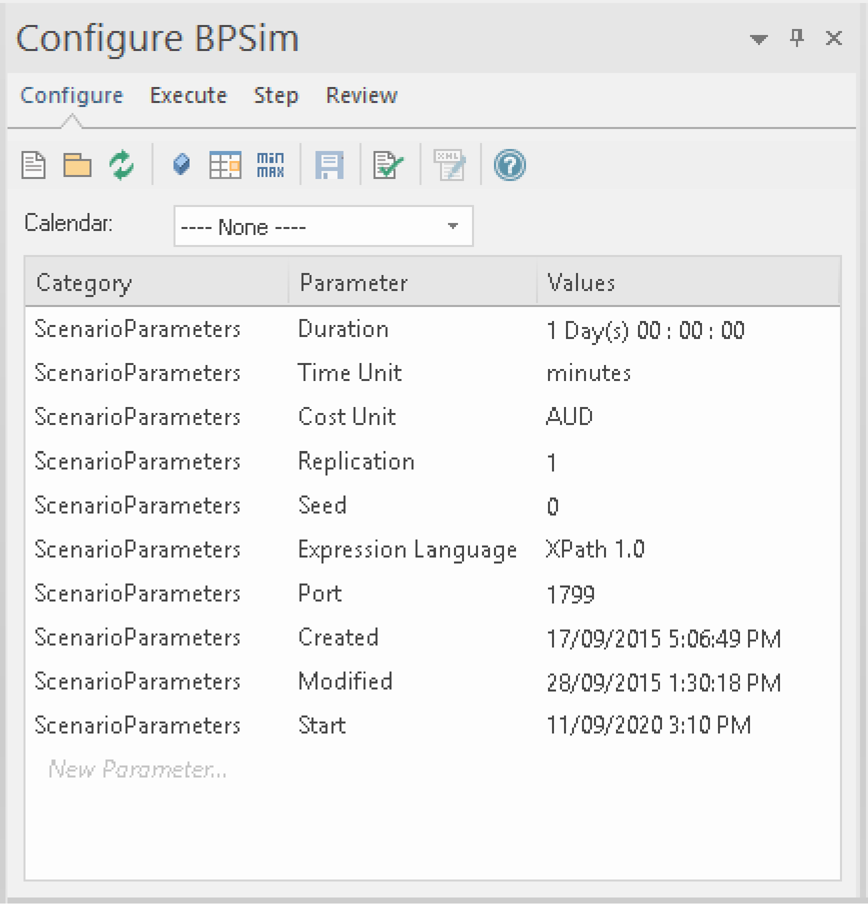
Option	Description
📄	Click on this button to select or create a BPSimConfiguration element.
📁	Click on this button to set a Package for the BPSim Artifact. All BPMN elements under this Package or its sub-Packages will be included.
🔄	Click on this button to reload BPMN elements from the configured Packages. For example, when some BPMN elements are modified, run this command to reload the Package so that the changes will be taken into account for BPSim Simulation.
🔷	Click on this button to define Properties, which can be used as Property Parameters on BPMN elements.
📅	Click on this button to define Calendars, which can be used to configure element parameters.
min max	Click on this button to show or hide the 'Result Request' column. Result Request configuration is required for a Custom simulation. The execution report will only contain results that are requested.
💾	Click on this button to save the Configure BPSim window information to a BPSim Artifact element.
✓	Click on this button to validate the BPMN model and the BPSim configurations. Error or warning messages might display in the System Output window if they are generated.
XML	Click on this button to export the BPMN model with the BPSim configuration. This exported BPMN file conforms to the BPMN and BPSim specifications and can be used by third-party BPSim Execution engines.

Scenario Parameters

A scenario is composed of a collection of element parameters. The scenario itself defines parameters used by all elements as global settings. Not all parameters will display for an element, but you can bring them in to the list by:

1. Clicking on the *New Parameter* text, clicking on the drop-down arrow and selecting 'ScenarioParameter'.
2. Clicking on the drop-down arrow in the 'Parameter' field and then selecting the parameter type from the list.

Note that once you have added all possible parameters for a scenario, the Configure BPSim window does not allow you to attempt to add more.



Calendar: ---- None ----

Category	Parameter	Values
ScenarioParameters	Duration	1 Day(s) 00 : 00 : 00
ScenarioParameters	Time Unit	minutes
ScenarioParameters	Cost Unit	AUD
ScenarioParameters	Replication	1
ScenarioParameters	Seed	0
ScenarioParameters	Expression Language	XPath 1.0
ScenarioParameters	Port	1799
ScenarioParameters	Created	17/09/2015 5:06:49 PM
ScenarioParameters	Modified	28/09/2015 1:30:18 PM
ScenarioParameters	Start	11/09/2020 3:10 PM
New Parameter...		

Name	Description
Start	The date and time at which the process starts to take effect. You can edit this by overtyping the values or, for the date, by selection from a drop-down calendar.
Duration	The length of time the process takes. The 'Duration' parameter is a required value. It must be long enough to accommodate a complete simulation; for example, if a process (and hence its simulation) takes three hours to complete, the 'Duration' parameter must be set to a value greater than three hours. You can edit this by overtyping the appropriate segment in the format 'days hours:minutes:seconds'.
Time Unit	The base unit in which periods of time are expressed in this scenario. All numeric

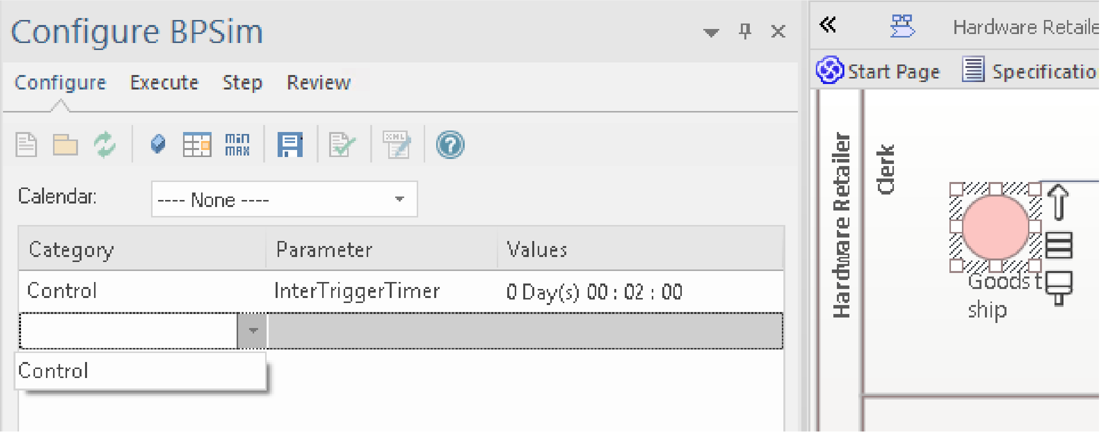
	and floating values representing time should be considered as being expressed in this unit, unless overridden locally. You can edit this by clicking on the drop-down arrow and selecting the unit.
Cost Unit	The currency unit of any costs recorded in the process. All numeric and floating values representing a cost should be considered as being expressed in that currency code, unless overridden locally. You can edit this by clicking on the drop-down arrow and selecting the unit abbreviation.
Replication	The number of replications of the scenario to be executed. Defaults to 1. You can edit this by simply typing a value in.
Seed	A random seed to be used to initialize a pseudo-random number generator. You can edit this by simply typing a value in.
Expression Language	XPath 1.0 and Java - XPath 1.0 is the default language. If Java is specified as the expression language, JDK Home must be set. You can edit this by clicking on the drop-down arrow and selecting the language.
DMN Module	When business rule tasks are used in the BPMN Model, you can implement these tasks as a DMN Model. You might first create a DMN Model and generate a DMN Server in Java, then click on the … button to specify the generated DMN Server file.
JRE Home	The Enterprise Architect BPSim Execution Engine runs in a Java Environment, therefore a JRE Home has to be specified. Click on the … button to choose a directory; for example, C:\Program Files\Java\jre7. You can edit this by clicking on the … button again to browse the directory.
JDK Home	When the expression language is Java, the Enterprise Architect BPSim Execution Engine will generate Java code and compile with javac as the vendor extension. So a JDK Home must be specified. Use the … button to choose a directory (such as C:\Program Files\Java\jdk1.7.0_80). You can edit this by again clicking on the … button to browse the directory.
Port	The Port number that Enterprise Architect used to communicate with the BPSim Execution Engine. The default Port number is 1799.
Created	Read only field. The timestamp when the BPSim Artifact was created.
Modified	Read only field. The timestamp for when the BPSim Artifact was last modified.

Control Parameters

To begin defining Control parameters for the appropriate element (such as an Event or Gateway):

1. Select the element on the diagram, then click on the *New Parameter* text and on the drop-down arrow in the

'Category' column, and select 'Control'.

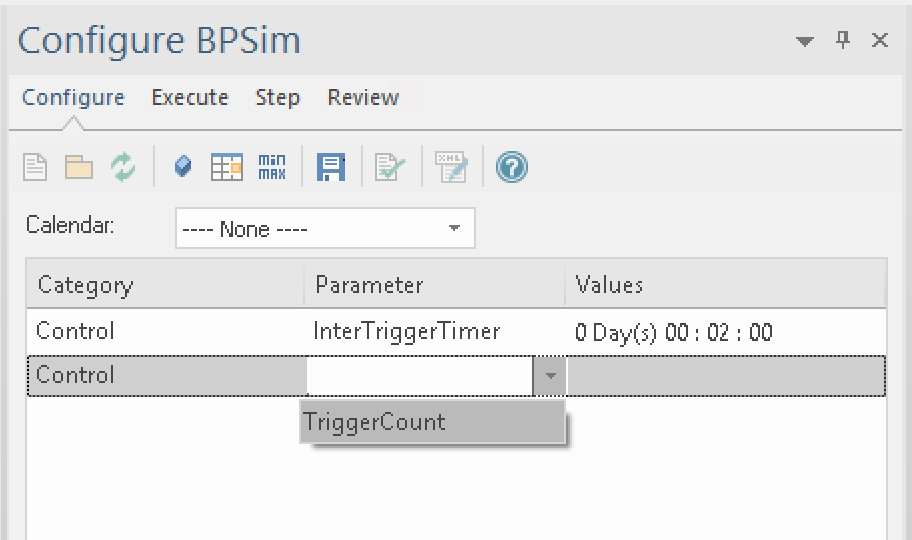


Calendar: ---- None ----

Category	Parameter	Values
Control	InterTriggerTimer	0 Day(s) 00 : 02 : 00

Control

2. Click on the drop-down arrow in the 'Parameter' field, which will display the available unassigned parameters for the selected element.

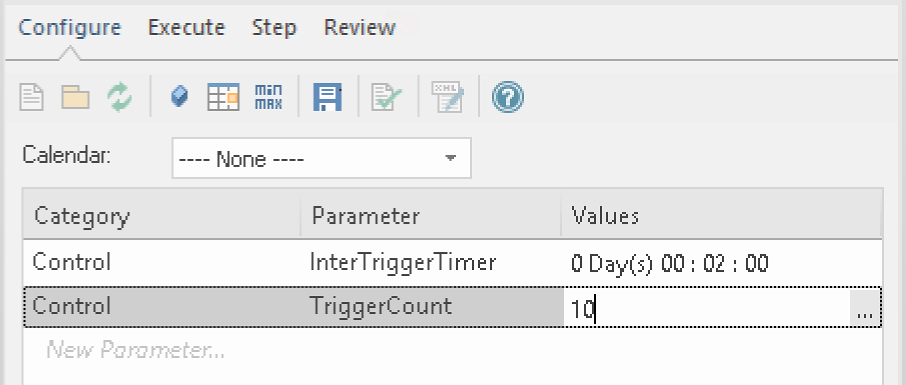


Calendar: ---- None ----

Category	Parameter	Values
Control	InterTriggerTimer	0 Day(s) 00 : 02 : 00
Control		

TriggerCount

3. Select the appropriate parameter, and then click on the 'Values' field; you can either type the parameter value in the field or use the ... button to open the 'Parameter Value' dialog.



Calendar: ---- None ----

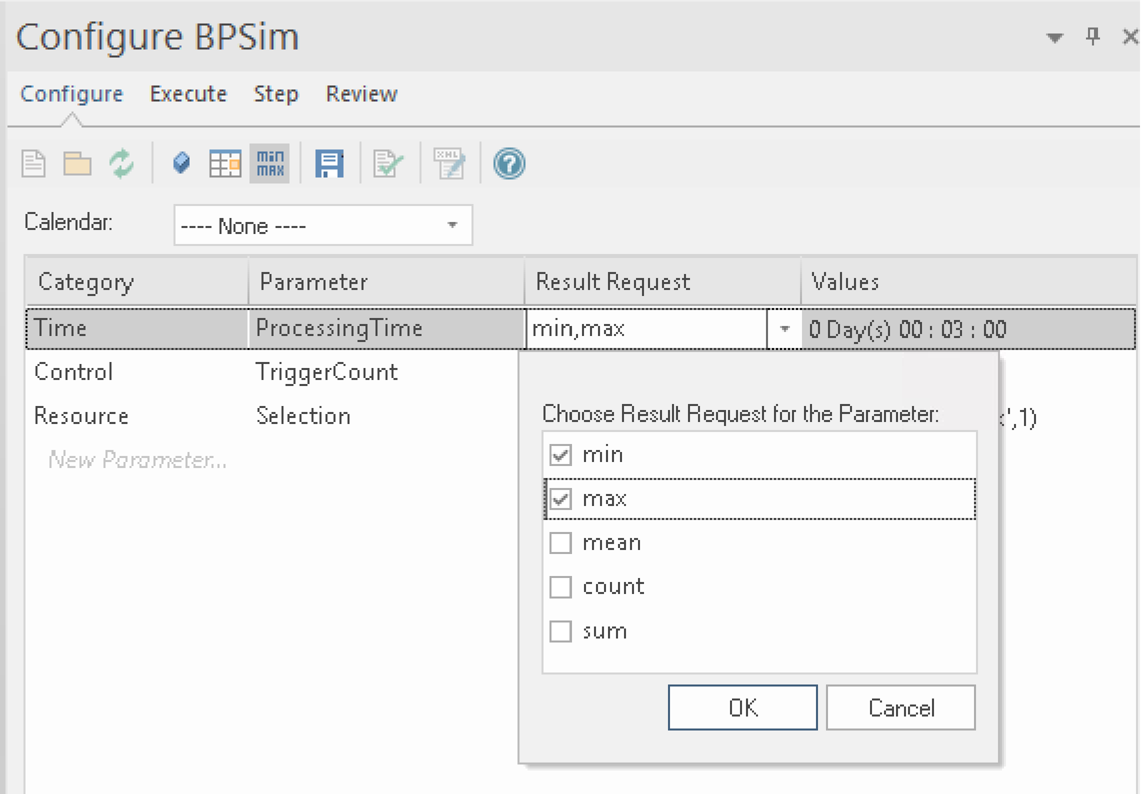
Category	Parameter	Values
Control	InterTriggerTimer	0 Day(s) 00 : 02 : 00
Control	TriggerCount	10 ...
New Parameter...		

Note that the page allows you to provide only the appropriate parameters for the element. Once you have specified these parameters, the fields do not allow further input or selection.

Temporal Parameters

To begin defining Time parameters for the appropriate element (such as a BPMN Task):

1. Select the element on the diagram.
2. Click on the *New Parameter* text and on the drop-down arrow, and select 'Time' from the list.
3. After you select 'Time', click on the drop-down arrow in the 'Parameter' field and select from the available parameters for the element.
4. In the 'Values' field, either type the value or click on the ... button to open the 'Parameter Value' dialog.
5. You can toggle the 'Result Request' column by clicking on the min/max button on the toolbar to customize the simulation output by requiring certain results



Calendar: ---- None ----

Category	Parameter	Result Request	Values
Time	ProcessingTime	min,max	0 Day(s) 00 : 03 : 00
Control	TriggerCount		
Resource	Selection		',1)
New Parameter...			

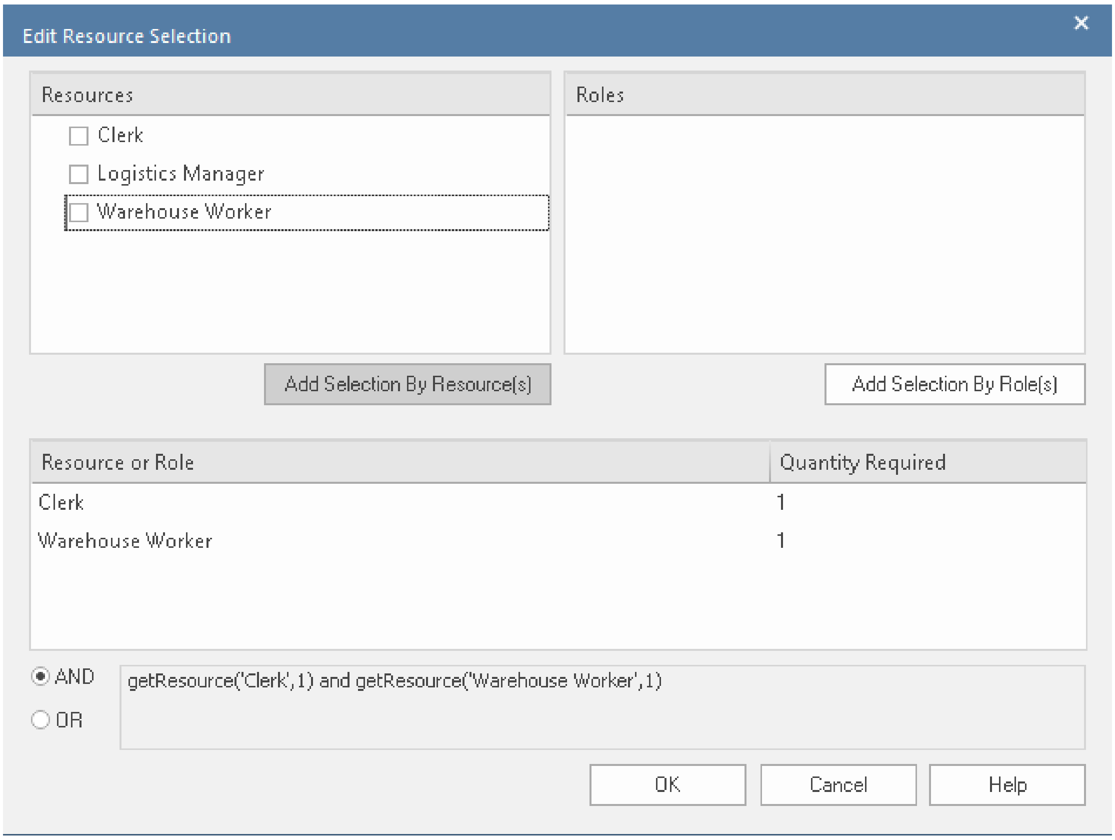
Choose Result Request for the Parameter:

☑ min
☑ max
☐ mean
☐ count
☐ sum

Resource Parameters

To begin defining the Resource parameters for the appropriate element (such as a BPMN Task):

1. Select the element on the diagram.
2. Click on the *New Parameter* text and on the drop-down arrow, and select 'Resource' from the list.
3. In the 'Parameter' field, click on the drop-down arrow and click on 'Selection' in the list.
4. In the 'Values' field, click on the ... button to open the 'Edit Resource Selection' dialog.



Resources

- □ Clerk
- □ Logistics Manager
- □ Warehouse Worker

Roles

Resource or Role	Quantity Required
Clerk	1
Warehouse Worker	1

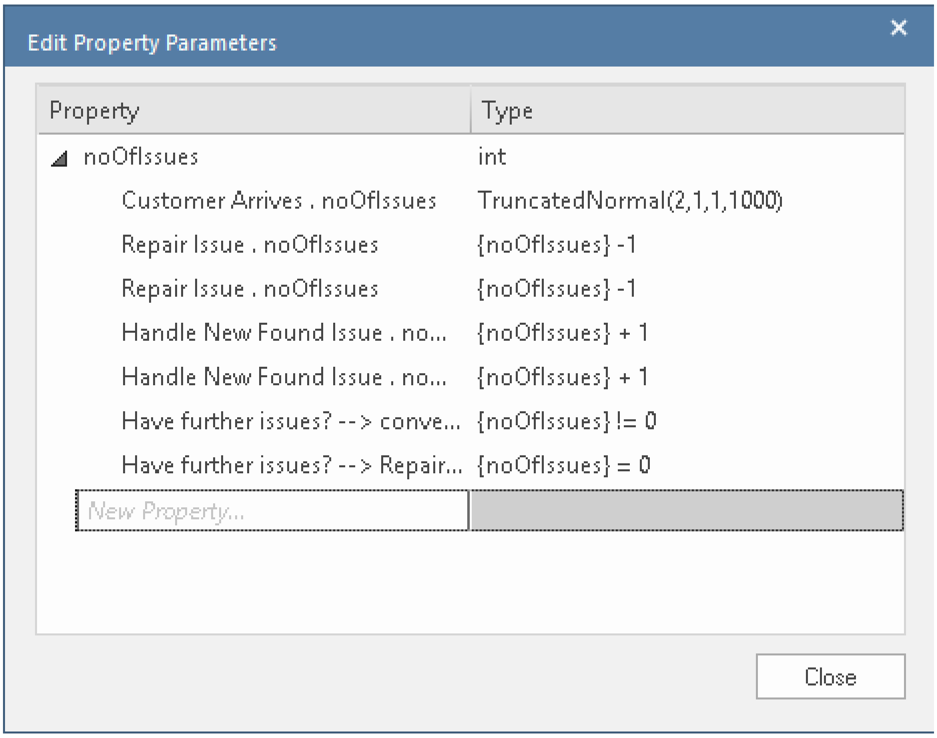
● AND
○ OR

getResource('Clerk',1) and getResource('Warehouse Worker',1)

- The top left panel lists the defined Resource elements; click on the resources to assign and on the Add Selection By Resource(s) button to move the selection to the 'Resource or Role' panel
- The top right panel lists the defined Roles (if any) for the Resource elements; click on the required roles and on the Add Selection By Role(s) button to move the selection to the 'Resource or Role' panel
- The 'Quantity Required' column defaults to 1 for each resource/role; if a larger quantity is required, overtype this value with the appropriate number
- Click on the appropriate radio button to set the logical relationship to AND or OR for the selection
- The final expression for Resource selection is composed and shown in the text field
- Click on the OK button to return to the Configure BPSim window, where the expression is shown in the 'Values' field

Property Parameters

To begin defining the Property parameters, click on the button on the toolbar. The 'Edit Property Parameters' dialog displays.

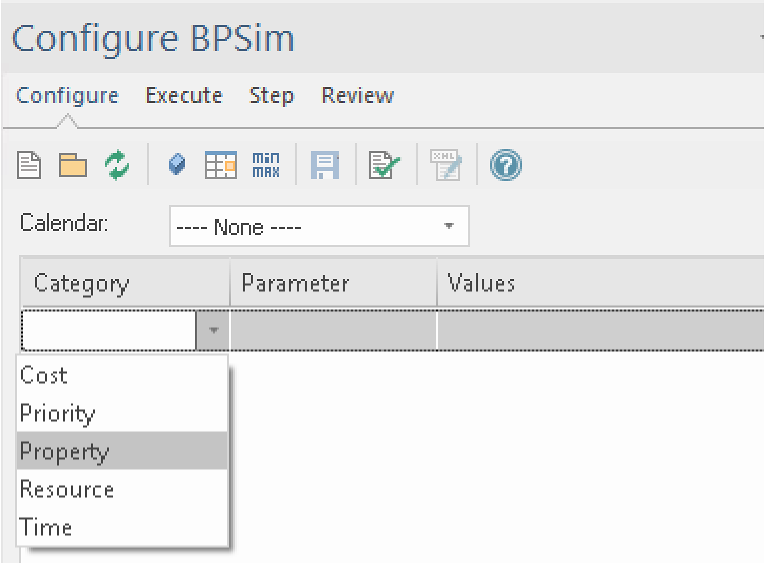


Property	Type
noOfIssues	int
Customer Arrives . noOfIssues	TruncatedNormal(2,1,1,1000)
Repair Issue . noOfIssues	{noOfIssues} -1
Repair Issue . noOfIssues	{noOfIssues} -1
Handle New Found Issue . no...	{noOfIssues} + 1
Handle New Found Issue . no...	{noOfIssues} + 1
Have further issues? --> conve...	{noOfIssues} != 0
Have further issues? --> Repair...	{noOfIssues} = 0
New Property...	

The defined properties and their references are listed.

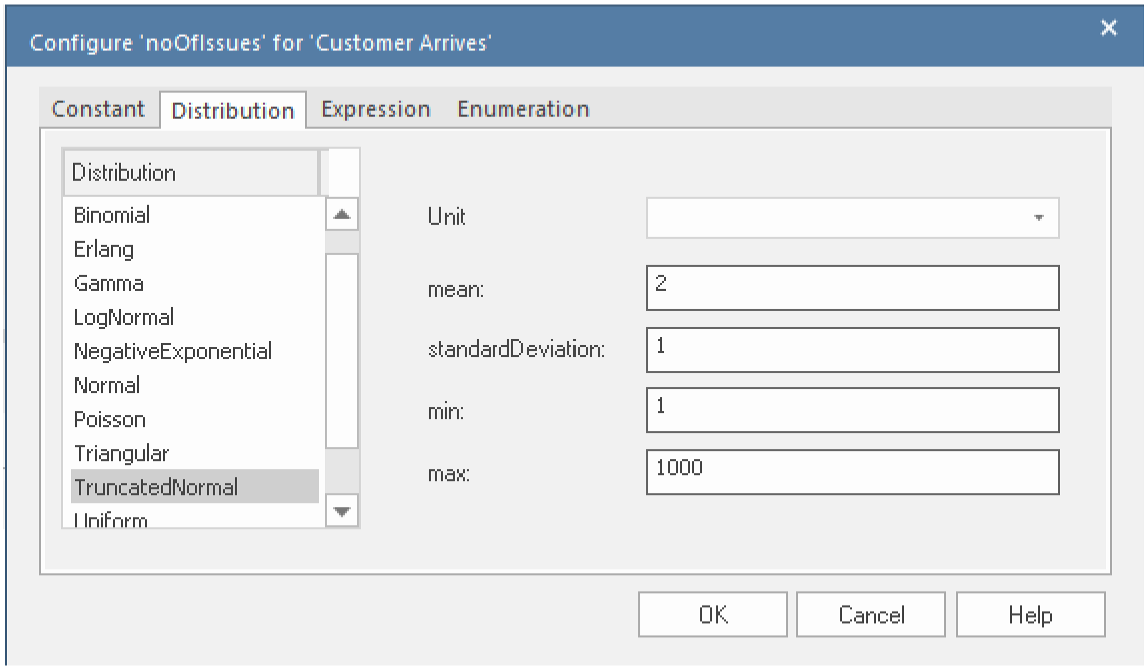
You can add a new property, delete a selected property (using the context menu option), overtype a property's name or select a different type for a property.

Having checked the defined properties, you can set Property parameters on BPMN elements.



Choose 'Property' as a category, then click on the drop-down arrow in the 'Parameter' field and select a property.

Click on the ... button on the 'Values' column to display the parameter value dialog (named from the property and the parent element).



Configure 'noOfIssues' for 'Customer Arrives'

Unit:

mean: 2

standardDeviation: 1

min: 1

max: 1000

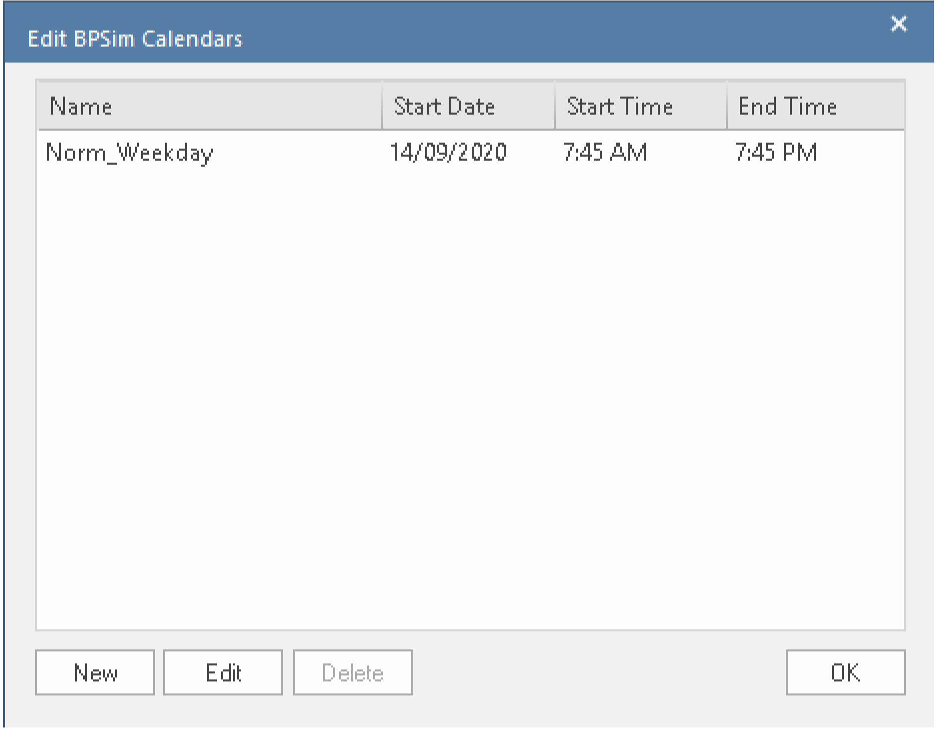
Navigate to the appropriate tab to select and define the type of value and actual value, then click on the OK button. The value displays in the 'Value' field.

Calendars

Calendars help you to define any number of special periods of time that can influence the process, such as working days, shifts, holidays or periodic events (for example, stock-taking, inventory or auditing).

To begin defining the Calendars:

1. Click on the button on the toolbar; the 'Edit BPSim Calendars' dialog displays, showing any existing calendars.

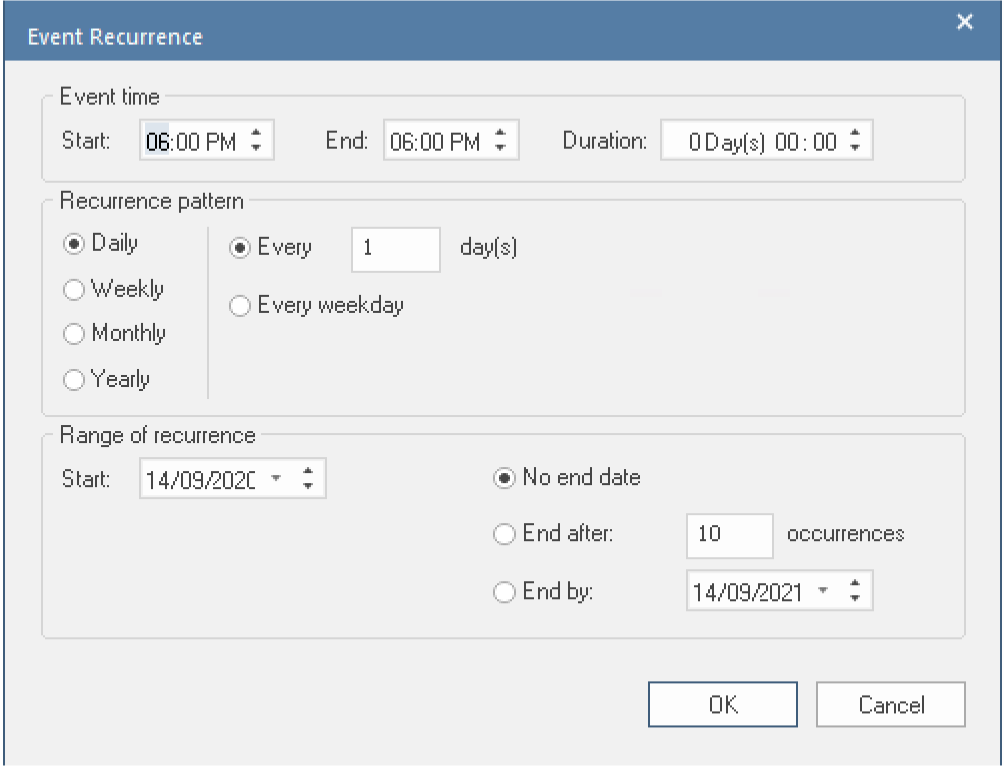


Edit BPSim Calendars

Name	Start Date	Start Time	End Time
Norm_Weekday	14/09/2020	7:45 AM	7:45 PM

You can add a new calendar, or edit or delete a selected calendar.

2. To add a new calendar period, click on the New button to display the 'Event Recurrence' dialog.



Event Recurrence

Event time

Start: 06:00 PM End: 06:00 PM Duration: 0 Day(s) 00 : 00

Recurrence pattern

● Daily
○ Weekly
○ Monthly
○ Yearly

● Every 1 day(s)
○ Every weekday

Range of recurrence

Start: 14/09/202C

● No end date
○ End after: 10 occurrences
○ End by: 14/09/2021

OK Cancel

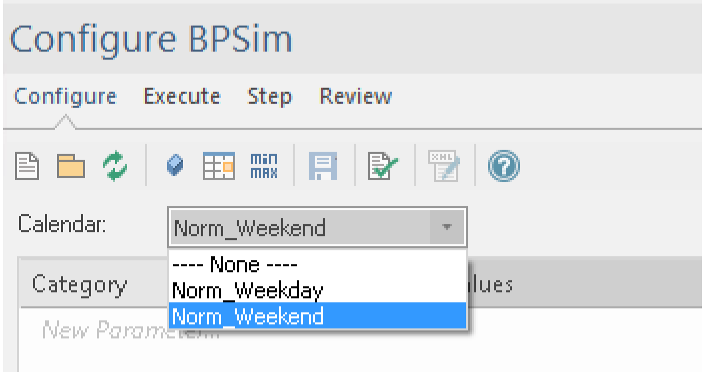
3. In the 'Event time' panel, the 'Start' and 'End' fields both default to the current time. The 'Start' field is the anchor; a change to either the 'End' field or the 'Duration' field automatically updates the other field, in reference to the 'Start' field. Click on the hour and minute segments of each field (and, for the 'Duration' field, the 'Day(s)' segment) separately, and use the 'spin' arrows to set the start time and the end time or duration of the period.

4. In the 'Recurrence pattern' panel, select the radio button for the interval at which the calendar period recurs. Each option displays an appropriate set of fields in the right of the panel for refining that interval to every day/week/month or every two/three/four days/weeks/months, on a particular day of the week, or day or date of the month, or day or date in the year. Select the checkboxes or values in drop-down lists as appropriate.

5. In the 'Range of recurrence' panel, select the date on which the calendar period takes effect and select the appropriate radio button to define when the period ceases to apply - never, after a set number of occurrences, or on a specific date. You can select an end date either from a drop-down calendar or using the 'spin' arrows on each segment of the date.

6. Click on the OK button to set the calendar period.

As you define calendar periods, they are listed in order of the start date and/or time, earliest first.



Configure BPSim

Configure Execute Step Review

Calendar: Norm_Weekend

---- None ----
Norm_Weekday
Norm_Weekend

Category ...lues

New Parameter...

With defined calendars, you can configure parameters on a selected calendar.

Validation

After configuring BPSim parameters for some BPMN elements, click on the ✓ button to run a validation of the simulation. Any BPMN or BPSim errors/warnings will be displayed in the System Output window. Fix the issues according to the messages.

After doing this, proceed to the next Help topic: *BPSim Execute Page*.

BPSim - Execute Page

After defining the configuration, you can choose to conduct a Standard simulation or a Customized simulation. The execution will generate a result report and a list of records used to replay (step through) the simulation.

Access

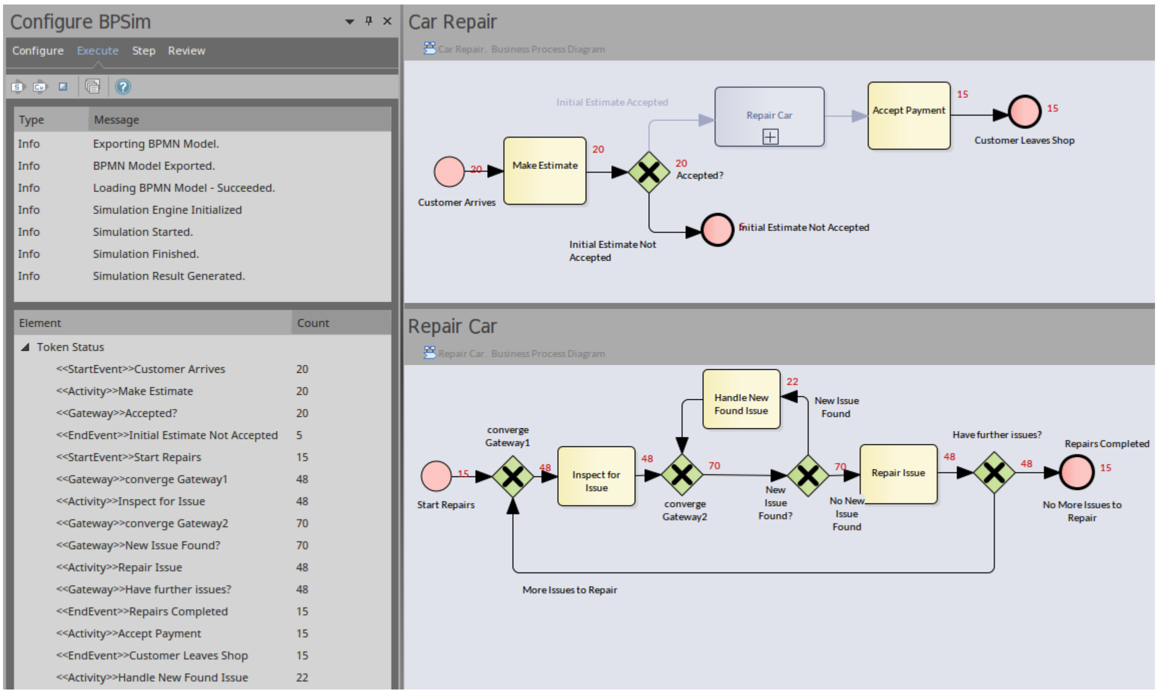
Ribbon	Simulate > Process Analysis > BPSIM > Open BPSIM Manager > Execute page

Toolbar Options

Option	Description
S	Click on this button to execute the BPMN model with the BPSim configuration and generate a standard report.
Cu	Click on this button to execute the BPMN model with the BPSim configuration and generate a customized report based on the 'Result Request' settings made on the 'Configure' page.
■	Click on this button to stop the execution and exit the simulation.
🗐	Click on this button open the generated report in the 'Review' page.

Execution

When you click on the Run Simulation button or Run Customized Simulation button, the BPMN model with BPSim Configuration will be exported and loaded into the Execution engine.



Type	Message
Info	Exporting BPMN Model.
Info	BPMN Model Exported.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Count
Token Status	
<<StartEvent>>Customer Arrives	20
<<Activity>>Make Estimate	20
<<Gateway>>Accepted?	20
<<EndEvent>>Initial Estimate Not Accepted	5
<<StartEvent>>Start Repairs	15
<<Gateway>>converge Gateway1	48
<<Activity>>Inspect for Issue	48
<<Gateway>>converge Gateway2	70
<<Gateway>>New Issue Found?	70
<<Activity>>Repair Issue	48
<<Gateway>>Have further issues?	48
<<EndEvent>>Repairs Completed	15
<<Activity>>Accept Payment	15
<<EndEvent>>Customer Leaves Shop	15
<<Activity>>Handle New Found Issue	22

During the simulation:

- The token status list will flash with runtime values
- The diagram will flash with runtime token counts

However, the simulation might execute too quickly to be able to see this. You can see these changes if you use the 'Step' page to run through the simulation step by step.

In this example, BPMN elements under the process 'Car Repair' and sub-process 'Repair Car' get triggered as new customers arrive at regular intervals.

BPSim - Step Page

After successfully running the execution, the system generates an execution report that tells you the process status in general, such as (for the Car Repair example) the average time of a task, the total waiting time of customers and how many issues are repaired.

In addition, you can inspect the process from various angles. For example:

- From the timestamp - what was the status of this process at 9:30 AM?
- From the token - what did the 3rd customer do in the shop?
- From the property - how does the number of issues decrease and increase for the 2nd car?
- From multiple threads - can I see customers walk in and simulate automatically on the diagram?
- From the resources - when is a support person busy or idle? Why is a customer waiting for 40 minutes?

All of these kinds of question can be answered on the 'Step' page.

Access

Ribbon	Simulate > Process Analysis > BPSIM > Open BPSIM Manager > Step page

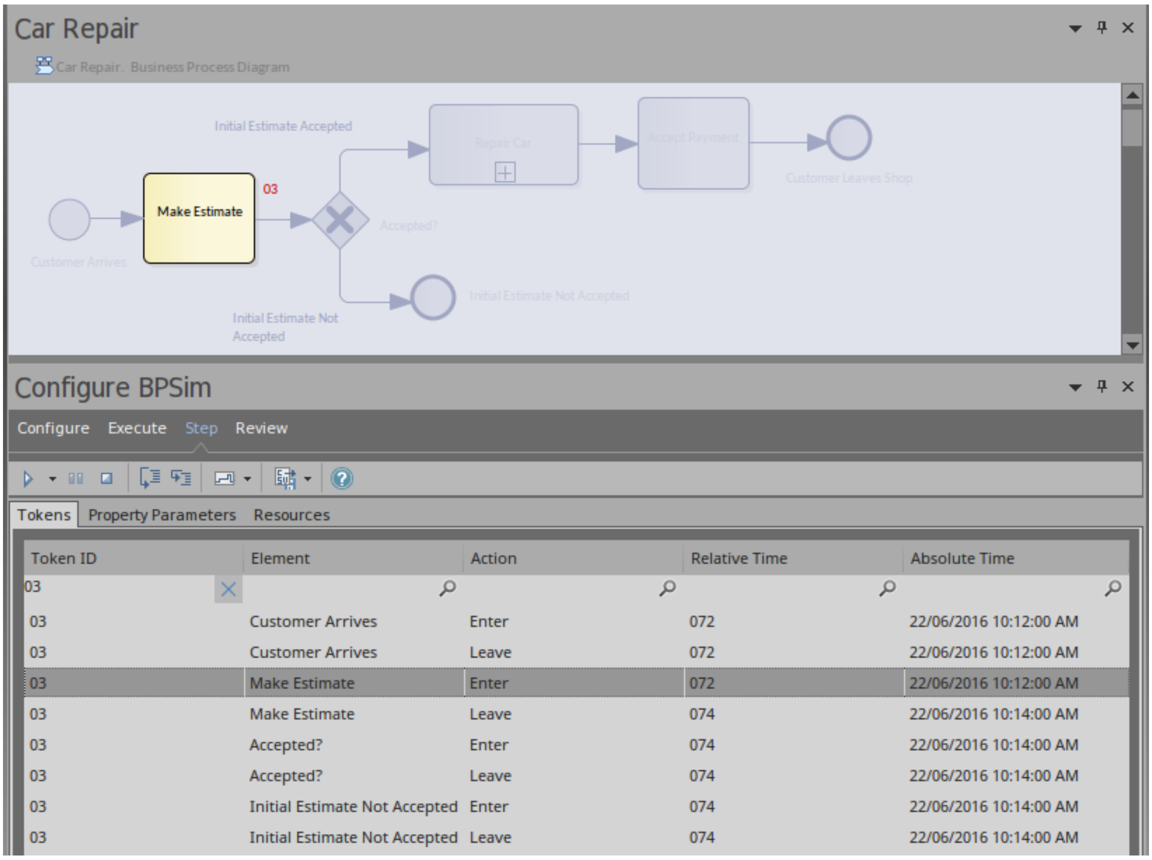
Toolbar Options

Option	Description
▷ ▾	Click on this button to simulate the process automatically based on the execution result. Click on the drop-down arrow and on the menu option 'Set Speed for Replay', and adjust the simulation speed as a multiple of normal. For example, typing '60' make the simulation 60 times faster than the actual activity; 1 minute in real life will be simulated in 1 second.
⏸	Click on this button to pause the auto-replay simulation.
◼	Click on this button to stop the simulation.
	Click on this button to 'Step over' to the next timestamp. Each 'Step over' can contain multiple 'steps'.
	Click on this button to play a single step. This represents a single movement of a token in the process.
▾	Click on this button to generate a Timing Diagram for the simulation. You can choose from the menu, either 'Generate a single timeline for each token' or 'Generate multiple timelines for each token'. See *Generate Timing Diagram* later in this topic.
	Click on this button to export the filtered records on this step page to a CSV file.

	You can choose from which tab ('Tokens', 'Property Parameters' or 'Resources') the data is exported.

Tokens Tab

After running the execution, this page will be filled with token information during the simulation; the sequence of entries is in order of triggering time.



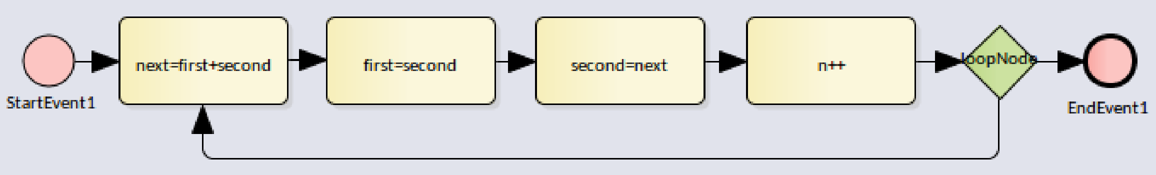
Token ID	Element	Action	Relative Time	Absolute Time
03				
03	Customer Arrives	Enter	072	22/06/2016 10:12:00 AM
03	Customer Arrives	Leave	072	22/06/2016 10:12:00 AM
03	Make Estimate	Enter	072	22/06/2016 10:12:00 AM
03	Make Estimate	Leave	074	22/06/2016 10:14:00 AM
03	Accepted?	Enter	074	22/06/2016 10:14:00 AM
03	Accepted?	Leave	074	22/06/2016 10:14:00 AM
03	Initial Estimate Not Accepted	Enter	074	22/06/2016 10:14:00 AM
03	Initial Estimate Not Accepted	Leave	074	22/06/2016 10:14:00 AM

- Using the Filter Bar in the Header band (right-click on the column heading and select 'Toggle Filter Bar') you can filter the results shown; for example, typing 03 in the 'Token ID' column will show only the records for token 03
- If you click once on the Step in button, one record in the list will play
- If you double-click on a record the simulation will 'Step to' that record from the beginning
- If time parameters are set on the elements, clicking on the Step over button will run to the last record of the next time event
- When a record in the list is played, the simulation snapshot will show on the diagram

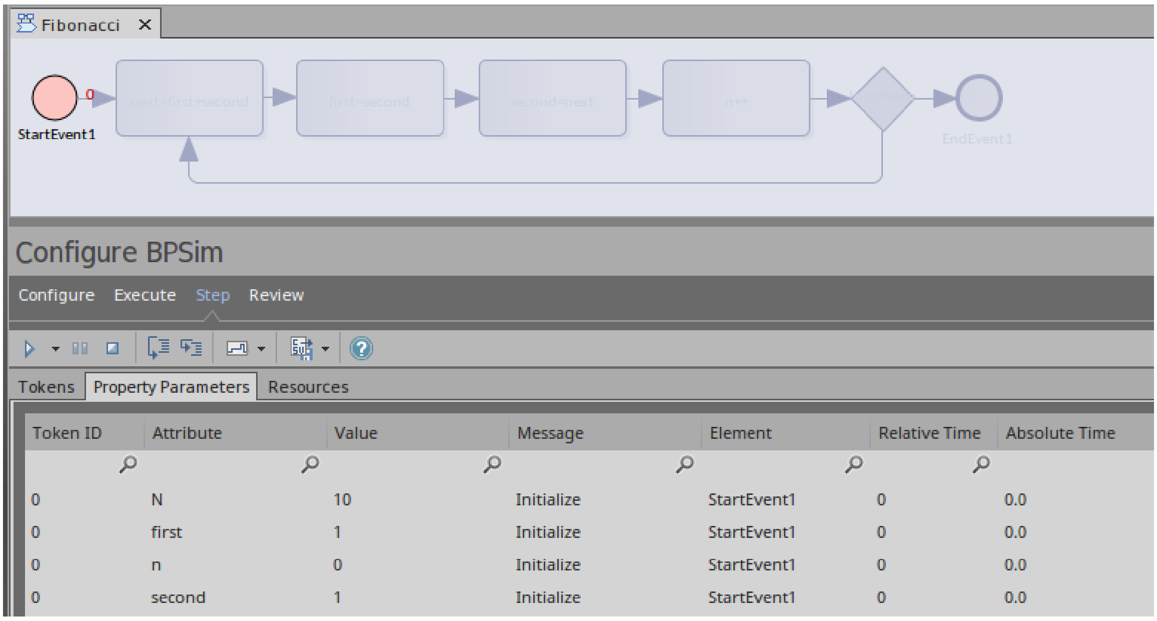
Property Parameters Tab

While the records on the 'Tokens' tab are played, the 'Property Parameters' tab will show the runtime value of properties at the timestamp.

For example, a BPMN process to calculate Fibonacci numbers might be modeled in this way:

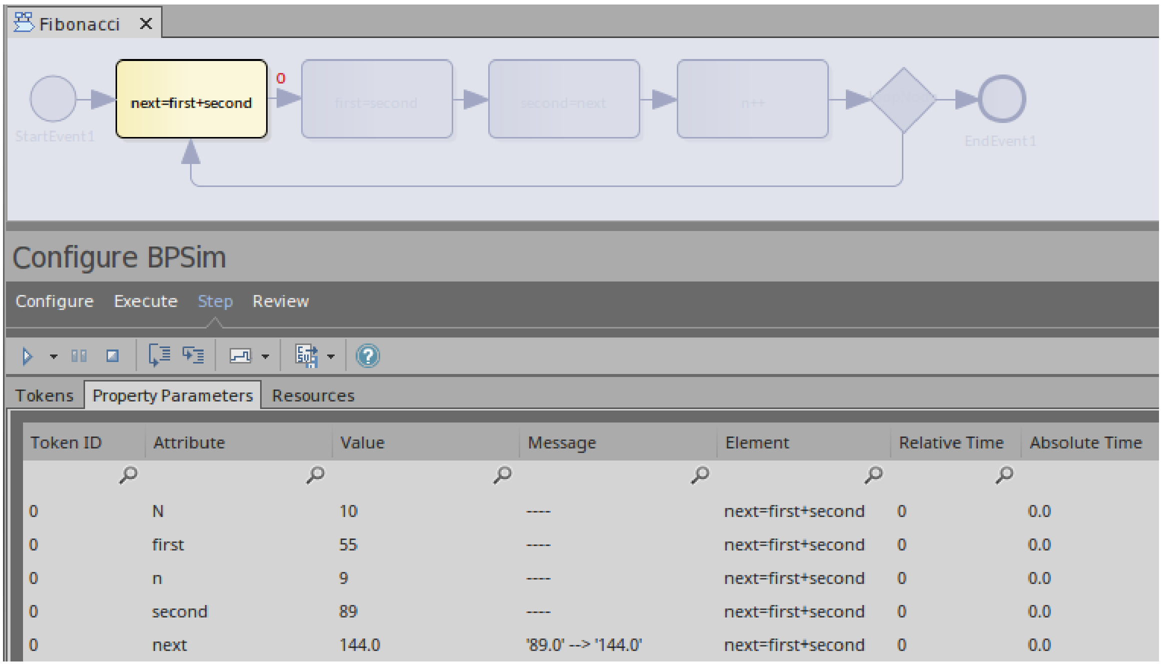


After defining property parameters, configuring BPSim parameters for each element and executing the model, we are ready for the step simulation:



Token ID	Attribute	Value	Message	Element	Relative Time	Absolute Time
0	N	10	Initialize	StartEvent1	0	0.0
0	first	1	Initialize	StartEvent1	0	0.0
0	n	0	Initialize	StartEvent1	0	0.0
0	second	1	Initialize	StartEvent1	0	0.0

The 'Message' column indicates that properties 'N', 'first', 'n' and 'second' are initialized.

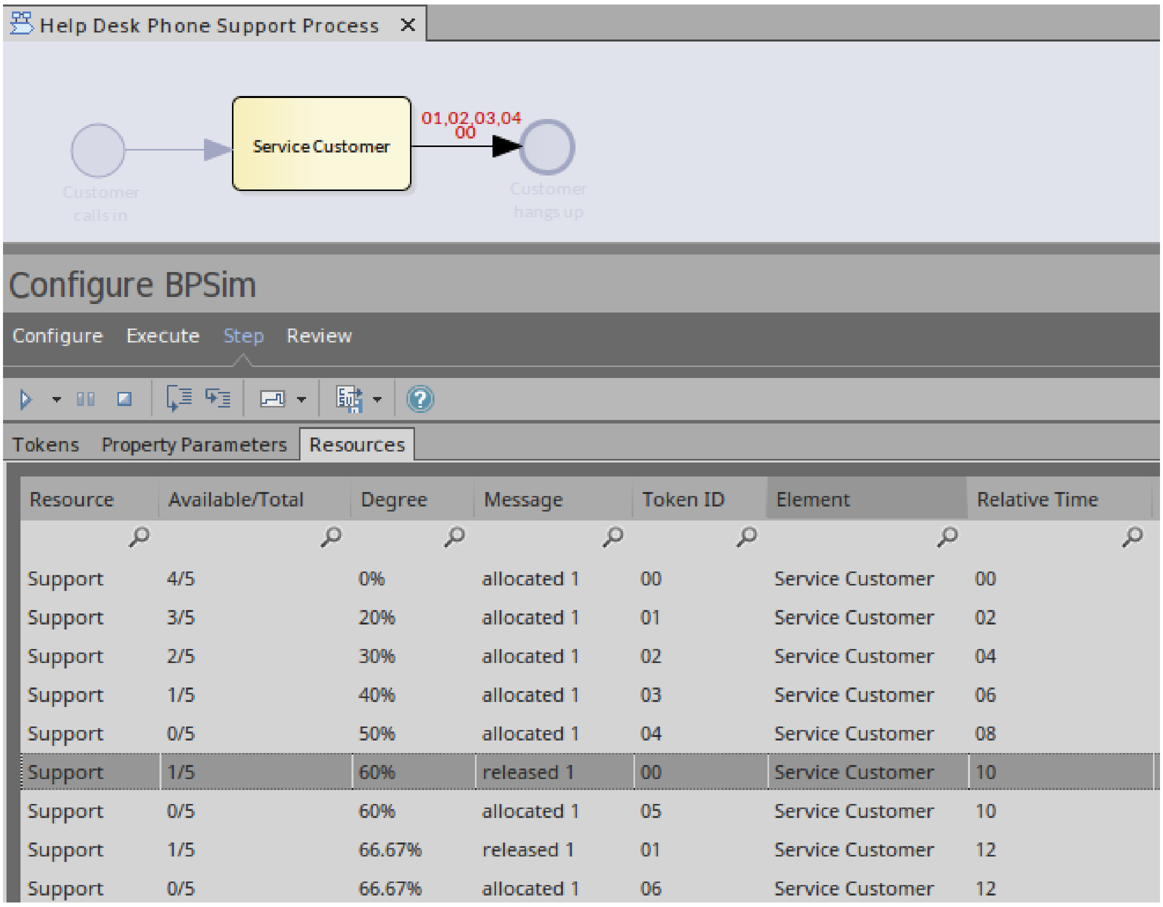


Token ID	Attribute	Value	Message	Element	Relative Time	Absolute Time
0	N	10	----	next=first+second	0	0.0
0	first	55	----	next=first+second	0	0.0
0	n	9	----	next=first+second	0	0.0
0	second	89	----	next=first+second	0	0.0
0	next	144.0	'89.0' --> '144.0'	next=first+second	0	0.0

If you keep clicking on the Step in button, the properties in the list will change their values. The illustration shows that on entering task 'next = first + second', the value of property 'next' changes from 89 to 144.

Resources Tab

While the records on the 'Tokens' tab are played, the 'Resources' tab will show the runtime resource available, the quantity of that resource available and the allocation or release events at the timestamp.



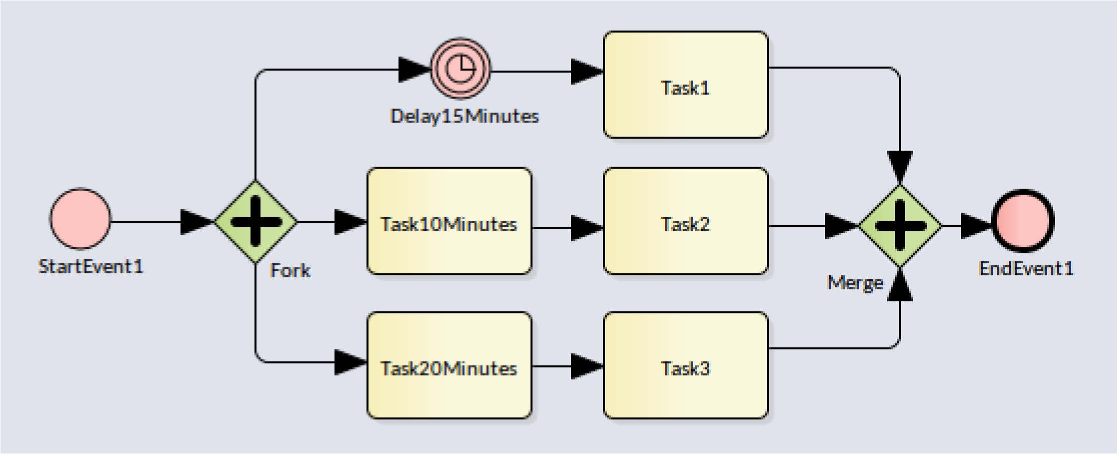
Resource	Available/Total	Degree	Message	Token ID	Element	Relative Time
Support	4/5	0%	allocated 1	00	Service Customer	00
Support	3/5	20%	allocated 1	01	Service Customer	02
Support	2/5	30%	allocated 1	02	Service Customer	04
Support	1/5	40%	allocated 1	03	Service Customer	06
Support	0/5	50%	allocated 1	04	Service Customer	08
Support	1/5	60%	released 1	00	Service Customer	10
Support	0/5	60%	allocated 1	05	Service Customer	10
Support	1/5	66.67%	released 1	01	Service Customer	12
Support	0/5	66.67%	allocated 1	06	Service Customer	12

Generate Timing Diagram

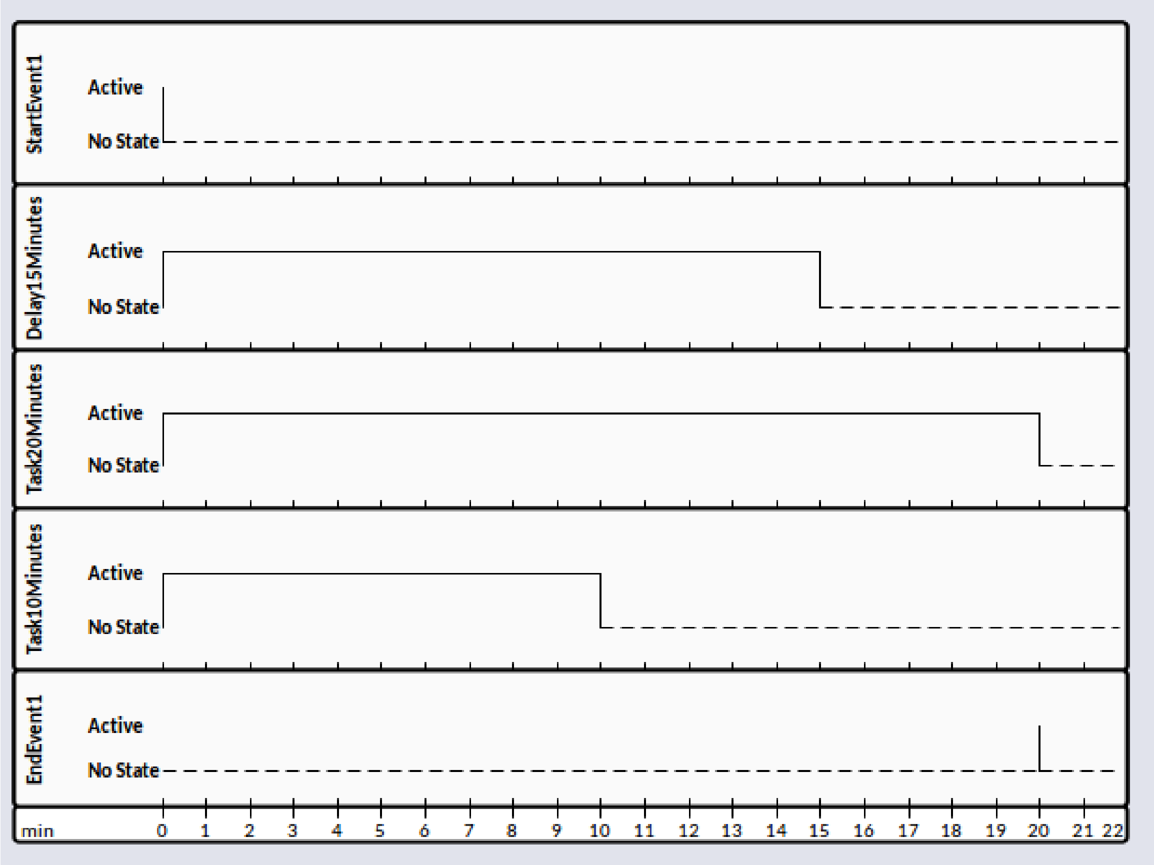
When time parameters are configured on the BPMN elements, Enterprise Architect can generate a Timing diagram for the simulation process.

- Generate a single timeline for each token - use this option for a 'single threaded' process; that is, no Parallel Gateway or Event sub-processes
- Generate multiple timelines for each token - use this option in cases where the 'Generate a single timeline for each token' option does not apply

For example:



Execute this model and click on 'Generate multiple timelines for each token' the generated Timing diagram resembles this:



BPSim - Review Page

This review page contains three tabs:

- Configuration Summary
- Standard Results Report
- Custom Results Report

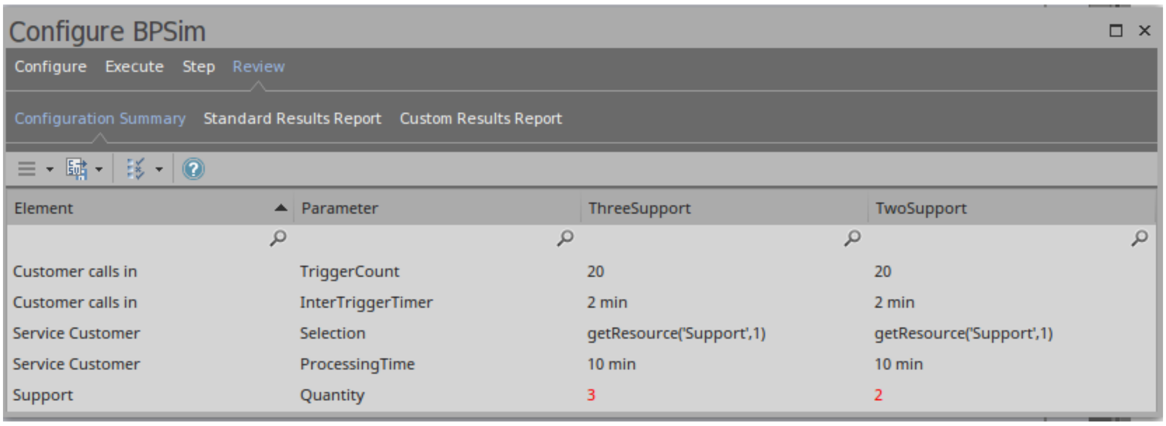
These tabs work in a similar way: add one Artifact for review or multiple Artifacts for comparison. This makes it easy for you to do what-if analyses.

Access

Ribbon	Simulate > Process Analysis > BPSIM > Open BPSIM Manager > Review

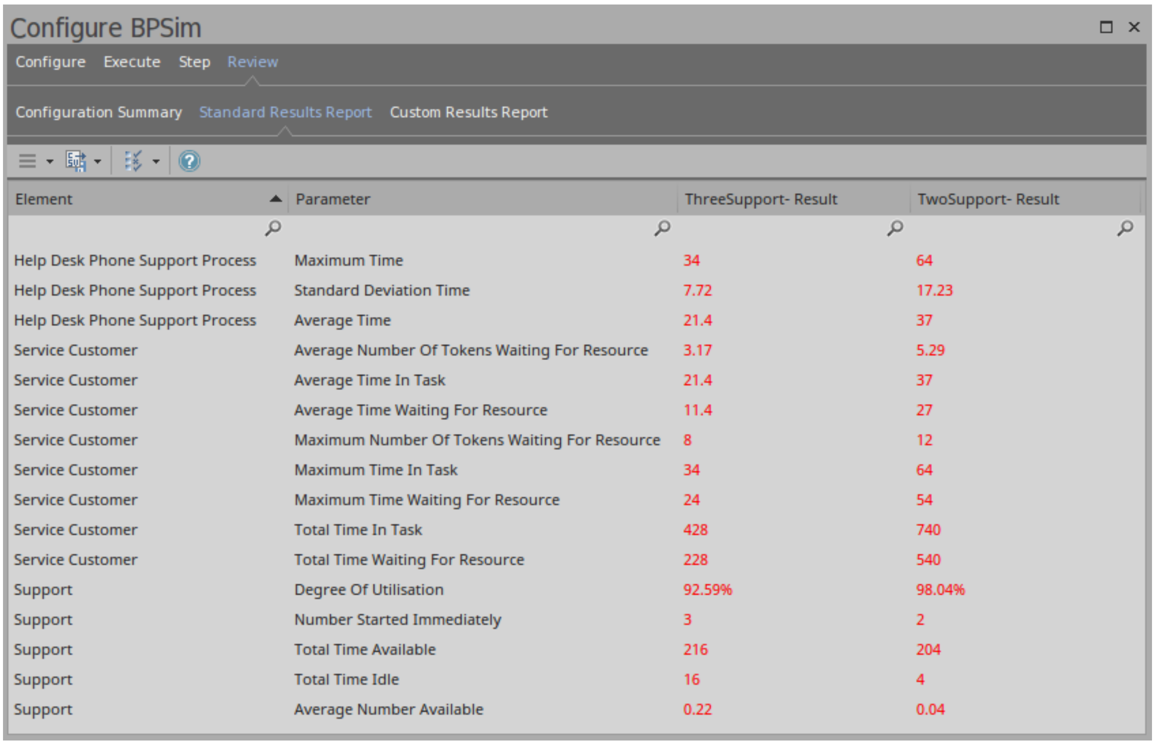
What-If Analysis

In the Help Desk Support example, we can compare two Artifacts and their corresponding results.



Element	Parameter	ThreeSupport	TwoSupport
Customer calls in	TriggerCount	20	20
Customer calls in	InterTriggerTimer	2 min	2 min
Service Customer	Selection	getResource('Support',1)	getResource('Support',1)
Service Customer	ProcessingTime	10 min	10 min
Support	Quantity	3	2

In this illustration, we have clicked on the icon in the toolbar and selected the 'Show only different items' option to see what differences the changed parameter values have caused.



Element	Parameter	ThreeSupport- Result	TwoSupport- Result
Help Desk Phone Support Process	Maximum Time	34	64
Help Desk Phone Support Process	Standard Deviation Time	7.72	17.23
Help Desk Phone Support Process	Average Time	21.4	37
Service Customer	Average Number Of Tokens Waiting For Resource	3.17	5.29
Service Customer	Average Time In Task	21.4	37
Service Customer	Average Time Waiting For Resource	11.4	27
Service Customer	Maximum Number Of Tokens Waiting For Resource	8	12
Service Customer	Maximum Time In Task	34	64
Service Customer	Maximum Time Waiting For Resource	24	54
Service Customer	Total Time In Task	428	740
Service Customer	Total Time Waiting For Resource	228	540
Support	Degree Of Utilisation	92.59%	98.04%
Support	Number Started Immediately	3	2
Support	Total Time Available	216	204
Support	Total Time Idle	16	4
Support	Average Number Available	0.22	0.04

We see that when the number of support staff decreases from 3 to 2, the average time waiting for resources increases from 11.4 minutes to 27 minutes.

Using the Parameter Value Dialog

The 'Parameter Value' dialog helps you to define values for a wide range of parameters throughout the BPSim Configuration. It supports the definition of simple fixed values through to distributions and expressions that yield a derived value. Not all types of value or derivation are appropriate to all types of parameter.

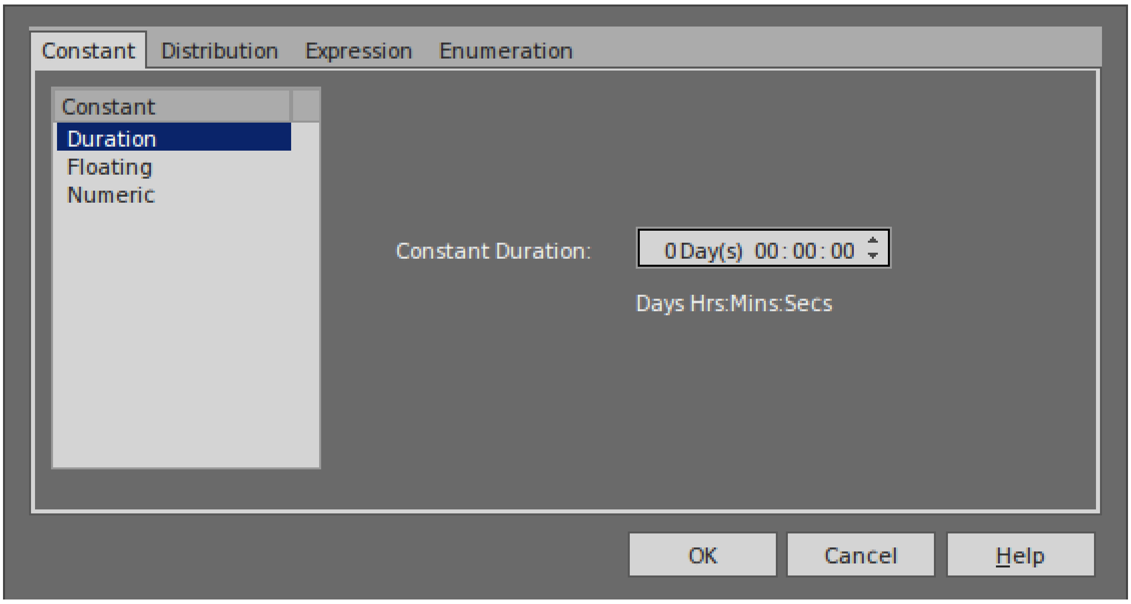
The dialog name is taken from the object name and the name of the parameter being defined; for example, Configure 'Processing' for 'Activity1'.

Access

With a BPSim Artifact loaded in the Configure BPSim window, select a BPMN Element on the diagram or Browser window, then click on ... in the 'Values' field. (If the parameter is not already created, choose Category and Parameter in the list to create a new one.)

Constant Tab

Use this tab to define a specific value for the parameter - a numeral, text string or time, for example.



Constant Duration: 0 Day(s) 00 : 00 : 00

In the 'Constant' panel, select the type of constant:

- Floating
- Numeric
- String
- DateTime
- Boolean, or
- Duration

Appropriate fields display to the right of the panel; type the value and, if required, the unit in which the value is expressed (for example, a unit of time or of currency). For some types of parameter a drop-down list is available from which you can select a value.

Distribution Tab

On this tab, you can apply a statistical sampling method to obtain the parameter value; for each type of distribution available, the appropriate fields display for you to enter the parameters of the distribution. All distributions require you to identify the unit of expression.

The distribution parameters are not necessary for the business process you are developing, but (if you are deriving values from a distribution) are required for the simulation.

You can select from these types of distribution:

- **Beta** - a continuous probability distribution providing 'real' values within a short range, commonly 0 to 1
- **Weibull** - a continuous probability distribution providing 'real' values, commonly used for object lifetime analysis
- **Gamma** - a continuous probability distribution providing 'real' values, useful for modeling exponentially distributed random variables
- **Binomial** - an 'integer' distribution, providing values based on the number of trials and the probability of a certain outcome
- **Erlang** - provides 'real' values based on the Erlang *K* value and the mean of the distribution
- **Normal** - provides 'real' values based on the mean and standard deviation of the distribution
- **LogNormal** - a continuous probability distribution of 'real', random variables whose logarithm is normally distributed
- **Poisson** - a discrete ('integer') probability distribution that expresses the probability of a given number of events occurring independently in a fixed interval of time or space (volume, distance or area)
- **NegativeExponential** - provides 'real' values based on the mean of the distribution
- **Triangular** - provides 'real' values based on the mode of the distribution and the minimum and maximum values of a range
- **TruncatedNormal** - provides 'real' values based on the mean and standard deviation of points within the minimum and maximum values of a range
- **Uniform** - provides 'real' values between the minimum and maximum values in a range

Expression Tab

On this tab, you type in an XPATH 1.0 expression to combine explicit values, operators and functions to be processed at runtime to provide a value. Each property parameter of an expression must be enclosed in braces - {xxx}.

Example 1: In order to represent c = a + b + 10, we assign this expression to a property 'c':

{a} + {b} + 10

where 'a' and 'b' are properties defined in the BPSim model.

Example 2: In order to represent c = t - p * (a - b)^2, we assign this expression to a property 'c':

{t} - {p} * Math.pow({a} - {b}, 2.0)

Note: When simulating a model with this expression, please select 'Java' as the language in order to use the java built-in function Math.pow().

Enumeration Tab

On the 'Enumeration' tab, you can define an enumeration to supply a collection of constant values. You would have obtained these values from real-world, historical data or from analysis and simulation of a model. Every time the parameter is evaluated, the next enumeration value is returned.

As you define each numeration value, click on the Save button to add it to the list of possible values, and click on the New button to clear the data fields ready to enter another value. For some types of enumeration value you might be asked to define the unit in which the value is expressed. The types of enumeration you can define include:

- String
- Floating
- Numeric
- Duration
- DateTime
- Boolean

Using the BPSim Execution Engine

The BPSim Execution Engine is an Add-In integrated with the Unified and Ultimate editions of Enterprise Architect, to execute the simulations that you have defined using the Business Process Simulation (BPSim) facility. The Engine is a prerequisite for accessing and using the facilities of BPSim.

Access

Open the Configure BPSim window using one of the methods in this table, click on the 🗎 button and browse for a Business Process Simulation Artifact.

Ribbon	Simulate > Process Analysis > BPSim > Open BPSim Manager (or Simulate > Process Analysis > BPSim > Find BPSim Configuration Artifacts)
Context Menu	Right-click on a Business Process Simulation Artifact element | Configure BPSim
Other	Right-click on a Business Process Simulation Artifact element | Simulate BPMN with BPSim

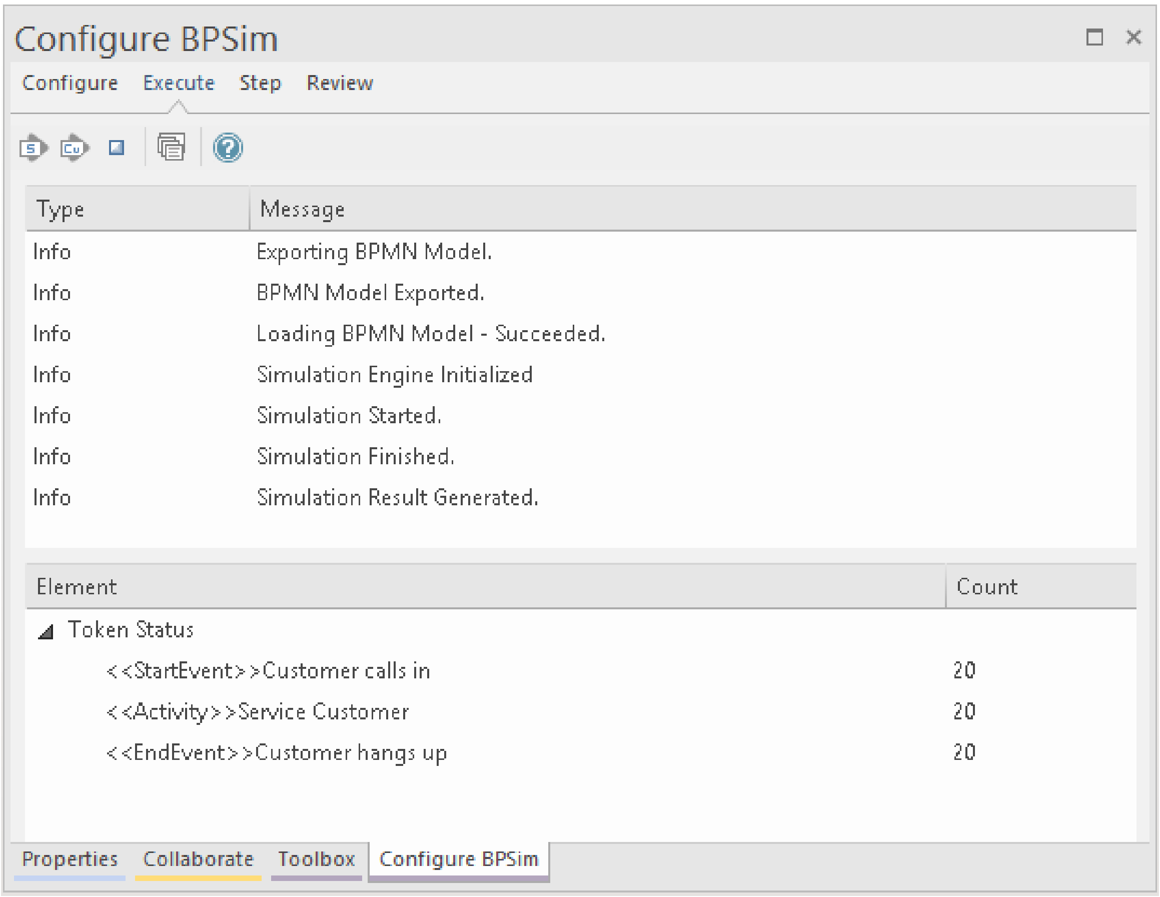
Execute and Control a Simulation

Click on the 'Execute' tab and on either:

- S to start a Standard Simulation or
- Cu to start a Customized Simulation

These options trigger the same processing, except that while a Standard Simulation generates a report on all built-in parameters set in the simulation, a Customized Simulation extracts results for only the parameters you have specifically flagged using the 'Result Request' columns in the configuration.

The simulation executes, displaying processing messages in the top section of the dialog, and the elements and parameters processed with the runtime values used from the configuration.



Type	Message
Info	Exporting BPMN Model.
Info	BPMN Model Exported.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Count
◢ Token Status	
<<StartEvent>>Customer calls in	20
<<Activity>>Service Customer	20
<<EndEvent>>Customer hangs up	20

While the simulation is in progress, you can click on the ■ icon to cancel the simulation.

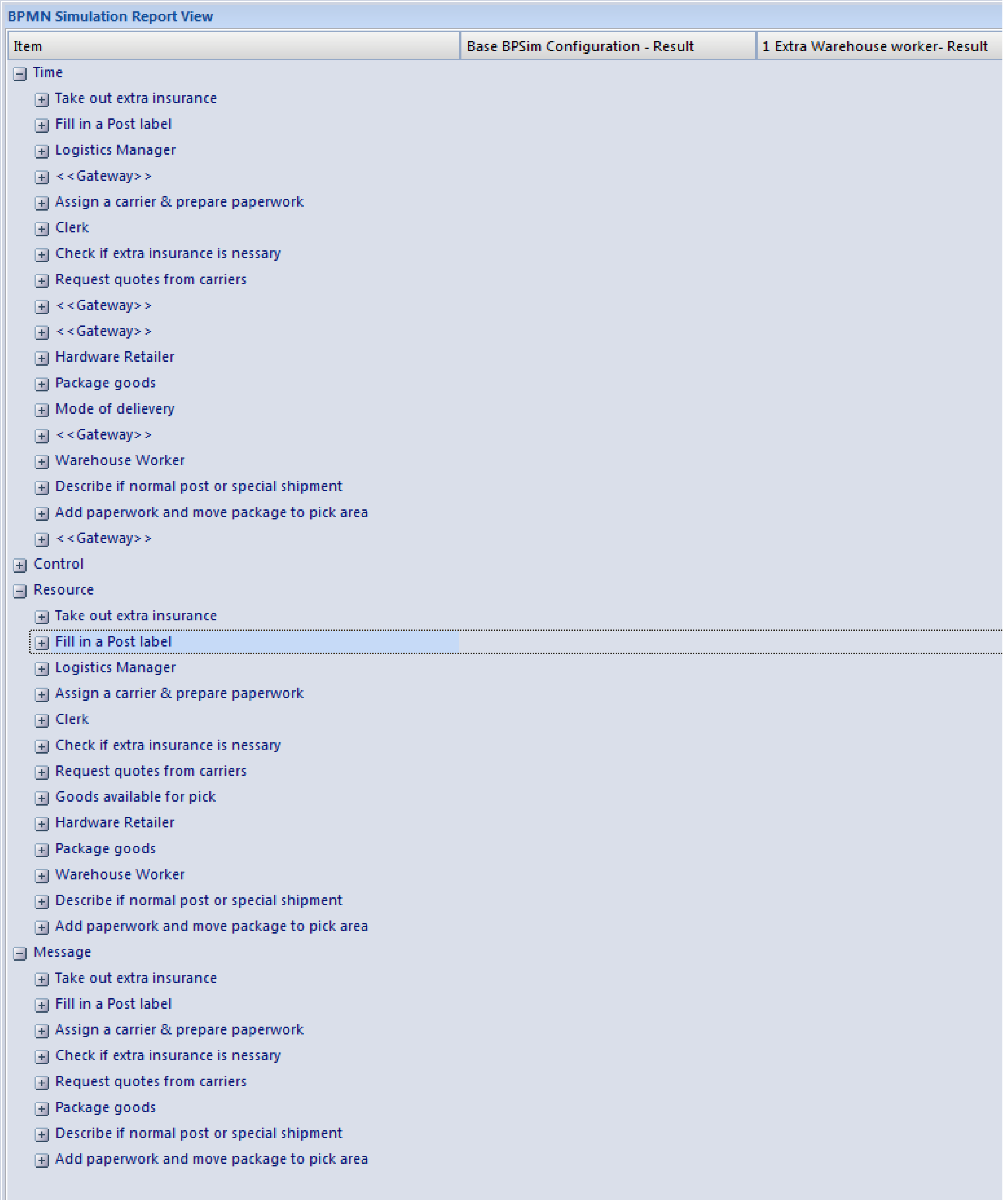
The results of the simulation are written to an Artifact element added to the Business Process parent Package. A Standard simulation writes to a <<BPSimReport>>-stereotyped Artifact, whilst a Customized simulation writes to a <<BPSimCustomReport>>-stereotyped Artifact.

Track Property Values

As well as the built-in parameters, you can define your own process-specific Property parameters (attributes) in the configuration. When the simulation has finished, and if you have defined property parameters, the Attributes button is enabled. When you click on this button the 'BPSim PropertyParameter Values' dialog displays, through which you can track how the run-time values of all the property parameters accrue or change through the business process.

Review a Simulation

When the simulation has finished processing, click on the Open Result button. The 'BPMN Simulation Report View' tab opens in the main work area, showing the results for the built-in parameters in the current simulation (but not for the user-defined property parameters). If you have already run a simulation of another configuration based on the same business process, that is also displayed in the report as an additional column. Otherwise, you can click on the report Artifact element and drag it onto the report tab, to compare the runtime values of the built-in parameters under two (or more) configurations.



BPMN Simulation Report View

Item	Base BPSim Configuration - Result	1 Extra Warehouse worker- Result
⊟ Time		
⊞ Take out extra insurance		
⊞ Fill in a Post label		
⊞ Logistics Manager		
⊞ <<Gateway>>		
⊞ Assign a carrier & prepare paperwork		
⊞ Clerk		
⊞ Check if extra insurance is nessary		
⊞ Request quotes from carriers		
⊞ <<Gateway>>		
⊞ <<Gateway>>		
⊞ Hardware Retailer		
⊞ Package goods		
⊞ Mode of delievery		
⊞ <<Gateway>>		
⊞ Warehouse Worker		
⊞ Describe if normal post or special shipment		
⊞ Add paperwork and move package to pick area		
⊞ <<Gateway>>		
⊞ Control		
⊟ Resource		
⊞ Take out extra insurance		
⊞ Fill in a Post label		
⊞ Logistics Manager		
⊞ Assign a carrier & prepare paperwork		
⊞ Clerk		
⊞ Check if extra insurance is nessary		
⊞ Request quotes from carriers		
⊞ Goods available for pick		
⊞ Hardware Retailer		
⊞ Package goods		
⊞ Warehouse Worker		
⊞ Describe if normal post or special shipment		
⊞ Add paperwork and move package to pick area		
⊟ Message		
⊞ Take out extra insurance		
⊞ Fill in a Post label		
⊞ Assign a carrier & prepare paperwork		
⊞ Check if extra insurance is nessary		
⊞ Request quotes from carriers		
⊞ Package goods		
⊞ Describe if normal post or special shipment		
⊞ Add paperwork and move package to pick area		

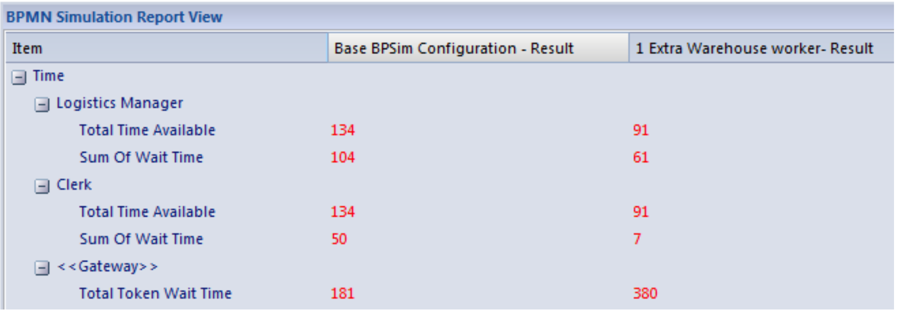
To make it easier to view the data from the report, you can drag the 'BPMN Simulation Report View' tab out of the main view so that it becomes a floating window, and enlarge the window to a suitable size.

Click on the expansion boxes against the parameters you want to check. You can also expose and filter the information using right-click context menu options.

You can represent specific differences between the results from separate simulations as Charts. The simulation result Artifacts (<<BPSimReport>>*name* - Result elements) must exist before you can set up the Chart Artifacts. There is a template Chart Artifact for Standard simulations and one for Customized simulations.

BPMN Simulation Report Options

Option	Description

<table>
<tr><td>Collapse All</td><td>Select this option to collapse the parameter hierarchy down to just the parent tab names.</td></tr>
<tr><td>Expand All</td><td>Select this option to expand the parameter hierarchy out to the lowest value type.</td></tr>
<tr><td>Show Only Different items</td><td>(When you have two or more simulations shown.) Select this option to restrict the display to those parameters where the values differ between the simulations. Click on the option again to deselect it.</td></tr>
<tr><td>Highlight Different Items</td><td>(When you have two or more simulations shown and where some of their parameter values are different.) Displays the differing parameter values in red. This option is disabled if you select the 'Show Only Different Items' option.

BPMN Simulation Report View
<table>
<tr><th>Item</th><th>Base BPSim Configuration - Result</th><th>1 Extra Warehouse worker- Result</th></tr>
<tr><td>⊟ Time</td><td></td><td></td></tr>
<tr><td>⊟ Logistics Manager</td><td></td><td></td></tr>
<tr><td>Total Time Available</td><td>134</td><td>91</td></tr>
<tr><td>Sum Of Wait Time</td><td>104</td><td>61</td></tr>
<tr><td>⊟ Clerk</td><td></td><td></td></tr>
<tr><td>Total Time Available</td><td>134</td><td>91</td></tr>
<tr><td>Sum Of Wait Time</td><td>50</td><td>7</td></tr>
<tr><td>⊟ <<Gateway>></td><td></td><td></td></tr>
<tr><td>Total Token Wait Time</td><td>181</td><td>380</td></tr>
</table>
</td></tr>
<tr><td>Show Only Non Empty Items</td><td>Select this option to filter the display to show only parameters that have a specific value other than 0.</td></tr>
<tr><td>Remove Model</td><td>(When you have selected a specific result, which identifies the simulation in the report.) Select this option to remove the simulation column from the report.</td></tr>
</table>

BPSim Execution Engine - Simulation Language

The BPSim Execution Engine supports simulation on XPath 1.0 or Java, where the appropriate language is defined as the Expression Language in the simulation configuration. It also supports the use of process instance data in BPSim Property Parameters, where the actual value is only determined during execution.

XPath 1.0 Operators

These operators can be used in BPSim expression parameters.

Operator	Description
|	The Union operator, used for resource acquisition. Example: getResource('w1',1) | getResource('w2',1)
+	Addition. Example: 4 + 6
-	Subtraction. Example: 6 - 4
*	Multiplication. Example: 6 * 4
div	Division. Example: 8 div 4
=	Equality. Example: 4 = 4 (True)
!=	Not Equal. Example: 5 != 3
<	Less than. Example: 6 < 9
<=	Less than or equal to. Example: x <= 6
>	Greater than. Example: 9 > 6
>=	Greater than or equal to. Example: n >= 7
or	Alternative. Example: n = 6 or n <= 6

and	Combination. Example: n = 5 and m < 8
mod	Modulus division. Example: 5 mod 2
getProperty	Get a property value. Example: getProperty ("amount")
getResource	Get a resource assignment. Example: getResource ('w1',1)

Note

The Expression Language can be set in the Configure BPSim window, on the 'Configure' tab; the two options 'XPath 1.0' and 'Java' are available as values of the 'Expression' parameter.

If you select 'Java', you must set the property 'JDK Home' to a valid JDK directory.

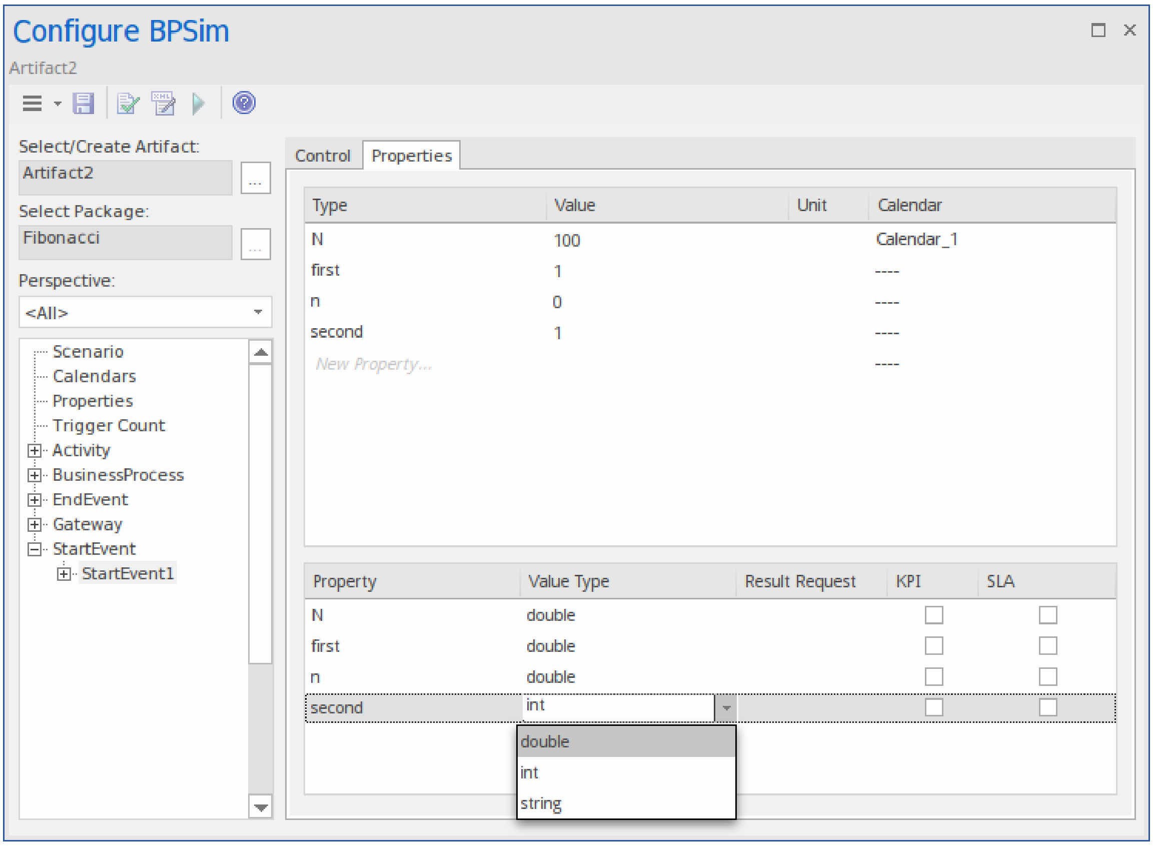
Tip: You can use {PropertyParameterName} as a short form of getProperty('PropertyParameterName'), which is useful when writing the value for the expressions; for example:

{n} < {N}   instead of   getProperty('n') < getProperty('N')

The short form of the getProperty operator can be used in both XPath 1.0 and Java.

BPSim Property Parameters

From Enterprise Architect release 13.0 onwards, BPSim property parameters can hold process instance data to which no value is assigned until run time. You can define the property parameter type on the 'Properties' page of the Configure BPSim window; the supported types are 'int', 'double' and 'string'.



Configure BPSim

Artifact2

Select/Create Artifact: Artifact2

Select Package: Fibonacci

Perspective: <All>

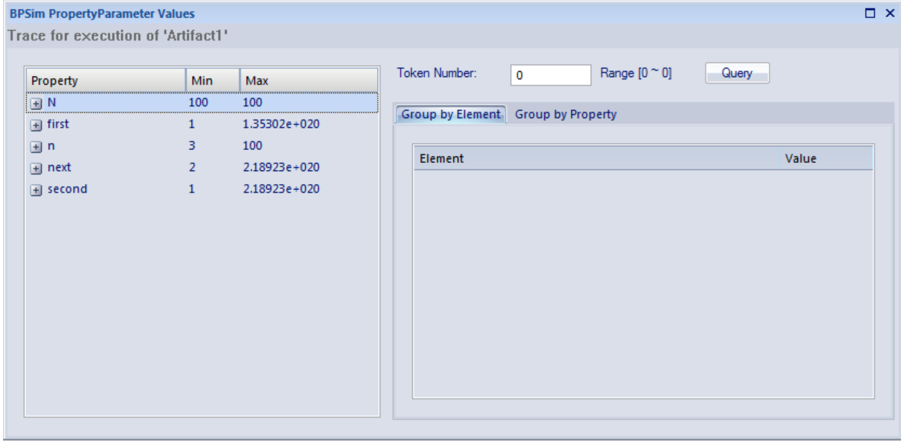
Type	Value	Unit	Calendar
N	100		Calendar_1
first	1		----
n	0		----
second	1		----
New Property...			----

Property	Value Type	Result Request	KPI	SLA
N	double		□	□
first	double		□	□
n	double		□	□
second	int		□	□

Tracking Property Parameter Values

The Business Process Simulator (BPSim) helps you to model and test the operating details of a business process, such as the resourcing of activities and tasks, the intervention of events, and the impact of decision points and the decisions made at those points. You add these process-specific property parameters, or attributes, to the BPSim configuration and, as you run simulations on the model according to the configuration, the BPSim engine helps you to capture the run-time values of the property parameters for every iteration of the simulation, and to filter the results to examine specific pathways or decision points. This gives you an incredibly detailed insight into what might actually happen in your business process under a specific condition or combination of conditions, either to generate a result or to show what processing path produces that result.

Access

<table>
<tr>
<td>Context Menu</td>
<td>
<p>Right-click on a defined Business Process Simulation configuration Artifact |Simulate BPMN with BPSim... : Run (select simulation type) : Attributes</p>
<p>(The Attributes button is not available if the simulation configuration does not contain any property parameters)</p>

<p>BPSim PropertyParameter Values</p>
<p>Trace for execution of 'Artifact1'</p>
<table>
<tr><th>Property</th><th>Min</th><th>Max</th></tr>
<tr><td>⊞ N</td><td>100</td><td>100</td></tr>
<tr><td>⊞ first</td><td>1</td><td>1.35302e+020</td></tr>
<tr><td>⊞ n</td><td>3</td><td>100</td></tr>
<tr><td>⊞ next</td><td>2</td><td>2.18923e+020</td></tr>
<tr><td>⊞ second</td><td>1</td><td>2.18923e+020</td></tr>
</table>
<p>Token Number: 0 Range [0 ~ 0]</p>

<table>
<tr><th>Element</th><th>Value</th></tr>
</table>
</td>
</tr>
</table>

BPSim PropertyParameter Values dialog fields

Option	Action
Property	This table lists the properties defined for the process, and shows the minimum and maximum possible values for each property for the whole process. If you click on the expansion box for a property, the table shows the minimum and maximum values for the property at each activity or event (element) during the process.
Token Number	Type the number of the 'token' to examine; this number must lie within the range shown to the right of the field. A 'token' is an independent trigger, such as a customer or an order entering a business and initiating the business process under review. There can be any number of customers or orders each having no relationship with any other customer or order, and each potentially following a different route through the business process. Where there is only one possible instance of one possible trigger event, such as an

	on/off switch being thrown, the token is regarded as being 0.
Query	Click on this button to initiate a query on the token simulation, to populate the 'Group by Element' and 'Group by Property' tabs.
Group by Element	Displays the results from the perspective of how a property parameter's value changes within a selected element. The tab shows a list of the elements in the process and, for each element, the value of each property applied in the element on each iteration of the simulation. Using the 'Toggle Filter Bar' option on the header bar, you can refine the display to show only a particular property and see how often it is used by the element and with what values.
Group by Property	Displays the results from the perspective of how the value of each property changes during the whole process. The tab shows a list of properties that are applied during the process and, for each property, the value in each activity (element) on each iteration of the process.

Examples

In the EAExample Model, you can study two examples of generating information on property parameters from a simulation of a BPMN business process model. These will demonstrate how you define property parameters in the configuration, based on the model. You can initially just run a simulation on each example, and examine the output as described here. You can then examine the business processes and configurations themselves, and change or add to the property parameters provided.

The examples are described in the *Tracking Property ParameterValues - Examples* topic. Briefly, they are:

- 'Fibonacci' - a very simple recursive business process that calculates a series of Fibonacci numbers through ten iterations; you can see how the property parameters increment in each iteration through the elements of the process (in Example Model > Model Simulation > BPSim Models > Fibonacci)
- 'Car Repair' - a more complex and realistic process that represents what might happen when a series of individual 'walk-in' customers bring vehicles into a car repair shop for estimation and repair (in Example Model > Model Simulation > BPSim Models > Car Repair Process)

There is also a small example of the behavior of the Time Parameters (in Example Model > Model Simulation > BPSim Models > Time Parameter).

Notes

- If a BPSim configuration contains result requests and a Custom simulation is performed on it, the 'BPMN Simulation Report' shows only the built-in parameters requested in the configuration; in contrast, the 'BPSim PropertyParameter Values' dialog lists all property parameters regardless of any result request settings or type of simulation

Tracking Property Parameter Values - Examples

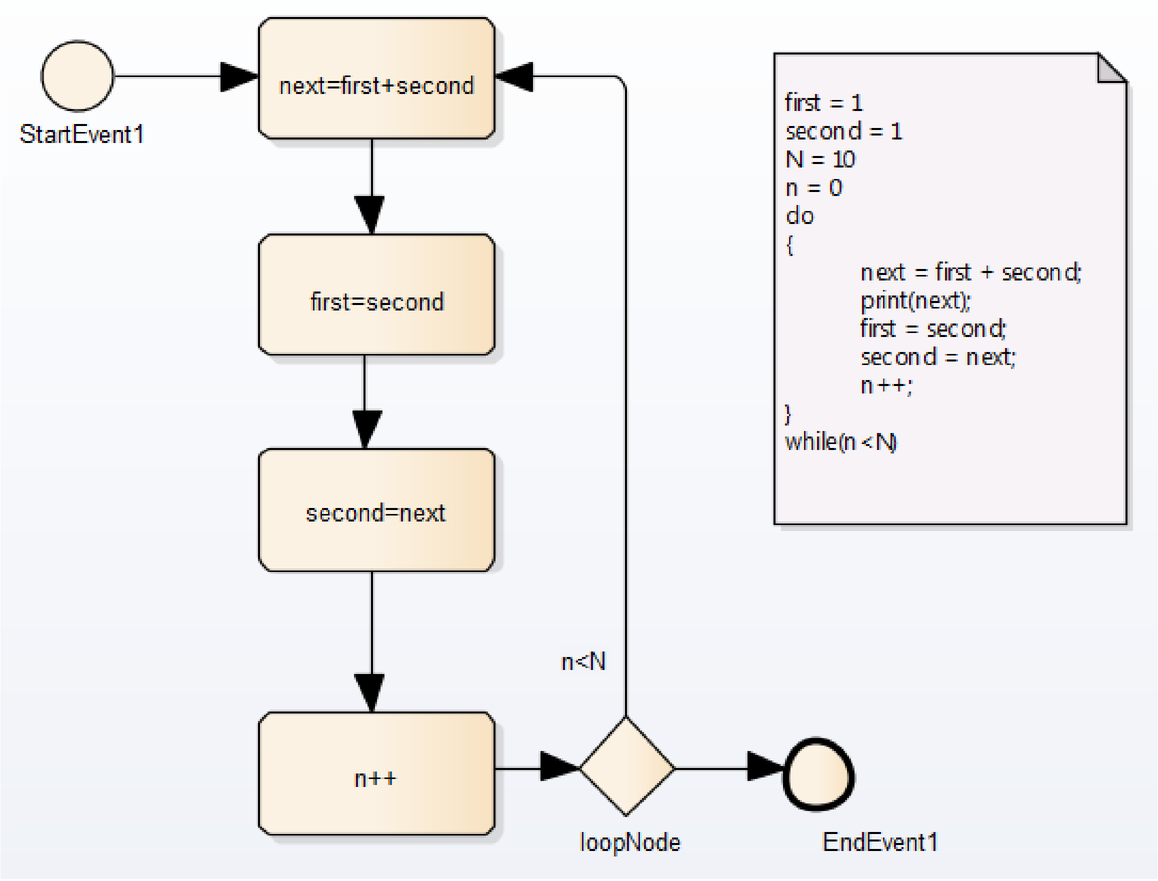
To help you learn about the facility for generating information on property parameters from a simulation of a BPMN business process model, Sparx Systems provide two examples that you can explore in the EAExample model. These are the:

- Fibonacci process - a very simple example to help you become familiar with the parameter tracking facilities
- Car Repair process - a more complex example that you can manipulate to see how a real-life process might be investigated

At the end of this topic is a section that briefly discusses how you might work with an integer-based process containing parameters that are initialized by 'real' distributions, and a section describing the example of the behavior of the Time Parameters.

The Fibonacci Example

This is a very simple recursive business process that calculates a series of Fibonacci numbers through ten iterations; you can see how the property parameters increment in each iteration through the elements of the process. Open Example Model > Model Simulation > BPSim Models > Fibonacci.

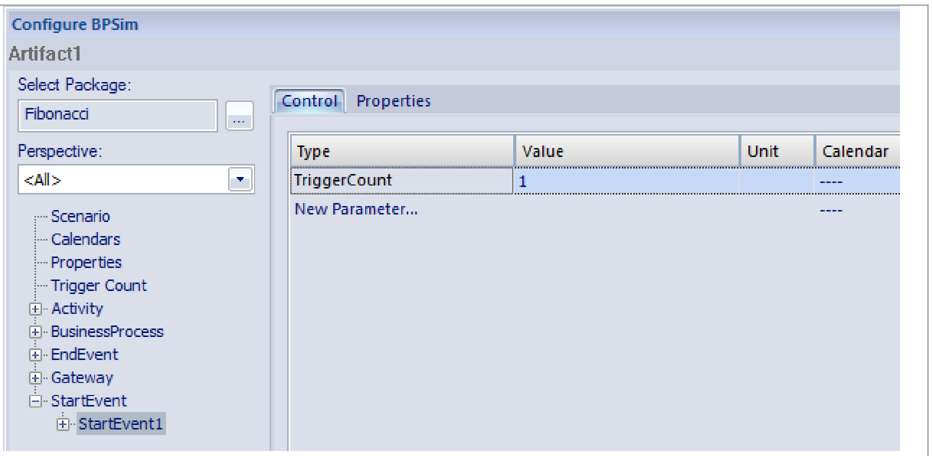


```
first = 1
second = 1
N = 10
n = 0
do
{
        next = first + second;
        print(next);
        first = second;
        second = next;
        n++;
}
while(n < N)
```

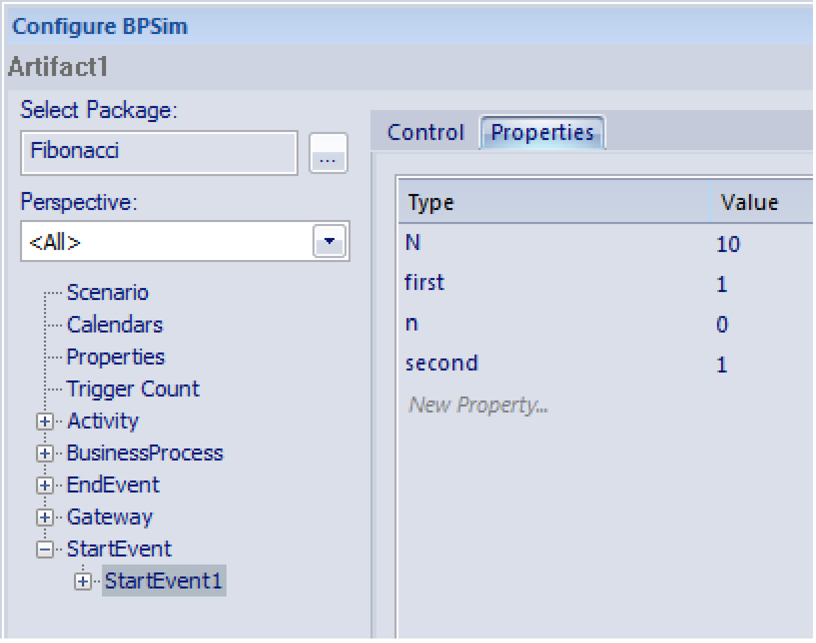
The pseudocode of the process is shown in the Notes element on the diagram. The statement 'print(next)' will output the number series 2, 3, 5, 8, 13, 21, 34, 55, 89, 144.

The BPSim configuration for this process is set up as described here.

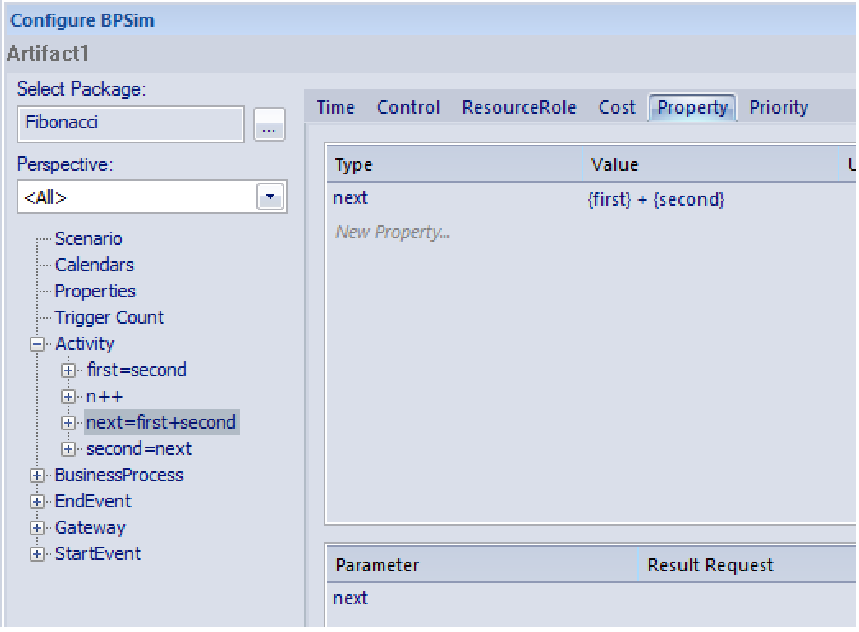
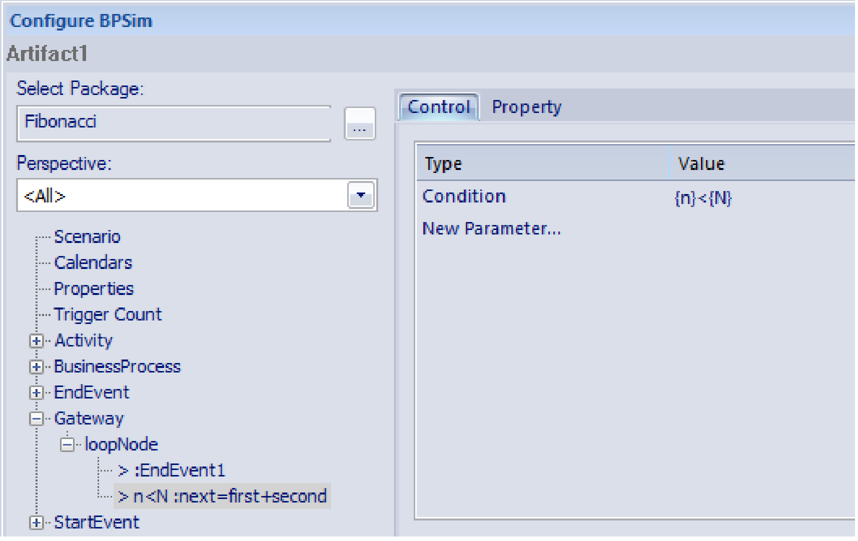
Step	Action
1	

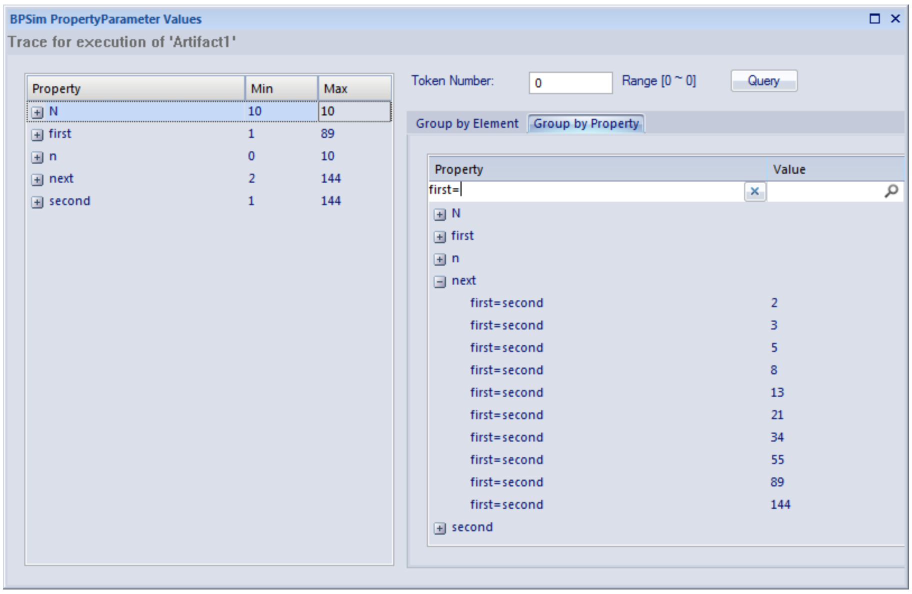
<table>
<tr>
<td></td>
<td>


Select Package: Fibonacci
Perspective: <All>
<table>
<tr><th>Type</th><th>Value</th><th>Unit</th><th>Calendar</th></tr>
<tr><td>TriggerCount</td><td>1</td><td></td><td>----</td></tr>
<tr><td>New Parameter...</td><td></td><td></td><td>----</td></tr>
</table>

On the 'Control' tab for the StartEvent element, set 'TriggerCount' to '1', and on the 'Properties' tab create and initialize the properties:
- 'N' as '10'
- 'first' as '1'
- 'second' as '1'
- 'n' as '0'


Select Package: Fibonacci
Perspective: <All>
<table>
<tr><th>Type</th><th>Value</th></tr>
<tr><td>N</td><td>10</td></tr>
<tr><td>first</td><td>1</td></tr>
<tr><td>n</td><td>0</td></tr>
<tr><td>second</td><td>1</td></tr>
<tr><td>New Property...</td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>2</td>
<td>
Now define the properties for each of the Activities in the process, on the 'Properties' tab. Note that the values for these properties are derived from <b>Expressions</b>, the components of which must be enclosed in braces - {xxx}. For the Activity:
- next=first+second - set the property 'next' and define the value as the Expression {first} + {second}
</td>
</tr>
</table>

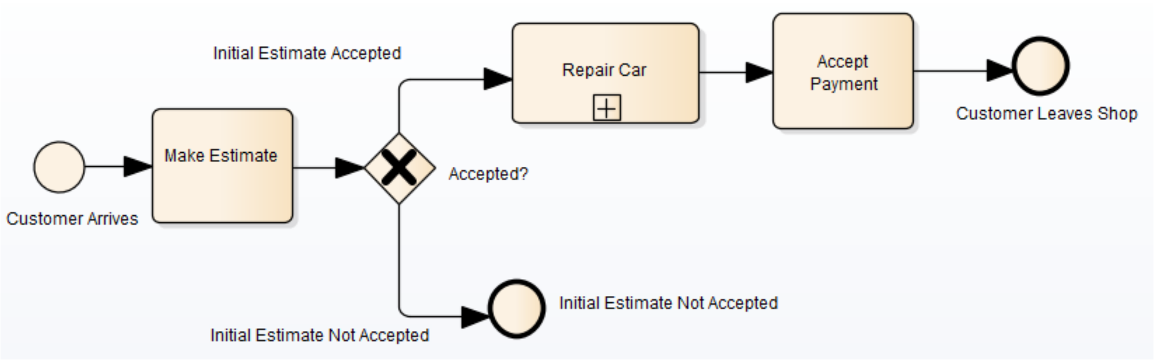
	 Configure BPSim Artifact1 Select Package: Fibonacci Perspective: <All> Scenario Calendars Properties Trigger Count Activity first=second n++ next=first+second second=next BusinessProcess EndEvent Gateway StartEvent Time Control ResourceRole Cost Property Priority Type Value next {first} + {second} New Property... Parameter Result Request next • first=second - set the property 'first' and define the value as the Expression {second} • second=next - set the property 'second' and define the value as the Expression {next} • n++ - set the property 'n' and define the value as the Expression {n} + 1
3	Set the 'Condition' property parameters for the two Sequenceflow connectors issuing from the 'loopNode' Gateway element, on the 'Control' tab. Expand the Gateway | loopNode element and for the link to: • next=first+second - set the Control parameter to 'Condition' and define the value as the Expression {n}<{N}  Configure BPSim Artifact1 Select Package: Fibonacci Perspective: <All> Scenario Calendars Properties Trigger Count Activity BusinessProcess EndEvent Gateway loopNode > :EndEvent1 > n<N :next=first+second StartEvent Control Property Type Value Condition {n}<{N} New Parameter... • EndEvent1 - set the Control parameter to 'Condition' and define the value as the Expression {n}=>{N}
4	Having completed the configuration, click on the Run button on the Configure BPSim window and on the 'BPSim Simulation Controller' dialog, selecting a Standard simulation. When the simulation is complete, click on the Attributes button.

	On the 'BPSim PropertyParameter Values' dialog, set the 'Token number' field to '0' and click on the Query button.
5	Now examine the values of the 'next' property on entering the first=second Activity in each iteration of the simulation. Click on the 'Group by Property' tab and expand the 'next' item. The list of values is long, so right-click on the column headers and select the 'Toggle Filter Bar' option. Under the 'Property' column heading, type 'first='. This filters the list to show only the property parameter values on entering the first=second Activity.  BPSim PropertyParameter Values Trace for execution of 'Artifact1' Property · Min · Max N · 10 · 10 first · 1 · 89 n · 0 · 10 next · 2 · 144 second · 1 · 144 Token Number: 0 Range [0 ~ 0] Query Group by Element · Group by Property Property · Value first= N first n next first=second · 2 first=second · 3 first=second · 5 first=second · 8 first=second · 13 first=second · 21 first=second · 34 first=second · 55 first=second · 89 first=second · 144 second

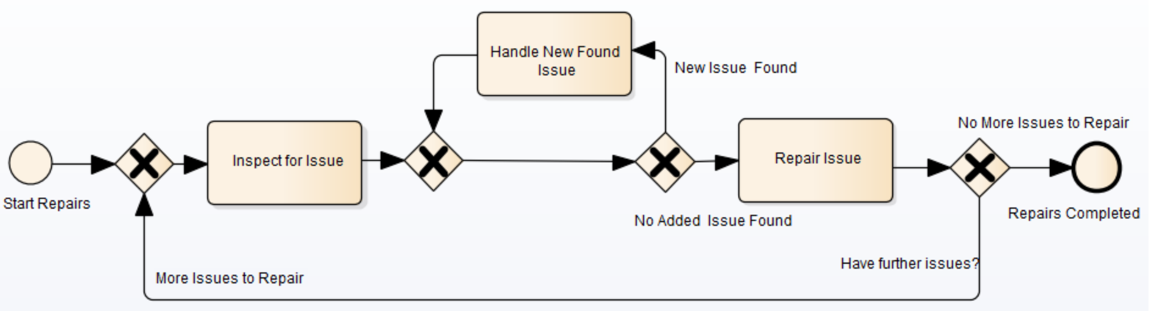
The Car Repair Example

This more complex example is based on a realistic model of a car repair process, where a number of individual customers request an estimate for repair and either proceed with the repair or decline to continue; you can see how the property parameters vary as different decisions are made during the process. Open Example Model > Model Simulation > BPSim Models > Car Repair Process.

The overall process is represented by this diagram:


Initial Estimate Accepted
Repair Car
Accept Payment
Customer Leaves Shop
Make Estimate
Accepted?
Customer Arrives
Initial Estimate Not Accepted
Initial Estimate Not Accepted

The *Repair Car* Activity is a composite element that contains this sub process diagram:



Step	Action
1	In the Browser window, expand the 'BPSim' child Package underneath the 'Car Repair Process' Package, and double-click on the 'Scenario 1: Main Flow' Artifact. The Configure BPSim window displays. On the 'Scenario' tab look at the 'Duration' field; this has been set to 2 days and 12 hours (that is, 60 hours).
2	In the process hierarchy on the left of the window, expand the 'Start Event' category and click on 'Customer Arrives'. Select the 'Control' tab and look at the 'InterTriggerTimer' parameter, which has the value 24 minutes; that is, a customer arrives every 24 minutes (so over the 60 hour duration, 150 customers pass through the repair shop). Each customer enters the repair shop with one or more issues to be evaluated and repaired. The number of issues each customer presents can be randomly generated using one of the Distributions supported by BPSim. As the issue number is counted in discrete units (rather than measured on a continuous scale) we would use an 'integer' distribution. If you select the 'Properties' tab for the 'Customer Arrives' Start Event, you will see that the 'noOfIssues' property value is initialized from a Poisson distribution with a mean of 3.
3	Now expand the decision Gateway 'Accepted?' and its connectors, in the process hierarchy. 'Initial Estimate Accepted' has a Control parameter 'Probability', set to 0.67. The alternative connector, 'Initial Estimate Not Accepted' has a similar Control parameter 'Probability' set to 0.33. That is, we expect an average of one issue in three to be withdrawn - or not pursued - by the customer.
4	Further into the process, when an issue is being assessed on the vehicle, there is a possibility of another issue being discovered. In the list of Gateway elements, the last 'unnamed element' has two paths: 'New Issue Found' and 'No Added Issue Found'. Click on each of these and look at the 'Control' tab; the 'Probability' parameter for 'New Issue Found' is set to 0.25 and, for 'No Added Issue Found', to 0.75. So on average, for every four issues reported and assessed, one new one is discovered. The 'New Issue Found' path takes the process to the 'Handle New Found Issue' Activity, which adds 1 to the number of issues to be processed for the current customer. Expand the Activity group, and click on the 'Handle New Found Issue' element and on the 'Property' tab. You will see that the property 'noOfIssues' here has the Expression value {noOfIssues} **+ 1**.
5	When a problem with the vehicle is resolved, the 'Repair Issue' Activity deducts 1 from the number of issues to be repaired for the current customer. Click on the 'Repair Issue' element in the Activity group and on the 'Property' tab. You will see that the property 'noOfIssues' here has the Expression value {noOfIssues} **- 1**.
6	The value from the 'Repair Issue' Activity is tested at the 'Have Further Issues?' Gateway. Click on the 'More Issues to Repair' connector and on the 'Control' tab; the Condition parameter for following this path is set to the Expression value {noOfIssues} **> 0**; flow passes to the Gateway prior to the 'Inspect for Issue' Activity. Similarly, if you click on the 'No More Issues to Repair' connector and on the 'Control' tab, the Condition parameter for following that path is set to the Expression value {noOfIssues} **=< 0**, and flow passes to the

	'Repairs Completed' End Event. Now that you have examined the process flow and configuration settings, you can run a simulation and review the results.
7	On the Configure BPSim window, click on the Run button, and then on the 'BPSim Simulation Controller' dialog click again on the Run button, selecting the 'Standard' simulation (however, the type of simulation makes no difference for reviewing property parameters). On the 'BPSim Simulation Controller' dialog, you can review the Token status (and see that an additional customer manages to enter the shop at the very last minute) but it is difficult to see exactly how this summary data resulted. Click on the Attributes button to obtain the detailed property parameter values information on the 'BPSim PropertyParameter Values' dialog.
8	On the left hand side of the dialog is a summary of the minimum and maximum values for the property parameter (attribute) for each element in the process. For example, for the 'Customer Arrives' element the 'noOfIssues' parameter has a minimum of 0 and a maximum of 8, as generated by the Poisson (3) distribution. In the 'Token Number' field, type in a number (N) between 0 and 150 to select for the Nth customer who entered the repair shop. Click on the Query button to obtain the property parameter values used in the process for that customer. Review the results on each of the two tabs: • On the 'Group by Element' tab, see how the attribute's value changes in each element; for example, for customer 24 the 'noOfIssues' parameter is initialized with a value of 4 by the random distribution, and the 'Inspect for Issue' Activity is called six times with the parameter value being adjusted to 3 for three of those calls before cycling to 1, and the 'Handle New Found Issue' Activity is called twice with the parameter value at 3 both times • On the 'Group by Property' tab see how the parameter value changes as the process cycles through the Activities to completion, starting at 4, being adjusted between 3 and 4 a number of times and then decrementing to 0 at the end
9	Continue to explore the results as required, selecting different customers (Tokens). You can also return to the BPSim configuration and change the parameter initializations and add new ones, or change the decision points, to experiment with the process.

Responding to real numbers in the simulation of an integer-based process

In some cases, you might need to generate property parameter values using a distribution that returns 'real' numbers when the activities in the process operate with integers, or when you want to see what impact forcing integer values has on the process.

One mechanism to apply in such cases is to set conditions to avoid absolute numbers. So, for example, you might have a counter that decrements by 1, that is initialized to a 'real' number. If you set a condition to 'value==0' (equals 0) or 'value !=0' (does not equal 0), the two conditions might never be True or might always be True, respectively, causing an infinite loop. To avoid that, in the conditions you would use operators such as:

'value > 0'

'value < 0'

'value >= 0'

'value <= 0'

Another mechanism is to edit the code template used by the BPSim engine, to intercept and replace the real numbers provided to specific parameters with integers, as shown:

1. Select the 'Develop > Source Code > Options > Edit Code Templates' ribbon option.
2. In the Code Template Editor, in the 'Language' field, click on the drop-down arrow and select 'MDGBPSimExecutionEngineExtension'.

3. In the list of (Java) templates, click on 'MDGBPSimExecutionEngineExtension Compute Value'. The template contents display in the 'Template' panel.
4. Find this line:

   ```
   double %bpsimPropertyParameterName% = (double) distribution.next();
   ```

   Change it to:

   ```
   %if bpsimPropertyParameterName == "noOfIssues" or  bpsimPropertyParameterName == "noOfVisitors"%
   double %bpsimPropertyParameterName% = (int) distribution.next();
   //double %bpsimPropertyParameterName% = Math.ceil(distribution.next());
   //double %bpsimPropertyParameterName% = Math.floor(distribution.next());
   //double %bpsimPropertyParameterName% = Math.round(distribution.next());
   %else%
   double %bpsimPropertyParameterName% = (double) distribution.next();
   %endIf%
   ```
5. Replace the property parameter names with your own property parameters.
6. Click on the Save button, close the Code Template Editor and reload the project.

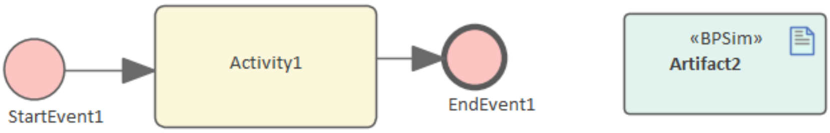
As presented, for each specified parameter the code template will simply replace any 'real' number initialized by the distribution with an integer. If you prefer, you can use one of the commented lines instead:

- Math.ceil() will take the 'real' number and convert it to the next highest integer
- Math.floor() will take the 'real' number and convert it to the next lowest integer
- Math.round() will take the 'real' number and round it up or down depending on whether it is greater than or less than n.5

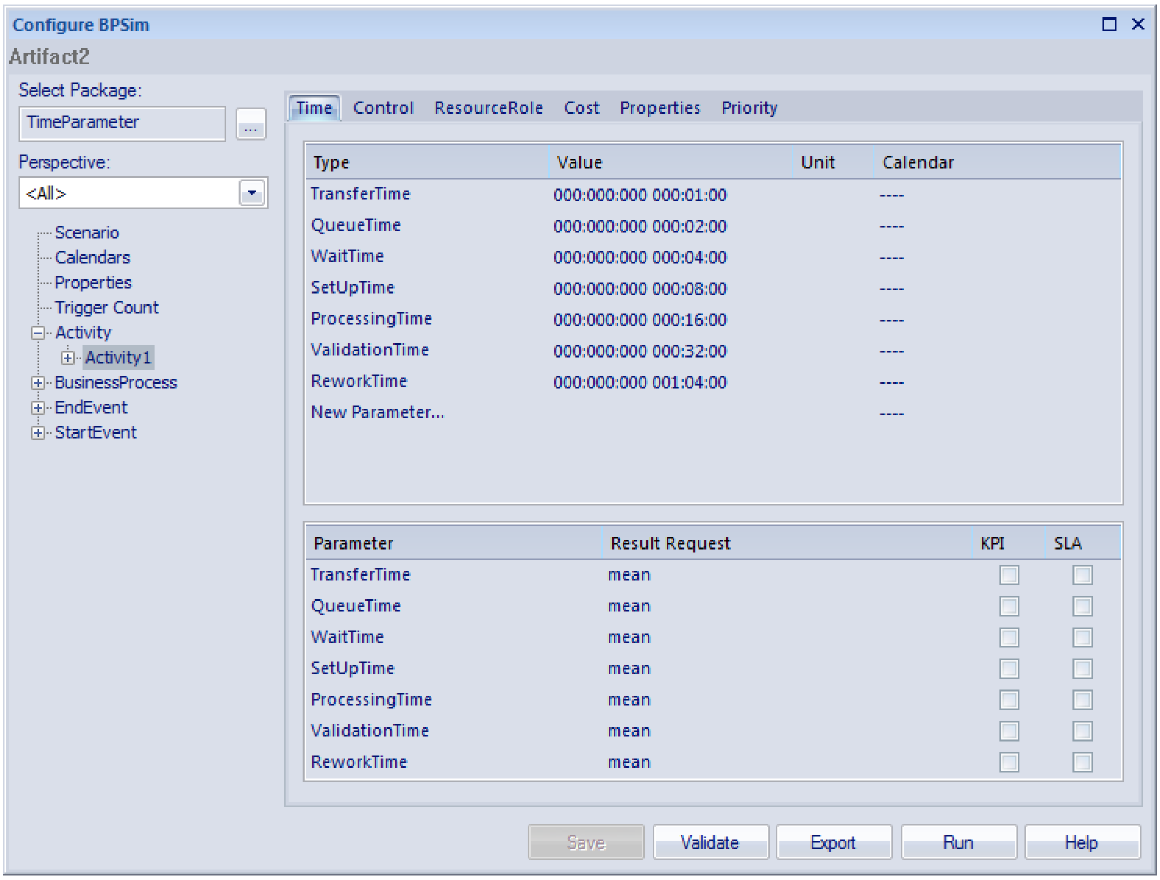
Time Parameter Behavior

In the BPSim configuration, you can set a number of Time parameters for an Activity, such as Queue Time and Wait Time. You can also set a Result Request on each of these, for a Custom simulation. However, the BPSim simulation engine combines these parameters into a single 'Processing Time' quantity.

Consider the simple model TimeParameter in the Example Model (Example Model > Model Simulation > BPSim Models > Time Parameter), represented by this diagram:



If you double-click on the Artifact2 element, the Configure BPSim window displays. Click on the Activity1 element in the diagram to expand the Activity group, to select Activity1 in the hierarchy at the left of the dialog, and to display the first tab, 'Time', for the element in the configuration, as shown.



Select Package: TimeParameter

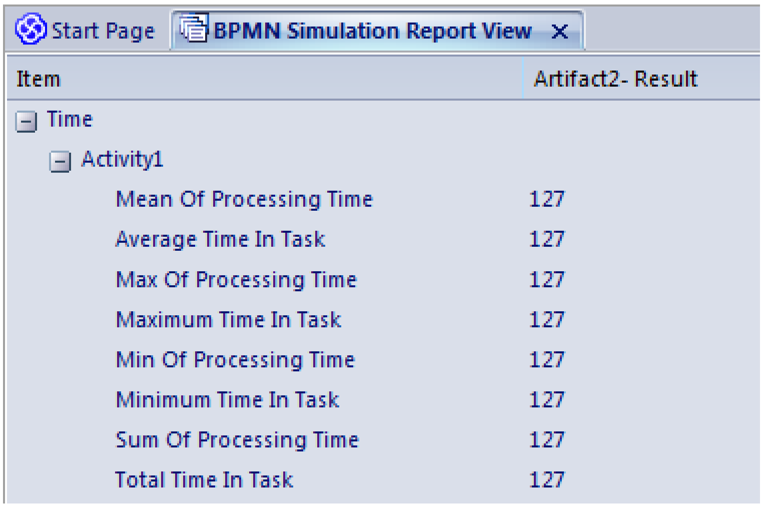
Perspective: <All>

Type	Value	Unit	Calendar
TransferTime	000:000:000 000:01:00		----
QueueTime	000:000:000 000:02:00		----
WaitTime	000:000:000 000:04:00		----
SetUpTime	000:000:000 000:08:00		----
ProcessingTime	000:000:000 000:16:00		----
ValidationTime	000:000:000 000:32:00		----
ReworkTime	000:000:000 001:04:00		----
New Parameter...			----

Parameter	Result Request	KPI	SLA
TransferTime	mean	□	□
QueueTime	mean	□	□
WaitTime	mean	□	□
SetUpTime	mean	□	□
ProcessingTime	mean	□	□
ValidationTime	mean	□	□
ReworkTime	mean	□	□

Note that in the upper panel there are seven system-supplied 'Time' parameters, which have been given initial values of - in order - 1, 2, 4, 8, 16, 32 and 64 minutes (the 64 minutes is 1 hour and 4 minutes). Note also that in the lower panel, each of these has a Result Request for the mean runtime value of the parameter.

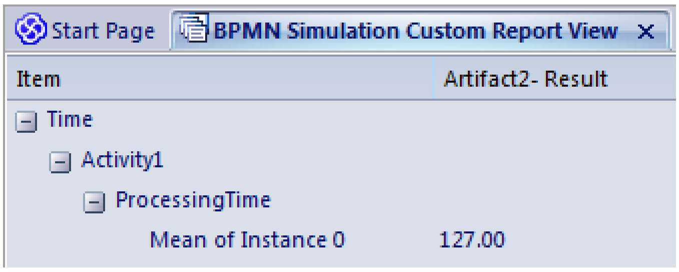
Click on the Run button, and on the 'BPSim Simulation Controller' dialog click on the Run button and select 'Standard Simulation'. The simulation is configured to cycle through the process once. When the simulation is complete, click on the Open Result button, and on the 'BPMN Simulation Report View' right-click and select the 'Show Only Non-Empty Items' option. This gives you, for the Activity1 element on which the parameters were set, these results:



Item	Artifact2- Result
Time	
Activity1	
Mean Of Processing Time	127
Average Time In Task	127
Max Of Processing Time	127
Maximum Time In Task	127
Min Of Processing Time	127
Minimum Time In Task	127
Sum Of Processing Time	127
Total Time In Task	127

All of these derived results are 127 minutes, the sum of the initial values of the original seven 'Time' parameters. The individual parameters are not processed separately.

If you return to the 'BPSim Simulation Controller' dialog and click on the Run button, selecting 'Custom Simulation' this time, the Open Result button displays the 'BPMN Simulation Custom Report View'. In the configuration, the Result Requests were for the mean values of the seven parameters. In the Report View for the simulation, you only see the mean of the single aggregated parameter, ProcessingTime, as 127 minutes.



Item	Artifact2- Result
⊟ Time	
⊟ Activity1	
⊟ ProcessingTime	
Mean of Instance 0	127.00

Compare BPSim Configurations

When you develop a BPSim configuration, you can define a wide range of parameters to set prior to running simulations and observing the effects both of those settings and of changes in selected settings. To facilitate using and managing multiple 'what-if' scenarios, it is recommended that you create copies of the original configuration (as Artifact elements) and make the setting changes in the copies.

A useful facility in creating variations of a configuration is to apply inheritance, whereby the data and parameters you do NOT intend to vary are held in one configuration, and only those parameters that you change are held in another. The 'variable' configuration uses (inherits) the common data held in the base configuration, so you do not have to re-create that common data in the 'variable' configuration.

You can then run simulations on the changed configurations and on the original 'baseline' and compare the simulation reports to see what differences occurred in the run-time variables, and then run and display the comparisons of the configurations to see what changes in parameter settings gave rise to those run-time differences.

By running simulations under the original and copy configurations, comparing the results and the changes that caused the results, and modifying the model accordingly, you can achieve a very high degree of control in streamlining the business process you are developing.

Access

Context Menu	In a diagram or the Browser window | Right-click Business Process Simulation Artifact | Show BPSim Configuration > Review tab > Configuration Summary tab

Configuration Summary Tab

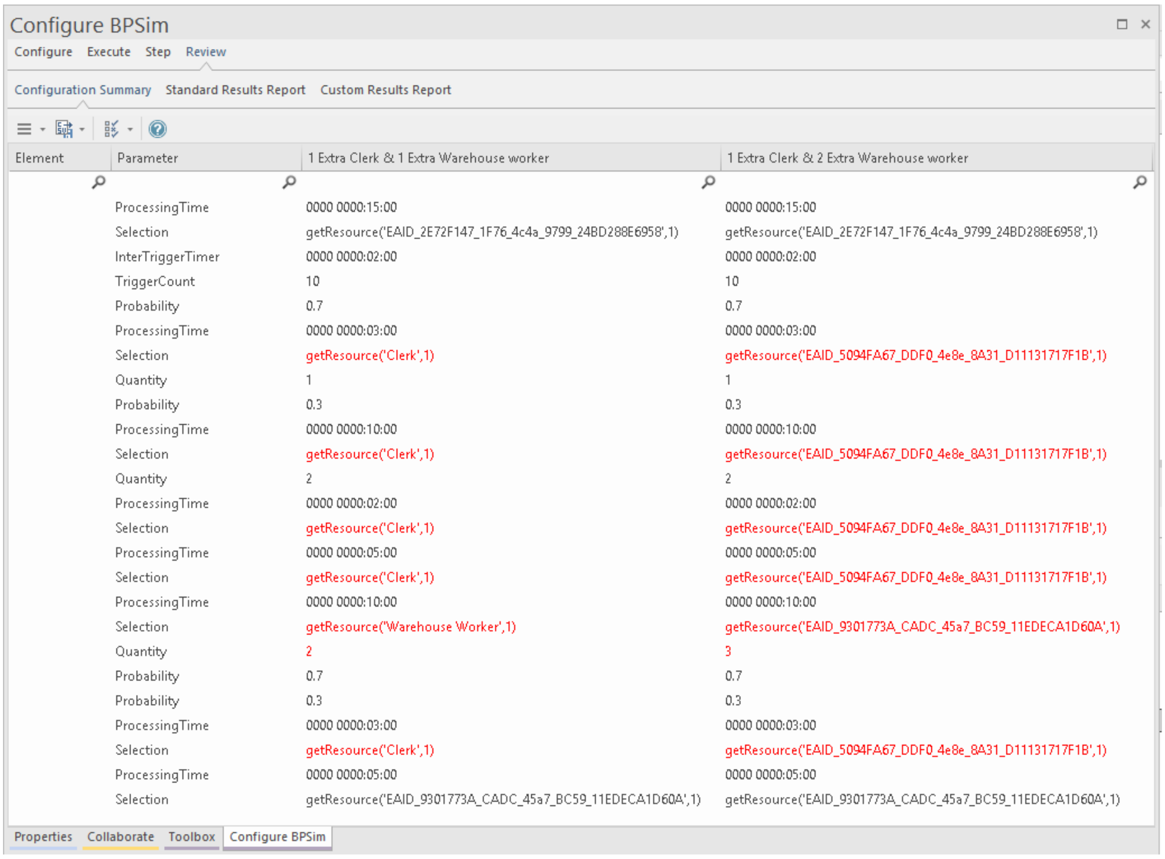
This view initially displays the parameters in the selected configuration that have values and, in the column under the configuration name, the values set for those parameters. If the window is docked you can review the results more easily by dragging the tab off the workspace to become a floating display, and expand the display to full screen size.

To compare two (or more) configurations, click on another BPSim Configuration Artifact element in the Browser window or diagram and either:

- Drag it onto the report view or
- Select the 'Show BPSim Configuration' option

You can also click on the ☰ icon on the 'Configuration Summary' tab, click on the 'Add BPSim Configuration' option and browse for and select the Artifact element on the 'Select Element' dialog.

The parameter hierarchy now contains any additional parameters in that configuration, and its parameter values display in a column to the left of the original configuration values, with the name of the Artifact in the column header.

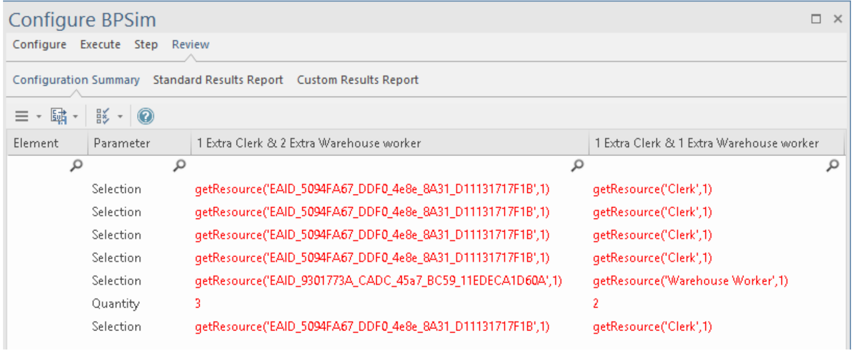


Element	Parameter	1 Extra Clerk & 1 Extra Warehouse worker	1 Extra Clerk & 2 Extra Warehouse worker
	ProcessingTime	0000 0000:15:00	0000 0000:15:00
	Selection	getResource('EAID_2E72F147_1F76_4c4a_9799_24BD288E6958',1)	getResource('EAID_2E72F147_1F76_4c4a_9799_24BD288E6958',1)
	InterTriggerTimer	0000 0000:02:00	0000 0000:02:00
	TriggerCount	10	10
	Probability	0.7	0.7
	ProcessingTime	0000 0000:03:00	0000 0000:03:00
	Selection	getResource('Clerk',1)	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)
	Quantity	1	1
	Probability	0.3	0.3
	ProcessingTime	0000 0000:10:00	0000 0000:10:00
	Selection	getResource('Clerk',1)	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)
	Quantity	2	2
	ProcessingTime	0000 0000:02:00	0000 0000:02:00
	Selection	getResource('Clerk',1)	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)
	ProcessingTime	0000 0000:05:00	0000 0000:05:00
	Selection	getResource('Clerk',1)	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)
	ProcessingTime	0000 0000:10:00	0000 0000:10:00
	Selection	getResource('Warehouse Worker',1)	getResource('EAID_9301773A_CADC_45a7_BC59_11EDECA1D60A',1)
	Quantity	2	3
	Probability	0.7	0.7
	Probability	0.3	0.3
	ProcessingTime	0000 0000:03:00	0000 0000:03:00
	Selection	getResource('Clerk',1)	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)
	ProcessingTime	0000 0000:05:00	0000 0000:05:00
	Selection	getResource('EAID_9301773A_CADC_45a7_BC59_11EDECA1D60A',1)	getResource('EAID_9301773A_CADC_45a7_BC59_11EDECA1D60A',1)

You can review and manipulate the information on the report using options available from the toolbar and the right-click context menu.

BPMN Simulation Configuration Summary Options

Option	Description
≡ ▾	Select this toolbar option to display a short menu of options: • Add BPSim Configuration - displays the 'Select Element' dialog, through which you can browse for and select another BPSim Artifact and display its parameters next to those already on the page • Reload - refresh the display and return column changes to the original widths and positions • Clear List - clear the selected Artifacts and their parameters from the display • Remove Report <name> - clears the named Artifact and its parameters from the page; one of these options is provided for each Artifact currently shown on the page
	Click on this toolbar icon to export the records on this page to a CSV file or XML file. The 'Save As' browser displays, from which you select the location into which to save the file.
☑ ▾	Displays two options for filtering the information displayed on the page; the options are enabled when you have two or more configurations shown: • Show Only Different Items - select this option to restrict the display to only those parameters where the values differ between the configurations; click on

	the option again to deselect it ● Highlight Different Items - (when two or more configurations have different parameter values) displays the differing parameter values in red; this option is disabled if you have already selected the 'Show Only Different Items' option 

Configure BPSim

Element	Parameter	1 Extra Clerk & 2 Extra Warehouse worker	1 Extra Clerk & 1 Extra Warehouse worker
	Selection	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)	getResource('Clerk',1)
	Selection	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)	getResource('Clerk',1)
	Selection	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)	getResource('Clerk',1)
	Selection	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)	getResource('Clerk',1)
	Selection	getResource('EAID_9301773A_CADC_45a7_BC59_11EDECA1D60A',1)	getResource('Warehouse Worker',1)
	Quantity	3	2
	Selection	getResource('EAID_5094FA67_DDF0_4e8e_8A31_D11131717F1B',1)	getResource('Clerk',1)

BPSim Charts

The 'Charts' page of the Diagram Toolbox provides two icons specifically to generate Charts that reflect selected results from BPSim simulations. These are:

- BPSim Result Chart - to generate a Chart that reflects selected results from a series of standard BPSim simulations
- BPSim Custom Result Chart - to generate a Chart that reflects results from a series of customized BPSim simulations

As for other Chart Artifacts, both BPSim Chart types can be quickly configured to display the simulation results in variations of a Time Line chart, 2-dimensional Bar Chart or 3-dimensional Bar Chart.

Prerequisites

To populate the Charts created from the Business Process Simulation Artifacts, you select the Result Artifacts created during the simulation of each configuration that you want to show. Therefore, the initial simulations must be performed first, and the Report Artifacts generated.

Access

Display the 'Charts' page of the Diagram Toolbox using any of the methods outlined in this table.

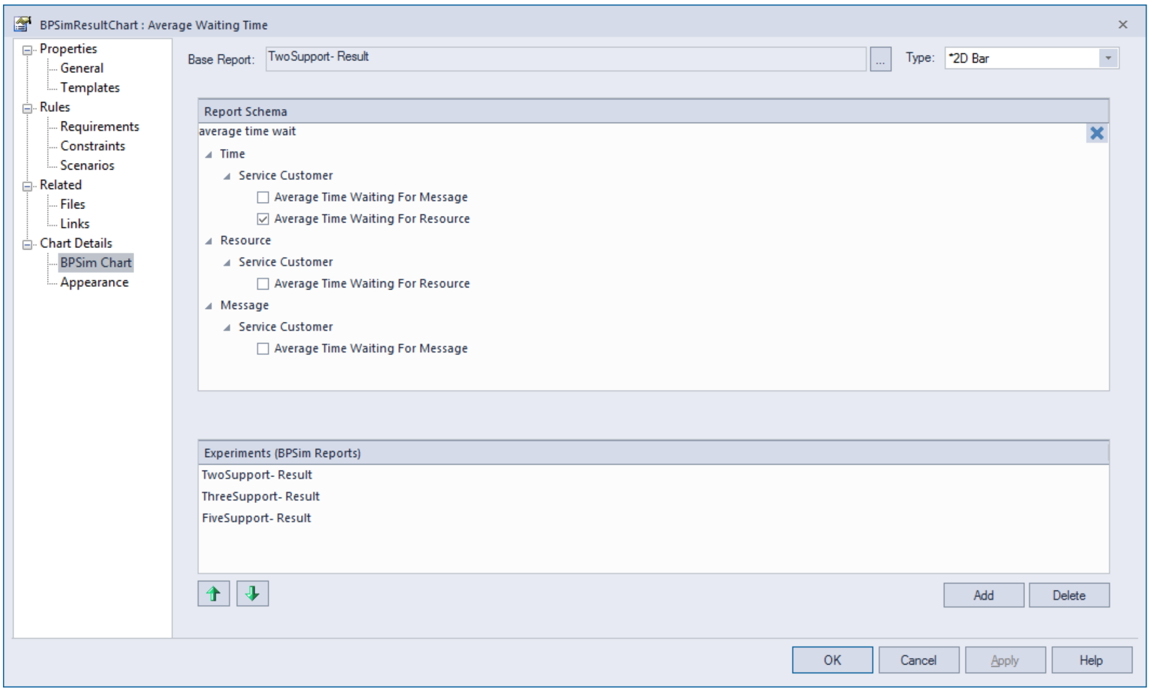
Then, drag the BPSim <type> Chart Artifact icon onto the diagram - a new Chart element is created.

Double-click on the new Chart element to open the 'Properties' dialog, showing the 'BPSim Chart' page.

Ribbon	Design > Diagram > Toolbox > Charts
Keyboard Shortcuts	Ctrl+Shift+3 |Charts
Other	You can display or hide the Diagram Toolbox by clicking on the » or « icons at the left-hand end of the Caption Bar at the top of the Diagram View.

Select Results to Display In Chart

Complete the fields on the 'BPSim Chart' page of the Chart 'Properties' dialog.



BPSimResultChart : Average Waiting Time

Base Report: TwoSupport- Result

Type: *2D Bar

Report Schema

average time wait

- Time
  - Service Customer
    - □ Average Time Waiting For Message
    - ☑ Average Time Waiting For Resource
- Resource
  - Service Customer
    - □ Average Time Waiting For Resource
- Message
  - Service Customer
    - □ Average Time Waiting For Message

Experiments (BPSim Reports)

TwoSupport- Result
ThreeSupport- Result
FiveSupport- Result

Option	Action
Base Report	Click on the ... button to display the 'Select <<BPSimReport>>Artifact' dialog, and select the report Artifact for the simulation results against which other simulation results will be compared. This Result Artifact is also added to the 'Experiments' panel as the first in the list of report results to be compared.
Type	Click on the drop-down arrow and select the format type of the Chart in which to display the results - 2D Bar, 3D Bar or Line. After you have specified the report parameters to compare, you can select the 'Appearances' page of the dialog and define the appearance of the Bar Chart or Time Line graph.
Report Schema	Expand the hierarchy as necessary and select the checkbox against each property to display on the Chart. Each property will be represented by a separate line or group of bars on the Chart. Usually you would select similar objects (such as different resources) and the same single property for each object (such as the degree of utilization of the resource). You have a wealth of properties to examine and compare, but any more than a couple on the same Chart makes the Chart hard to read.
Experiments (BPSim Reports)	This panel lists the simulation report results (as BPSim Result Artifacts) that you have selected to compare using the Chart, in the sequence in which their selected parameter will be shown on the Chart. Usually, the Base Report remains as first in the list and the result for its parameter is shown at the left of the Chart. If you want to change the sequence, click on the Result Artifact name and click on the appropriate Up/Down green arrow button. To add further Result Artifact names to the list, click on the Add button and browse for and select the Artifact from the 'Select <<BPSimReport>> Artifact' dialog.

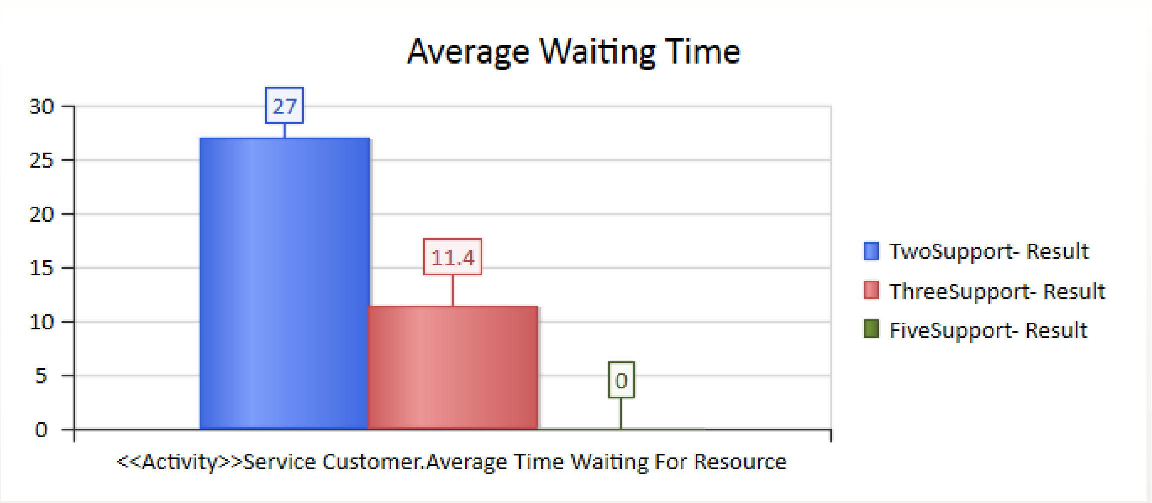
A BPSim Chart Example

In the Help Desk Phone Support example, we created three BPSim Artifacts to do a what-if analysis of 'How many support resources do we need to answer the customer's questions over the phones in an economical way?'

We started with two support resources, then tried three and five resources. After simulation, we have a BPSim Report based on different configurations: TwoSupport-Result, ThreeSupport-Result and FiveSupport-Result.

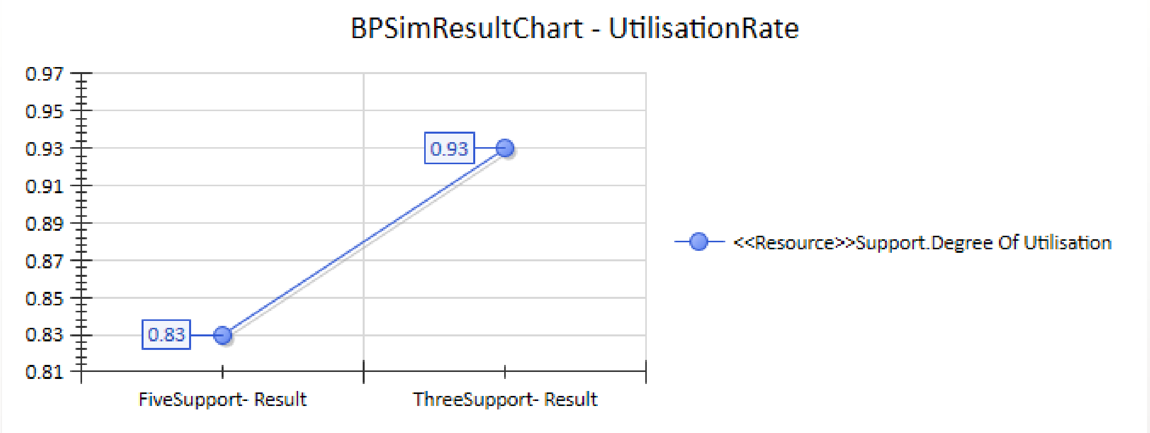
These are the steps to create a Chart to compare the customer's average time waiting for support:

1. Create a BPSim Result Chart on the diagram, and name it *Average Waiting Time.*
2. Double-click on the Chart to open the Properties window, then open the 'BPSim Chart' tab.
3. Click on the ... button to select a Base Report, from which we define the schema (legends) to use in the Chart. Select 'TwoSupport-Result'.
4. Choose this schema:
   - Time | Service Customer | Average Time Waiting For Resource
5. Click on the Add button to add another two BPSim Reports: 'ThreeSupport-Result' and 'FiveSupport-Result'.
6. Click on the OK button, and adjust the size of the Chart element. This Chart gives us direct information.



These are the steps to create a Chart to compare the degree of use of Support:

1. Create a BPSim Result Chart on the diagram, and name it *Utilisation Rate.*
2. Double-click on the Chart to open the Properties window, then open the 'BPSim Chart' tab.
3. Click on the ... button and select a Base Report, from which to define the schema (legends) to use in the Chart. Select 'TwoSupport-Result'.
4. Choose this schema:
   - Resource | Support | Degree Of Utilisation
5. Click on the Add button to add another two BPSim Reports: 'ThreeSupport-Result' and 'FiveSupport-Result'.
6. Click on the OK button and adjust the size of the Chart element. This Chart provides specific information.


BPSimResultChart - UtilisationRate
0.97
0.95
0.93
0.91
0.89
0.87
0.85
0.83
0.81
0.83
0.93
FiveSupport- Result
ThreeSupport- Result
<<Resource>>Support.Degree Of Utilisation

BPSim Examples

This section provides several examples of BPMN modeling, BPSim Configuration and analysis of simulation results.

These examples can all be accessed from the EAExample model.

To run the example simulations you must have the BPSim Execution Engine and Java Runtime Environment (JRE) 1.7 or higher installed. To work with Property Parameters you must also have Java Development Kit (JDK) version 1.7 or higher installed.

Meal Order Collaboration Version 1

In this example, we create a very simple model to simulate the communication between a customer and a restaurant for a meal order.

For the customer's process:

1. A customer sends a message to the restaurant to order a meal.
2. The customer will wait for the delivery.
   If the delivery is not fulfilled within 60 minutes, they will call the restaurant, and then continue to wait.
3. Upon the delivery, the customer will have dinner.

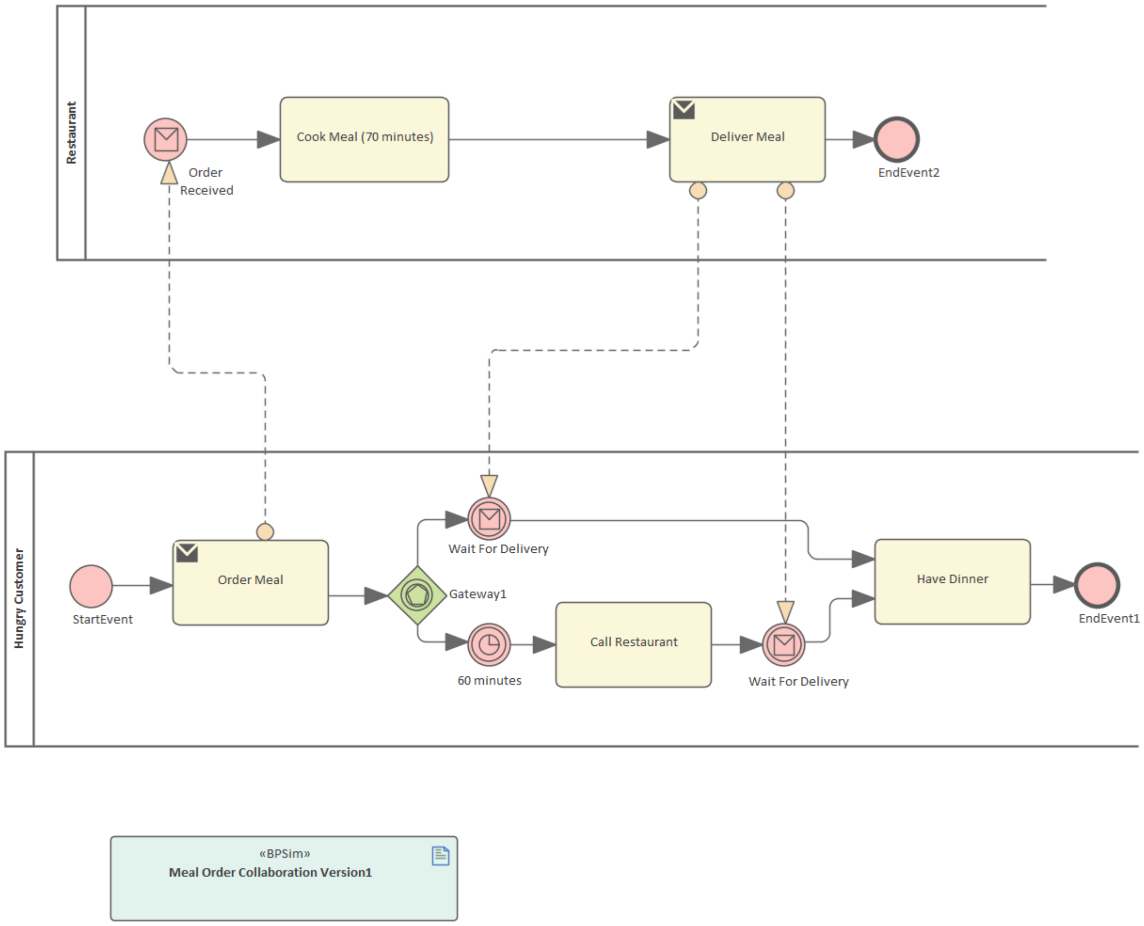
For the restaurant's process:

1. The process is started on receiving a meal order from the customer.
2. The cooking time can be set by the user. This allows experimentation with different events times, e.g. 30 minutes, 70 minutes
3. The restaurant delivers the meal and finishes the process.

Create BPMN Model

To set up a BPMN model that can be used for this BPSim simulation you:

- Create a Collaboration Model with 2 Pools
- Within each Pool create an element for each process
- Connect the elements with Message Flows for the process communication
- Include a BPSim Artifact for setting the simulation details.

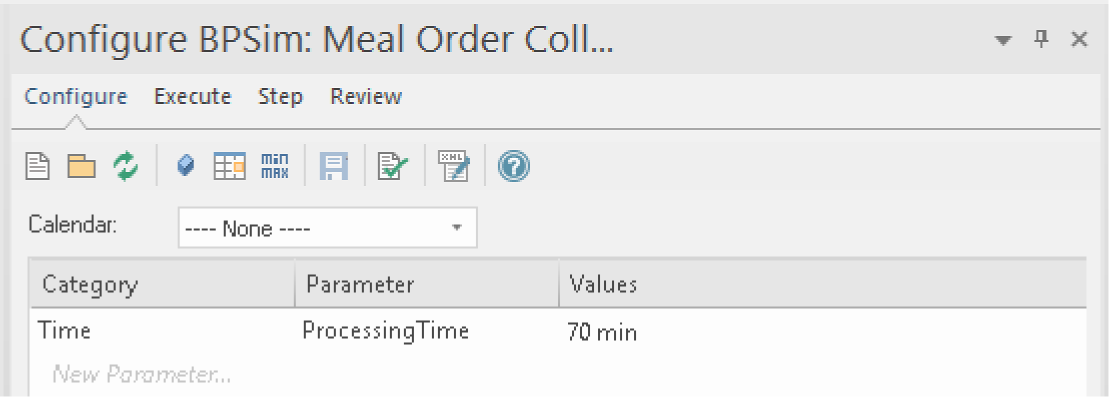


Configure BPSim

In this example we configure the following BPSim parameters:

- Set the TriggerCount of StartEvent in Hungry Customer to 1

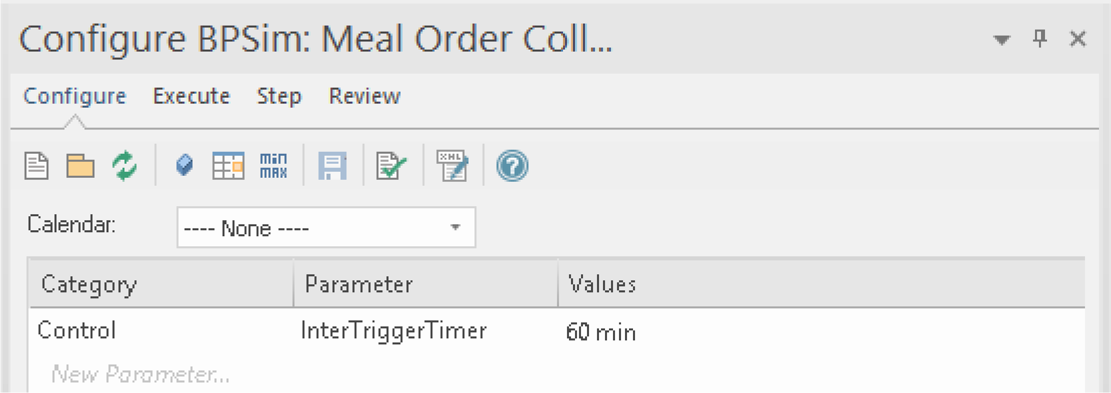
- Set the ProcessingTime of Cook Meal to 70 minutes



Calendar: ---- None ----

Category	Parameter	Values
Time	ProcessingTime	70 min
New Parameter...		

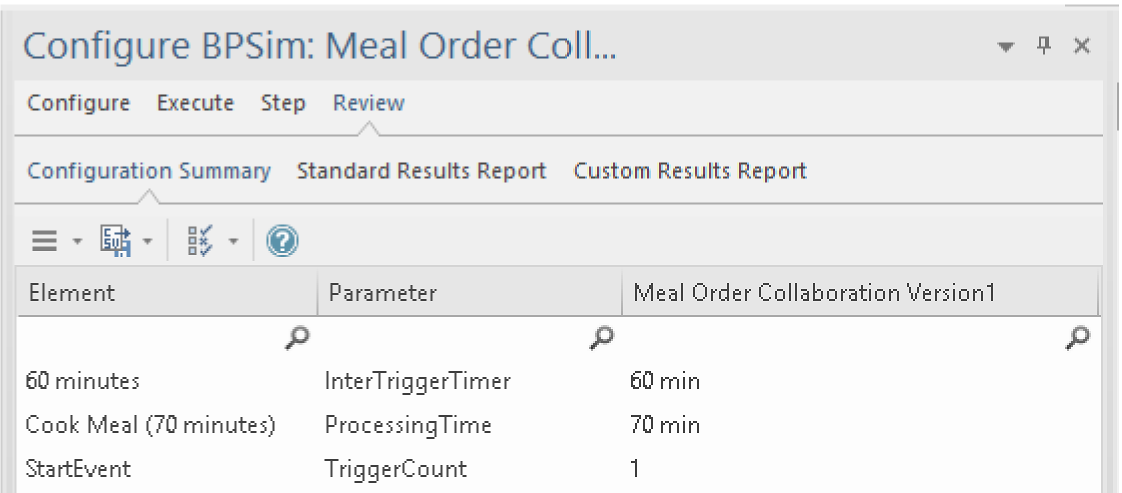
- Set the InterTriggerTimer of Intermediate Event to 60 minutes



Calendar: ---- None ----

Category	Parameter	Values
Control	InterTriggerTimer	60 min
New Parameter...		

- Default settings on other BPSim parameters, here is a list of configurations, you can view thourgh Review > Configuration Summary

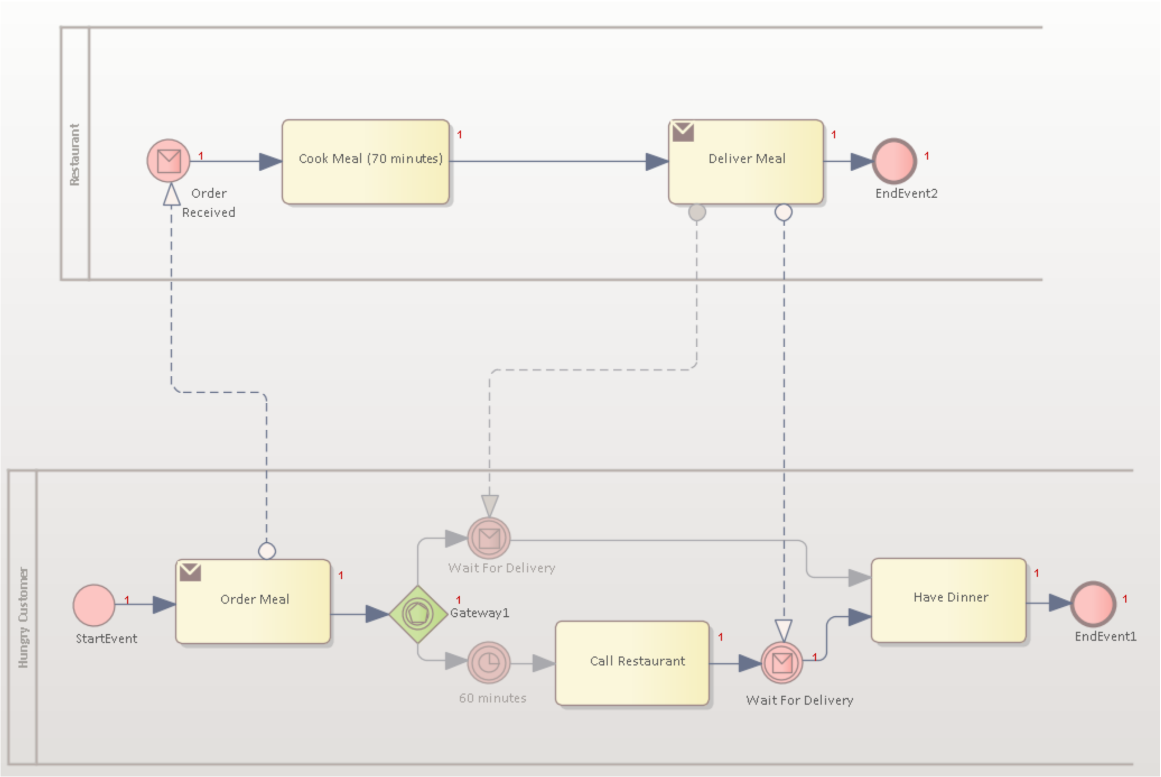


Element	Parameter	Meal Order Collaboration Version1
60 minutes	InterTriggerTimer	60 min
Cook Meal (70 minutes)	ProcessingTime	70 min
StartEvent	TriggerCount	1

Simulation

Ensure the Config BPSim window is open (Simulate > Process Analysis > BPSIM > Open BPSIM Manager).

Navigate to Execute tab and run the Standard Simulation:



The Exclusive Event Gateway was triggered by the 60 minutes Timer event when the Cooking Meal task took 70 minutes.

If we change the BPSim setting for Task: *Cook Meal*: ProcessingTime from 70 minutes to 30 minutes, the Exclusive Event Gateway will be triggered by the message event *Wait For Delivery* and *Call Restaurant* task will not be activated at all.

Meal Order Collaboration Version 2

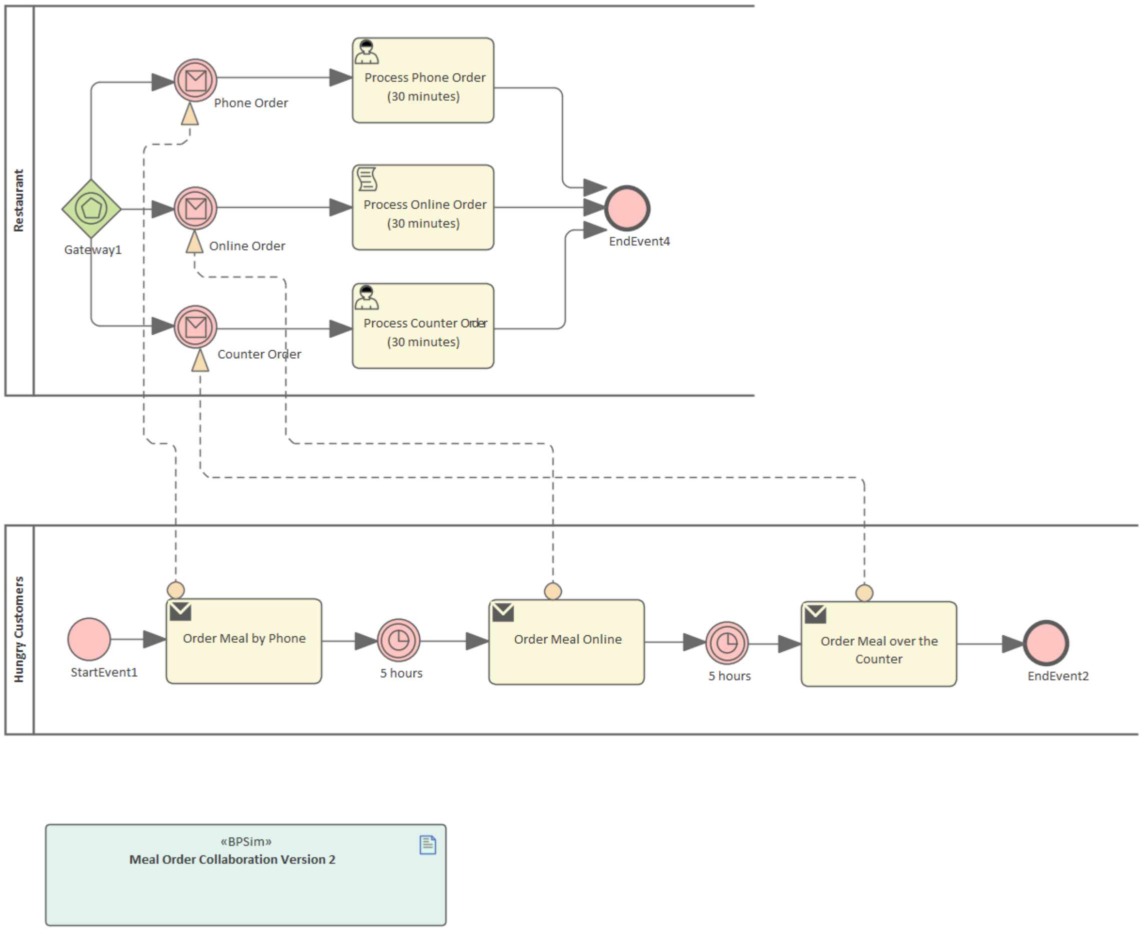
In this example, we create a model to simulate the communication between a customer and a restaurant for a meal order. The customer places three different orders by: phone, online and over the counter.

Create BPMN Model

To set up a BPMN model that can be used for this BPSim simulation you:

- Create a Collaboration Model with 2 Pools
- Within each pool create an element for each process
- Create Message Flows depicting the Process communication
- Include a BPSim Artifact for setting the simulation details.

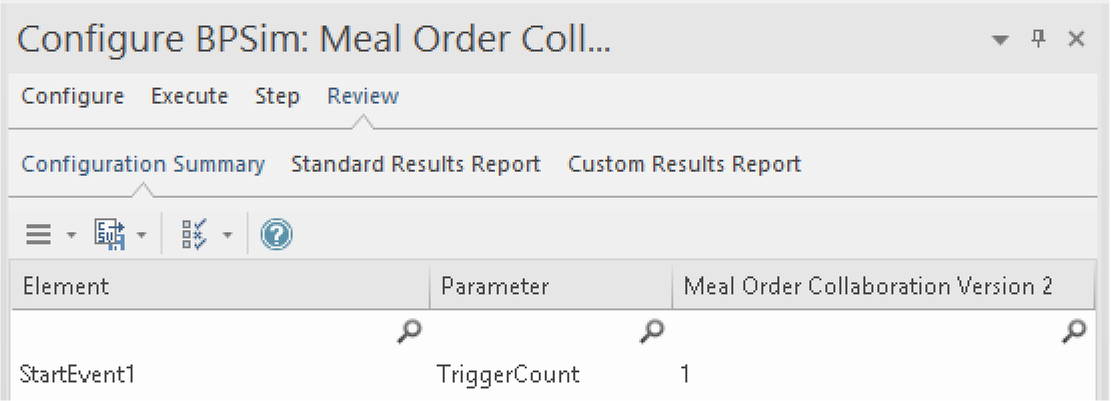
Please note: the Event-Exclusive gateway's *instantiate* attribute was set to **true**, which means the process was started by this *Start Event Gateway*.


Restaurant
Phone Order
Process Phone Order (30 minutes)
Gateway1
Online Order
Process Online Order (30 minutes)
EndEvent4
Counter Order
Process Counter Order (30 minutes)
Hungry Customers
StartEvent1
Order Meal by Phone
5 hours
Order Meal Online
5 hours
Order Meal over the Counter
EndEvent2
«BPSim»
Meal Order Collaboration Version 2

Configure BPSim

In this example we configure the following BPSim parameters:

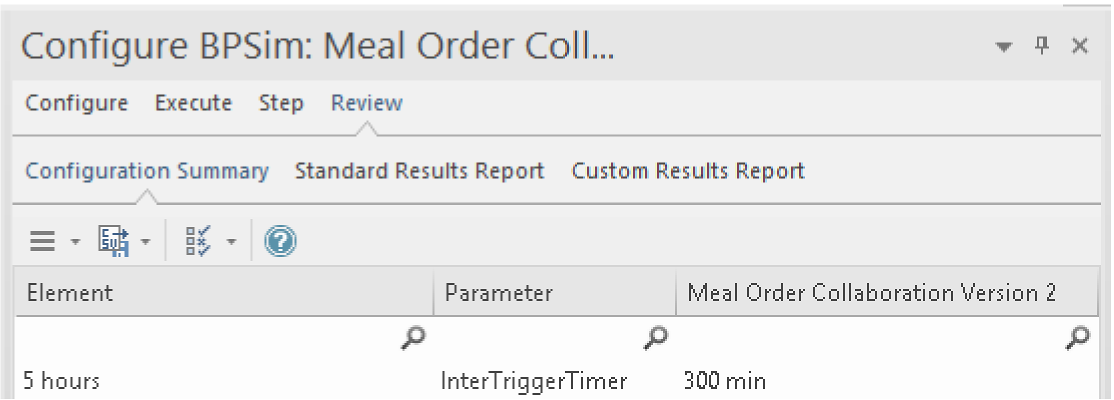
- For the Start Event, set the parameter TriggerCount to value 1



Configure BPSim: Meal Order Coll...

Element	Parameter	Meal Order Collaboration Version 2
StartEvent1	TriggerCount	1

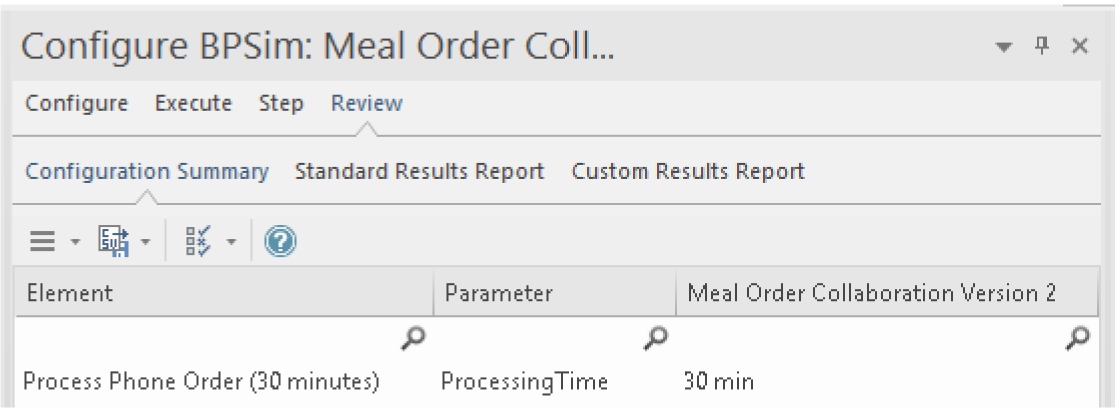
- For two of the Intermediate Event 5 hours, set the parameter InterTriggerTimer to value 300 min



Configure BPSim: Meal Order Coll...

Element	Parameter	Meal Order Collaboration Version 2
5 hours	InterTriggerTimer	300 min

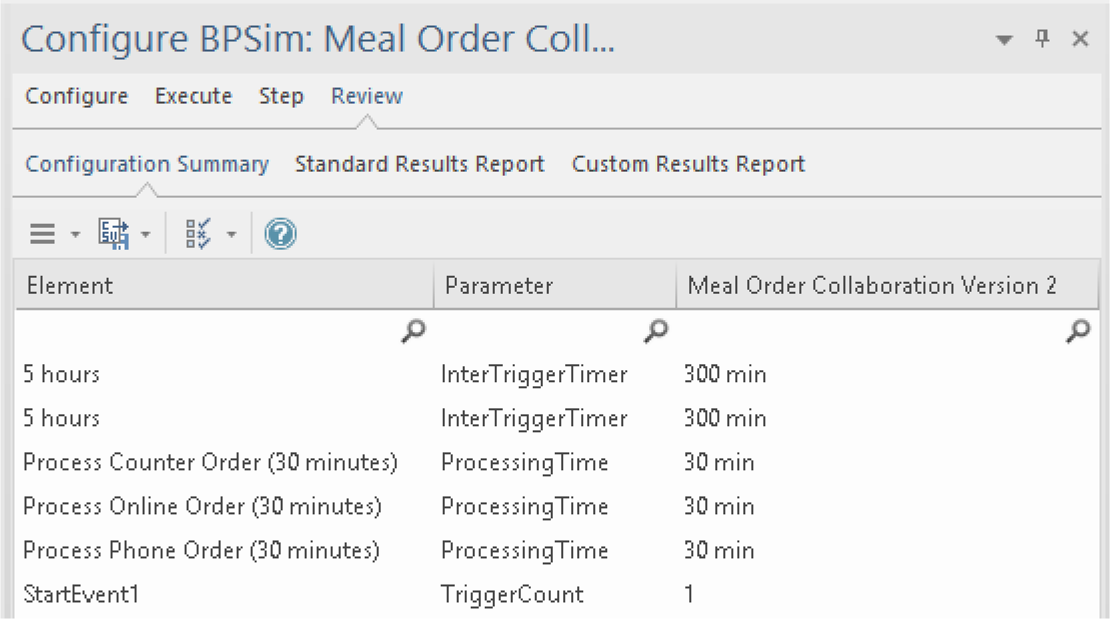
- For three of the tasks in Restaurant, set parameter ProcessingTime to value 30 min



Configure BPSim: Meal Order Coll...

Element	Parameter	Meal Order Collaboration Version 2
Process Phone Order (30 minutes)	ProcessingTime	30 min

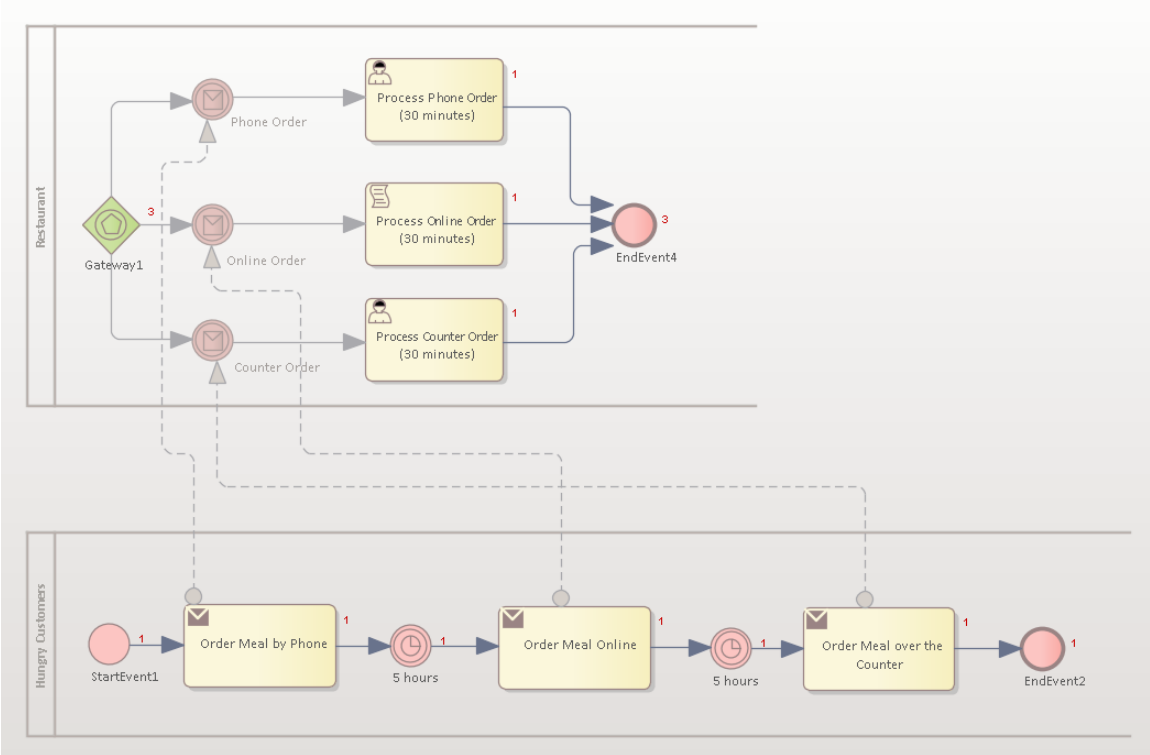
- Default settings on other BPSim parameters, here is a list of configurations, you can view thourgh Review > Configuration Summary



Element	Parameter	Meal Order Collaboration Version 2
5 hours	InterTriggerTimer	300 min
5 hours	InterTriggerTimer	300 min
Process Counter Order (30 minutes)	ProcessingTime	30 min
Process Online Order (30 minutes)	ProcessingTime	30 min
Process Phone Order (30 minutes)	ProcessingTime	30 min
StartEvent1	TriggerCount	1

Simulation

Navigate to Execute tab and run the Standard Simulation:



The *Restaurant* process was instantiated 3 times by the Event-Ecxlusive gateway through 3 different messages.

Help Desk Phone Support Simulation

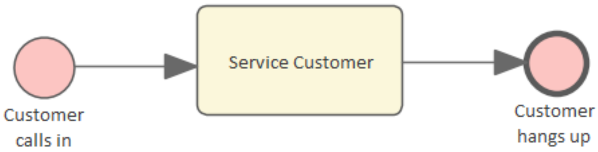
In this example, we create a very simple model to simulate a Help Desk phone support process.

We set up a scenario in which resources are limited and the requests have to be put in a waiting queue for a resource. Then we try to seek a balance point between a customer's waiting time and the number of resources, using a what-if analysis.

Firstly, we model this process step-by-step, starting with a simple parameter setting than can be calculated with pen and paper, then verifying it with BPSim. After that, we perform a what-if analysis that might help the manager to make a decision.

Create BPMN Model

The model itself is very simple, consisting of a Start Event, a Task and an End Event.



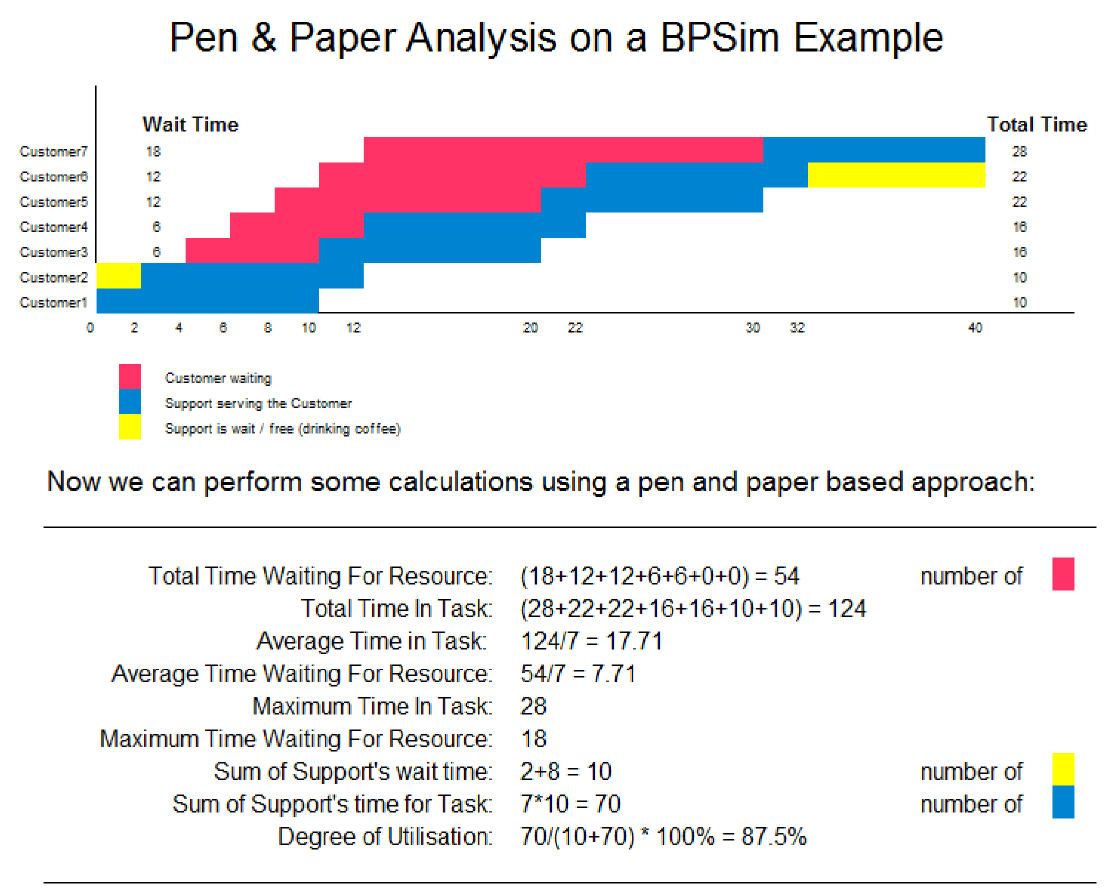
- Create a Start Event called *Customer calls in*
- Add a Sequence Flow to a target Abstract Task called *Service Customer*
- Add a Sequence Flow to a target End Event called *Customer hangs up*

Create a BPMN2.0 Resource named *Support;* this element will be used in the BPSim Configuration.

Pen and Paper Analysis

We will use pen and paper to analyze this case:

- 7 customers call in at 2 minute intervals
- 2 support resources are available
- Each service will take 10 minutes



Pen & Paper Analysis on a BPSim Example

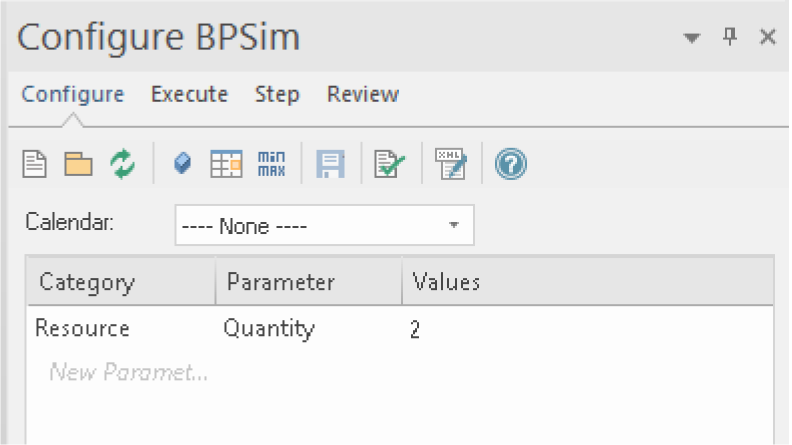
Now we can perform some calculations using a pen and paper based approach:

Total Time Waiting For Resource:	(18+12+12+6+6+0+0) = 54	number of 🟥
Total Time In Task:	(28+22+22+16+16+10+10) = 124	
Average Time in Task:	124/7 = 17.71	
Average Time Waiting For Resource:	54/7 = 7.71	
Maximum Time In Task:	28	
Maximum Time Waiting For Resource:	18	
Sum of Support's wait time:	2+8 = 10	number of 🟨
Sum of Support's time for Task:	7*10 = 70	number of 🟦
Degree of Utilisation:	70/(10+70) * 100% = 87.5%	

Seen from this result, it is already a very complicated computation for such a simple model when resource constraints are applied. When the process expands and more constraints are applied, analyzing the process with pen and paper will quickly become impossible. We will demonstrate how BPSim can help.

BPSim Configuration

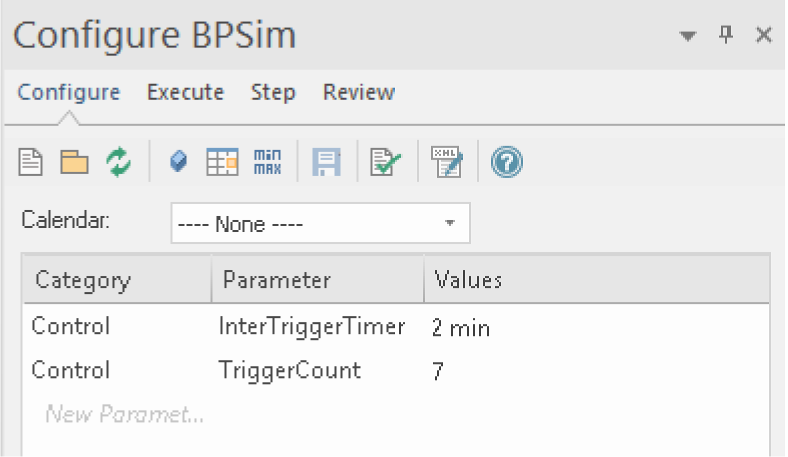
1. Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager').
2. Click on the 📄 icon and the Add New button, and create a Business Process Simulation Artifact named *Pen & Paper Analysis 7 Customers.*
3. Click on the 📁 icon and browse for and select the Package containing the corresponding BPMN 2.0 model.
4. Open the model diagram and click on the Resource element called 'Support'.
5. In the 'Category' column on the window, click on the 'New Parameter' drop-down arrow and select 'Resource', then click on the 'Parameter' drop-down arrow and select 'Quantity', and in the 'Values' field type '2'.
6. Click on the 💾 icon.



Calendar: ---- None ----

Category	Parameter	Values
Resource	Quantity	2
New Paramet...		

1. In the diagram, click on the 'Customer calls in' Start Event element.
2. In the 'Category' column in the window click on the 'New Parameter' drop-down arrow and select 'Control'.
3. Click on the 'Parameter' drop-down arrow and select 'InterTriggerTimer'.
4. In the 'Value' field, click on the ... button, select the 'Constant' tab and 'Numeric', type '2' in the 'Constant Numeric' field and select 'minutes' in the 'TimeUnit' field, then click on the OK button.
5. Repeat steps 2 and 3, selecting 'TriggerCount' in the 'Parameter' field, and in the 'Value' field type '7'.
6. Click on the 💾 icon.

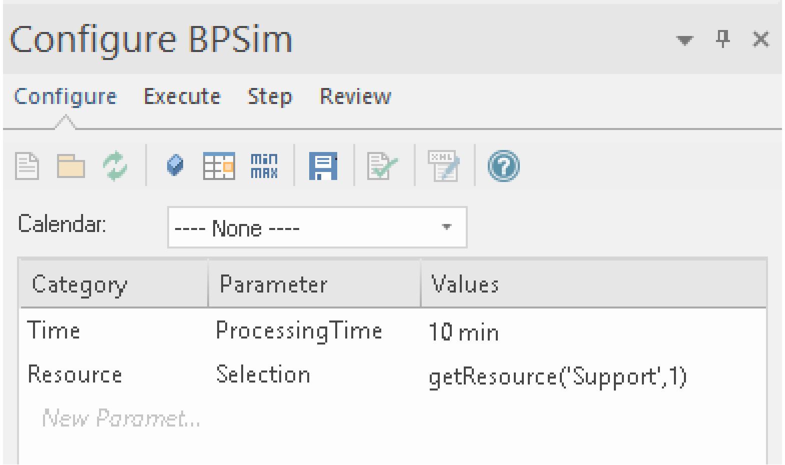


Calendar: ---- None ----

Category	Parameter	Values
Control	InterTriggerTimer	2 min
Control	TriggerCount	7
New Paramet...		

1. In the diagram, click on the 'Service Customer' Activity element.
2. In the 'Category' column in the window click on the 'New Parameter' drop-down arrow and select 'Time'.
3. Click on the 'Parameter' drop-down arrow and select 'ProcessingTime'.
4. In the 'Value' field, click on the ... button, select the 'Constant' tab and 'Numeric', type '10' in the 'Constant Numeric' field and select 'minutes' in the 'TimeUnit' field, then click on the OK button.
5. In the 'Category' column in the window click on the 'New Parameter drop-down arrow and select 'Resource'.
6. Click on the 'Parameter' drop-down arrow and select 'Selection'.
7. In the 'Value' field, click on the ... button, select 'Support' (the name of the Resource element you created in the model) and click on the Add Selection by Resource(s) button to move 'Support' into the 'Resource or Role' column.
8. In the 'Quantity Required' column type 1.
9. Click on the OK button. In the 'Values' field the automatically-generated expression *bpsim:getResource('Support',1)* displays.

10. Click on the icon.



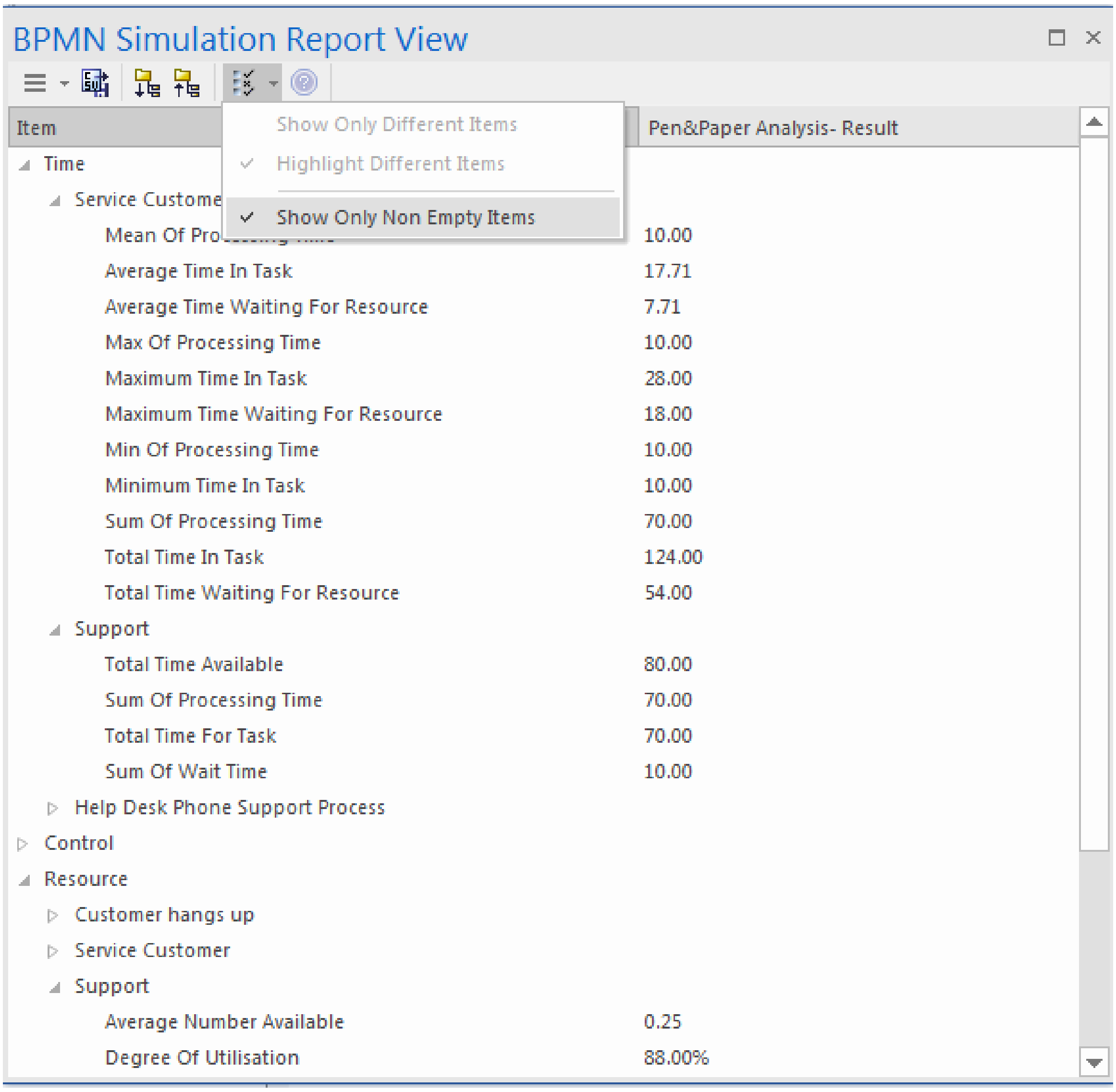
Calendar: ---- None ----

Category	Parameter	Values
Time	ProcessingTime	10 min
Resource	Selection	getResource('Support',1)
New Paramet...		

Run the Simulation

1. On the Configure BPSim window click on the 'Execute' tab and on the icon in the toolbar.

2. When the simulation is complete, click on the icon to open the 'Configuration Summary' tab of the 'Results' tab'


BPMN Simulation Report View
Show Only Different Items
✓ Highlight Different Items
✓ Show Only Non Empty Items

Item	Pen&Paper Analysis- Result
◢ Time	
◢ Service Custome	
Mean Of Pro	10.00
Average Time In Task	17.71
Average Time Waiting For Resource	7.71
Max Of Processing Time	10.00
Maximum Time In Task	28.00
Maximum Time Waiting For Resource	18.00
Min Of Processing Time	10.00
Minimum Time In Task	10.00
Sum Of Processing Time	70.00
Total Time In Task	124.00
Total Time Waiting For Resource	54.00
◢ Support	
Total Time Available	80.00
Sum Of Processing Time	70.00
Total Time For Task	70.00
Sum Of Wait Time	10.00
▷ Help Desk Phone Support Process	
▷ Control	
◢ Resource	
▷ Customer hangs up	
▷ Service Customer	
◢ Support	
Average Number Available	0.25
Degree Of Utilisation	88.00%

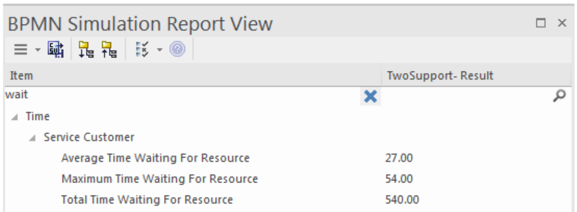
The results match the pen and paper analysis.

Simulation - 2 Support Resources for 20 customers

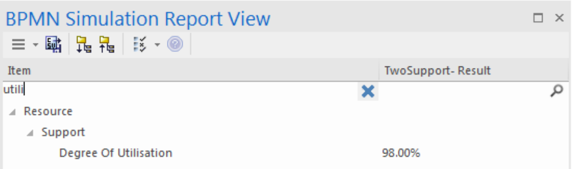
You can create a new Business Process Simulation Artifact by copying an existing BPSim Configuration. Copy the *Pen & Paper Analysis 7 customers* element and press Ctrl+Shift+V to paste, giving the new element the name *TwoSupport*.

1. Double-click on *TwoSupport* to open the Configure BPSim window; you can see that all the configurations are kept from the copied source
2. In the diagram, click on the 'Customer calls in' Start Event element.
3. In the 'Category' column in the window click on the 'Value' field for 'Control' - 'Trigger Count' and change the value to '20'.
4. Click on the 💾 icon.

Run simulation and analyze results



Item	TwoSupport- Result
wait	
Time	
Service Customer	
Average Time Waiting For Resource	27.00
Maximum Time Waiting For Resource	54.00
Total Time Waiting For Resource	540.00



Item	TwoSupport- Result
utili	
Resource	
Support	
Degree Of Utilisation	98.00%

From the report, you can see that:

- The 'Average Time Waiting For Resource' is 27 minutes and the 'Maximum Time Waiting For Resource' is 54 minutes
- The two Support resources - are they busy? If they were not, we might have to change the process to use all of their time and reduce the customer's waiting time; however, the 'Degree Of Utilisation' is at 98%, which indicates that the resources had almost no idle time

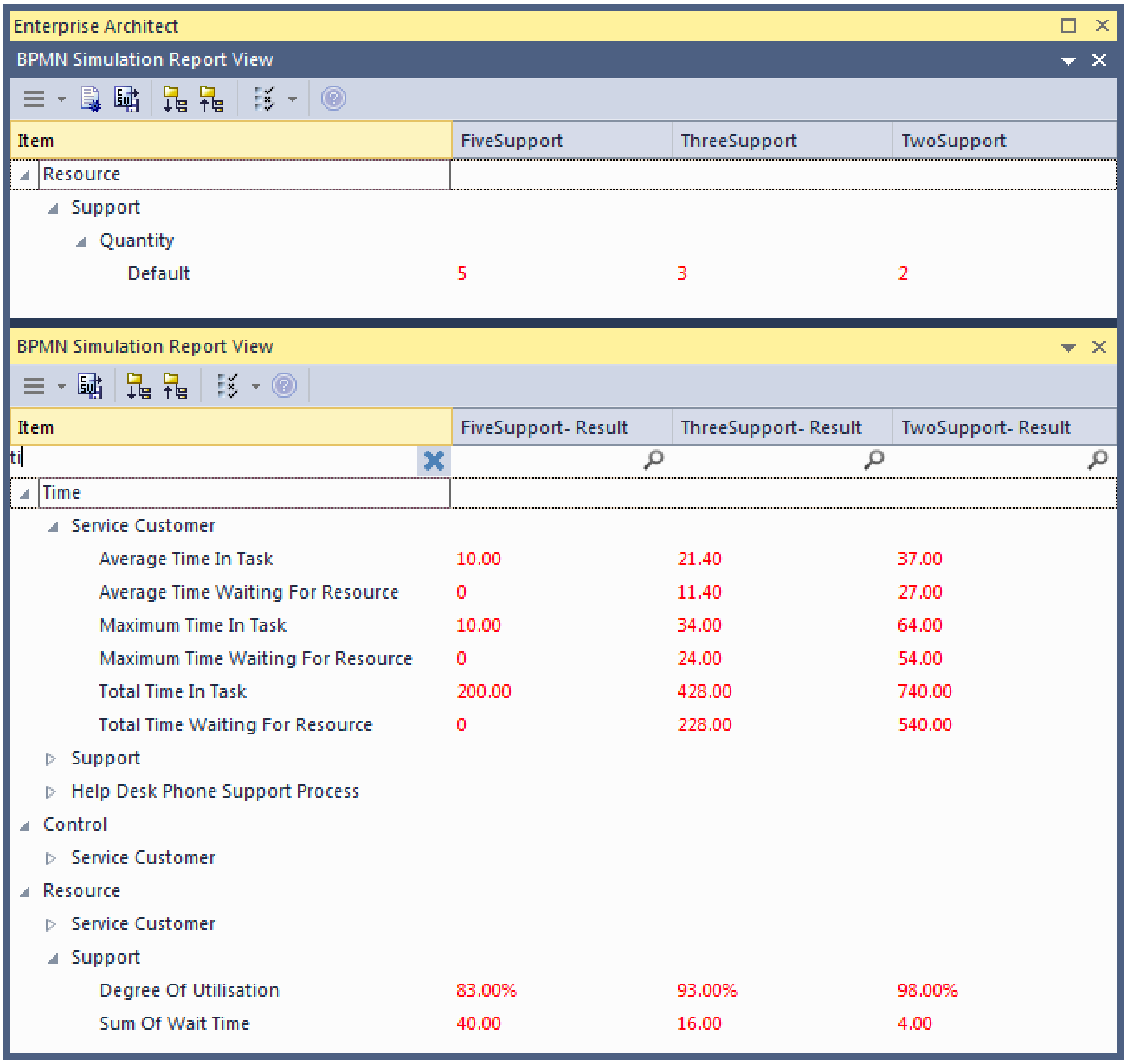
'What-If' I have more staff? Compare 2 Support Resources with 3 and 5 Support Resources

1. Copy *TwoSupport* and press Ctrl+Shift+V to paste, giving the new element the name *ThreeSupport.*
2. Double-click on *ThreeSupport* to open the 'Configure BPSim' dialog.
3. In the diagram, click on the 'Support' Resource element.
4. In the Configure BPSim window, in the 'Values' field for 'Resource' - 'Quantity', type '3'.

1. Again, copy *TwoSupport* and press Ctrl+Shift+V to paste, giving this new element the name *FiveSupport.*
2. Double-click on *FiveSupport* to open the 'Configure BPSim' dialog.
3. In the diagram, click on the 'Support' Resource element.
4. In the Configure BPSim window, in the 'Values' field for 'Resource' - 'Quantity', type '5'.

Run the simulations and do a comparison; in the Browser window:

1. Ctrl+click on *TwoSupport*, *ThreeSupport* and *FiveSupport*, then right-click and select the 'Show BPSim Configuration' option.
2. Ctrl+click on *TwoSupport-Result*, *ThreeSupport-Result* and *FiveSupport-Result*, then right-click and select the 'Show BPSim Report' option.



Item	FiveSupport	ThreeSupport	TwoSupport
Resource			
Support			
Quantity			
Default	5	3	2

ti

Item	FiveSupport- Result	ThreeSupport- Result	TwoSupport- Result
Time			
Service Customer			
Average Time In Task	10.00	21.40	37.00
Average Time Waiting For Resource	0	11.40	27.00
Maximum Time In Task	10.00	34.00	64.00
Maximum Time Waiting For Resource	0	24.00	54.00
Total Time In Task	200.00	428.00	740.00
Total Time Waiting For Resource	0	228.00	540.00
Support			
Help Desk Phone Support Process			
Control			
Service Customer			
Resource			
Service Customer			
Support			
Degree Of Utilisation	83.00%	93.00%	98.00%
Sum Of Wait Time	40.00	16.00	4.00

Tips:

- Click on the button and on the 'Show Only Different Items' option for both views
- You can dock the views together, so they provide direct comparisons: THESE are the differences in results caused by THOSE differences in configuration
- Toggle the filter bar to filter for the items that interest you

Analysis

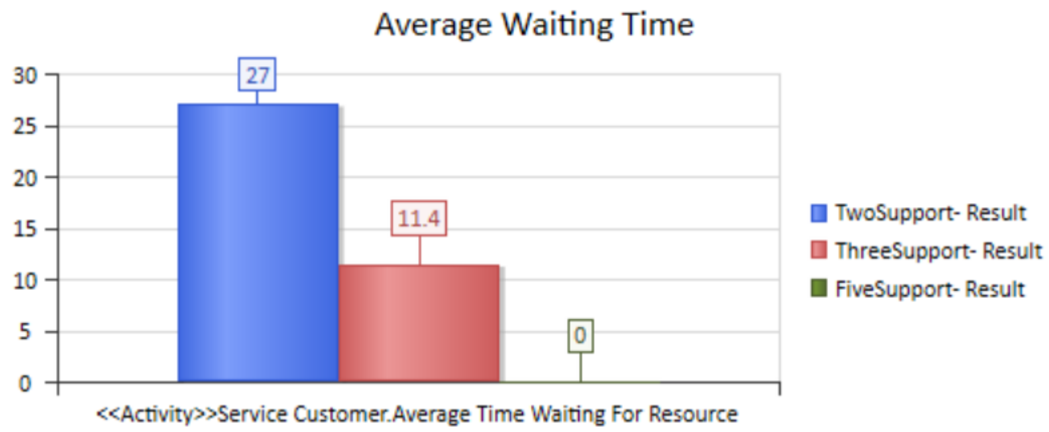
The docked comparison views show the configuration differences and the corresponding result differences.

- The customer's waiting time dropped from 27 minutes (2 Support resources) to 11.4 minutes (3 Support resources) and further down to 0 minutes (5 Support resources)
- The 'Degree Of Utilisation' dropped from 98% (2 Support resources) to 93% (3 Support resources) and further down to 83% (5 Support resources)

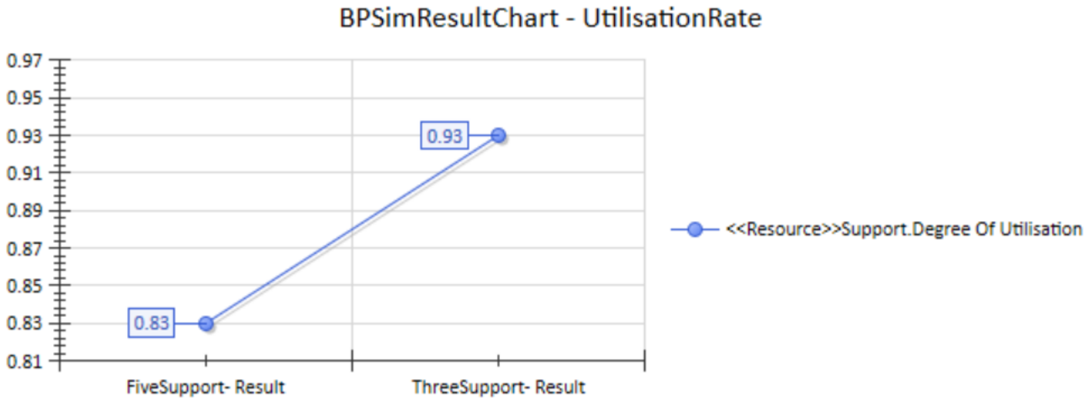
The customers will most likely be satisfied with 5 Support resources; however, the cost might be out of budget. So 3 or possibly 4 Support resources might be a balance point for this case. Try copying one of the Business Process Simulation Artifacts and configure and run a simulation for 4 Support resources.

Show Result with Chart

1. Drag a 'BPSim Result Chart' icon from the Toolbox onto the diagram and create a BPSim Result Chart Artifact; call it *Average Waiting Time.*
2. Right-click on the Artifact and select the 'Properties' option to display the element 'Properties' dialog; click on the 'BPSim Chart' page.
3. Click on the … button and select a Base Report from which to define the schema (legends) to use in the chart; select *TwoSupport-Result.*
4. Choose the schema 'Time' | 'Service Customer' | 'Average Time Waiting For Resource'.
5. Click on the Add button to add another two BPSim Reports: *ThreeSupport-Result* and *FiveSupport-Result*
6. Click on the OK button and adjust the size of the Chart element; this Chart gave us very straightforward information



1. Create another BPSim Result Chart Artifact on the diagram, called *Utilisation Rate.*
2. Double-click on the Artifact to display the element 'Properties' dialog and click on the 'BPSim Chart' tab.
3. Click on the … button and select a Base Report from which to define the schema (legends) to use in the Chart; select *TwoSupport-Result.*
4. Choose the schema 'Resource' | 'Support' | 'Degree Of Utilisation'.
5. Click on the Add button to add another two BPSim Reports: *ThreeSupport-Result* and *FiveSupport-Result.*
6. Click on the OK button and adjust the size of the Chart element.



Calendar-Based Help Desk Phone Support Simulation

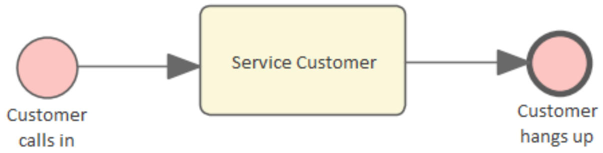
In this example, we create a very simple model to simulate a Help Desk telephone support process, based on calendar settings. We assume that:

- Customers call in at different intervals on weekdays and weekends
- Processing times differ between weekdays and weekends
- There are different numbers of support resources on weekdays and weekends

We model this process step by step, then create calendars and configure the Business Process simulation, which is simple enough to be calculated with pen and paper. After that, we run the simulation to compare that result with the pen and paper analysis.

Create BPMN Model

The model itself is very simple, consisting of a Start Event, a Task and an End Event.

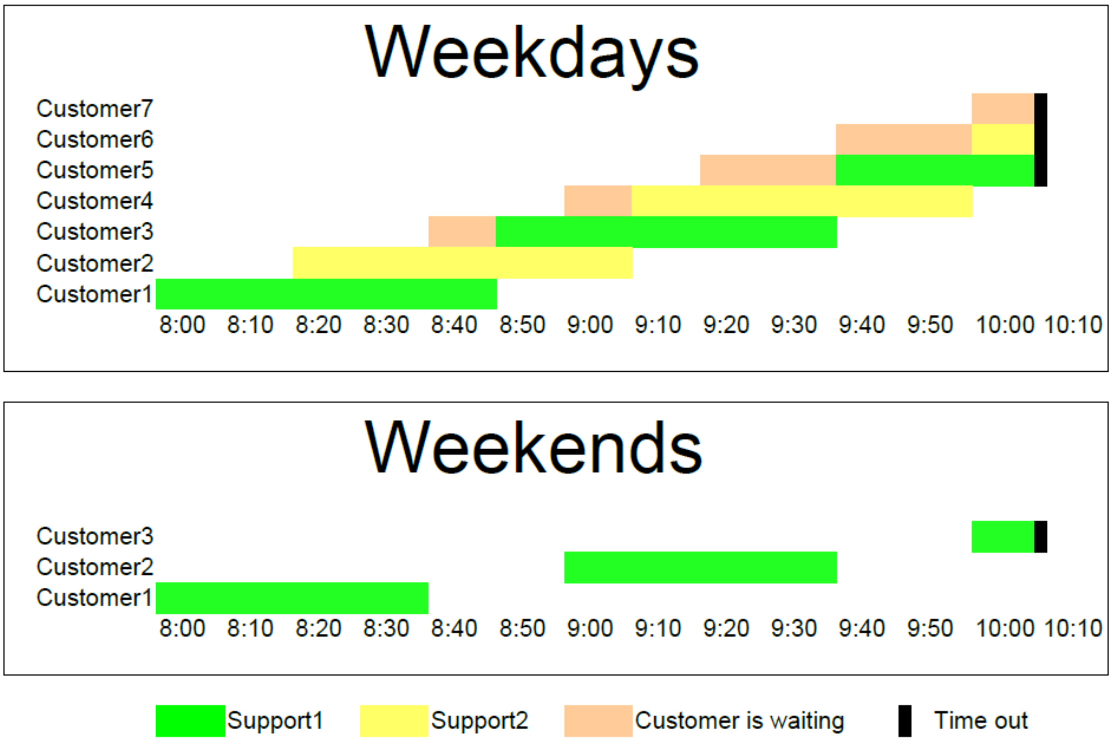


1. Create a Start Event *Customer calls in.*
2. Add a Sequence Flow to the target abstract task Activity *Service Customer.*
3. Add a Sequence Flow to the target End Event *Customer hangs up.*
4. Create a BPMN2.0 Resource named *Support.*
5. Create a BPMN2.0::ResourceRole inside *Service Customer*, give it the name *support* and set the tag *resourceRef* to the name of the Resource element *Support*.

Pen and Paper Analysis

We can use pen and paper to analyze this case:

- The simulation duration is 2 hours and 10 minutes, from 8:00AM to 10:10AM
- A customer calls in every 20 minutes on weekdays
- A customer calls in every 60 minutes on weekends
- To service each customer takes 50 minutes on weekdays
- To service each customer takes 40 minutes on weekends
- There are 2 support resources on weekdays
- There is 1 support resource on weekends



Looking at this result, when resource constraints are applied the computation is quite complicated for such a simple model.

On weekdays

- 7 customers called at intervals of 20 minutes over a duration of 2 hours and 10 minutes
- 4 customer calls were terminated normally
- 2 customer calls were interrupted because of timeout
- 1 customer call was not answered
- Support1 worked continuously for 130 minutes, Support2 worked continuously for 110 minutes

On weekends

- 3 customers called at intervals of 60 minutes over a duration of 2 hours and 10 minutes
- 2 customer calls were terminated normally
- 1 customer call was interrupted because of timeout
- Support1 worked 90 minutes, in 40-minute blocks with an interval of 20 minutes between calls

Now we will see how BPSim can help.

BPSim Configuration

In this section, we first create the Calendars, then we set up the Duration and Start parameters.

For element parameters, you can specify one or more calendars for a given parameter. However, **if any calendar is set for a parameter value, a default value (without any calendar specified) must exist**, otherwise the simulation will not work.

Clicking on the button on the Configure BPSim window toolbar will automatically check this constraint for you.

Task	Action
Create BPSim Artifact and Set Package	1. Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager'). 2. Create a Business Process Simulation Artifact named *Calendar Based Support Process Simulation.* 3. Select the Package containing the corresponding BPMN 2.0 model. 4. Open the diagram containing the model to be simulated.
Calendars	1. On the 'Configure' tab of the Configure BPSim window, click on the icon in the toolbar. The 'Edit BPSim Calendars' dialog displays. 2. Click on the New button to display the 'Event Recurrence' dialog and complete the fields as described here, to create a calendar. (You will create two calendars.) 3. In the 'Event Time' panel, set 'Start' to 08:00AM and 'End' to 05:00PM. 4. In the 'Recurrence Pattern' panel select 'Weekly' and select the checkboxes against 'Monday' through to 'Friday'. 5. In the 'Range of recurrence' panel set 'Start' to '02/11/2020' and select 'No end date'. 6. Click on the OK button. You are prompted to enter a Calendar name; overtype *Calendar_1* with 'Weekdays' and click on the OK button. 7. Click on the New button again and repeat steps 3 to 6 with these values: - 'Start' - 08:00AM - 'End' - 05:00PM - 'Weekly' - 'Saturday' and 'Sunday' - 'Start' to '07/11/2020' and 'No end date' - Overtype *Calendar_2* with 'Weekend' 8. Click on the OK button.
Duration	On the diagram, click on the *Calendar Based Support Process Simulation* BPSim Artifact and, on the 'Configure' tab of the Configure BPSim window, with the 'Calendar' field set to '----None----', create or edit this ScenarioParameter: • Duration - with a Constant value of 0000 002:10:00, which means 0 days, 2 hours and 10 minutes
Customer arrivals	On the diagram, click on the *Customer calls in* StartEvent and, on the 'Configure' tab of the Configure BPSim window, create or edit this Control parameter: • InterTriggerTimer - Value: 0 00:00:00, with the 'Calendar' field set to '----None----' (this default value is necessary) • InterTriggerTimer - Value: 0 00:20:00, with the 'Calendar' field set to 'Weekdays' • InterTriggerTimer - Value: 0 01:00:00, with the 'Calendar' field set to 'Weekends'
Processing Times	On the diagram, click on the *Service Customer* Activity and, on the 'Configure' tab of the Configure BPSim window, create or edit this Time parameter: • ProcessingTime - Value: 0 00:00:00, with the 'Calendar' field set to '----None----' (this default value is necessary) • ProcessingTime - Value: 0 00:50:00, with the 'Calendar' field set to 'Weekdays'

	• ProcessingTime - Value: 0 00:40:00, with the 'Calendar' field set to 'Weekends'
Resources	On the diagram, click on the *Support* Resource and, on the 'Configure' tab of the Configure BPSim window, create or edit this Resource parameter • Quantity - Value: 0; Calendar, with the 'Calendar' field set to '----None----' (this default value is necessary) • Quantity - Value: 2; Calendar, with the 'Calendar' field set to 'Weekdays' • Quantity - Value: 1; Calendar, with the 'Calendar' field set to 'Weekends'
Resource Selection (allocation)	On the diagram, click on the *Service Customer* Activity and, on the 'Configure' tab of the Configure BPSim window, with the Calendar field set to '----None----' check that the 'Values' field for the Resource parameter 'Selection' is set to: **bpsim::getResource('Support',1)** as an expression This expression is loaded from your BPMN model by default. You can do some advanced configurations for resource selection for a task.

Run Simulation

Weekdays

1. Click on the 'Calendar' field and select 'Weekday'.
2. Click on the 'Execute' tab and on the toolbar icon.

A file with the name *Calendar Based Support Process Simulation - Result* is generated. This report file contains the result for a weekday simulation, which is displayed on the 'Review' tab of the Configure BPSim window, on the 'Standard Results Report' tab.

Weekends

1. Click on the 'Calendar' field and select 'Weekend'.
2. Click on the 'Execute' tab and on the toolbar icon.

The *Calendar Based Support Process Simulation - Result* file is updated to show the result for a weekend simulation and displayed on the 'Review' tab of the Configure BPSim window, on the 'Standard Results Report' tab.

In each case, check the match between the result file and our analysis with pen and paper.

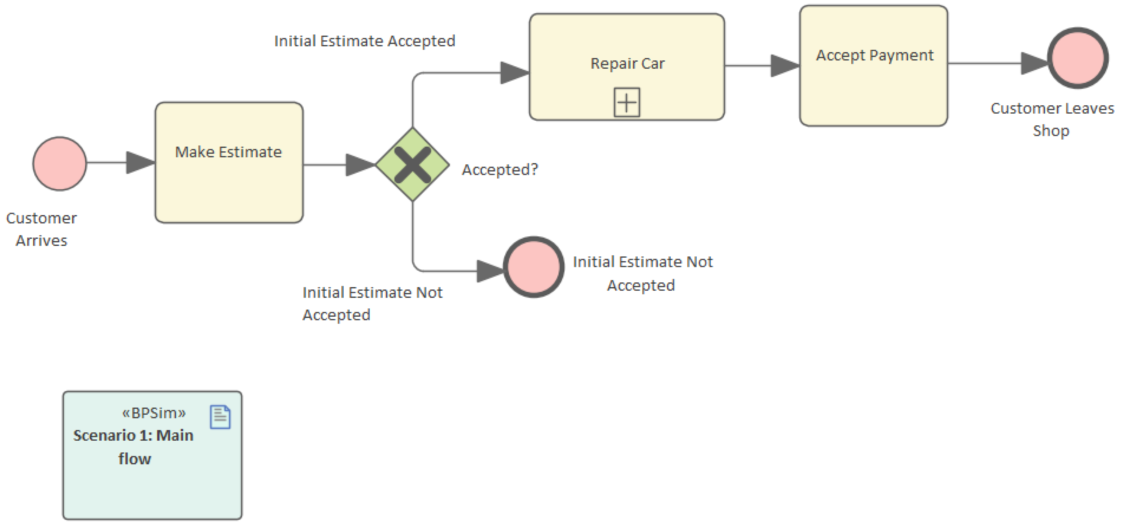
Car Repair Process

This example simulates the process flow of a Car Repair shop. The BPSim configuration:

- Uses a property parameter initialized by distribution to generate a random number of issues for each customer
- Applies probability to simulate:
  - Acceptance of the initial estimate or not
  - Whether new issues will be found during repair
- Increments or decrements the property parameter's value in each task
- Uses the property parameter's value on conditions of Sequences outgoing from Gateways
- Simulates the customer arrivals for a given start and duration

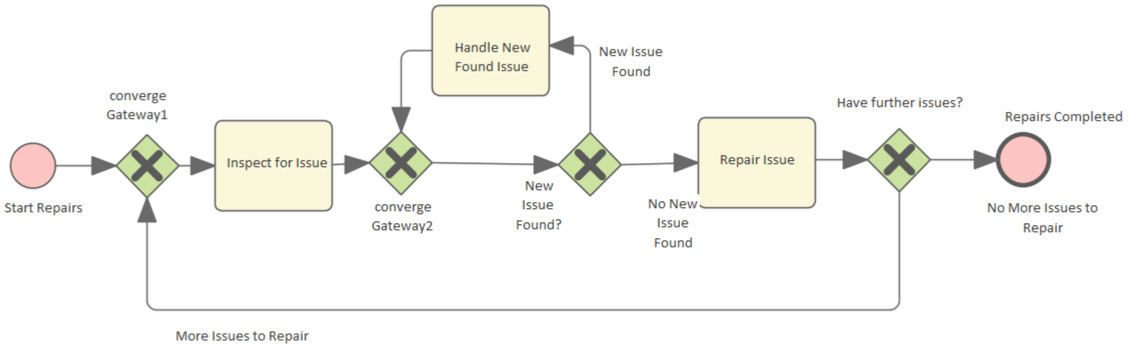
Create BPMN Model

Create the main process



1. Create a Start Event *Customer Arrives.*
2. Add a Sequence Flow to a target abstract task Activity *Make Estimate.*
3. Add a Sequence Flow to a target Exclusive Gateway *Accepted?.*
4. Add Sequence Flows to:
   - A target End Event *Initial Estimate Not Accepted*
   - A target subProcess *Repair Car*
5. From *Repair Car*, add a Sequence Flow to a target abstract task Activity *Accept Payment.*
6. Add a Sequence Flow to a target End Event *Customer Leaves Shop.*

Create the sub process Repair Car



1. Create a Start Event *Start Repairs.*
2. Add a Sequence Flow to a target Exclusive Gateway *converge Gateway1*.
3. Add a Sequence Flow to an abstract task Activity *Inspect for Issue.*
4. Add a Sequence Flow to an Exclusive Gateway *converge Gateway2.*
5. Add a Sequence Flow to an Exclusive Gateway *New Issue Found?*.
6. Add Sequence Flows to:
   - A target abstract task Activity *Handle New Found Issue*, then add a Sequence Flow back to *converge Gateway2*
   - A target abstract task *Repair Issue*, then add a Sequence Flow to a target Exclusive Gateway *Have further issues?*
7. From the Gateway *Have further issues?* add Sequence Flows to:
   - The target End Event *Repairs Completed*
   - *converge Gateway1*

Configure BPSim

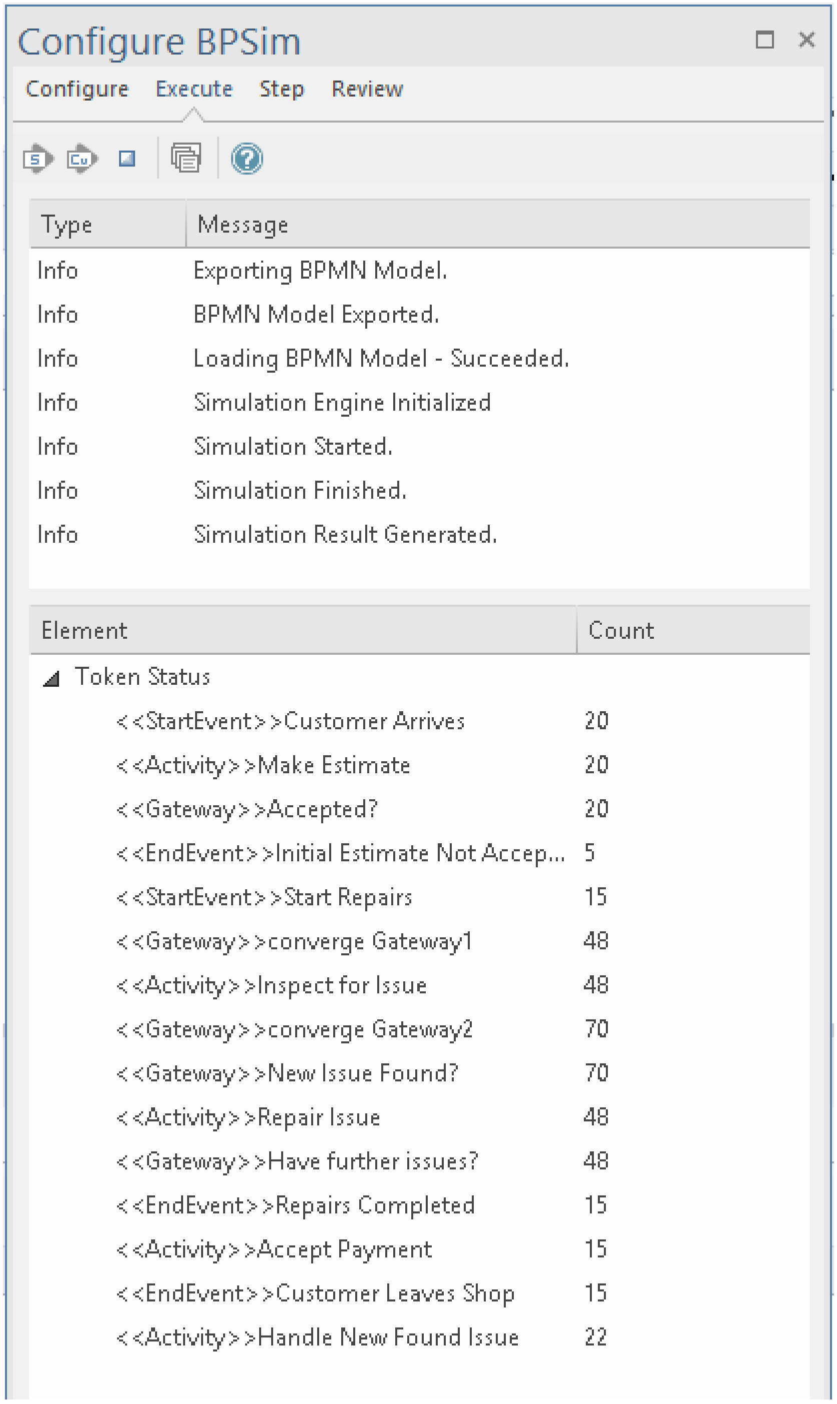
Task	Description
Artifact and Package	1. Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager'). 2. Create a Business Process Simulation Artifact named *Scenario 1: Main flow.* 3. Select the Package containing the corresponding BPMN 2.0 model.
Start and Duration	We will simulate the processes in a car repair shop whose opening hours are from 9:00 am to 5:00 pm, which is a period of 8 hours. We also suppose that a customer walking in after 4:50 pm will not be served on that day. Therefore the simulation Start time is 9:00 am and the duration is 7 hours and 50 minutes. On the 'Car Repair' diagram, click on the Business Process Simulation Artifact named *Scenario 1: Main Flow* and, on the Configure BPSim window, update these ScenarioParameters: • Start - overtype the 'Values' field with any date (in the format dd/mm/yyyy) and change the time section to '9:00 AM' • Duration - click on the … button in the 'Values' field and set it to a Constant Duration of '0 07:50:00'
Customer Arrives	We will simulate a customer arriving every 24 minutes. The first customer arrives at 9:00 AM and the last arrives at 4:36 PM (the customer

	arriving at 5:00 PM will not be served today because that is constrained by the 'Duration' setting). With a pen and paper, we can calculate that there are 20 customers served (9:00 AM to 4:36 PM = 456 minutes; number of customers is 456/24 + 1 = 19 + 1 = 20). We will verify this with the simulation result later. On the 'Car Repair' diagram, click on the Start Event element *Customer Arrives*, and in the Configure BPSim window: 1. Click on the *New Parameter* drop-down arrow, and select 'Control'. 2. Click on the 'Parameter' drop-down arrow and select 'InterTriggerTimer'. 3. In the 'Values' field click on the ... button and set a Constant Numeric value of '24 minutes'. Click on the OK button and the Save toolbar icon.
Property Parameters	We suppose each customer's car might initially have a different number of issues. This could be reflected using a random number generator. BPSim provides a number of distributions to suit your needs. In this example, we use a Truncated Normal distribution to initialize the property *noOfIssues.* Tasks *Repair Issue* and *Handle New Found Issue* will decrement and increment the value of *noOfIssues* respectively. 1. On the 'Car Repair' diagram, click on the Start Event *Customer Arrives*. 2. On the 'Configure' tab of the Configure BPSim window, click on the *New Parameter* drop-down arrow and create a Property parameter called *noOfIssues.* 3. In the 'Values' field click on the ... button; the 'Configure 'noOfIssues' for 'CustomerArrives'' dialog displays. 4. Click on the 'Distribution' tab and select 'TruncatedNormal'; in the fields: - 'Mean', type '2' - 'StandardDeviation', type '1' - 'Min', type '1' - 'Max', type '1000' **Important Note:** Distributions such as 'TruncatedNormal', return a floating point value, but the property is used as an integer. Setting the property's type is important, especially in condition expressions when testing with equality. For example, the condition expression *getProperty('noOfIssues') = 0* will almost never be satisfied because *noOfIssues* was initialized by a floating point distribution. **Tip: How to customize the type for a property** After you create the property and set a value, click on the 💾 icon on the toolbar, and then click on the ◆ icon to display the 'Edit Property Parameters' dialog. In the 'Type' field for the property, click on the drop-down arrow and select the value 'int' instead of the default 'double'. 1. On the 'Repair Car' diagram, click on the Activity *Repair Issue.* 2. On the 'Configure' tab of the Configure BPSim window, click on the *New Parameter* drop-down arrow and create a Property parameter called *noOfIssues.* 3. In the 'Values' field click on the ... button. The 'Configure 'noOfIssues' for 'Repair Issue'' dialog displays. 4. Click on the 'Expression' tab and, in the 'Expression' field, type *{noOfIssues} -1*; click on the OK button. 5. On the 'Repair Car' diagram, click on the Activity *Handle New Found Issue*. 6. On the 'Configure' tab of the Configure BPSim window, click on the *New*

	Parameter drop-down arrow and create a Property parameter called *noOfIssues.* 7. In the 'Values' field click on the ... button. The 'Configure 'noOfIssues' for 'Handle New Found Issue'' dialog displays. 8. Click on the 'Expression' tab and, in the 'Expression' field, type *{noOfIssues} +1*; click on the OK button.
Probability on Sequence Flows	We estimate that one in three of the customers will not accept the initial estimate for repairs and the remaining two will accept it. We also estimate that for one in four of the repairs new issues will be found, and for the remaining three repairs no new issues will be found. On the 'Car Repair' diagram, refer to the Gateway element *Accepted?*. Click on the: • *Initial Estimate Accepted* Sequence Flow and in the Configure BPSim window click on the *New Parameter* drop-down arrow, and create a Control parameter called 'Probability'; in the 'Values' field type '0.67' • *Initial Estimate Not Accepted* Sequence Flow, and in the Configure BPSim window click on the *New Parameter* drop-down arrow, and create a Control parameter called 'Probability'; in the 'Values' field type '0.33' On the 'Car Repair' diagram, refer to the Gateway element *New Issue Found?*. Click on the: • *No More Issues to Repair* Sequence Flow and in the Configure BPSim window click on the *New Parameter* drop-down arrow, and create a Control parameter called 'Probability'; in the 'Values' field type '0.75' • *More Issues to Repair* Sequence Flow and in the Configure BPSim window click on the *New Parameter* drop-down arrow, and create a Control parameter called 'Probability'; in the 'Values' field type '0.25'
Condition on Sequence flows	We use an expression to return a Boolean value as a Sequence Flow's condition, which plays a key role in the flow's logic. On the 'Repair Car' diagram, refer to the *Have Further Issues?* Gateway element. Click on the: • *More Issues to Repair* Sequence Flow and in the Configure BPSim window click on the *New Parameter* drop-down arrow, and create a Control parameter called 'Condition'; in the 'Values' field click on the ... button, click on the 'Expression' tab and type **{noOfIssues} != 0** in the 'Expression' field • *No More Issues to Repair* Sequence Flow and in the Configure BPSim window click on the *New Parameter* drop-down arrow, and create a Control parameter called 'Condition'; in the 'Values' field click on the ... button, click on the 'Expression' tab and type **{noOfIssues} = 0** in the 'Expression' field Note: All the outgoing transitions from a Gateway should include 100% of the logic; for example, you would not enter **{noOfIssues} > 10** and **{noOfIssues} < 5** as condition expressions, because values in the range **[5, 10]** will not be handled by any outgoing Sequence Flows.

Run Simulation

1. On the Configure BPSim window click on the 'Execute' tab and on the icon in the toolbar.
2. When the simulation is complete, the Execute tab provides results similar to these:



Type	Message
Info	Exporting BPMN Model.
Info	BPMN Model Exported.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Count
Token Status	
<<StartEvent>>Customer Arrives	20
<<Activity>>Make Estimate	20
<<Gateway>>Accepted?	20
<<EndEvent>>Initial Estimate Not Accep...	5
<<StartEvent>>Start Repairs	15
<<Gateway>>converge Gateway1	48
<<Activity>>Inspect for Issue	48
<<Gateway>>converge Gateway2	70
<<Gateway>>New Issue Found?	70
<<Activity>>Repair Issue	48
<<Gateway>>Have further issues?	48
<<EndEvent>>Repairs Completed	15
<<Activity>>Accept Payment	15
<<EndEvent>>Customer Leaves Shop	15
<<Activity>>Handle New Found Issue	22

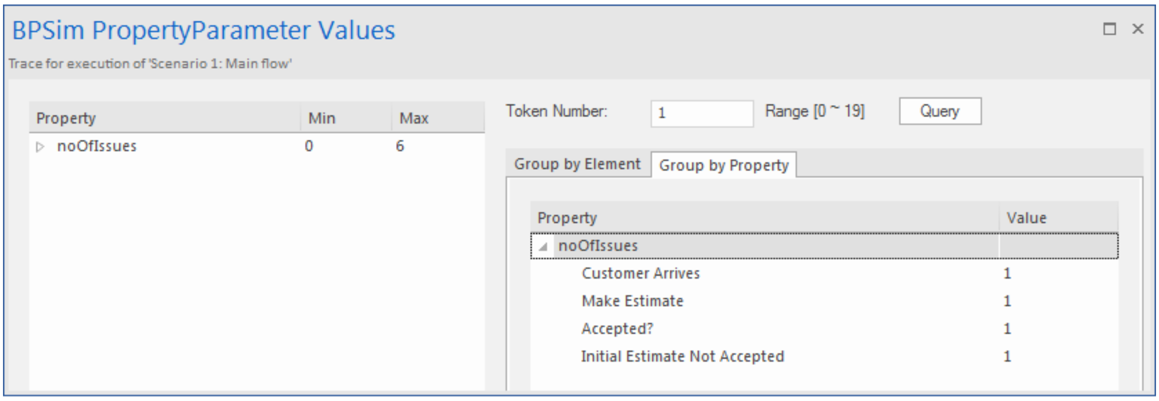
Token Analysis

- 20 customers arrived, matching the number we calculated manually (see *Customer Arrives* in the *Configure BPSim* table)
- 8 customers out of the 20 did not accept the initial estimate, whilst 12 of the 20 accepted and had their car repaired; these figures roughly match the 1/3 and 2/3 probabilities
- 64 tokens passed the Gateway *New Issue Found?*, of which 19 had new issues and 45 did not; these figures roughly match the 1/4 and 3/4 probabilities

Analysis on individual customers

Click on the button on the toolbar to open the 'BPSim PropertyParameter Values' dialog. As there are 20 customers (tokens), you can input a value between 0 and 19 in the 'Token Number' field and click on the Query button to do some analysis:

- This customer did not accept the initial estimate, as shown on the 'Group by Property' tab:



BPSim PropertyParameter Values

Trace for execution of 'Scenario 1: Main flow'

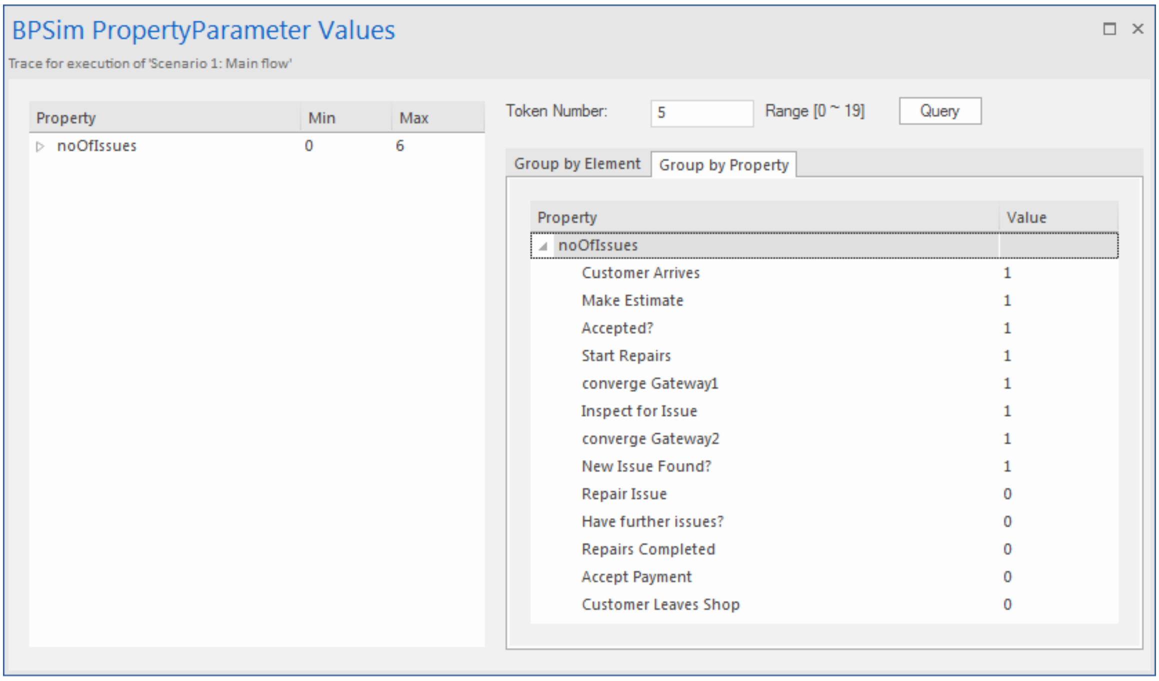
Property	Min	Max
noOfIssues	0	6

Token Number: 1 Range [0 ~ 19] Query

Group by Element | Group by Property

Property	Value
noOfIssues	
Customer Arrives	1
Make Estimate	1
Accepted?	1
Initial Estimate Not Accepted	1

- This customer's car had only one issue, which was fixed:



BPSim PropertyParameter Values

Trace for execution of 'Scenario 1: Main flow'

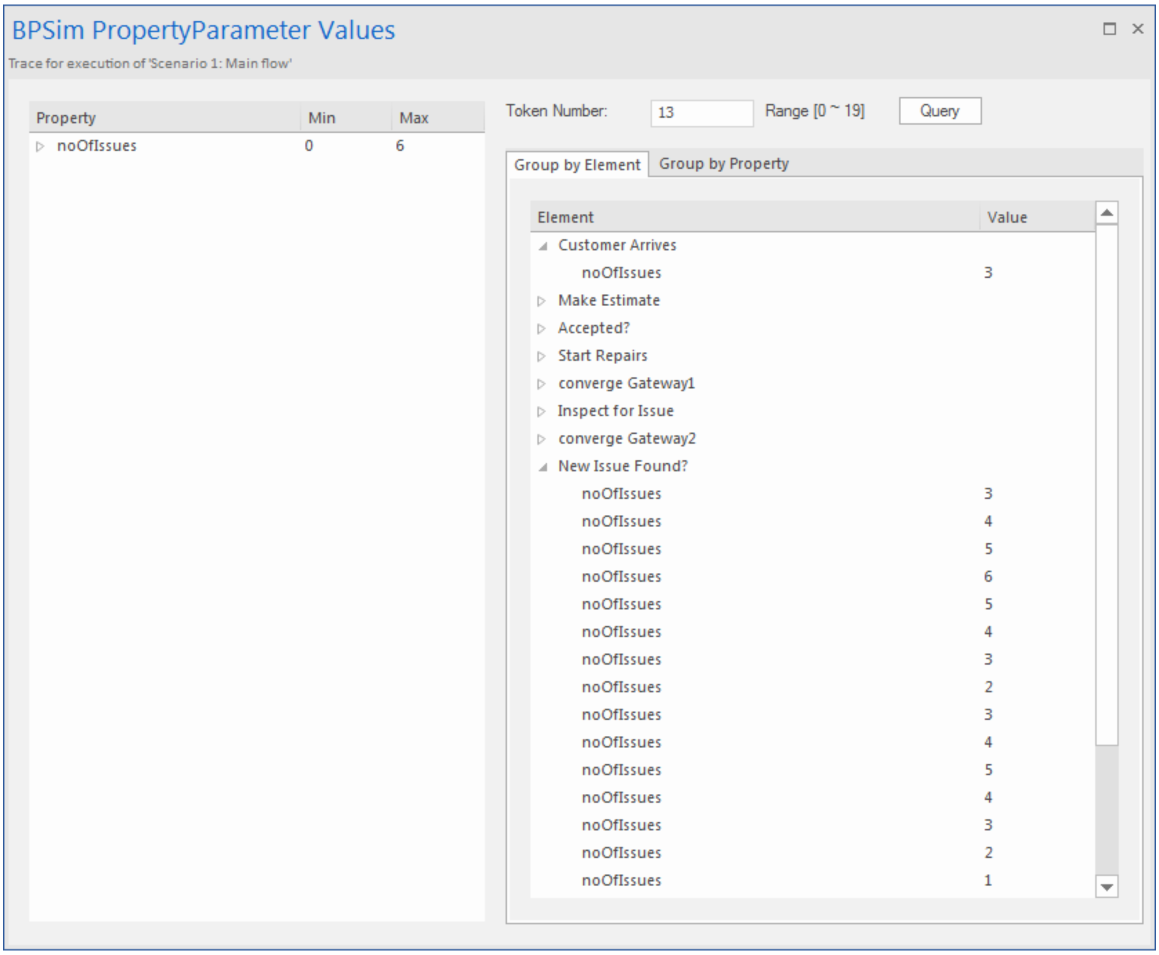
Property	Min	Max
noOfIssues	0	6

Token Number: 5 Range [0 ~ 19] Query

Group by Element | Group by Property

Property	Value
noOfIssues	
Customer Arrives	1
Make Estimate	1
Accepted?	1
Start Repairs	1
converge Gateway1	1
Inspect for Issue	1
converge Gateway2	1
New Issue Found?	1
Repair Issue	0
Have further issues?	0
Repairs Completed	0
Accept Payment	0
Customer Leaves Shop	0

- This customer's car had three known issues and three other issues were found during repair, so in total six issues got fixed (perhaps this is a really old car); switching to the 'Group by Element' tab:



BPSim PropertyParameter Values

Trace for execution of 'Scenario 1: Main flow'

Property	Min	Max
▷ noOfIssues	0	6

Token Number: 13 Range [0 ~ 19]

Element	Value
◢ Customer Arrives	
noOfIssues	3
▷ Make Estimate	
▷ Accepted?	
▷ Start Repairs	
▷ converge Gateway1	
▷ Inspect for Issue	
▷ converge Gateway2	
◢ New Issue Found?	
noOfIssues	3
noOfIssues	4
noOfIssues	5
noOfIssues	6
noOfIssues	5
noOfIssues	4
noOfIssues	3
noOfIssues	2
noOfIssues	3
noOfIssues	4
noOfIssues	5
noOfIssues	4
noOfIssues	3
noOfIssues	2
noOfIssues	1

BPMN2.0 Event Examples

An Event is something that happens during the course of a Process. Events affect the flow of the Process, usually having a cause or an impact, and in general requiring or allowing for a reaction. For example, the start of an Activity, the end of an Activity, the change of state of a document, or the arrival of a Message could all be considered to be Events.

Events allow for the description of 'event-driven' Processes. In these Processes, there are three main types of Event:

- Start Events, which indicate where a Process will start
- End Events, which indicate where the path of a Process will end
- Intermediate Events, which indicate where something happens between the start and end of a Process

Within these three types, Events can be one of two subtypes:

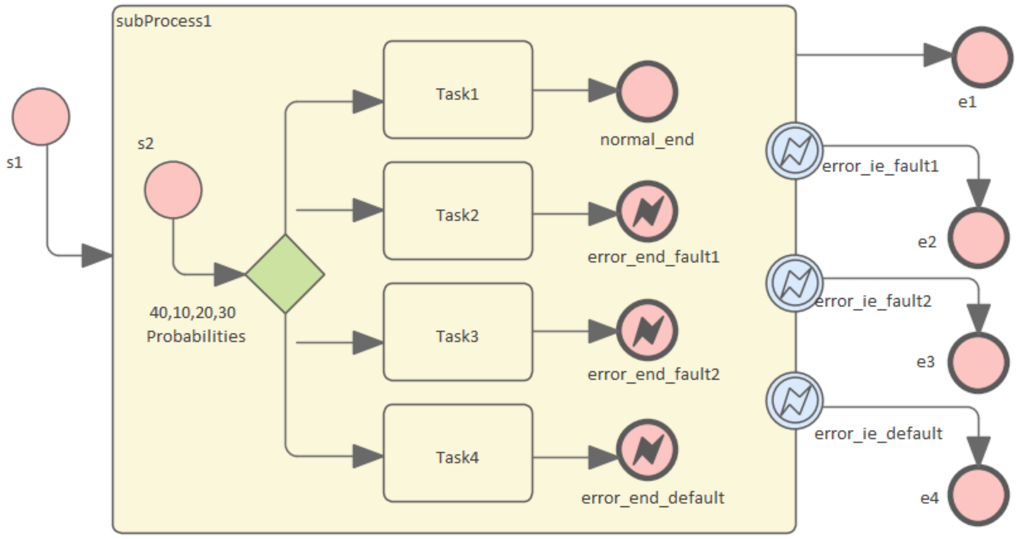
- Events that catch a trigger - all Start Events and some Intermediate Events are catching Events
- Events that throw a Result - all End Events and some Intermediate Events are throwing Events that could eventually be caught by another Event

In this section, we provide examples illustrating many of the commonly-used BPMN 2.0 events. In each example, we provide step-by-step BPMN modeling and BPSim configuration instructions, and a thorough analysis of the simulation result. All the examples are available in the EAExample model.

Error Event

When an Intermediate Error Event connects to the boundary of an Activity, it becomes part of an exception flow. The event is triggered when a token causes a fault name to be raised in the normal flow, going through to an Error End Event.

Create BPMN Model



Create the main process

- Create a Start Event *s1*
- Add a Sequence Flow to a target Activity element *subProcess1;* enlarge the Activity and right-click, selecting the 'Is Expanded' option, then open the 'Properties' dialog and set 'Type' to 'subprocess'
- Add a Sequence Flow to a target End Event element *e1* ('Type' set to 'None')
- Create three Boundary Intermediate Events, dragging the elements from the Toolbox and dropping them on *subProcess1*; from the instant menus select 'Edge-Mounted' and 'Error':
  - *error_ie_fault1*; add a Sequence Flow to a target EndEvent element *e2* ('Type' set to 'None')
  - *error_ie_fault2*; add a Sequence Flow to a target EndEvent element *e3* ('Type' set to 'None')
  - *error_ie_default*; add a Sequence Flow to a target EndEvent elemenr *e4* ('Type' set to 'None')

Create the subprocess

Within the *subProcess1* Activity:

- Create a Start Event *s2*, 'Standalone' and set 'Type' to 'None'
- Create a Sequence Flow to a target Gateway element set to 'Exclusive' and with the name '*40,10,20,30 Probabilities'*
- Create Sequence Flows to four target Activiy elements of Type 'abstractTask' called:
  - *Task1*, and add a Sequence Flow to a target EndEvent called *normal_end*, 'Type' set to 'None'
  - *Task2*, and add a Sequence Flow to a target EndEvent called *error_end_fault1,* 'Type' set to 'Error'
  - *Task3*, and add a Sequence Flow to a target EndEvent called *error_end_fault2,* 'Type' set to 'Error'
  - *Task4*, and add a Sequence Flow to a target EndEvent called *error_end_default,* 'Type' set to 'Error'

Create BPMN2.0::Error elements

Create the Error elements *Fault1* and *Fault2*, which will be used as error code by Events.

- Double-click on the *error_end_fault1* element and, in the 'Properties' dialog 'BPMN2.0' tab, locate the 'errorRef' tag
- In the 'Value' field, click on the ... button and browse to the Package containing this model
- Click on the Add New button and, in the 'Name' field, type the name *Fault1*, then click on the Save button
- Again click on the Add New button and, in the 'Name' field, type the name *Fault2*, then click on the Save button
- Click on the OK button, and again on the next OK button

Set up Events for Error Codes

- Double-click on the *error_end_fault1* element and, in the 'Properties' dialog 'BPMN2.0' tab, locate the 'errorRef' tag
- In the 'Value' field, click on the ... button and browse to the Package containing this model
- Click on *Fault1*, then on the OK button, and again on the OK button.

Do the same for these elements:

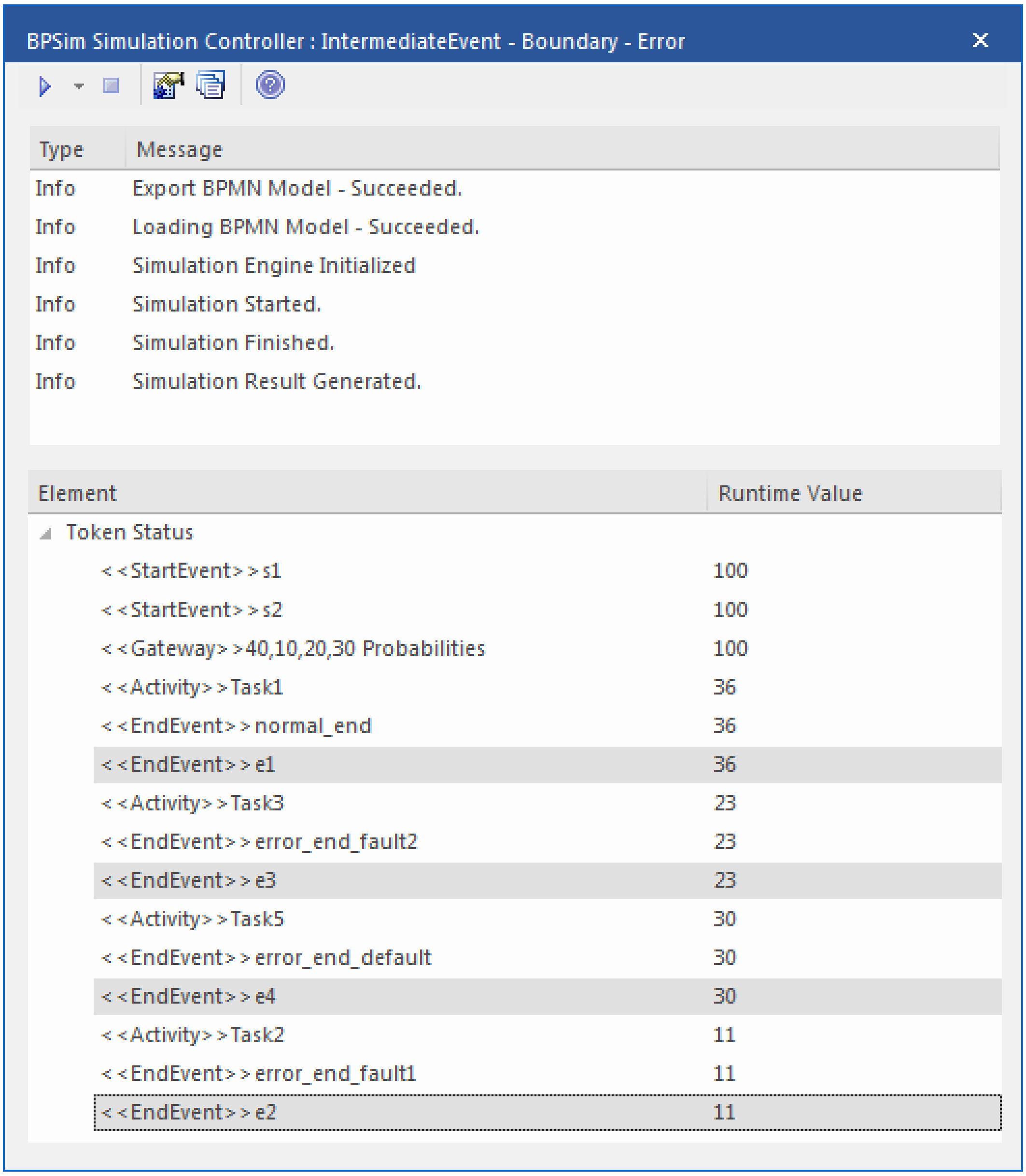
- *error_end_fault2*, clicking on *Fault2*
- *error_ie_fault1*, clicking on *Fault1*
- *error_ie_fault2*, clicking on *Fault2*

Configure BPSim

Object	Action
Artifact & Package	• Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager') • Create an Artifact named 'IntermediateEvent - Boundary - Error' (in the 'Select/Create Artifact' field, click on the ... button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the Configure BPSim window.
s1	• From the tree on the left of the Configure BPSim window, expand 'StartEvent' and click on 's1' • On the 'Control' tab, in the 'New Parameter...' field, click on the drop-down arrow and select 'TriggerCount' • In the 'Value' field, type '100'
Probability	From the tree on the left of the Configure BPSim window, expand 'Gateway | 40,10,20,30 Probabilities'. *Tips: You can also float the Configure BPSim window, then click on the element or connectors on the BPMN diagram; the element in the Configure BPSim window will be automatically selected.* For each of the *Taskn* elements, in the 'Control' tab click on the 'New Parameter' drop-down arrow and select 'Probability', then type the corresponding value in the 'Value' field: • For Task1 type '0.4' • For Task2 type '0.1' • For Task3 type '0.2' • For Task4 type '0.3'

Run Simulation

- On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Controller' dialog
- Click on the Run button and select 'Standard Simulation'
- The results of the simulation resemble this:



BPSim Simulation Controller : IntermediateEvent - Boundary - Error

Type	Message
Info	Export BPMN Model - Succeeded.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Runtime Value
Token Status	
<<StartEvent>>s1	100
<<StartEvent>>s2	100
<<Gateway>>40,10,20,30 Probabilities	100
<<Activity>>Task1	36
<<EndEvent>>normal_end	36
<<EndEvent>>e1	36
<<Activity>>Task3	23
<<EndEvent>>error_end_fault2	23
<<EndEvent>>e3	23
<<Activity>>Task5	30
<<EndEvent>>error_end_default	30
<<EndEvent>>e4	30
<<Activity>>Task2	11
<<EndEvent>>error_end_fault1	11
<<EndEvent>>e2	11

Analysis:

The Probability set on the Sequence Flows outgoing from *40,10,20,30 Probabilities* are 0.4, 0.1, 0.2 and 0.3 respectively.

- 36 out of 100 passes finished at *normal_end*, which flowed to *e1*
- 11 out of 100 passes finished at *error_end_fault1*, which triggered *error_ie_fault1* by the ErrorRef *Fault1,* and the exception flowed to *e2*
- 23 out of 100 passes finished at *error_end_fault2*, which triggered *error_ie_fault2* by ErrorRef *Fault2,* and the

exception flowed to *e3*

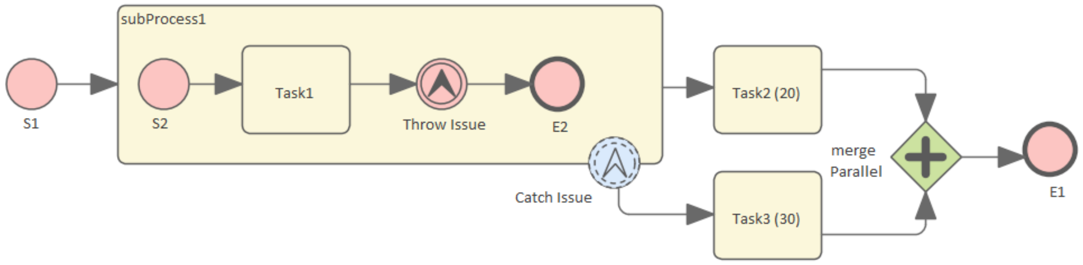
- 30 out of 100 passes finished at *error_end_default*, which triggered *error_ie_default* because they did not set ErrorRef and the exception flowed to *e4*

The numbers 36, 11, 23 and 30 add up to 100, which was set as the TriggerCount in *s1,* so they match the 100% probability

Escalation Event

In BPMN, Escalation is the non-interrupting counterpart of Error, with similar throw-catch behavior. Unlike Error, however, the normal flow and exception flow exits from the Activity are parallel paths, not alternative.

Create BPMN Model



Create the main process

- Create a Start Event *S1*
- Add a Sequence Flow to a target Activity *subProcess1;* enlarge the Activity and right-click, selecting the 'Is Expanded' option, then open the 'Properties' dialog and set 'Type' to 'subprocess'
- Add a Sequence Flow to a target abstractTask Activity element *Task2 (20)* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')
- Add a Sequence Flow to a target parallel Gateway element *merge Parallel* (open the 'Properties' dialog and set the 'Type' field to 'parallel')
- Add a Sequence Flow to a target End Event *E1*
- On *subProcess1*, add a boundary non-interrupting Escalation Event *Catch Issue* (drag the 'Intermediate Event' icon onto *subProcess1*, and from the instant menus select 'Edge Mounted' and 'Escalation'; double-click on the element to display the 'Properties' dialog and add the name, then in the 'Type' field select 'Boundary Non-Interrupting > Escallation')
- Add a Sequence Flow to a target abstractTask Activity element *Task3 (30)* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')
- Add a Sequence Flow to the target element *merge Parallel*

Create the sub process

- Within (or under) *subProcess1*, create a Start Event *S2*
- Add a Sequence Flow to a target abstractTask Activity element *Task1* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')
- Add a Sequence Flow to a target Throwing Escalation Intermediate Event *Throw Issue* (open the 'Properties' dialog and in the 'Type' field select 'Throwing > Escalation')
- Add a Sequence Flow to a target End Event *E2*

Create BPMN2.0::Escalation elements

From the Diagram Toolbox, expand the 'BPMN 2.0 Types' page, drag the 'Escalation' icon onto the diagram, and give the element the name *Escalation1*; this will be used as the escalation code by the Events.

Set up Events for Escalation Codes:

- Double-click on *Throw Issue* and in the 'Value' field for the escalationRef tag click on the ... icon and locate and select *Escalation1*
- Double-click on *Catch Issue* and, again, in the 'Value' field for the escalationRef tag click on the ... icon and locate and select *Escalation1*

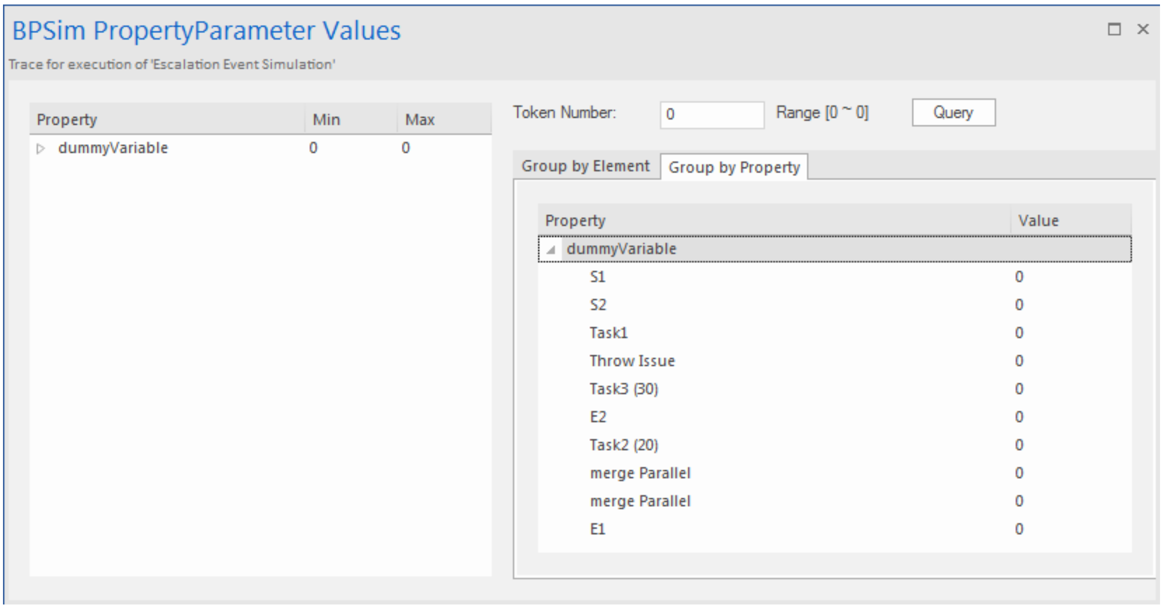
(The exception flow exits from the Activity are parallel.)

Configure BPSim

Task	Action
Artifact & Package	• Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager') • Create an Artifact named 'Escalation Event Simulation' (in the 'Select/Create Artifact' field, click on the ... button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the Configure BPSim window.
Trigger Count of Start Event	• From the tree on the left of the Configure BPSim window, expand 'StartEvent' and click on *S1* • On the 'Control' tab, in the 'New Parameter...' field, click on the drop-down arrow and select 'TriggerCount' • In the 'Value' field, type '1'
ProcessingTime	• In the left-hand tree expand 'Activity' and click on *Task2 (20)*; in the 'Value' field for 'Processing Time' type '20' and in the 'Unit' field type 's' (for 20 seconds) • Click on *Task3 (30);* in the same way, set 'ProcessingTime' to 30 seconds
dummyVariable for Trace	In order to show the exact trace of a given token, you must set a dummy variable on *S1*. • In the left-hand hierarchy click on *S1*, and on the 'Properties' tab overtype the *New Property* text with the name of a variable (such as 'dummyVariable') • In the 'Value' field, click on the ... button and, on the '<<StartEvent>>S1 : <variable name>' dialog click on 'Numeric' and type a 'Constant Numeric' value of '0'; click on the OK button

Run Simulation

- On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog
- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'
- After simulation, click on the button on the tool bar to display the 'BPSim PropertyParameter Values' dialog
- Click on the Query button and on the 'Group by Property' tab, and expand 'dummyVariable' (or the name you assigned to the variable)



BPSim PropertyParameter Values
Trace for execution of 'Escalation Event Simulation'
Token Number:
Range [0 ~ 0]
Query
Group by Element
Group by Property

Token Number: 0

Property	Min	Max
dummyVariable	0	0

Property	Value
dummyVariable	
S1	0
S2	0
Task1	0
Throw Issue	0
Task3 (30)	0
E2	0
Task2 (20)	0
merge Parallel	0
merge Parallel	0
E1	0

Analysis:

Unlike *Error*, the normal flow and exception flow exits from *subProcess1* are not alternative paths but parallel. This feature can be easily discovered from the trace:

- *E2* and *Task2 (20)* still get traversed after*Task3 (30)* started
- *E1* was reached after *mergeParallel* was traversed twice

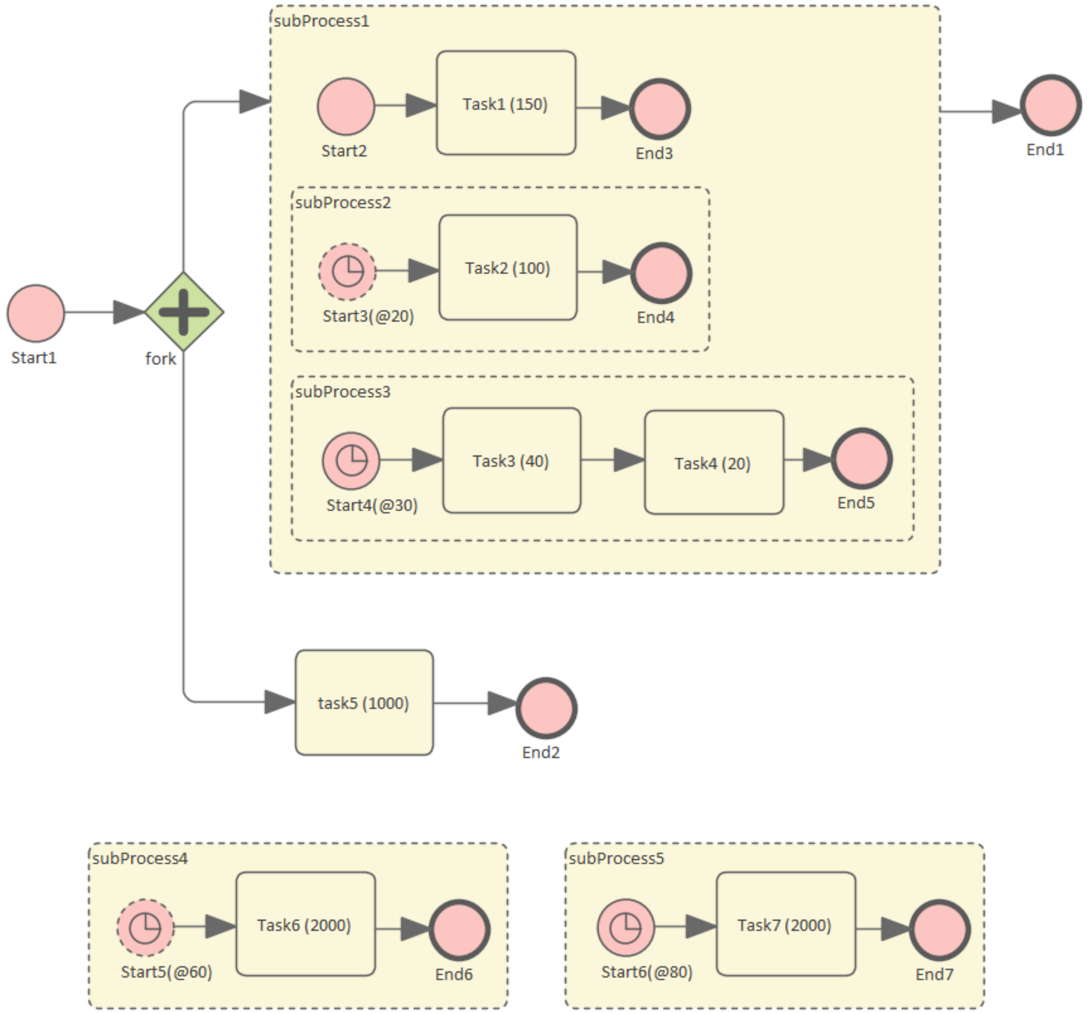
Event Sub-Process

Event Sub-Processes enable your system to handle an Event within the context of a given Sub-Process or Process. An Event Sub-Process always begins with a Start Event; it is not instantiated by normal control flow, but only when the associated Start Event is triggered. Event Sub-Processes are self-contained and MUST not be connected to the rest of the Sequence Flows in the Sub-Processes.

- If the isInterrupting attribute of its Start Event is set, an Event Sub-Process cancels execution of the enclosing Sub-Process
- If the isInterrupting attribute is not set, execution of the enclosing Sub-Process continues in parallel to the Event Sub-Process

In this example, we demonstrate how Interrupting and Non-Interrupting Event Sub-Processes affect the life line of the enclosing Sub-Process and Process.

Create BPMN Model



Model the main process

- Create a StartEvent *Start1*
- Add a Sequence Flow to a target Parallel Gateway element *fork*
- Add a Sequence Flow to
  - a Sub-Process *subProcess1*, and from that add a *S*equence Flow to a target End Event element

End1
  - an abstract Task *Task5*, and from that add a Sequence Flow to a target End Event element *End2*

Tips on how to model an Event Sub-Process

- Drag an Activity from the 'BPMN2.0 - Business Process' toolbox onto the diagram
- Double-click on the Activity to display the 'Properties' dialog and, in the 'Type' field, select 'subProcess'; set 'triggeredByEvent' to 'true' and click on the OK button
- Right-click on the element and select the 'Is Expanded' option; this will display the element name on the top left corner

Model the Event Sub-Processes for the main process

- Create an Event Sub-Process *subProcess4*
  - Create a Timer Start Event *Start5(@60)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Non-Interrupting > Timer'; click on the OK button
  - Add a Sequence Flow to a target abstract task Activity *Task6(2000)*
  - Add a Sequence Flow to a target End Event element *End6*
- Create an Event Sub-Process *subProcess5*
  - Create a Timer Start Event *Start6(@80)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Interrupting > Timer'; click on the OK button
  - Add a Sequence Flow to a target abstract task Activity *Task7(2000)*
  - Add a Sequence Flow to a target End Event element *End7*

Model the Sub-Process subProcess1 and the enclosed Event Sub-Processes

- Create a StartEvent *Start2*
  - Add a *S*equence Flow to a target abstract task Activity *Task1(150)*
  - Add a Sequence Flow to a target End Event *End3*
- Create an Event Sub-Process *subProcess2*
  - Create a Timer Start Event *Start3(@20)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Non-Interrupting > Timer'
  - Add a Sequence Flow to a target abstract task Activity *Task2(100)*
  - Add a Sequence Flow to a target End Event element *End4*
- Create an Event Sub-Process *subProcess3*
  - Create a Timer Start Event *Start4(@30)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Interrupting > Timer'
  - Add a Sequence Flow to a target abstract task Activity *Task3(40)*
  - Add a Sequence Flow to a target abstract task Activity *Task4(20)*
  - Add a Sequence Flow to a target End Event element *End5*

Configure BPSim

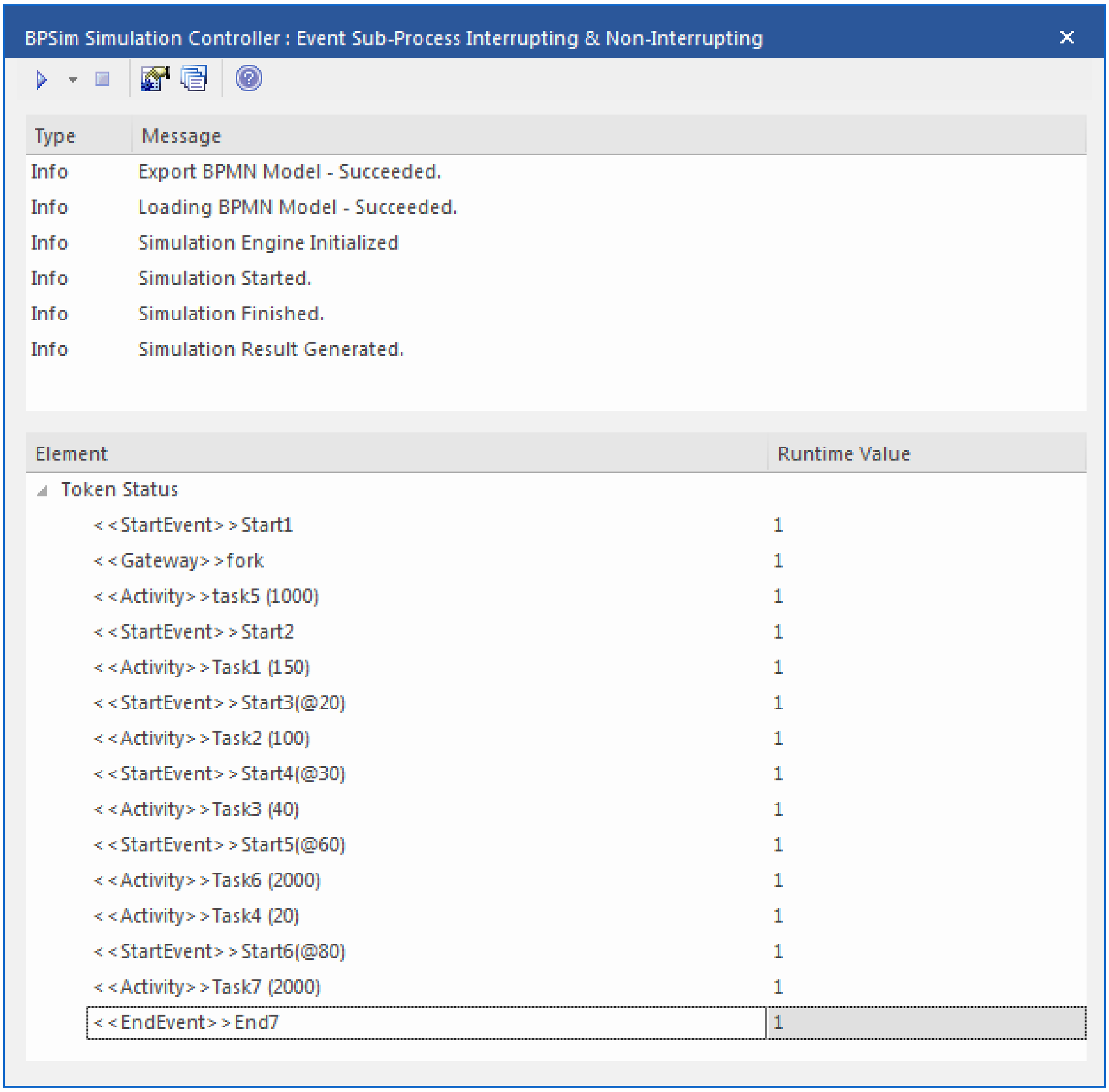
Using this table, we create the Artifact in the configuration Package and configure the parameter values of each element.

Task	Action
Create Artifact	• Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager') • Create an Artifact named 'Event Sub Process Interrupting and Non-Interrupting' (in the 'Select/Create Artifact' field, click on the ... button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button)

	Then all the BPMN elements will be loaded in to the Configure BPSim window.
InterTriggerTimer for Start Events in Event Sub-Process	From the tree on the left of the 'Configure BPSim' dialog, expand 'StartEvent'. For each of the elements listed here, on the 'Control' tab click on the drop-down arrow in the 'New Parameter...' field and select the parameter 'InterTriggerTimer'. Click on the ... button in the 'Value' field to open the 'Parameter' dialog and select 'Constant > Numeric', then type in the value and select 'seconds'. • Start3(@20): 20 seconds • Start4(@30): 30 seconds • Start5(@60): 60 seconds • Start6(@80): 80 seconds
ProcessingTime for Tasks	From the tree on the left of the Configure BPSim window, expand 'Activity'. For each of the elements listed here, on the 'Time' tab click on the drop-down arrow in the 'New Parameter...' field and select the parameter 'ProcessingTime'. Click on the ... button in the 'Value' field to open the 'Parameter' dialog and select 'Constant > Numeric', then type in the value and select 'seconds'. • Task1 (150): 150 seconds • Task2 (100): 100 seconds • Task3 (40): 40 seconds • Task4 (20): 20 seconds • Task5 (1000): 1000 seconds • Task6 (2000): 2000 seconds • Task7 (2000): 2000 seconds

Run Simulation

- On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog
- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'



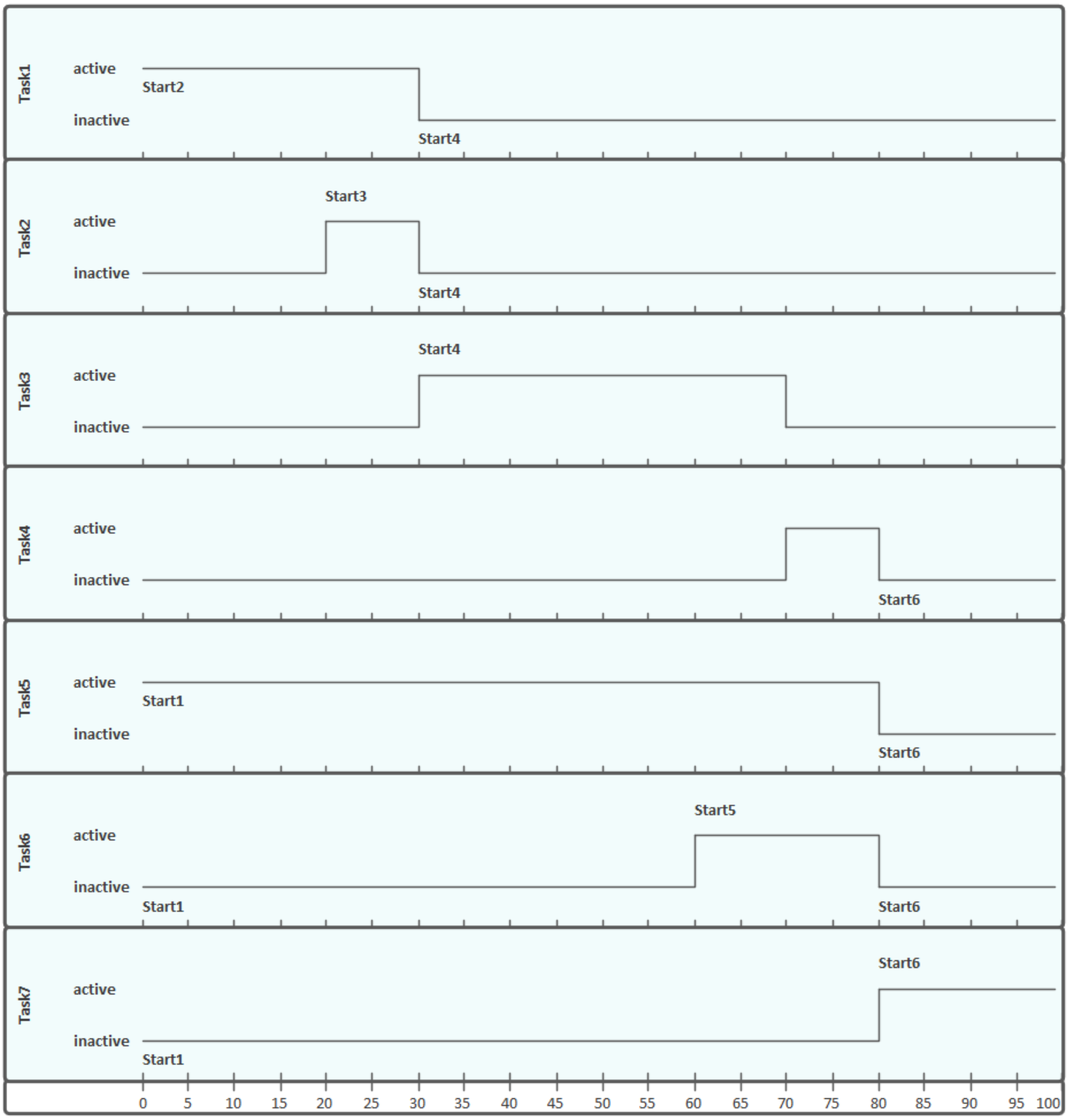
BPSim Simulation Controller : Event Sub-Process Interrupting & Non-Interrupting

Type	Message
Info	Export BPMN Model - Succeeded.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Runtime Value
Token Status	
<<StartEvent>>Start1	1
<<Gateway>>fork	1
<<Activity>>task5 (1000)	1
<<StartEvent>>Start2	1
<<Activity>>Task1 (150)	1
<<StartEvent>>Start3(@20)	1
<<Activity>>Task2 (100)	1
<<StartEvent>>Start4(@30)	1
<<Activity>>Task3 (40)	1
<<StartEvent>>Start5(@60)	1
<<Activity>>Task6 (2000)	1
<<Activity>>Task4 (20)	1
<<StartEvent>>Start6(@80)	1
<<Activity>>Task7 (2000)	1
<<EndEvent>>End7	1

Analysis

Reading the results, it might not be completely straightforward to see what has happened; however, if we draw the lifeline for each task in a Timing diagram, it becomes clearer.



- Event *Start3(@20)* is Non-Interrupting, it did not stop *Task1* at 20 seconds
- Event *Start4(@30)* is Interrupting, it stopped *Task1* and *Task2* at 30 seconds; it did not affected *Task5* because *Task5*'s enclosing process (main process) level is higher than *Start4*'s enclosing Sub-Process (*subProcess1*)
- Event *Start5(@60)* is Non-Interrupting, it started *Task6* at 60 seconds without affecting *Task3* or *Task5*
- Event *Start6(@80)* is Interrupting, it started *Task7* at 80 seconds and interrupted the running Tasks (*Task4*, *Task5*, *Task6*) that were in the same or lower level of its enclosing process
- Only *End7* is reached as expected

Fibonacci Number Generator with Link Event

A Link Event is a mechanism for connecting two sections of a Process. Link Events can be used:

- To create looping scenarios, as generic 'Go To' objects within the Process level
- To avoid long Sequence Flow lines; paired Link Events can be used as 'off-page' connectors for printing a Process across multiple pages

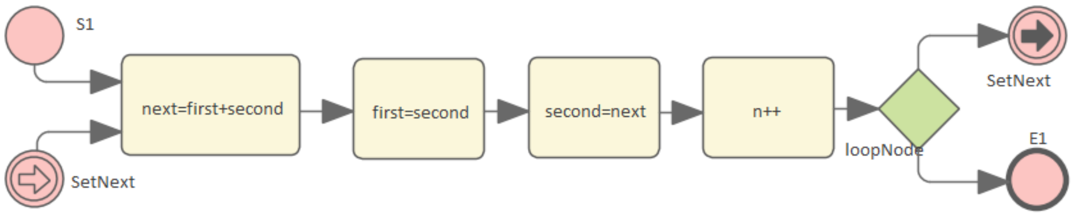
The use of Link Events is limited to a single Process level (that is, they cannot link a parent Process with a sub-Process).

There can be multiple source Link Events, but there can only be one target Link Event.

- The target Link Event marker is unfilled, to 'catch' from the source link
- The source Link Event marker is filled to 'throw' to the target link

When the EABPSim Execution Engine is running the simulation, the source-target Link Events are paired by element NAME, so they cannot be empty.

Create BPMN Model



- Create a StartEvent *S1*
- Add a Sequence Flow to a target abstractTask Activity element *next=first+second* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')
- Add a Sequence Flow to a target abstractTask Activity element *first=second*
- Add a Sequence Flow to a target abstractTask Activity element *second=next*
- Add a Sequence Flow to a target abstractTask Activity element *n++*
- Add a Sequence Flow to a target exclusive Gateway element *loopNode* (on the instant menu, select 'Exclusive')
- Add a Sequence Flow to each of these target elements:
  - A Throwing Link Intermediate Event element *SetNext* (open the 'Properties' dialog and set the 'Type' field to 'Throwing > Link') and
  - An End Event element *E1*
- Create a Catching Link Intermediate Event element *SetNext* (open the 'Properties' dialog and set the 'Type' field to 'Catching > Link')
- Add a Sequence Flow to the target element *next=first+second*

Configure BPSim

We will use Property Parameters to define how the sequence flow forms a loop during which a Fibonacci Number will be generated. The loop mechanism is implemented via the pair of Link Events.

Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager')

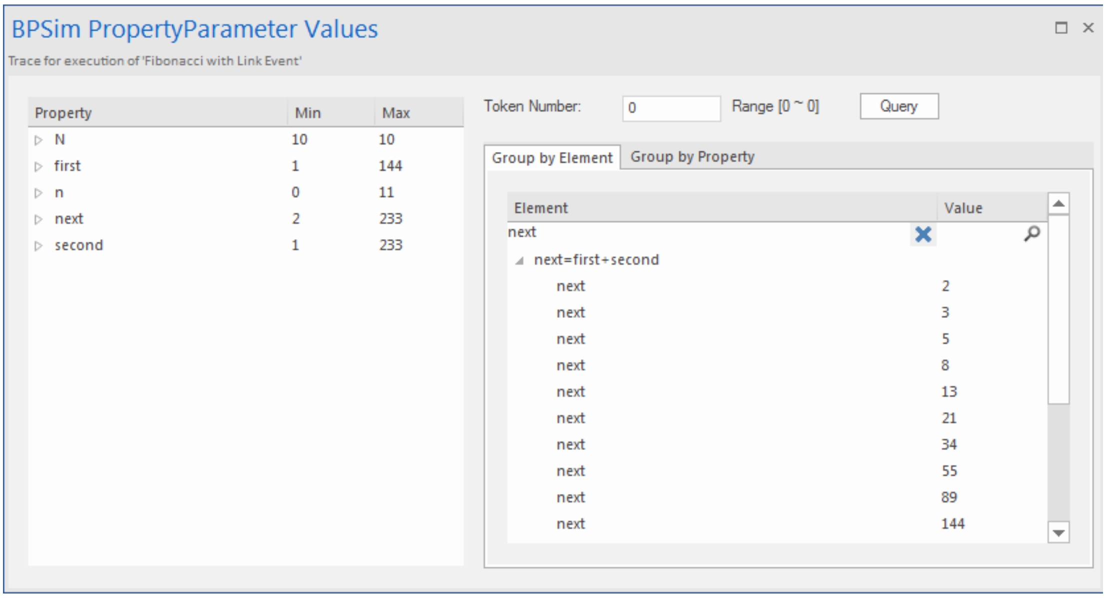
Task	Action
Element: S1	In the element type list on the left, expand the Start Event group and click on *S1*.

	Click on the 'Control Tab' and on the 'New Parameter' drop-down arrow; select 'TriggerCount'. In the 'Value' field type '1'. **Click on the 'Properties' tab** Overtype the *New Property* text to create these properties: • N - and type '10' in the 'Value' field as the total number of Fibonacci Numbers to be generated • first - and type '1' in the 'Value' field • second - and type '1' in the 'Value' field • n - and type '0' in the 'Value' field as the *n*th new Fibonacci Number
Element: next=first+second	In the element type list, expand the Activity group and click on *next=first+second*. Click on the 'Properties' tab and overtype the *New Property* text with 'next '. In the 'Value' field, click on the ... button, click on the 'Expression' tab and type the expression '{first}+{second}'. Click on the OK button.
Element: first=second	In the element type list, in the Activity group click on *first=second*. Click on the 'Properties' tab and overtype the *New Property* text with 'first '. In the 'Value' field, click on the ... button, click on the 'Expression' tab and type the expression '{second}'. Click on the OK button.
Element: second=next	In the element type list, in the Activity group click on *second=next*. Click on the 'Properties' tab and overtype the *New Property* text with 'second'. In the 'Value' field, click on the ... button, click on the 'Expression' tab and type the expression '{next}'. Click on the OK button.
Element: n++	In the element type list, in the Activity group click on *n++*. Click on the 'Properties' tab and overtype the *New Property* text with 'n'. In the 'Value' field, click on the ... button, click on the 'Expression' tab and type the expression '{n}+1'. Click on the OK button.
Conditions of Gateway	In the element type list, expand the Gateway group and the LoopNode element and click on *SetNext*. Click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Condition'. In the 'Value' field, click on the ... button, click on the 'Expression' tab and type the expression '{n} <={N}'. Click on the OK button. Now click on *E1*. Click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select

	'Condition'. In the 'Value' field, click on the ... button, click on the 'Expression' tab and type the expression '{n} > {N}'. Click on the OK button.

Run Simulation

- On the 'Configure BPSim' dialog, in the toolbar, click on the 'Run' icon; the 'BPSim Simulation Controller' dialog displays
- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'
- When the simulation is complete, click on the icon in the toolbar; the 'BPSim PropertyParameter Values' dialog displays
- Click on the Query button and on the 'Group by Element' tab, and expand 'next=first+second'; all the attributes's snapshot values are listed
- Apply a filter 'next' (right-click on the list header, select 'Toggle Filter Bar' and type 'next' under the 'Element' heading); the results will resemble this image:



BPSim PropertyParameter Values

Trace for execution of 'Fibonacci with Link Event'

Property	Min	Max
N	10	10
first	1	144
n	0	11
next	2	233
second	1	233

Token Number: 0 Range [0 ~ 0] Query

Group by Element | Group by Property

Element	Value
next	
next=first+second	
next	2
next	3
next	5
next	8
next	13
next	21
next	34
next	55
next	89
next	144

Ten more Fibonacci numbers are generated:

2,3,5,8,13,21,34,55,89,144

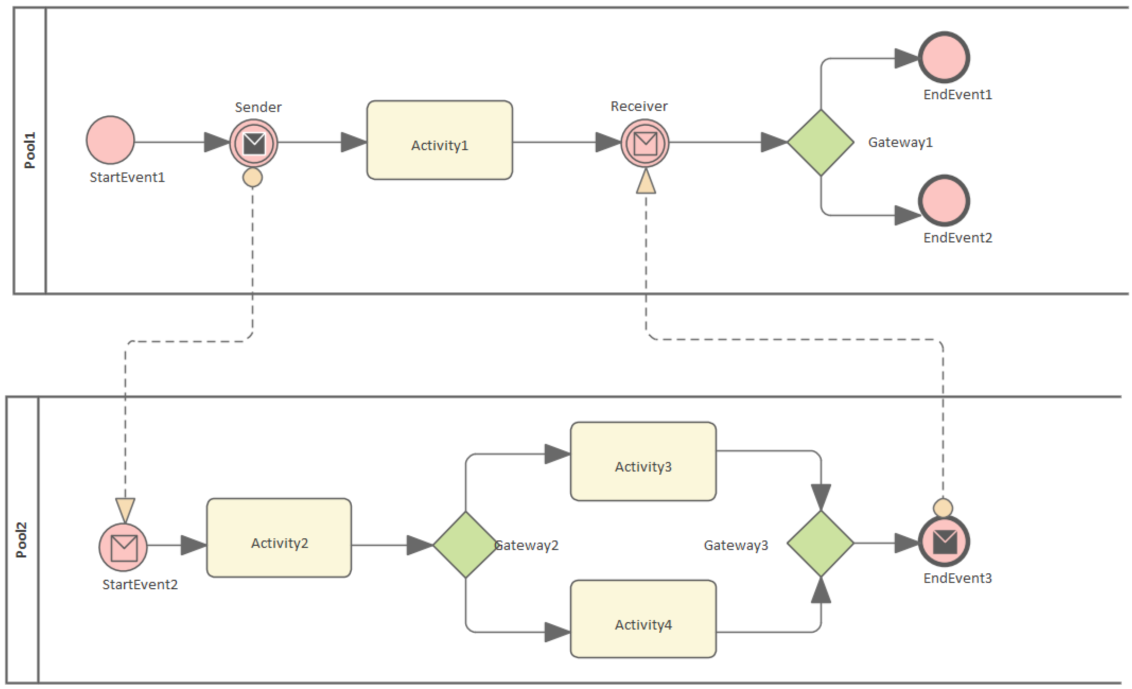
Message Event

When used in normal sequence flow, the Message Event can be used to either send or receive a message.

- When sending a message to a participant, the values of all property parameters are copied; once the message is sent the token continues along the sequence flow
- When receiving a message, the event is triggered when a message is received.

This example demonstrates the Message Event features. We will first create the BPMN model, then configure BPSim and run the simulation.

Create BPMN Model



Sequence

Pool1

- The token starts from StartEvent1
- On receiving the token, Sender (a Throwing Intermediate Message Event) creates a message and copies the current property values to the message
- Sender sends the *message* to the 'To' participant (Pool2, StartEvent2)
- Sender forwards the *token* along its sequence flow, as far as Receiver
- The token waits at Receiver until a message arrives

Pool2

- StartEvent2 receives a message and starts a token
- StartEvent2 copies the values from the message and sets these in the token
- StartEvent2 forwards the token along its sequence flow until EndEvent3
- EndEvent3 creates a message and copies the current property values to the message

- EndEvent3 sends the message to the 'To' participant (Pool1, Receiver)

Pool1 continued

- The waiting Receiver is awakened and the property values are updated from the arriving message

Create Diagram

- Create a BPMN 2.0 Collaboration diagram
- Select the option 'Create this diagram within a new CollaborationModel'
- Create *Pool1* and *Pool2* by dragging the 'Pool' icon from the Toolbox onto the diagram

Within Pool1

- Create a Start Event of type 'None', named *StartEvent1*
- Add a Sequence Flow to the target Intermediate Event of type 'Throwing Message', called *Sender*
- Add a Sequence Flow to the target Activity of type 'abstract', called *Activity1*
- Add Sequence Flows to the target:
  - End Event of type 'None', called *EndEvent1*
  - End Event of type 'None', called *EndEvent2*

Within Pool2

- Create a Start Event of type 'Message', called *StartEvent2*
- Add a Sequence Flow to the target Activity of type 'abstract', called *Activity2*
- Add a Sequence Flow to the target Gateway of type 'Exclusive', called *Gateway2*
- Add Sequence Flows to the target:
  - Activity of type 'abstract', called *Activity3*
  - Activity of type 'abstract', called *Activity4*
- Add Sequence Flows from *Activity3* and *Activity4* to the target Gateway of type 'Exclusive', called *Gateway3*
- Add a Sequence Flow to the End Event of type 'Message', called *EndEvent3*

Message Flows

- Add a Message Flow from *Sender* to *StartEvent2*
- Add a Message Flow from *EndEvent3* to *Receiver*

Configure BPSim

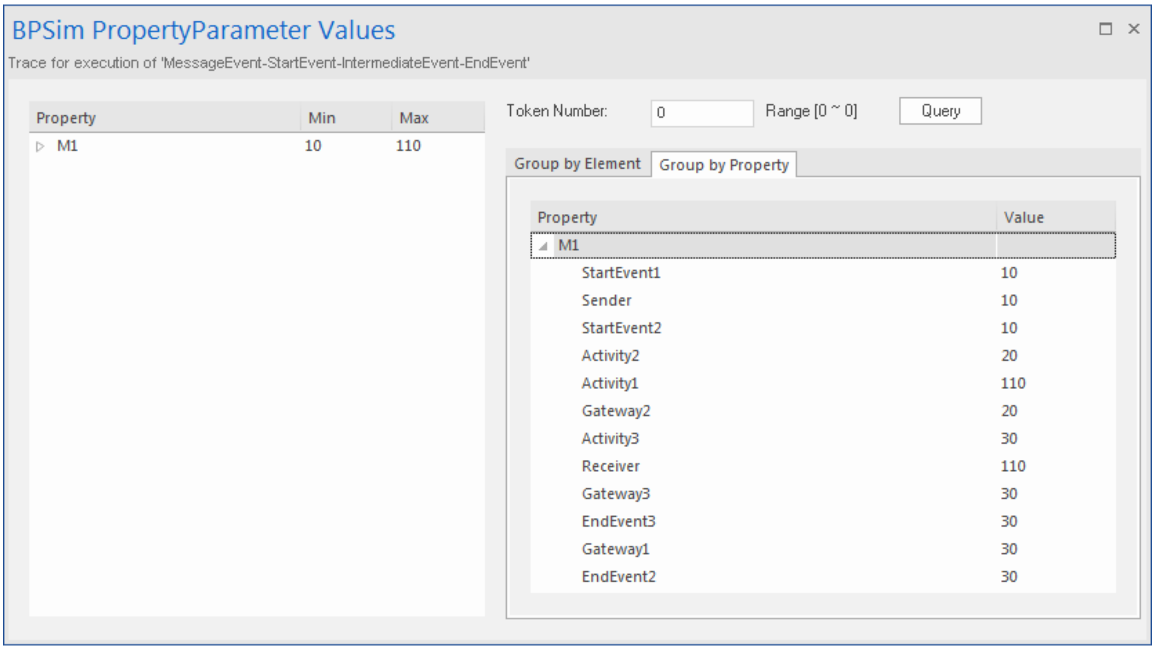
In order to show the Message Flow's ability to carry values, we create a Property Parameter 'M1' and change its value in each Activity. We then use the value of 'M1' as part of the expression for the condition of the Sequence Flow.

Task	Description
Create Artifact and Package	• Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager') • Click on the ... button in the 'Select/Create Artifact' field and create an Artifact called 'MessageEvent-StartEvent-IntermediateEvent-EndEvent' • In the 'Select Package' field select the Package containing the model All the BPMN elements from the model are loaded into the Configure BPSim

	window.
Property Values	We will give the Property Parameter 'M1' an initial value of 10 at *StartEvent1*. Then we change the value as the token flows through the processes and the value is copied across participants. In the element list on the left of the dialog: • Expand the 'StartEvent' group, click on *StartEvent1* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the ... button and select 'Constant' and 'Numeric, and type '10' in the 'Constant Numeric' field • Expand the 'Activity' group, click on *Activity1* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} + 100' in the 'Expression' field • Click on *Activity2* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} + 10' in the 'Expression' field • Click on *Activity3* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} + 10' in the 'Expression' field • Click on *Activity4* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} + 1' in the 'Expression' field *Tip: The format of* {PropertyName} *is a convenient short form of* getProperty("PropertyName").
Control Parameters	We only need one token in this simulation for evaluating the behavior of the model. • In the expanded 'StartEvent' group, click on *StartEvent1* and the 'Control' tab; click on the 'New parameter' drop-down arrow and select 'Trigger Count', and type in a 'Value' of '1' Now set up the conditions for the Gateways' outgoing Sequence Flows. In the element list on the left of the dialog, expand the 'Gateway' group: • Expand *Gateway1*, click on *EndEvent1* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} >= 50' in the 'Expression' field • Click on *EndEvent2* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} < 50' in the 'Expression' field • Expand *Gateway2*, click on *Activity3* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} >= 15' in the 'Expression' field • Click on *Activity4* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the ... button and select 'Expression', and type '{M1} < 15' in the 'Expression' field

Run Simulation

- On the 'Configure BPSim' dialog toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays
- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'; the simulation starts
- When the simulation is finished, click on the button; the 'Property Parameter Values' dialog displays, tracing the values for properties during simulation
- In the 'Token Number' field type '0', then click on the Query button and on the 'Group by Property' tab



BPSim PropertyParameter Values

Trace for execution of 'MessageEvent-StartEvent-IntermediateEvent-EndEvent'

Property	Min	Max
▷ M1	10	110

Token Number: 0 Range [0 ~ 0] Query

Group by Element | Group by Property

Property	Value
◢ M1	
StartEvent1	10
Sender	10
StartEvent2	10
Activity2	20
Activity1	110
Gateway2	20
Activity3	30
Receiver	110
Gateway3	30
EndEvent3	30
Gateway1	30
EndEvent2	30

Analysis

As the 'ProcessingTime' of *Activity1* was set as a distribution value, it turns out that:

- [Process1]'s 'M1' value after *Pool1.StartEvent1* is '10', as expected
- *[Process2] *Pool2.StartEvent2*'s 'M1' value is '10'; this value is carried from a message sent by *Pool1.Sender*

Now there are actually two 'M1's - *Process1.M1* and *Process2.M1*

- [Process2] *Pool2.Activity2* increased *Process2.M1* by 10; *[Process2.M1 == 20]*
- [Process1] *Pool1.Activity1* increased *Process1.M1* by 100; [*Process1.M1 == 110]*
- [Process2] The condition expressions are evaluated; as '20 > 15', the token will flow to *Activity3 [Process2.M1 == 20]*
- [Process2] *Pool2.Activity3* increased *Process2.M1* by 10; *[Process2.M1 == 30]*
- [Process1] *Pool1.Receiver* is reached and waiting *[Process1.M1 == 110]*
- [Process2] *Pool2.Gateway3* serves as a Merge node and continues to *EndEvent3 [Process2.M1 == 30]*
- *[Process1] *Pool1.Receiver is* woken up by a message(carrying *M1 == 30*) and *Process1.M1*'s value changes from 110 to 30
- [Process1] The condition expressions are evaluated; as '30 < 50', the token will flow to *EndEvent2 [Process1.M1 == 30]*

Notes

- The lines marked with an asterisk (*) are the effects of Message Flows
- The order within a process is defined; however, the order between two processes is not always predictable

- The Throwing Message Event forks another process; the Catching Message serves as thread synchronization

Signal Events

A Signal Event provides the facility of loosely coupling 'throwers' and 'catchers' by publish-subscribe integration. A 'thrower' will broadcast a signal rather than addressing it to a particular process; any process listening for that particular event could trigger a new instance using a Signal Start Event.

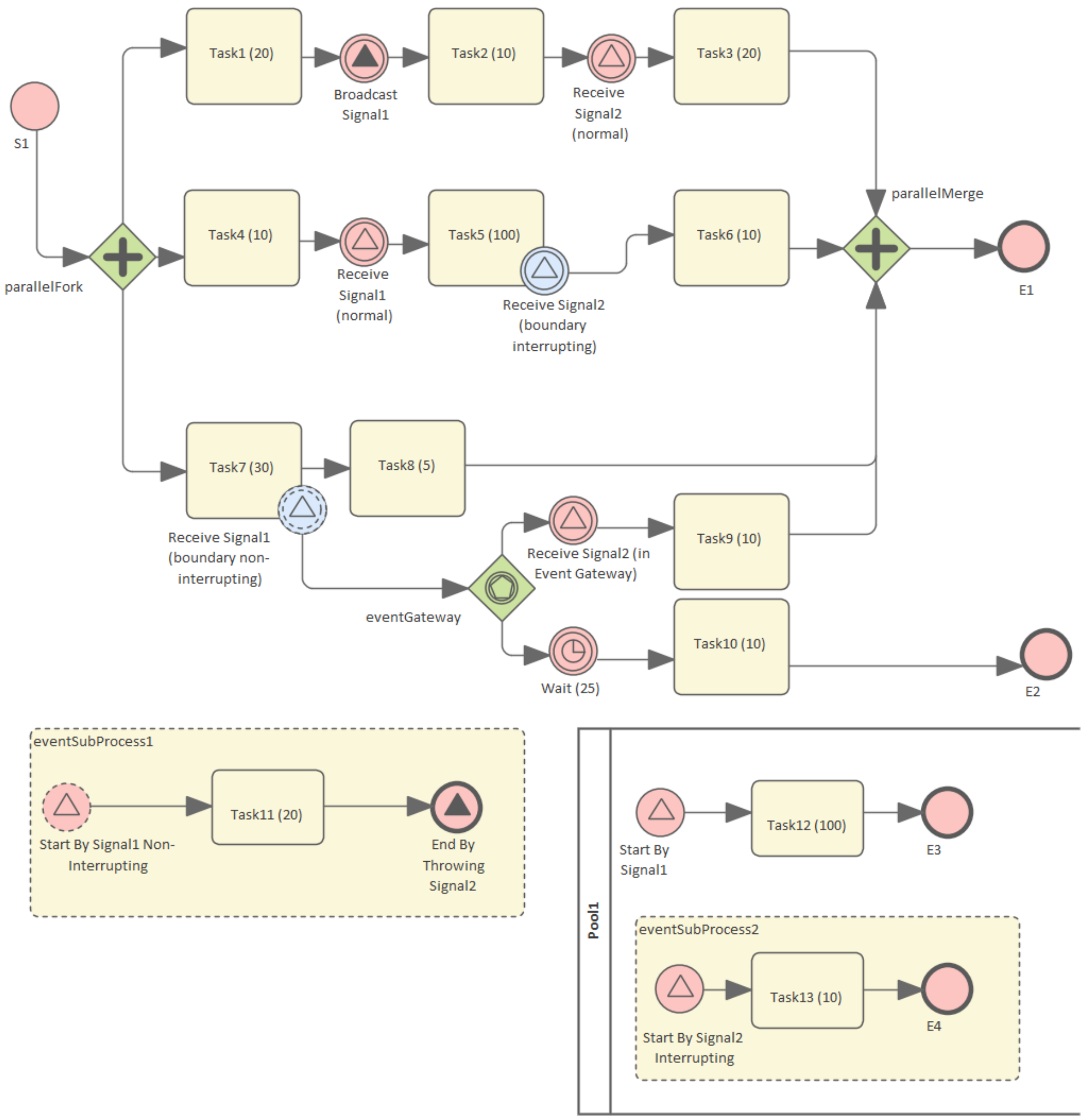
A Signal can be thrown from either a throwing Intermediate Event or a throwing End Event, and can be caught in a Start Event or a catching Intermediate Event (including a boundary Signal Event).

In this example, we demonstrate these Signal Events and their impact on the lifelines of tasks, via BPSim parameter settings.

- Start Signal Event:
  - *Start By Signal1* in top level process (Pool1)
  - *Start By Signal2* Interrupting in event sub-process *eventSubProcess2*
  - *Start By Signal1* Non Interrupting in event sub-process *eventSubprocess1*
- Throwing Intermediate Signal Event:
  - *Broadcast Signal1*
- Catching Intermediate Signal Event:
  - *Receive Signal1 (normal)*
  - *Receive Signal2 (normal)*
  - *Receive Signal2 (boundary Interrupting)*
  - *Receive Signal1 (boundary non-interrupting)*
  - *Receive Signal2 (in Event Gateway)*
- End Signal Event:
  - *End By Throwing Signal2*

Create BPMN Model

In order to demonstrate the ability to communicate across processes via a Signal Event, we create a Collaboration model with a main Pool and a process in another Pool (*Pool1*).



Create the Collaboration and main process

Create a new BPMN2.0 Collaboration diagram called *CollaborationForTestingSignalEvents*, (choose the option 'Create this diagram within a new Collaboration Model'). Right-click on the diagram name in the Browser window and select the 'Encapsulate Process' option.

A Pool *PoolMain* and a process *BusinessProcess_PoolMain* are created, and these tags are set with the automatic values:

- *CollaborationForTestingSignalEvents.mainPool* is set to *PoolMain*
- *PoolMain.processRef* is set to *BusinessProcess_PoolMain*

Create the elements for the main process

Create a Start Event *S1* and add a Sequence Flow to a Fork Parallel Gateway *parallelFork*

Add Sequence Flows to:

- An Abstract Task *Task1 (20)* and then add this chain of Sequence Flows:
    - To a Throwing Intermediate Signal Event *Broadcast Signal1*
    - Then to an Abstract Task *Task2 (10)*
    - Then to a Catching Intermediate Signal Event *Receive Signal2 (normal)*
    - Then to an Abstract Task *Task3 (20)*

  - Then to a Merge Parallel Gateway *parallelMerge*
  - Then to an End Event *E1*

- An Abstract Task *Task4 (10)* and then add this chain of Sequence Flows:
  - To a Catching Intermediate Signal Event *Receive Signal1 (normal)*
  - Then to an Abstract Task *Task5 (100),* on which you create a Boundary Interrupting Catching Intermediate Signal Event *Receive Signal2 (boundary interrupting)*
  - Then to an Abstract Task *Task6 (10)*
  - Then to the earlier Merge Parallel Gateway *parallelMerge*

- An Abstract Task *Task7 (30)*, and then add this chain of Sequence Flows:
  - To an Abstract Task *Task8 (5)*
  - Then to the earlier Merge Parallel Gateway *parallelMerge*

On *Task7 (30)*, create a Boundary Non-interrupting Catching Intermediate Signal Event *Receive Signal1 (boundary non-interrupting)*. Add a Sequence Flow to an Event Gateway *eventGateway*, and to that add Sequence Flows to:

- A Catching Intermediate Signal Event *Receive Signal2 (in Event Gateway),* and then this chain of Sequence Flows:
  - To an Abstract Task *Task9 (10)*
  - Then to the earlier Merge Parallel Gateway *parallelMerge*

- A Catching Intermediate Timer Event *Wait (25)*, and then this chain of Sequence Flows:
  - To an Abstract Task *Task10 (10)*
  - Then to an End Event *E2*

Create an Event sub-process (triggered by a non-interrupting Start Signal Event) in the main process

- Create an Activity *eventSubProcess1* and, in its 'Properties' dialog, set the 'Type' field to *subProcess* and change the attribute 'triggeredByEvent' to *true*
- Within *eventSubProcess1* create a Start Event *Start By Signal1 Non Interrupting* and, in its 'Properties' dialog, set the 'Type' field to *Event Sub-Process Non-Interrupting > Signal*
- Add a Sequence Flow to a target Abstract Task *Task11 (20)*
- Add a Sequence Flow to a target End Event *End By Throwing Signal2* and, in the element 'Properties' dialog, set the 'Type' field to *Signal*

Create another process

- From the Toolbox, drag and drop the 'Pool' icon onto the diagram and name the element *Pool1*
- Right-click on *Pool1* in the Browser window and select the 'Encapsulate Process' option; a process *BusinessProcess_Pool1* is created and the tag 'Pool1.processRef' is set to *BusinessProcess_Pool1*

Create the main process for *Pool1*

- Create a Signal Start Event *Start By Signal1*
- Add a Sequence Flow to a target Abstract Task *Task12 (100)*
- Add a Sequence Flow to a target End Event *E3*

Create an Event sub-process to interrupt *Pool1*

- Create an Activity *eventSubProcess1* and, in the 'Properties' dialog, set the 'Type' field to *subProcess*; change the attribute 'triggeredByEvent' to *true*
- Within *eventSubProcess2* create a Start Event *Start By Signal2 Interrupting* and, in the 'Properties' dialog, set the 'Type' field to *Event Sub-Process Interrupting > Signal*
- Add a Sequence Flow to a target Abstract Task *Task13 (10)*
- Add a Sequence Flow to a target End Event *E4*

Create the BPMN2.0 Signal Elements and configure for Signal Events

In the BPMN 2.0 Toolbox, expand the 'BPMN 2.0 - Types' page and drag the 'Signal' icon onto the diagram; name the element *Signal1*. Drag the icon onto the diagram again to create *Signal2*. These are root elements (which can be used by all processes) so they will be created directly under the model Package.

Double-click on each of the Signal Event elements and, in the 'Value' field for the 'signalRef' tag, click on the ... button and browse to the appropriate Signal element.

Tips: Alternatively, you can drag the Signal element from the Browser window and drop it on the Event elements in the diagram; a context menu displays, from which you select the 'set signalRef' option.

- Set signalRef to 'Signal1' on:
    - *Broadcast Signal1*
    - *Start By Signal1* in top level process (*Pool1*)
    - *Start By Signal1 Non Interrupting* in Event sub-process *eventSubprocess1*
    - *Receive Signal1 (normal)*
    - *Receive Signal1 (boundary non-interrupting)*
- Set signalRef to 'Signal2' on:
    - *Start By Signal2 Interrupting* in Event sub-process *eventSubProcess2*
    - *Receive Signal2 (normal)*
    - *Receive Signal2 (boundary Interrupting)*
    - *Receive Signal2 (in Event Gateway)*

Configure BPSim

In this section, we create the Configuration Artifact, specify the model Package and configure the parameter values of each element.

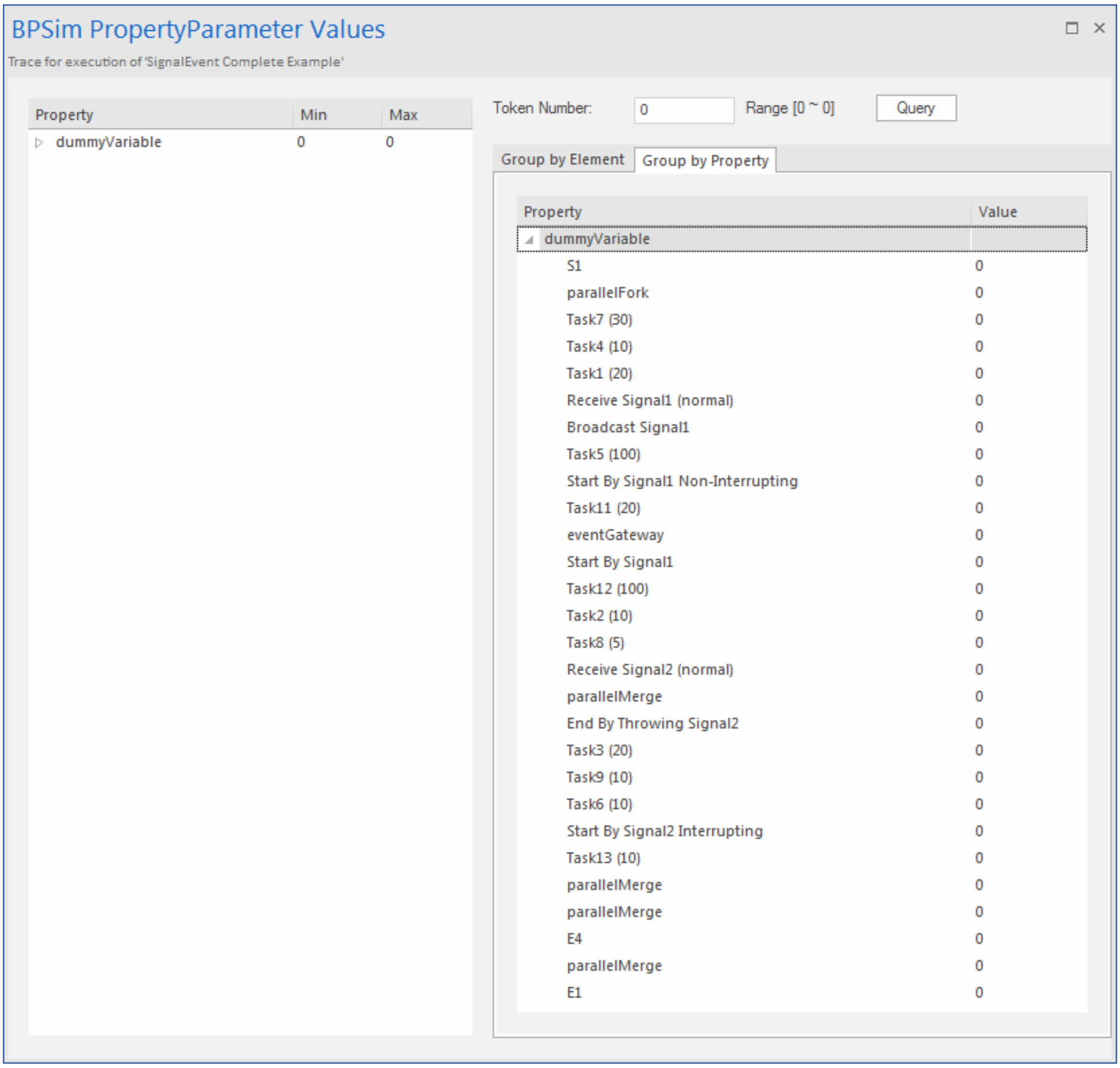
The configuration is quite simple because none of the Signal Events require any BPSim configurations. All we have to do is set the processing time for tasks so we can observe how processes, threads and tasks are started and interrupted.

Task	Description
Set up configuration	• Open the Configure BPSim window ('Simulate > Process Analysis > BPSim > Open BPSim Manager') • Create an Artifact named 'SignalEvent Complete Example' (in the 'Select/Create Artifact' field, click on the ... button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the Configure BPSim window.
Non-Signal Events	• In the element list on the left of the dialog, expand the 'StartEvent' group, then click on *S1* and on the 'Control' tab; click on the 'New Parameter' drop-down arrow and select 'Trigger Count', then type '1' in the 'Value' field • Expand the 'IntermediateEvent' group, then click on *Wait (25)* and on the 'Control' tab; click on the 'New Parameter' drop-down arrow and select 'InterTriggerTimer', then click on the ... button in the 'Value' field; select 'Constant' and 'Numeric', and type '25' in the 'Constant Numeric' field and 'seconds' in the 'TimeUnit' field
Dummy variable for Process	The simulation controller displays a list showing the runtime token count for each element. For example, 4 tokens have passed the Gateway element *parallelMerge* in a simulation. This is quite useful for certain statistics and analysis. However, it does not show WHEN *parallelMerge* was traversed during the simulation. In order to get the exact trace for a single token we use the property trace utility, which relies on property parameters. So we create a dummy parameter.

	In the 'Configuration BPSim' dialog, expand the 'BusinessProcess' group. • Click on *BusinessProcess_Main* and on the 'Properties' tab, and overtype *New Property* with *dummyVariable;* in the 'Value' field, click on the ... button and on 'Constant' and 'Numeric', and in the 'Constant Numeric' field type '0' • Click on *BusinessProcess_Pool1* and perform exactly the same actions as for *BusinessProcess_Main*
Processing Time for Tasks	Expand the 'Activity' group and for each Task element listed here: select the 'Time' tab, click on the 'New Parameter' drop-down arrow and select 'ProcessingTime', then click on the ... button on the 'Value' column, select 'Constant' and 'Numeric', type the value as indicated into the 'Constant Numeric' field and select 'seconds' in the 'TimeUnit' field. • Task1 (20): 20 seconds • Task2 (10): 10 seconds • Task3 (20): 20 seconds • Task4 (10): 10 seconds • Task5 (100): 100 seconds • Task6 (10): 10 seconds • Task7 (30): 30 seconds • Task8 (5): 5 seconds • Task9 (10): 10 seconds • Task10 (10): 10 seconds • Task11 (20): 20 seconds • Task12 (100): 100 seconds • Task13 (10): 10 seconds

Run Simulation

- On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog
- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'
- After simulation, click on the button on the tool bar to display the 'BPSim PropertyParameter Values' dialog
- Click on the Query button and on the 'Group by Property' tab, and expand 'dummyVariable'



BPSim PropertyParameter Values

Trace for execution of 'SignalEvent Complete Example'

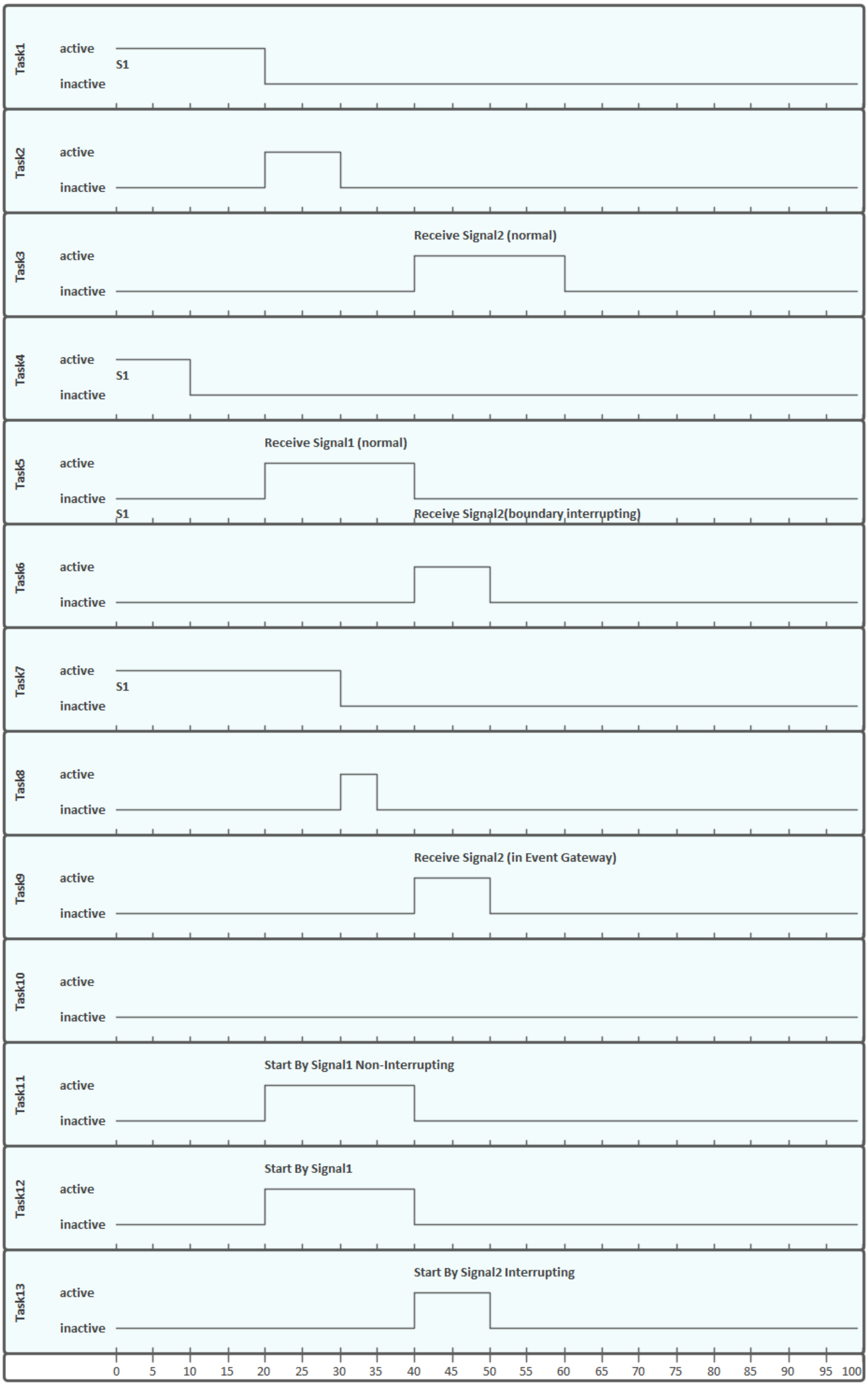
Property	Min	Max
dummyVariable	0	0

Token Number: 0 Range [0 ~ 0]

Property	Value
dummyVariable	
S1	0
parallelFork	0
Task7 (30)	0
Task4 (10)	0
Task1 (20)	0
Receive Signal1 (normal)	0
Broadcast Signal1	0
Task5 (100)	0
Start By Signal1 Non-Interrupting	0
Task11 (20)	0
eventGateway	0
Start By Signal1	0
Task12 (100)	0
Task2 (10)	0
Task8 (5)	0
Receive Signal2 (normal)	0
parallelMerge	0
End By Throwing Signal2	0
Task3 (20)	0
Task9 (10)	0
Task6 (10)	0
Start By Signal2 Interrupting	0
Task13 (10)	0
parallelMerge	0
parallelMerge	0
E4	0
parallelMerge	0
E1	0

Analysis

From the direct results of the simulation it might not be obvious what has taken place; however, if we draw the lifeline for each task, it becomes quite clear.



- *Task1, Task4* and *Task7* started in parallel

- *Task2* started immediately after *Task1* finished (without stopping at the throwing event)
- At 20 seconds, *Signal1* was broadcast by the Throwing Intermediate Event *Broadcast Signal1* and:
  - *Receive Signal1 (normal)* was activated and *Task5* started
  - *Start By Signal1 Non-Interrupting* was activated and *Task11* in *eventSubProcess1* started
  - *Start By Signal1* was activated and *Task12* in *Pool1 s*tarted
- At 40 seconds, *Signal2* was broadcast by the End Event *End By Throwing Signal2* and:
  - *Receive Signal2 (normal)* was activated and *Task3* started
  - *Task5* was interrupted and *Task6* started
  - *Receive Signal2 (in Event Gateway)* was activated and *Task9* started
  - *Start By Signal2 Interrupting* was activated, and:
    > The main process in *Pool1* was interrupted and *Task12* stopped
    > *Task13* in *eventSubProcess2* started
- The *eventSubProcess2* inside *BusinessProcess_Pool1* finished when *E4* was reached at 50 seconds
- The *BusinessProcess_MainPool* finished when *E1* was reached at 60 seconds
- The Intermediate Timer Event *Wait (25)* did not get activated because the signal event in the Gateway was activated first; as a result, *Task10* was never started

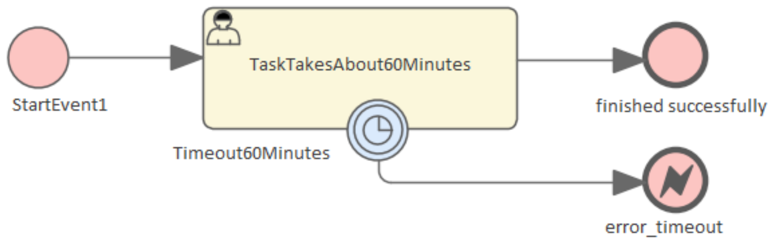
Note: The actual running time for each task can be observed from the generated BPSimReport element, by:

1. Double-clicking on the <<BPSimReport>> element.
2. Expanding the 'Time' group.
3. Expanding the task element.
4. Checking 'Total Time In Task'.

For example, for element *Task5 (100)*, although we set its processingTime to be 100 seconds, the **Total Time In Task** was 20 seconds, which was interrupted by *Receive Signal2 (boundary interrupting)* at 20 seconds.

Timer Event - Boundary

Create BPMN Model



- Create a Start Event *StartEvent1*
- Add a Sequence Flow to a target userTask *TaskTakesAbout60Minutes*
- Add a Sequence Flow to a target End Event *finished successfully*
- Create an Intermediate Event, dragging the icon from the Toolbox and dropping it onto *TaskTakesAbout60Minutes;* select 'Edge-Mounted' and 'Timer' from the automatic menus, and call the element *Timeout60Minutes*
- Add a Sequence Flow to a target End Event (Error) *error_timeout*

Configure BPSim

In this section, we create the Configuration Artifact, identify the parent Package and set the parameter values of each element.

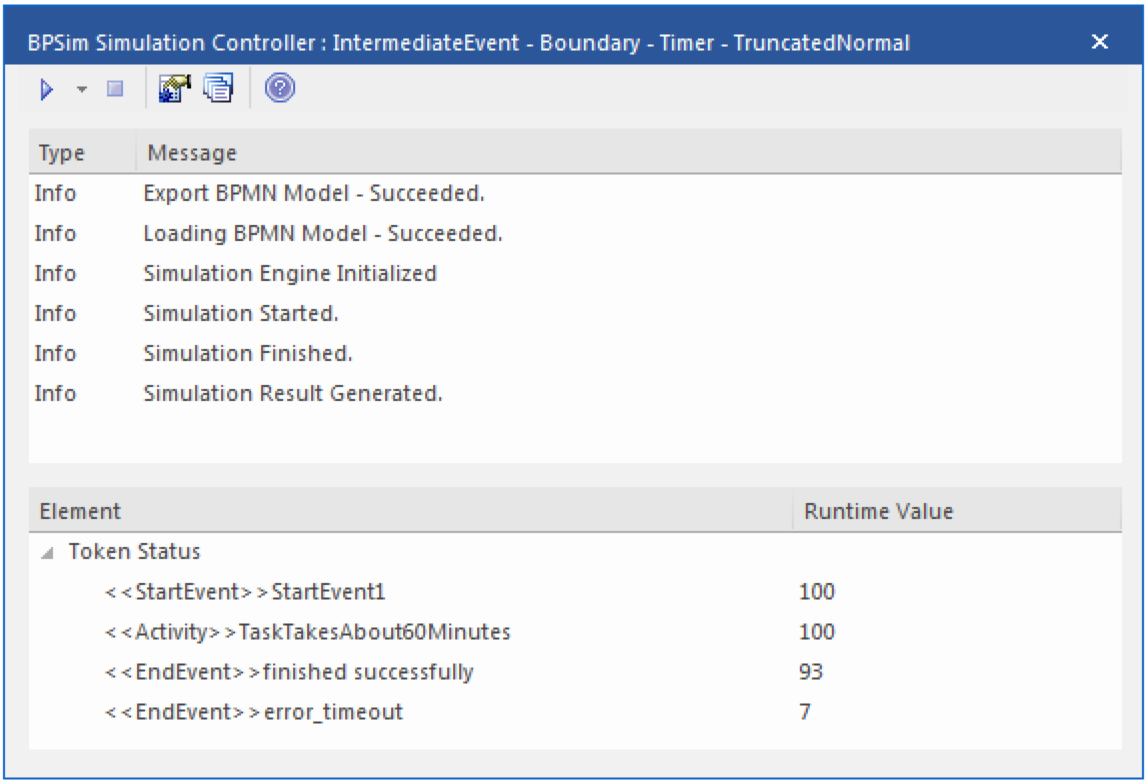
Objects	Action
Create Artifact and Package	• Open the 'Configure BPSim' dialog ('Simulate > Process Analysis > BPSim > Open BPSim Manager') • Create an Artifact named 'IntermediateEvent - Boundary - Timer - TruncatedNormal' (in the 'Select/Create Artifact' field, click on the ... button and select its parent Package, click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the 'Configure BPSim' dialog.
StartEvent1	In the element list on the left of the dialog, expand the 'StartEvent' group and click on *StartEvent1*. • Click on the 'Control' tab • Click on the 'New Parameter' drop-down arrow and select 'TriggerCount' • In the 'Value' field type '100'
TaskTakesAbout60Minutes	In the element list on the left of the dialog, expand the 'Activity' group and click on *TaskTakesAbout60Minutes.* • Click on the 'Time' tab • Click on the 'New Parameter' drop-down arrow and select 'ProcessingTime' • In the 'Value' field click on the ... button and select 'Distribution' and 'TruncatedNormal'

	- In the 'Mean' field type '50' - In the 'StandardDeviation' field type '10' - In the 'Min' field type '0' - In the 'Max' field type '1000' - Click on the OK button
Timeout60Minutes	In the element list on the left of the dialog, expand the 'IntermediateEvent' group and click on *Timeout60Minutes*. - Click on the 'Control' tab - Click on the 'New Parameter' drop-down arrow and select 'InterTriggerTimer' - Set the value to '000:000:000 001:00:00' (that is, 1 hour)

Run Simulation

- On the Configure BPSim window toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog
- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'
- After simulation, click on the button on the tool bar to display the 'BPSim PropertyParameter Values' dialog
- Click on the Query button and on the 'Group by Property' tab, and expand 'dummyVariable'

On simulation, we get this result:



BPSim Simulation Controller : IntermediateEvent - Boundary - Timer - TruncatedNormal

Type	Message
Info	Export BPMN Model - Succeeded.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Runtime Value
Token Status	
<<StartEvent>>StartEvent1	100
<<Activity>>TaskTakesAbout60Minutes	100
<<EndEvent>>finished successfully	93
<<EndEvent>>error_timeout	7

Analysis

Since the ProcessingTime of *TaskTakesAbout60Minutes* was set as a distribution value, it turns out that:

- 93 out of 100 finished in 1 hour, so the normal flow to *finished successfully* takes effect

- 7 out of 100 finished in more than 1 hour, so the exception flow to *error_timeout* takes effect

Other Configurations

In the example folder, there are two other Business Process Simulation Artifacts that set the ProcessingTime as a constant value of 50 minutes and 80 minutes, other settings remain the same.

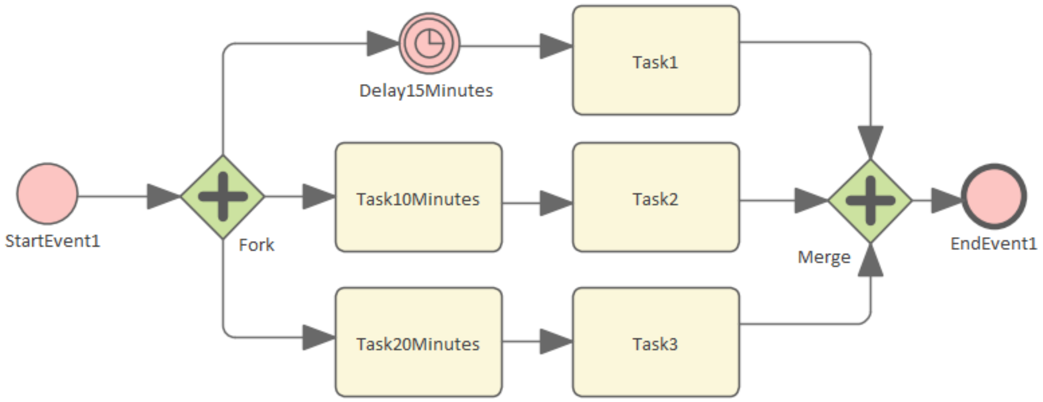
Run simulation on these two Artifacts:

- ProcessingTime configured to be 50 minutes always finishes in normal flow
- ProcessingTime configured to be 80 minutes always finishes in exception flow

Timer Event - Standalone Intermediate Event

When a Timer Intermediate Event is used within the normal sequence flow as a standalone element, it acts as a delay mechanism.

Create BPMN Model



- Create a Start Event called *StartEvent1*
- Add a Sequence Flow to a target Parallel Gateway called *Fork*
- Add Sequence Flows to:
  - A Standalone Timer Intermediate Event called *Delay15Minutes*, and from that a Sequence Flow to an Activity called *Task1*
  - An Activity called *Task10Minutes*, and from that a Sequence Flow to an Activity called *Task2*
  - An Activity called *Task20Minutes*, and from that a Sequence Flow to an Activity called *Task3*
- From *Task1*, *Task2* and *Task3* create Sequence Flows to a Merge Parallel Gateway called *Merge*
- Add a Sequence Flow to a target EndEvent called *EndEvent1*

Configure BPSim

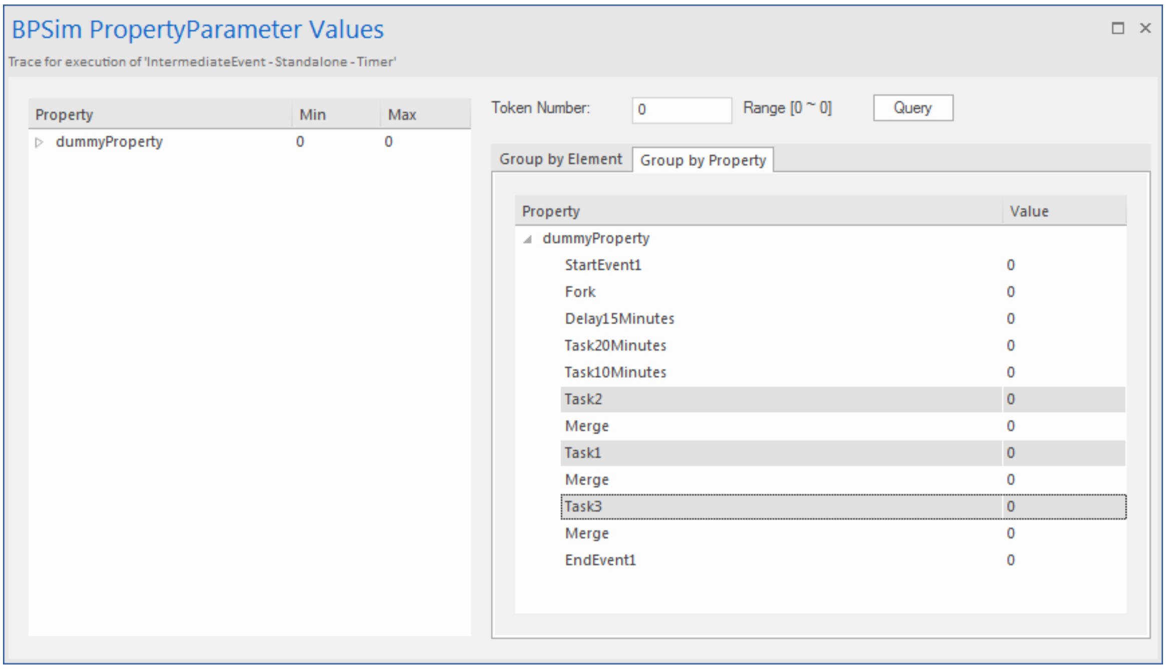
In this section, we create the Configuration Artifact, specify the model Package and configure the parameter values of each element.

Object	Action
Create Artifact and Package	• Open the 'Configure BPSim' dialog ('Simulate > Process Analysis > BPSim > Open BPSim Manager') • Create an Artifact named 'IntermediateEvent - Standalone - Timer' (in the 'Select/Create Artifact' field, click on the ... button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the 'Configure BPSim' dialog.
StartEvent1	• In the element list on the left of the dialog, expand the 'StartEvent' group, then click on *StartEvent1* and on the 'Control' tab • Click on the 'New Parameter' drop-down arrow and select 'Trigger Count', then

	type '1' in the 'Value' field - Click on the 'Properties' tab - Overtype *New Property* with *dummyProperty;* in the 'Value' field, click on the ... button and on 'Constant' and 'Numeric', and in the 'Constant Numeric' field type '0' With this property, the 'Property Trace' dialog will be able to show the sequence of element flows during simulation.
Delay15Minutes	- In the element list on the left of the dialog, expand the 'IntermediateEvent' group, then click on *Delay15Minutes* and on the 'Control' tab - Click on the 'New Parameter' drop-down arrow and select 'InterTriggerTimer', then set the 'Value' field to 15 minutes ('000:000:000 000:15:00')
Task10Minutes	- In the element list on the left of the dialog, expand the 'Activity' group, then click on *Task10Minutes* and on the 'Time' tab - Click on the 'New Parameter' drop-down arrow and select 'ProcessingTime', then set the 'Value' field to 10 minutes ('000:000:000 000:10:00')
Task20Minutes	- In the element list on the left of the dialog, expand the 'Activity' group, then click on *Task20Minutes* and on the 'Time' tab - Click on the 'New Parameter' drop-down arrow and select 'ProcessingTime', then set the 'Value' field to 20 minutes ('000:000:000 000:20:00')

Run Simulation

- On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog
- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'
- After simulation, click on the button on the tool bar to display the 'BPSim PropertyParameter Values' dialog
- Click on the Query button and on the 'Group by Property' tab



BPSim PropertyParameter Values

Trace for execution of 'IntermediateEvent - Standalone - Timer'

Property	Min	Max
dummyProperty	0	0

Token Number: 0 Range [0 ~ 0] Query

Property	Value
dummyProperty	
StartEvent1	0
Fork	0
Delay15Minutes	0
Task20Minutes	0
Task10Minutes	0
Task2	0
Merge	0
Task1	0
Merge	0
Task3	0
Merge	0
EndEvent1	0

Analysis

The *Fork* Parallel Gateway will activate the outgoing Sequence Flows simultaneously (the order is undefined and not important). However, we would expect the order of the Tasks to be exactly:

- *Task2*
- *Task1*
- *Task3*

This order is determined by the BPSim parameters set on two of the Activities (ProcessingTime) and the Timer Intermediate Event (InterTriggerTimer). The sequence shown in the 'BPSim PropertyParameter Values' dialog confirms that *Task2* comes ahead of *Task1*, which comes ahead of *Task3*.

Paint Wall Process Simulation (Call Activity)

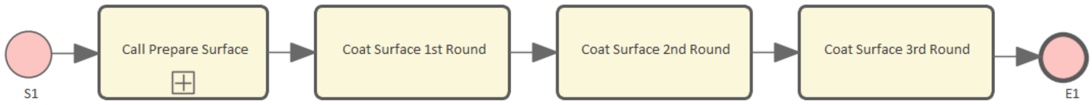
This is a simple example to simulate the process of painting a wall. We define the main process as preparing the surface and then painting it three times. Preparing the surface is further divided into tasks such as sanding and cleaning.

We assume that applying each of the three coats of paint is the same process, except that the time randomly spent on each coat might be different.

Create BPMN Model

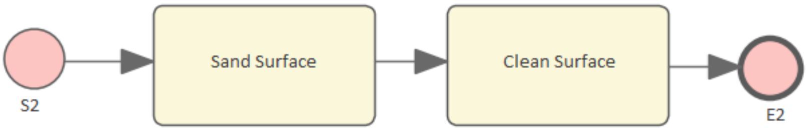
This simulation operates on two processes.

The main process - Paint Wall Process



1. Create a Start Event called *S1.*
2. Add a Sequence Flow to a target callProcessActivity called *Call Prepare Surface.*
3. Add a Sequence Flow to a target callGlobalTaskActivity called *Coat Surface 1st Round.*
4. Add a Sequence Flow to a target callGlobalTaskActivity called *Coat Surface 2nd Round.*
5. Add a Sequence Flow to a target callGlobalTaskActivity called *Coat Surface 3rd Round.*
6. Add a Sequence Flow to a target End Event called *E1.*

The re-used process - Prepare Surface Process



1. Create a Start Event called *S2.*
2. Add a Sequence Flow to a target Abstract Task called *Sand Surface.*
3. Add a Sequence Flow to a target Abstract Task called *Clean Surface.*
4. Add a Sequence Flow to a target End Event called *E2.*

Set Global Task and re-used process to call Activities

1. Create a Global Task Activity called *Coat Surface.*
2. Double-click on each of *Coat Surface 1st Round, Coat Surface 2nd Round* and *Coat Surface 3rd Round*, and set the tag 'calledActivityRef' to *Coat Surface.*
   Tip: You can also drag the Global Task 'Coat Surface' from the Browser window and drop it on the Call Activity element, clicking the 'Set calledActivityRef' option on the context menu.
3. Double-click on *Call Prepare Surface* and set the tag 'calledActivityRef' to *Prepare Surface Process.*
   Tip: You can also drag the process 'Prepare Surface Process' from the Browser window and drop it on the Call Activity element, clicking the 'Set calledActivityRef' option on the context menu.

Configure BPSim

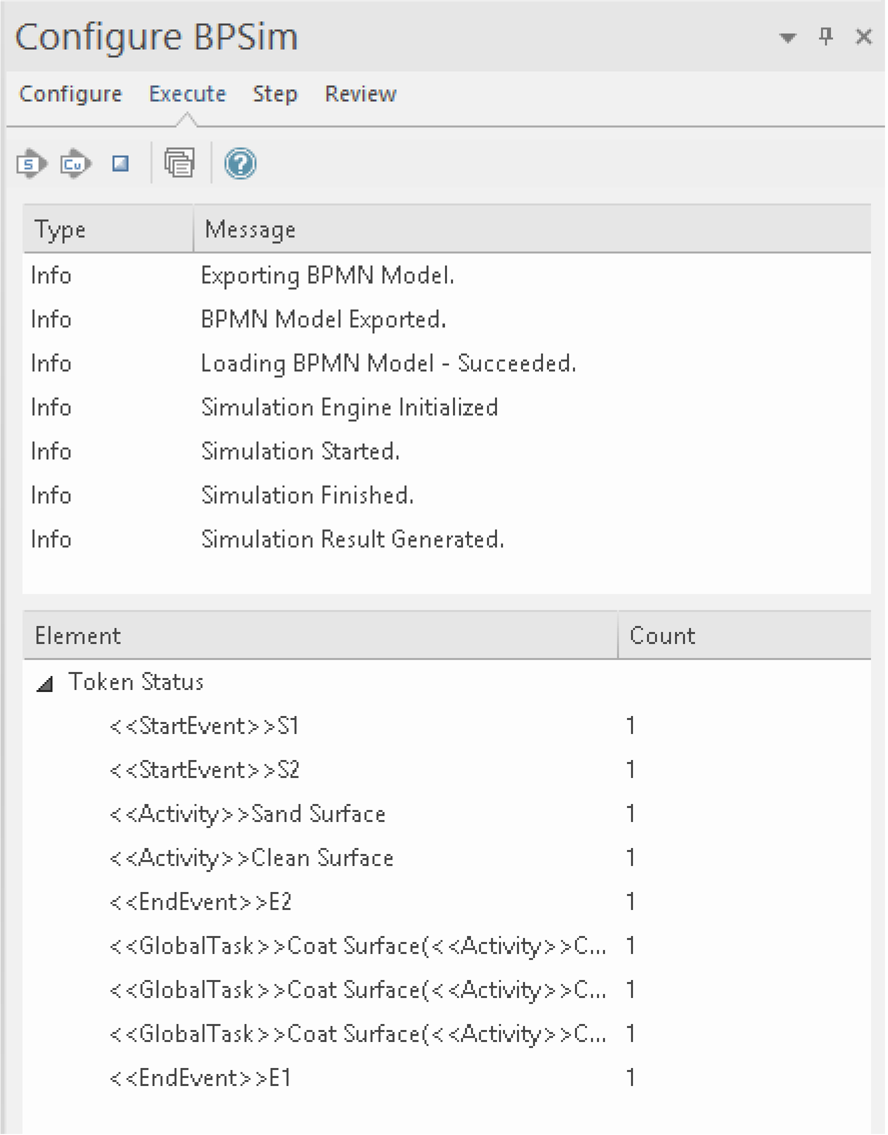
1. Open the 'Configure BPSim' dialog ('Simulate > Process Analysis > BPSim > Open BPSim Manager').
2. Click on the 📄 icon and create a Business Process Simulation Artifact named *Paint Wall Simulation.*
3. Click on the 📁 icon and select the Package containing the corresponding BPMN 2.0 model.

Object	Activity
Fixed Scaling Time	1. Open the 'Prepare Surface Process' diagram. 2. Click on the Activity *Sand Surface* and, on the Configure BPSim window, click on the *New Parameter* drop-down arrow and create a Time parameter called 'Processing Time'. 3. In the 'Values' field, change the setting to 0 00:30:00 (that is, 30 minutes). 4. Click on the Activity *Clean Surface* and, on the Configure BPSim window, click on the *New Parameter* drop-down arrow and create a Time parameter called 'Processing Time'. 5. In the 'Values' field, change the setting to 0 00:10:00, (that is, 10 minutes). 6. Click on the 💾 icon.
Random Coating Time	1. In the Browser window, click on the Global Task Activity *Coat Surface.* 2. In the Configure BPSim window, click on the *New Parameter* drop-down arrow and create a Time parameter called 'Processing Time'. 3. In the 'Values' field click on the ... button, and on the parameter dialog click on the 'Distribution' tab and on 'Poisson'. 4. In the 'Mean' field type '10', then click on the OK button. 5. Click on the 💾 icon. With this setting, the mean value of the random numbers generated by the Poisson distribution is 10. If you prefer, you can choose other types of distribution.
TriggerCount on S1	On the 'Paint Wall Process' diagram click on the Start Event *S1.* 1. In the Configure BPSim window, click on the *New Parameter* drop-down arrow and create a Control parameter called 'TriggerCount'. 2. In the 'Values' field type '1'. 3. Click on the 💾 icon.

Run Simulation

1. On the 'Execute' tab of the Configure BPSim window, click on the ▶ icon.

When the simulation is complete, it provides a result similar to this:

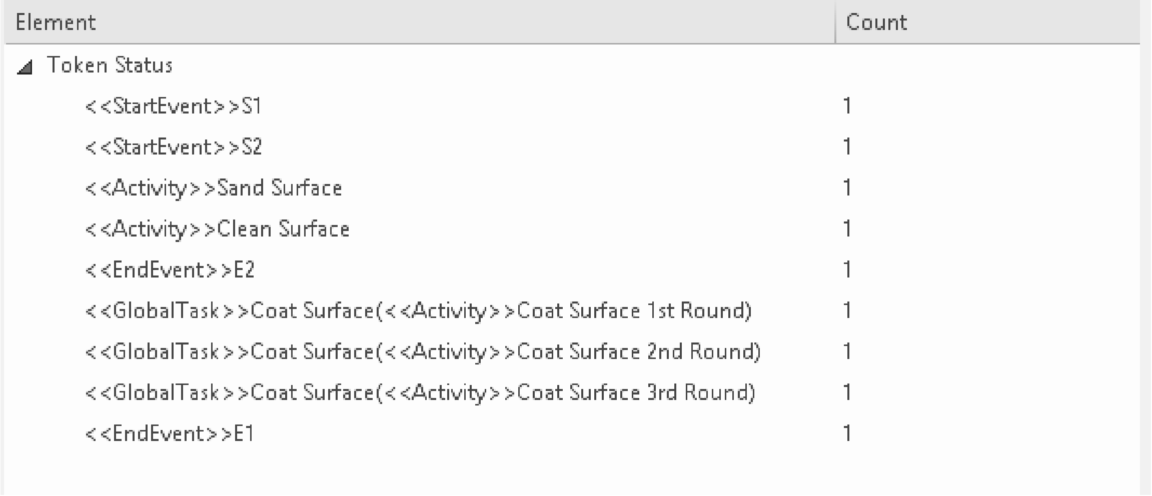


Type	Message
Info	Exporting BPMN Model.
Info	BPMN Model Exported.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Count
◢ Token Status	
<<StartEvent>>S1	1
<<StartEvent>>S2	1
<<Activity>>Sand Surface	1
<<Activity>>Clean Surface	1
<<EndEvent>>E2	1
<<GlobalTask>>Coat Surface(<<Activity>>C...	1
<<GlobalTask>>Coat Surface(<<Activity>>C...	1
<<GlobalTask>>Coat Surface(<<Activity>>C...	1
<<EndEvent>>E1	1

Flow Analysis

For the only token started on *S1*, we can see from the Configure BPSim window 'Execute' tab how the flow develops:



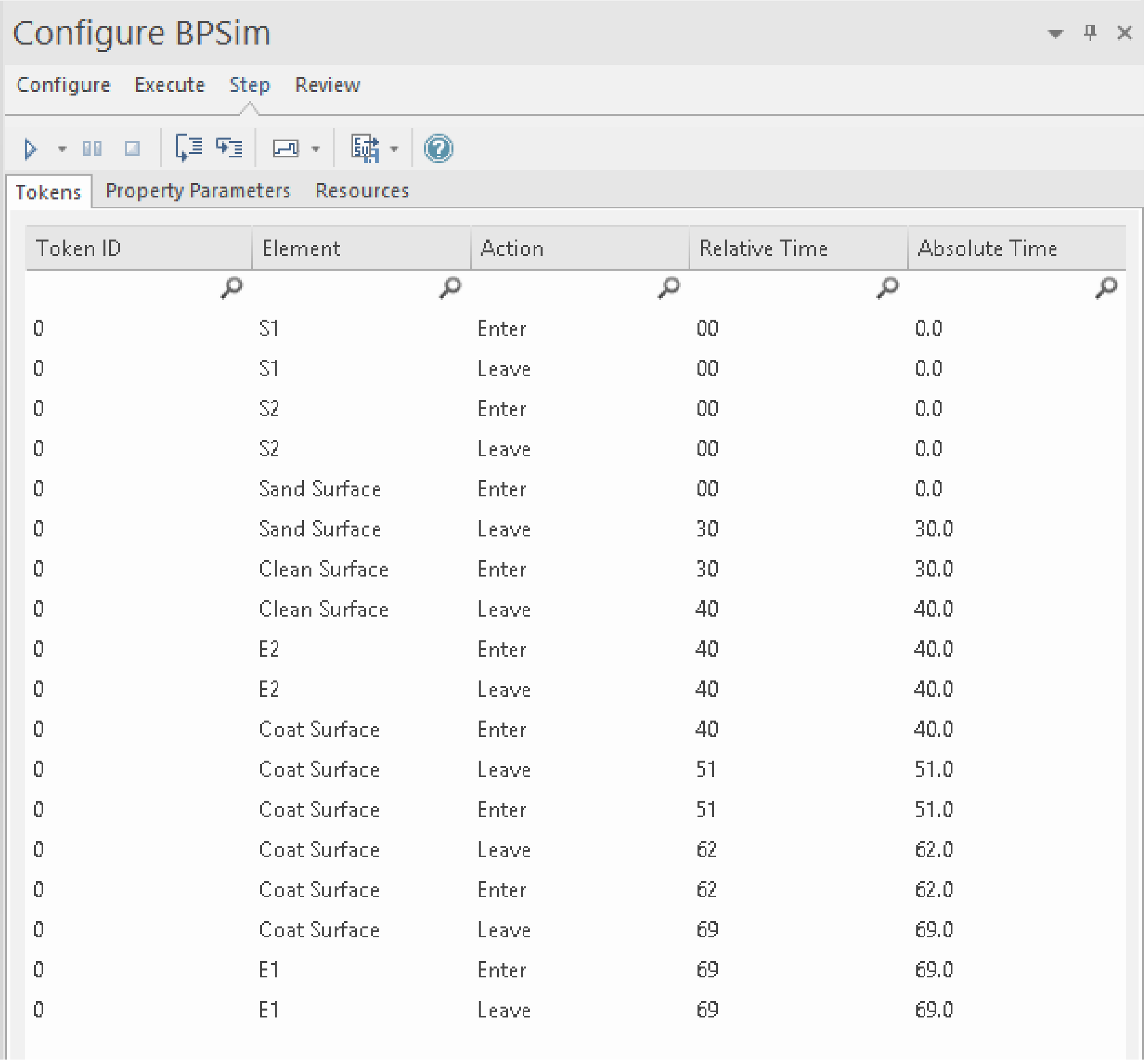
Element	Count
◢ Token Status	
<<StartEvent>>S1	1
<<StartEvent>>S2	1
<<Activity>>Sand Surface	1
<<Activity>>Clean Surface	1
<<EndEvent>>E2	1
<<GlobalTask>>Coat Surface(<<Activity>>Coat Surface 1st Round)	1
<<GlobalTask>>Coat Surface(<<Activity>>Coat Surface 2nd Round)	1
<<GlobalTask>>Coat Surface(<<Activity>>Coat Surface 3rd Round)	1
<<EndEvent>>E1	1

- When reaching the callProcessActivity, the called process is activated; so we have *S2 ~ E2*

- When reaching a callGlobalTaskActivity, the called Global Task is activated - the notation reads: *GlobalTask name (called activity name)*; the global *Coat Surface* was called three times:
  - *Coat Surface(Coat Surface 1st Round)*
  - *Coat Surface(Coat Surface 2nd Round)*
  - *Coat Surface(Coat Surface 3rd Round)*

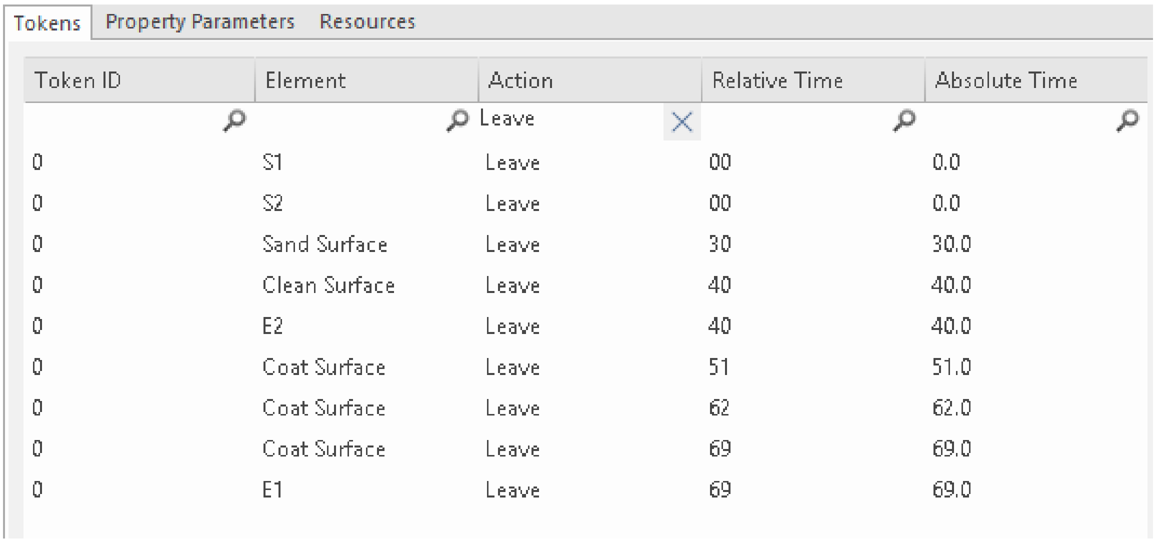
Time Analysis

Click on the 'Steps' tab of the Configure BPSim window, and on the Tokens tab, which resembles this illustration:



Token ID	Element	Action	Relative Time	Absolute Time
0	S1	Enter	00	0.0
0	S1	Leave	00	0.0
0	S2	Enter	00	0.0
0	S2	Leave	00	0.0
0	Sand Surface	Enter	00	0.0
0	Sand Surface	Leave	30	30.0
0	Clean Surface	Enter	30	30.0
0	Clean Surface	Leave	40	40.0
0	E2	Enter	40	40.0
0	E2	Leave	40	40.0
0	Coat Surface	Enter	40	40.0
0	Coat Surface	Leave	51	51.0
0	Coat Surface	Enter	51	51.0
0	Coat Surface	Leave	62	62.0
0	Coat Surface	Enter	62	62.0
0	Coat Surface	Leave	69	69.0
0	E1	Enter	69	69.0
0	E1	Leave	69	69.0

You can check timing in the list as it is, but to make the process easier type 'Leave' in the Filter bar field of the 'Action' column to show only the records containing that text string in that column.



Token ID	Element	Action	Relative Time	Absolute Time
		Leave		
0	S1	Leave	00	0.0
0	S2	Leave	00	0.0
0	Sand Surface	Leave	30	30.0
0	Clean Surface	Leave	40	40.0
0	E2	Leave	40	40.0
0	Coat Surface	Leave	51	51.0
0	Coat Surface	Leave	62	62.0
0	Coat Surface	Leave	69	69.0
0	E1	Leave	69	69.0

The report displays as shown, and we can make this analysis:

- The Call Activity *Call Prepare Surface* took 40 minutes, composed of *Sand Surface* (30 minutes) and *Clean Surface* (10 minutes), as defined
- *Coat Surface (1st Round)* took 11 minutes; *Coat Surface (2nd Round)* took 11 minutes; *Coat Surface (3rd Round)* took 7 minutes - the figures 11, 11, 7 are randomly generated by the Poisson(10) distribution; what is important here is that each call instance of the Global Task has its own values
- *Coat Surface* has a total time collected from all instances: 11 + 11 + 7 = 29
- The Sum Processing Time for the *Paint Wall Process* is 69 minutes, composed of the four Call Activities: 40 + 11 + 11 + 7 = 69

Customized Simulation

We can configure a 'Result Request' on BPMN elements to customize the simulation report so that we only report on the parameters we are interested in.

Configure Result Request

1. On the 'Paint Wall Process' diagram click on the Activity *Coat Surface 1st Round*.
2. In the Configure BPSim window, click on the *New Parameter* drop-down arrow and create a Time parameter called 'ProcessingTime'.
3. Click on the min max toolbar icon. The 'Result Request' column displays to the right of the 'Parameter' column; click on the drop-down arrow and select the 'sum' checkbox. Click on the OK button.
4. In the 'Values' field type '1'.
5. Click on the 💾 icon.
6. Repeat steps 1 to 5 for the Activities *Call Prepare Surface*, *Coat Surface 2nd Round*, *Coat Surface 3rd Round*

- Expand the 'Business Process' group and repeat these steps for *Paint Wall Process*

Run Simulation

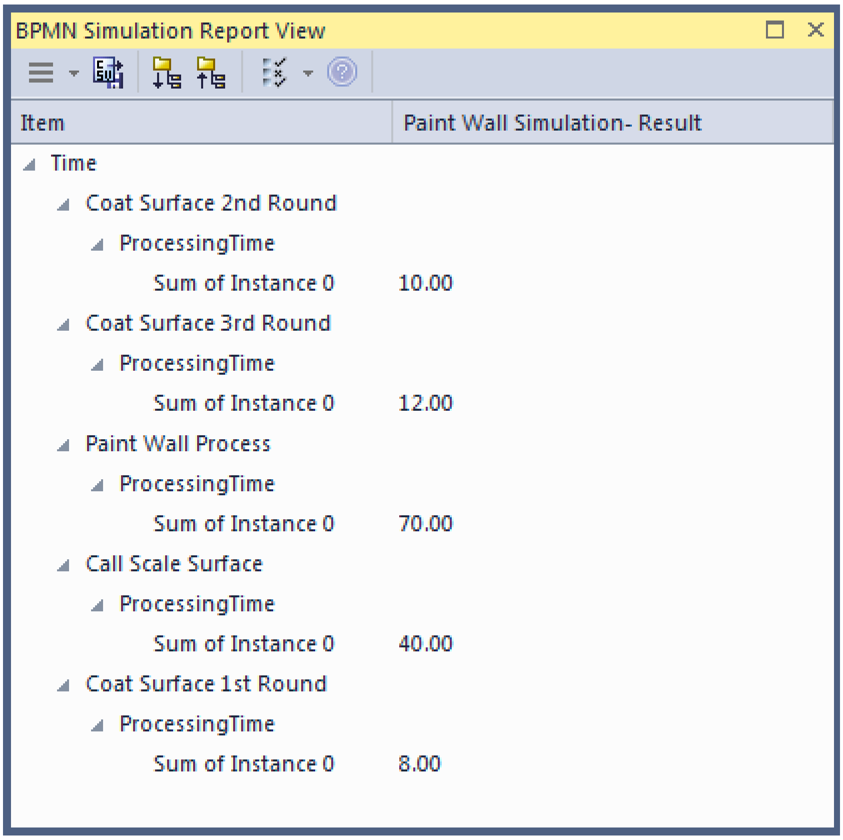
- On the 'Configure BPSim' dialog toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays
- Click on the Run button drop-down arrow and select, in this case, 'Customized Simulation'

Flow Analysis

The Flow Analysis is exactly the same as for a Standard Simulation.

Time Analysis

On the 'BPSim Simulation Controller' dialog toolbar, click on the button; the 'BPMN Simulation Report View' displays.



Item	Paint Wall Simulation- Result
Time	
Coat Surface 2nd Round	
ProcessingTime	
Sum of Instance 0	10.00
Coat Surface 3rd Round	
ProcessingTime	
Sum of Instance 0	12.00
Paint Wall Process	
ProcessingTime	
Sum of Instance 0	70.00
Call Scale Surface	
ProcessingTime	
Sum of Instance 0	40.00
Coat Surface 1st Round	
ProcessingTime	
Sum of Instance 0	8.00

The Time Analysis is the same as for a Standard Simulation; however, the report only contains the 'sum' results we requested.

Note: Currently, in the Time Analysis, we can not request ProcessingTime either on the called process itself or on the Activities contained by the called process. If you have this requirement, use the Standard Simulation.

BPSim Cost Parameters

BPSim 1.0 provides the means to set cost parameters and receive cost statistics from process simulation experiments. BPSim provides a framework to determine **variable** costs according to two parameters, both related to the level of activity performed in the simulated process. These parameters are:

- Completion Cost ('Fixed Cost' in the BPSim specification) - The cost incurred whenever an operation is completed; this cost can be related to Task, Process, Sub-Process, Call Activity or Resource elements
- Time Cost ('Unit Cost' in the BPSim specification) - The cost incurred whenever a task, process, sub-process, call activity or resource is busy for a period of time

The cost parameters are supported on Activities, resources and processes. On:

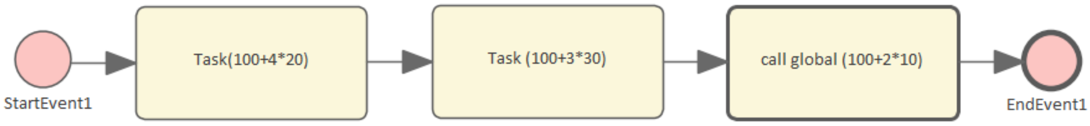
- An Activity, Completion Cost and Time Cost (Unit Cost * Time) are both incurred whenever an Activity finishes
- A Resource, Completion cost and Time Cost are both incurred whenever each involved resource finishes an Activity
- A Process, Completion cost and Time Cost are both incurred whenever a process finishes

Those costs that are known without the need for simulation - for example, overall labor employment costs - are not supported by BPSim.

The configuration and simulation of cost parameters are demonstrated by two examples:

- Set Cost Parameters on Activity
- Set Cost Parameters on Resource

Set Cost Parameters on Activity



Create the BPMN Model (Activities)

1. In the Browser window, create a *StartEvent1*, a *GlobalTask1*, two AbstractTasks, and an *EndEvent1*.
2. Ctrl+drag the elements from the Browser window onto a diagram, pasting *GlobalTask1* as an Invocation (Call Activity) called *call global (100+2*10)*.
3. Give the elements names and connect them with Sequence Flows; the two AbstractTasks should be called:
   - *Task (100+3*30)* and
   - *Task (100+4*20)*.

BPSim Configuration

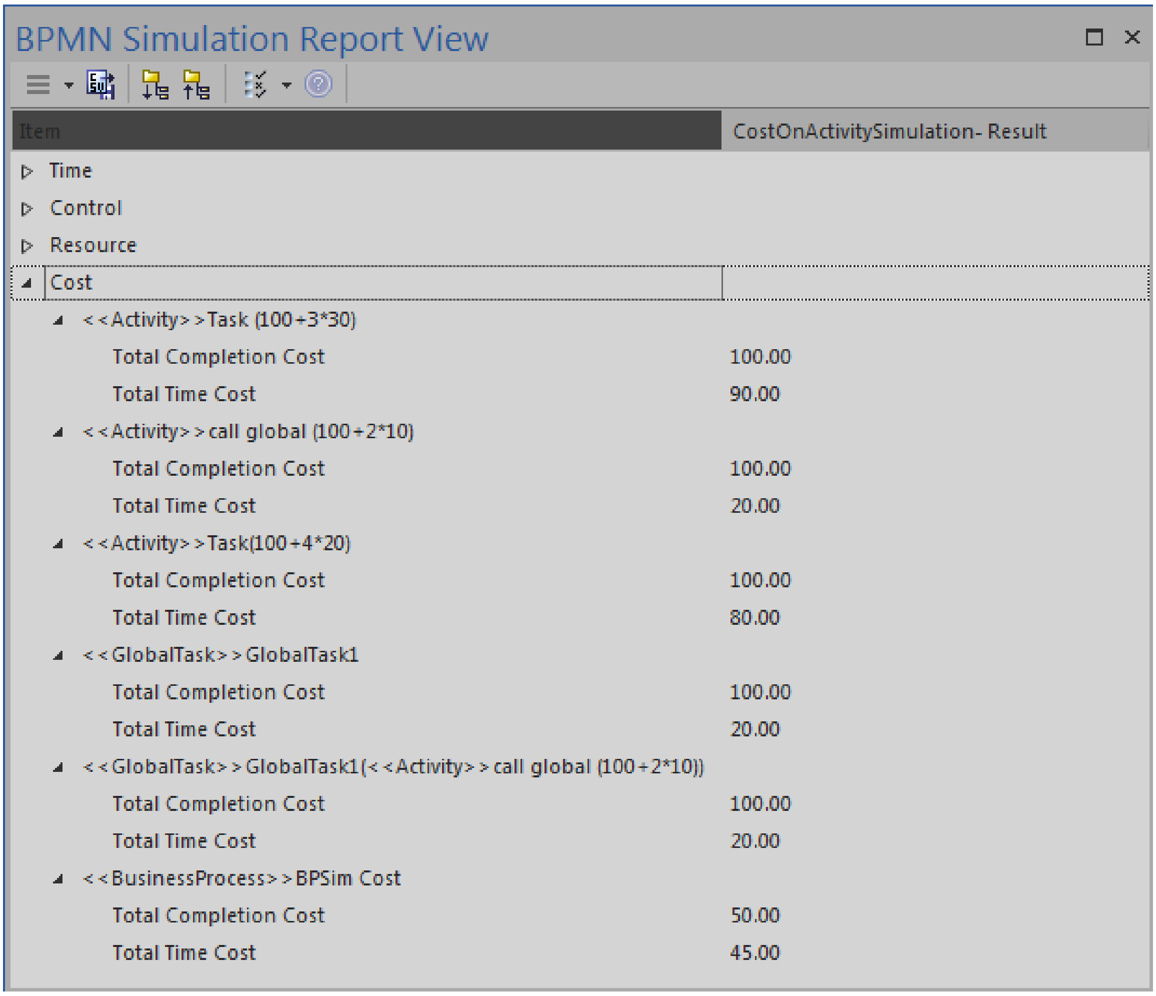
Create a Business Process Simulation configuration Artifact in the diagram, right-click on it and select the 'Configure BPSim' option. Set the configuration to link to the Package containing the BPMN model elements and configure these BPSim parameters as indicated.

Parameter	Settings
Scenario Parameters	1. Click on the BPSim Configuration Artifact and, for the Scenario Parameter 'Time Unit', click on the 'Value' drop-down arrow and select 'hours'. 2. In the 'Value' field for the 'Duration' parameter, set the value to '0001 00:00:00' (1 day). This Time Unit is used to calculate the Time Cost (Time cost = Unit Cost * Time), so make sure the Unit Cost is based on the correct Time Unit.
Control Parameters	1. On the diagram, click on *StartEvent1*. 2. Click on the *New Parameter* drop-down arrow and select 'Control'. 3. In the 'Parameter' field click on the drop-down arrow and select 'TriggerCount'. 4. In the 'Value' field type '1'.
Time Parameters	1. On the diagram click on *Task(100+4*20)*. 2. Click on the *New Parameter* drop-down arrow and select 'Time'. 3. Click on the 'Parameter' drop-down arrow and select 'ProcessingTime'. 4. In the 'Value' field set the value to '000:000:000 004:00:00' (4 hours). 5. Click on *Task(100+3*30)* on the diagram and repeat steps 2, 3 and 4, setting the 'Value' field to '000:000:000 003:00:00' (3 hours). 6. Click on *GlobalTask1* on the diagram and repeat steps 2, 3 and 4, setting the 'Value' field to '000:000:000 002:00:00' (2 hours).
Cost Parameters	1. On the diagram, click on *Task(100+4*20)*. 2. Click on the *New Parameter* drop-down arrow and select 'Cost'. 3. In the 'Parameter field click on the drop-down arrow and select, in turn:

	- 'FixedCost', then in the 'Value' field click on the ... button, select the 'Constant' tab and 'Floating', and in the 'Constant Floating' field type '100'; click on the OK button - 'UnitCost' - do the same, setting the 'Constant Floating' field to '20'. 4. On the diagram, click on *Task(100+3*30)* and repeat Steps 2 and 3, setting: - 'FixedCost' to '100 - 'UnitCost' to '30'. 5. On the diagram, click on *GlobalTask1* and repeat Steps 2 and 3, setting: - 'FixedCost' to '100 - 'UnitCost' to '10'. 6. On the diagram, click on *BPSim Cost* and repeat Steps 2 and 3, setting: - 'FixedCost' to '50' - 'UnitCost' to '5'.

Simulation

1. On the 'Configure BPSim' dialog, click on the 'Execute' tab.
2. Click on the S button'.
3. When the simulation is complete, click on the 'Review' tab, and on the 'Standard Results Report' tab.
4. Filter the report by clicking on the ☰ ▾ button and selecting the 'Show only Non-Empty Items' option.

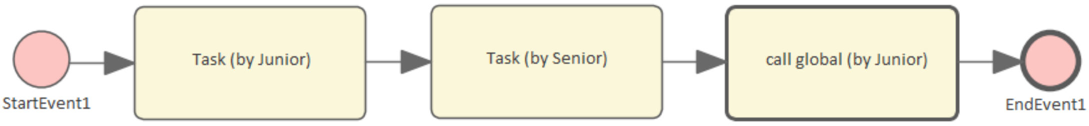


Item	CostOnActivitySimulation- Result
▷ Time	
▷ Control	
▷ Resource	
◢ Cost	
◢ <<Activity>>Task (100+3*30)	
Total Completion Cost	100.00
Total Time Cost	90.00
◢ <<Activity>>call global (100+2*10)	
Total Completion Cost	100.00
Total Time Cost	20.00
◢ <<Activity>>Task(100+4*20)	
Total Completion Cost	100.00
Total Time Cost	80.00
◢ <<GlobalTask>>GlobalTask1	
Total Completion Cost	100.00
Total Time Cost	20.00
◢ <<GlobalTask>>GlobalTask1(<<Activity>>call global (100+2*10))	
Total Completion Cost	100.00
Total Time Cost	20.00
◢ <<BusinessProcess>>BPSim Cost	
Total Completion Cost	50.00
Total Time Cost	45.00

Analysis

Activity	Analysis
Task(100+4*20)	• Total Completion Cost is 100, matching the FixedCost (100) setting in BPSim • Total Time Cost is 80, calculated as ProcessingTime (4 hours) * UnitCost (20/hour)
Task(100+3*30)	• Total Completion Cost is 100, matching the FixedCost (100) setting in BPSim • Total Time Cost is 90, calculated as ProcessingTime (3 hours) * UnitCost (30/hour)
call global (100+2*10)	• Total Completion Cost is 100, matching the FixedCost (100) on *GlobalTask1* setting in BPSim • Total Time Cost is 20, calculated as ProcessingTime (2 hours) * UnitCost (10/hour) on *GlobalTask1*
BPSim Cost process	• Total Completion Cost is 50, matching the FixedCost (50) setting in BPSim • Total Time Cost is 45, calculated as Total ProcessingTime of all tasks (4 + 3 + 2 = 9 hours) * UnitCost (5/hour)

Set Cost Parameters on Resource



Create the BPMN Model (Resources)

1. In the Browser window create a *StartEvent1*, a *GlobalTask1*, two abstractTasks called *Task (by Junior)* and *Task (by Senior)*, and an *EndEvent1.*
2. Ctrl+drag the elements from the Browser window onto a diagram, pasting *GlobalTask1* as an Invocation (Call Activity) named *call global (by Junior).*
3. Connect the elements with Sequence Flows.
4. Create two BPMN2.0 Resource elements: *Junior Developer* and *Senior Developer.*

BPSim Configuration

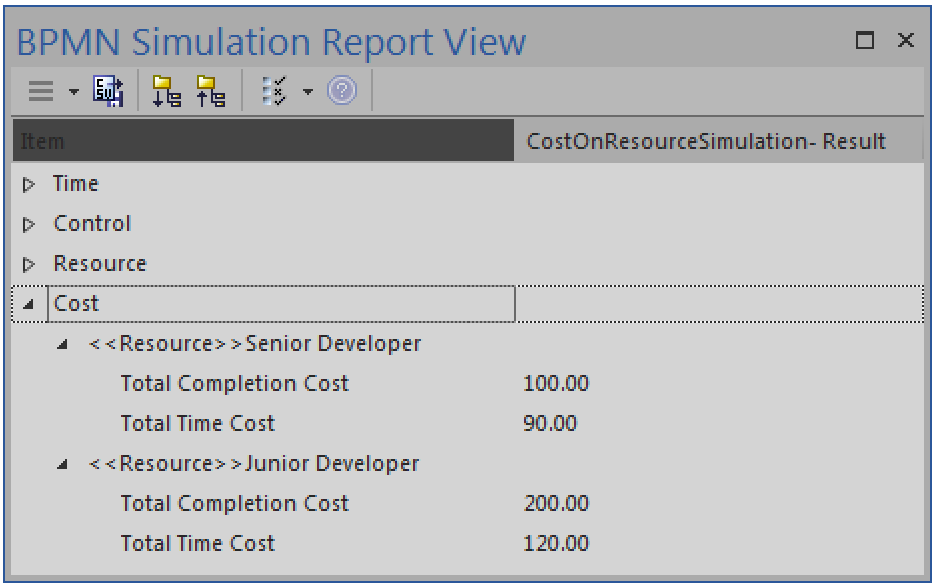
Create a Business Process Simulation configuration Artifact in the diagram, right-click on it and select the 'Configure BPSim' option, then set the configuration to link to the Package containing the BPMN model elements and configure these BPSim parameters as indicated.

Parameter	Setting
Scenario Parameters	1. Click on the BPSim Configuration Artifact and, for the Scenario Parameter 'Time Unit', click on the 'Value' drop-down arrow and select 'hours'. 2. In the 'Value' field for the 'Duration' parameter, set the value to '0001 00:00:00' (1 day). This Time Unit is used to calculate the Time Cost (Time cost = Unit Cost * Time), so make sure the Unit Cost is based on the correct Time Unit.
Control Parameters	1. On the diagram, click on *StartEvent1*. 2. Click on the *New Parameter* drop-down arrow and select 'Control'. 3. In the 'Parameter' field click on the drop-down arrow and select 'TriggerCount'. 4. In the 'Value' field type '1'.
Time Parameters	1. On the diagram click on *Task(by Junior)*. 2. Click on the *New Parameter* drop-down arrow and select 'Time'. 3. Click on the 'Parameter' drop-down arrow and select 'ProcessingTime'. 4. In the 'Value' field set the value to '000:000:000 004:00:00' (4 hours). 5. Click on *Task (by Senior)* on the diagram and repeat steps 2, 3 and 4, setting the 'Value' field to '000:000:000 003:00:00' (3 hours). 6. Click on *GlobalTask1* on the diagram and repeat steps 2, 3 and 4, setting the 'Value' field to '000:000:000 002:00:00' (2 hours).
Resource Parameters	1. On the diagram, click on the *Junior Developer* Resource. 2. Click on the *New Parameter* drop-down arrow and select 'Resource'.

	3. Click on the 'Parameter' drop-down arrow and select 'Selection'. 4. In the 'Values' field, click on the ... button to open the 'Edit Resource Selection' dialog. 5. Click on 'Junior Developer', and on the Add Selection By Resource(s) button to move the selection to the 'Resource or Role' panel. 6. The 'Quantity Required' column defaults to '1'; overtype this value with '10'. 7. Click on the AND radio button to set the logical relationship; the final expression for Resource selection is composed and shown in the text field. 8. Click on the OK button to return to the Configure BPSim window, where the expression is shown in the 'Values' field. 9. Click on the *Senior Developer* Resource and repeat steps 2 through to 8, typing '5' in the 'Quantity Required' field.
Cost Parameters	1. On the diagram click on *Junior Developer*. 2. Click on the *New Parameter* drop-down arrow and select 'Resource'. 3. Click on the 'Parameter' drop-down arrow and select, in turn: - 'FixedCost', then in the 'Value' field click on the ... button, select the 'Constant' tab and 'Floating', then in the 'Constant Floating' field type '100' and in the 'CurrencyUnit' field type 'AUD'; click on the OK button - 'UnitCost' - do the same, setting the 'Constant Floating' field to '20'. 4. On the diagram click on *Senior Developer* and repeat steps 2 and 3, setting: - 'FixedCost' to '100' - 'UnitCost' to '30'.

Simulation

1. On the 'Configure BPSim' dialog, click on the 'Execute' tab.
2. Click on the button'.
3. When the simulation is complete, click on the 'Review' tab, and on the 'Standard Results Report' tab.
4. Filter the report by clicking on the button and selecting the 'Show only Non-Empty Items' option.



Item	CostOnResourceSimulation- Result
▷ Time	
▷ Control	
▷ Resource	
◢ Cost	
◢ <<Resource>>Senior Developer	
Total Completion Cost	100.00
Total Time Cost	90.00
◢ <<Resource>>Junior Developer	
Total Completion Cost	200.00
Total Time Cost	120.00

Analysis

Resource	Results
Junior Developer	• Total Completion Cost is '200', calculated as FixedCost (100) * number of Activities involved (2) • Total Time Cost is '120', calculated as ProcessingTime (4 + 2 = 6 hours) * UnitCost (20/hour)
Senior Developer	• Total Completion Cost is '100', calculated as FixedCost (100) * number of Activities involved (1) • Total Time Cost is '90', calculated as ProcessingTime (3 hours) * UnitCost (30/hour)

Export a BPSim Configuration

When you have defined a BPSim Configuration in a model, you can export it to an XMI file to be imported into other projects. The BPMN 2.0 model on which the configuration is based is also exported with the configuration. The model is bound to the appropriate BPSim Configuration when you import the XMI file into another project.

Access

Context Menu	On a diagram or in the Browser window, right-click on the Business Process Simulation Artifact | Export BPSim Configuration
Other	Toolbar of Configure BPSim window | XML Export icon

Publish Model Package

The process of exporting a BPSim configuration and its model uses the 'Publish Model Package' dialog for publishing a model to an XMI file.

Option	Description
Package	Defaults to the name of the Package containing the Business Process Simulation Artifact.
Filename	Type in or browse for (click on the ... icon) the file path and XML filename into which to export the model.
XML Type	Select 'BPMN 2.0 XML'.
Export	Click on this button to export the configuration and BPMN 2.0 model. The export is complete when a confirmation message displays in the 'Progress' field.
Format XML Output	Defaults to selected; leave selected.
View XML	If you want to examine the exported XML, click on this button.

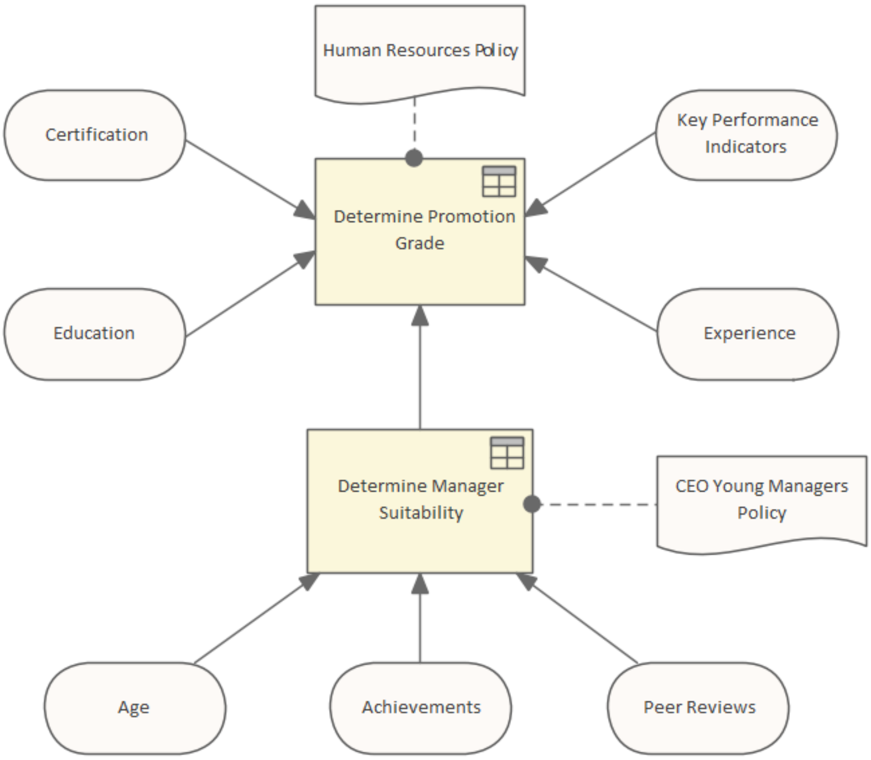
Notes

- To import the model from XMI into a new project, select the target Package in the new project and select the 'Publish > Model Exchange > Import > Native File' or 'Import > XMI File' ribbon option

Decision Model and Notation (DMN)

Create and Simulate Detailed Models of Enterprise Decisions

Organizations face increasingly difficult operating environments with fierce and often unpredictable competition from existing and new market players, changes in government and industry regulations and upheavals in the social fabric of their customer base. The decisions that an organization makes in this context are critical to the success of the organization and its ability to steer a safe path through these uncharted corporate waters. Using Enterprise Architects Decision Model and Notation (DMN) features you can not only model the decisions that your organization makes but you can also run simulations from these models to predict outcomes based on example data sets. The power of the language is that business people can readily understand and work with simple but expressive Decision Requirements diagrams that detail what the decisions are and what the inputs to the decisions are and what the expected outputs are. The rules can be documented in a number of ways including easy to define decision tables. Once completed these diagrams with accompanying input data examples can be simulated to show the results of the decisions.


Human Resources Policy
Certification
Key Performance Indicators
Determine Promotion Grade
Education
Experience
Determine Manager Suitability
CEO Young Managers Policy
Age
Achievements
Peer Reviews

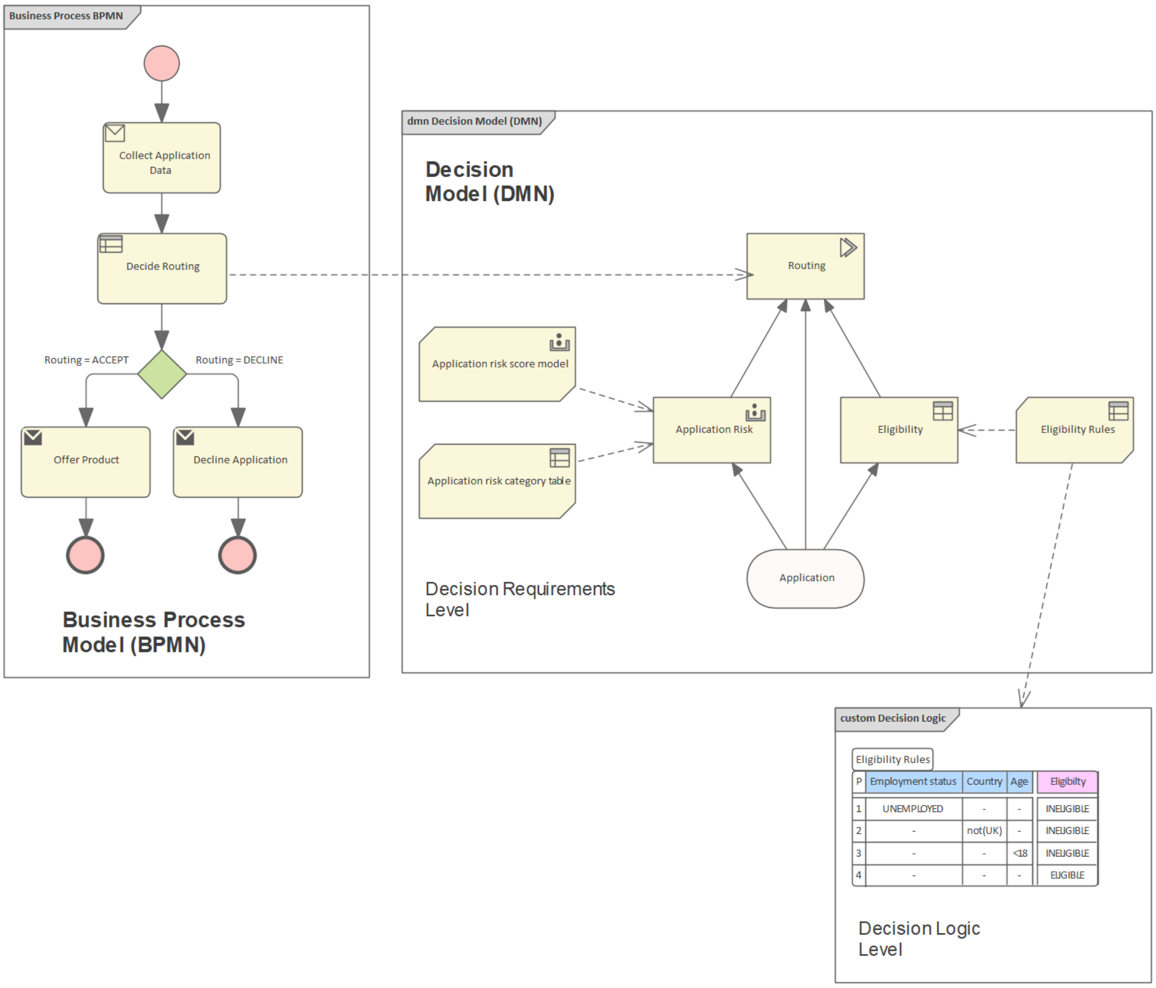
Decision Requirements diagram showing a Decision with a Business Knowledge Model and a number of inputs including another Decision.

Once these models have been defined, simulated and tested by the business, technologists and engineers can then refine these models and automatically generate software artifacts including programming code directly from the models reducing the possibility of errors of interpretation and reducing the time to implementation.

What is DMN?

DMN is intended to provide a bridge between business process models and decision logic models:

- Business process models will define tasks within business processes where decision-making is required to occur
- Decision Requirements Diagrams will define the decisions to be made in those tasks, their interrelationships, and their requirements for decision logic
- Decision logic will define the required decisions in sufficient detail to allow validation and/or automation



Eligibility Rules

P	Employment status	Country	Age	Eligibilty
1	UNEMPLOYED	-	-	INELIGIBLE
2	-	not(UK)	-	INELIGIBLE
3	-	-	<18	INELIGIBLE
4	-	-	-	ELIGIBLE

Taken together, Decision Requirements diagrams and decision logic allow you to build a complete Decision Model that complements a business process model by specifying - in detail - the decision-making carried out in process tasks.

DMN provides constructs spanning both decision requirements and decision logic modeling.

- For decision requirements modeling, it defines the concept of a Decision Requirements Graph (DRG) comprising a set of elements and their connection rules, and a corresponding notation: the Decision Requirements Diagram (DRD).
- For decision logic modeling it provides a language called FEEL for defining and assembling Decision Tables, calculations, if/then/else logic, simple data structures, and externally defined logic from Java and PMML into executable expressions with formally defined semantics.

Benefits of Using DMN in Enterprise Architect

Modeling decision-making processes using DMN allows you to record, specify and analyze complex decision processes as a system of interrelated decisions, business rules, data sets and knowledge sources. By doing so, you can decompose a highly complex decision making process into a network of supporting decisions and input data. This facilitates easier understanding of the overall process, supports refactoring of processes and simplifies the task of validating the process, by allowing you to easily validate the individual steps that make up the overall process.

When you build a Decision Model in Enterprise Architect using DMN, you can run simulations of the model to verify the correctness of the model. After you have verified your model, you can generate a DMN Module in Java, JavaScript, C++ or C#. The generated DMN Module can be used with the Enterprise Architect BPSim Execution Engine, Executable StateMachine, or within a separate software system that you are implementing.

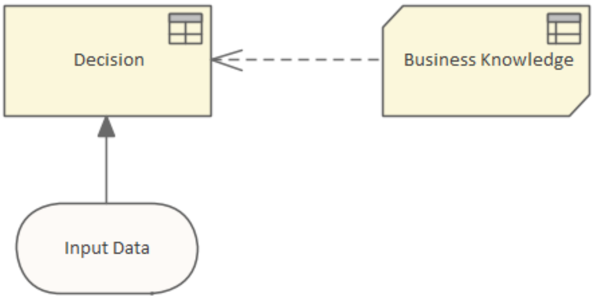
Enterprise Architect also provides a 'Test Module' facility, which is a preprocess for integrating DMN with BPMN. The aim is to produce BPMN2.0::DataObject elements, then use these to verify that a specified target decision is evaluated correctly with the DMN Module. You then configure BPSim by loading DataObjects and assigning DMN Module decisions to BPSim Properties.

This feature is available in the Unified and Ultimate Editions of Enterprise Architect, from Release 15.0.

Decision Requirements Graphs

The DMN decision requirement model consists of a Decision Requirements Graph (DRG) depicted in one or more Decision Requirements Diagrams (DRDs). The elements modeled are decisions, areas of business knowledge, sources of business knowledge, input data and decision services.

A DRG is a graph composed of elements connected by requirements, and is self-contained in the sense that all the modeled requirements for any Decision in the DRG (its immediate sources of information, knowledge and authority) are present in the same DRG. It is important to distinguish this complete definition of the DRG from a DRD presenting any particular view of it, which might be a partial or filtered display.



Getting Started

Decision Model and Notation (DMN) is a standard published and managed by the Object Management Group (OMG).

Portions of this topic have been used verbatim or are freely adapted from the DMN Specification, which is available on the OMG DMN web page (https://www.omg.org/spec/DMN). A full description of the DMN and its capabilities can be found on the OMG website.

The purpose of DMN is to provide the constructs that are needed to model decisions, so that organizational decision-making can be readily depicted in diagrams, accurately defined by business analysts, and (optionally) automated. It is also intended to facilitate the sharing and interchange of Decision Models between organizations.

Selecting the Perspective

Enterprise Architect partitions the tool's extensive features into Perspectives, which ensures that you can focus on a specific task and work with the tools you need without the distraction of other features. To work with the Decision Model and Notation features you first need to select this Perspective:

👁<perspective name> > Requirements > Decision Modeling

Setting the Perspective ensures that the Decision Model and Notation diagrams, their tool boxes and other features of the Perspective will be available by default.

Example Diagram

An example diagram provides a visual introduction to the topic and allows you to see some of the important elements and connectors that are created in specifying or describing the way decisions are modeled. The Decision Requirements diagram will introduce elements such as Decision Tables, Knowledge Sources, Input Date and more. Much of the power of Enterprise Architect relies in the ability to simulate or 'run' the decision models and to predict outcomes based on different data sets. This functionality will be described in later topics but starts with the creation of a Decision Requirements diagram.

Modeling with DMN

This topic introduces you to the most important elements that you need to create Decision models. This includes the creation of a Decision Requirements diagram which describes how decision are related and what inputs each decision has potentially including other decisions. You will learn about the mst important elements including: Decisions. Business Knowledge Models, Input Data Items Definitions, Data Set and Decision Services.

Code Generation and Test Module

This topic introduces you to the main concepts of the language including its structure, architecture and the elements and connectors that are used to create Decision Model and Notation (DMN) models. Understanding the intent and structure of the language will help analysts create meaningful and productive decision models.

Integrate Into BPSim for Simulation

Enterprise Architect allows Decision models to be run (simulated) which allows you to visualize the outcomes of decisions. In addition to this core facility you can also integrate the Decision models with BPSim which is a simulation engine for the simulation of BPMN diagrams. In this topic you will a number of different ways to integrate DMN with BPSIm.

Integrate Into UML Class Element

In this topic you will learn the process of integrating a DMN Model with a UML Class element. The DMN Module can be integrated with a UML Class element, so the code generated from that Class element can reuse the DMN Module and be well-structured

Importing DMN XML

This topic describes how to import a DMN XML file from a different Enterprise Architect repository or another DMN compliant tool. One of the promises of open standards is the ability to share models between different tools. Enterprise Architect often becomes the tool of choice for modeling because of the breadth of features and standards it supports. This allows Decision model to be related to Strategy, Requirements, Business Processes, Software Implementation elements and more.

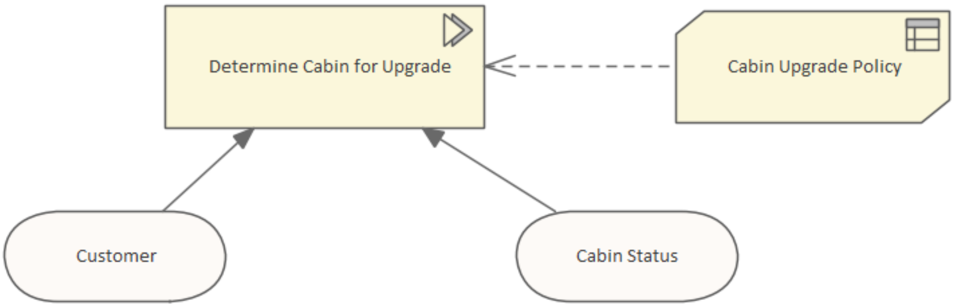
More Information

This section provides useful links to other topics and resources that you might find useful when working with the Decision Model and Notation tool features.

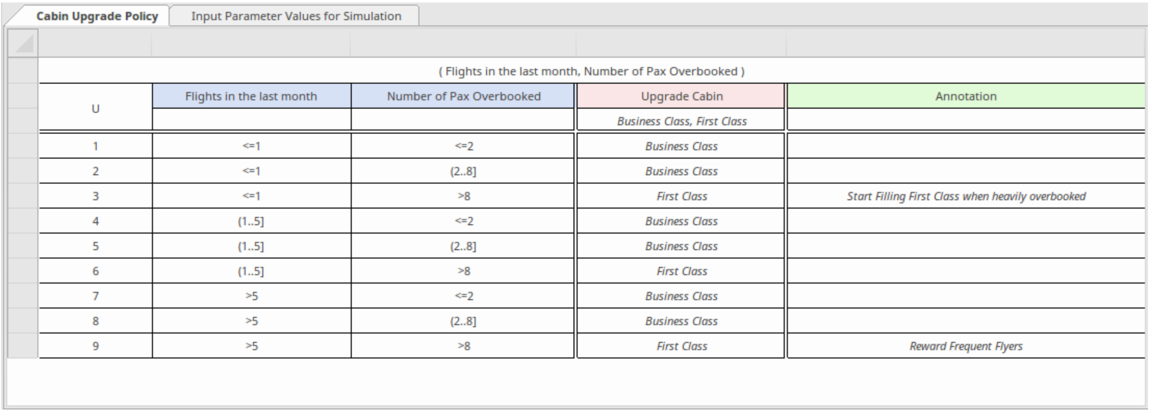
Example Diagram

Imagine you are an Airline reservation officer working at the check-in counter for a busy domestic airline. Getting the aircraft off on-time is critical as delays can result in fees applied by the airport controllers, needing to fly at a lower altitude increasing the cost of fuel, and other penalties.

A message from the supervisor appears on your screen saying that the economy cabin is overbooked; you will need to upgrade some passengers to Business or First Class — but which passengers should be chosen and which cabin should they be upgraded to? A decision needs to be made but what factors should be considered? This can be recorded in a Decision Model using a Decision Requirements diagram.



This is helpful but the busy check-in officer would still need to weigh up all the factors and make an unbiased decision. Should a disgruntled passenger be given priority over a Gold level frequent flyer, or should the fact that a particular passenger is connecting to an international flight take precedence. These 'rules' can all be recorded in a Decision Table, making it clear which passengers should get an upgrade and to which cabin: Business or First Class. This will make it much easier to make the decision and the rules can be formulated, agreed upon and checked for consistency back at head office. In this example we have kept it simple and used two factors: firstly the number of flights the passenger has made in the last month and secondly how overbooked the cabin is.



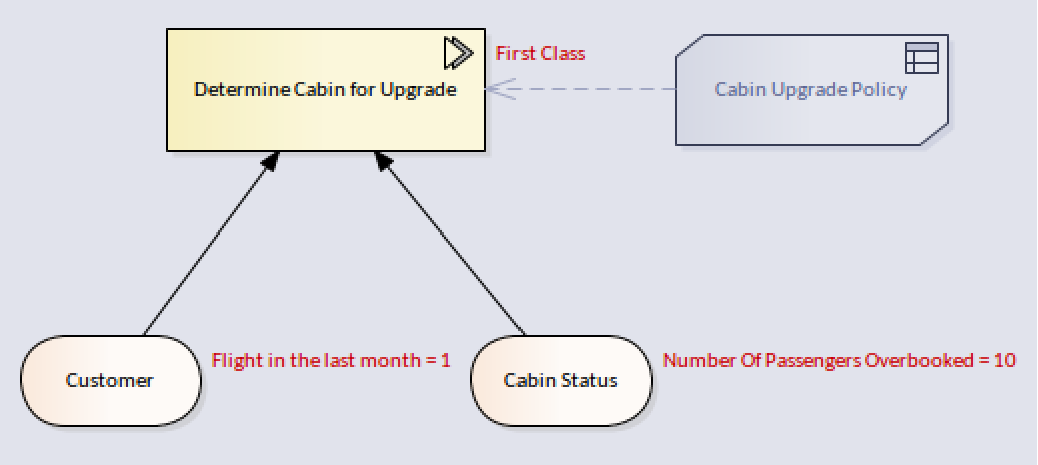
(Flights in the last month, Number of Pax Overbooked)

U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin	Annotation
			Business Class, First Class	
1	<=1	<=2	*Business Class*	
2	<=1	(2..8]	*Business Class*	
3	<=1	>8	*First Class*	*Start Filling First Class when heavily overbooked*
4	(1..5]	<=2	*Business Class*	
5	(1..5]	(2..8]	*Business Class*	
6	(1..5]	>8	*First Class*	
7	>5	<=2	*Business Class*	
8	>5	(2..8]	*Business Class*	
9	>5	>8	*First Class*	*Reward Frequent Flyers*

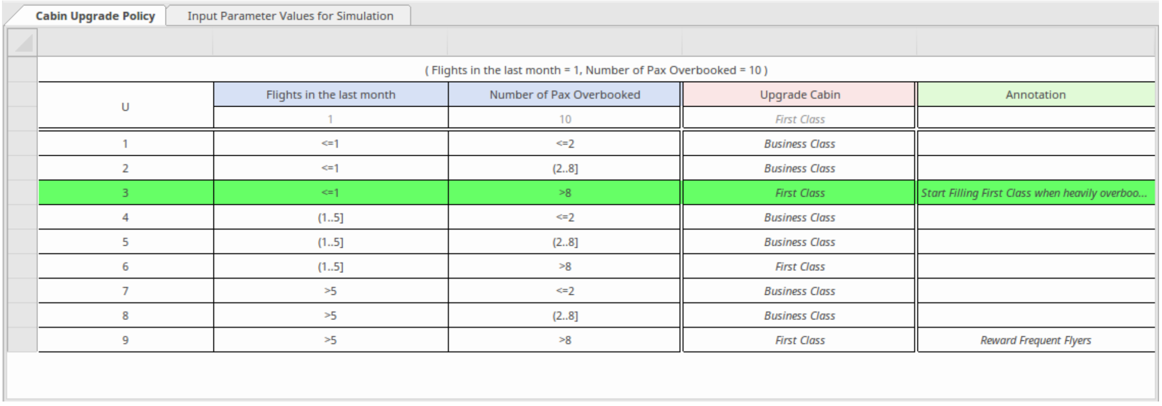
The table is divided into columns and rows. There are three types of column: inputs that are required to make the decision, outputs that are the result of applying the rules, and annotations.

This is again very helpful but still requires the busy check-in officer to be able to source all the required information required to find the right row in the Decision Table. Even if all this information were available, a wrong decision could still result from human error in selecting the wrong row in the table.

Fortunately the Decision Models can be automated and generated to programming code that can be executed by an application. So our busy check-in officer would not need to do anything or make any decisions; as he or she was checking in the passengers, if a particular passenger was entitled to an upgrade it would be visible on the computer screen. In the next diagram the model has been simulated so that the business and technical staff can agree that the model has been defined correctly. Any number of user-defined data sets can be used to test the model before generating the programming code that will run in the check-in system and display the result to the end user.


Determine Cabin for Upgrade
First Class
Cabin Upgrade Policy
Customer
Flight in the last month = 1
Cabin Status
Number Of Passengers Overbooked = 10

When developing the models a business or technical user can step through the simulation and the system will show that user which row in the Decision Table was fired to determine the output. This is very useful in models that are made up of multiple decisions.



(Flights in the last month = 1, Number of Pax Overbooked = 10)

U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin	Annotation
	1	10	*First Class*	
1	<=1	<=2	*Business Class*	
2	<=1	(2..8]	*Business Class*	
3	<=1	>8	*First Class*	*Start Filling First Class when heavily overboo...*
4	(1..5]	<=2	*Business Class*	
5	(1..5]	(2..8]	*Business Class*	
6	(1..5]	>8	*First Class*	
7	>5	<=2	*Business Class*	
8	>5	(2..8]	*Business Class*	
9	>5	>8	*First Class*	*Reward Frequent Flyers*

It is common for the rules that govern the upgrade decision to change. For example, the Marketing Department might decide they want to reward passengers that travel on long-haul flights. The Decision Requirements diagram can be altered to include the new input, the Decision Table modified, and the programming code regenerated. Once the changes have been pushed through to the airport systems, the right passengers will be automatically upgraded. The check-in officer could still view the Decision Tables during a training and briefing session to understand the rules.

Creating a Decision Model

In the model we described in *An Example of Decision Modeling*, we showed how a decision can be modeled using a Decision Table, in which a decision result is determined by finding a row in the table where the input values in the table match the input values under consideration, giving a particular output result.

We will now look at how such a model can be created in Enterprise Architect, by stepping through the process of creating the decision model for the Airline Cabin Upgrade example.

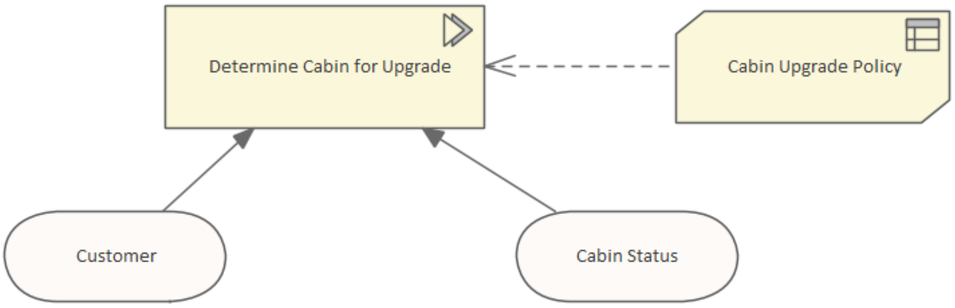
There are a number of model elements involved in this example, such as Input Data elements, Item Definitions that are used to describe the Input Data (defining the data types), a Decision element and also a Business Knowledge Model element that holds the Decision Table definition.

Create a Decision Requirements Diagram

These steps will guide you through the creation of a simple Decision Requirements Diagram (DRD). In this example, we will create the model from scratch, rather than using a pattern from the Model Wizard.

Step	Description
1	Select the perspective 'Requirements | Decision Modeling'. (The Start Page 'Create from Pattern' tab (Model Wizard) is displayed, but we will not use it for this example.)
2	Create a new DMN diagram. Name it 'Airline Cabin Upgrade'.
3	Using the diagram toolbox, place a Decision element on the diagram. Choose 'Invocation' as the type - we will use this element to 'invoke' a decision from a Business Knowledge Model element. Name the element 'Determine Cabin for Upgrade'.
4	Place an InputData element on the diagram. Name this element 'Customer'.
5	Place another InputData element on the diagram. Name this element 'Cabin Status'.
6	Place a Business Knowledge Model element on the diagram. Choose the type 'Decision Table'. Name this element 'Cabin Upgrade Policy'.
7	Draw an 'Information Requirement' connector **from** the decision 'Determine Cabin for Upgrade' **to** the input data 'Customer'.
8	Draw an 'Information Requirement' connector **from** the decision 'Determine Cabin for Upgrade' **to** the input data 'Cabin Status'.
9	Draw a 'Knowledge Requirement' connector **from** the decision 'Determine Cabin for Upgrade' **to** the BKM 'Cabin Upgrade Policy'.

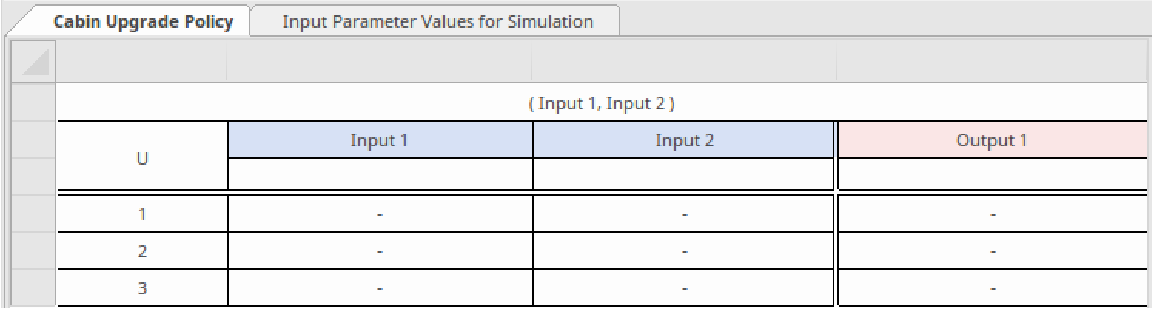
At this stage, we should have a simple DRD, that resembles this:



We can now specify the details for each of the elements making up this model.

Define the Decision Table

By double-clicking on the Business Knowledge Model element 'Cabin Upgrade Policy', the 'DMN Expression' window is displayed, showing an empty Decision Table. This is where we will define the rules of our cabin upgrade policy.



(Input 1, Input 2)			
U	Input 1	Input 2	Output 1
1	-	-	-
2	-	-	-
3	-	-	-

By default, new Decision Tables are created with two input columns and one output column, a header row and three empty rules rows.

The left-most column in the table displays the Hit Policy and also numbers the rules. By default, the Hit Policy is 'U' for 'Unique'. This is the policy that we will use for our example, so you do not need to change this column heading.

For more information on Hit Policies, refer to the *Decision Table Hit Policy* Help topic.

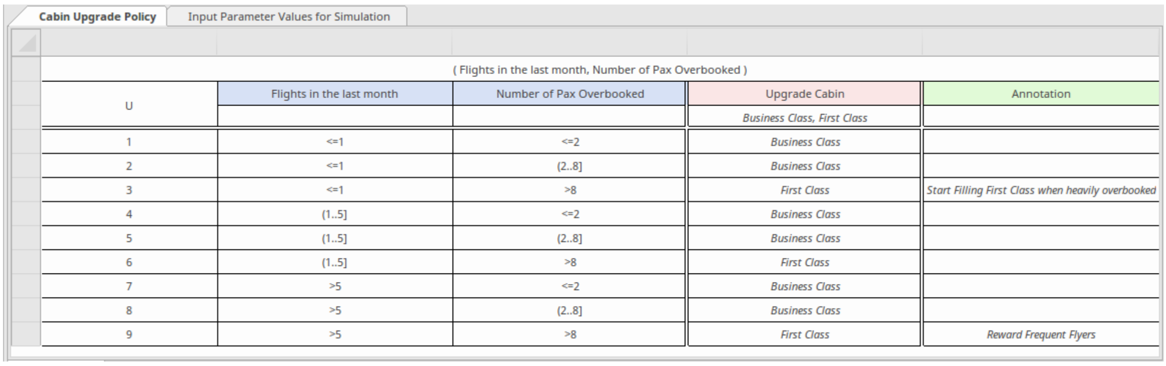
Name and Define Types for Decision Table Inputs and Outputs

Step	Description
1	On the toolbar of the 'DMN Expression' window, click on the 'Edit Parameters' button, . The 'Edit Parameters' dialog displays.
2	Replace the parameter name 'Input 1' with 'Num of Pax Overbooked'. If necessary, click on the 'Type' drop-down arrow and set the type of this parameter to 'number'.
3	Replace the parameter name 'Input 2' with 'Num of Flights in Last Month by Pass'. Set the type of this parameter to 'number' as well. Close the 'Edit Parameters' dialog.

4	Edit the input expression that will be evaluated for column 1. Select the header cell (containing the text 'Input 1') then click again or press F2 to enter 'Edit' mode. Select all of the cell text, then press the Spacebar. The list of input parameters is displayed. Click on 'Num of Pax Overbooked', then press 'Enter'. The *expression* for column 1 is set to 'Num of Pax Overbooked'. **Note**: The input expressions evaluated for each column typically just use the corresponding input parameter; however, you *can* use a complex expression.
5	Right-click on the column 1 expression and check that its data type is set to 'number'.
6	Edit the input expression that will be evaluated for column 2. Select all of the text, then press the Spacebar. The list of input parameters is displayed. Choose 'Num of Flights in Last Month for Pass', then press 'Enter'. The *expression* for column 2 is set to 'Num of Flights in Last Month for Pass'.
7	Right-click on the column 2 expression and set its data type to 'number'.
8	Edit the name of the decision table output. Replace 'Output 1' with 'Upgrade Cabin', then press 'Enter'.
9	Set the data type of the decision output. Right-click on the output column header and choose 'string'.
10	Set the allowable values for the decision output. In the cell directly beneath the output column header (but above row 1), define the allowable values for output. Enter 'Business Class, First Class'. **Note**: There is no need for quote marks around the values, as the data type has been specified as 'string'.

Define the Rules of the Decision Table

Enter values into the table cells to match this image.



(Flights in the last month, Number of Pax Overbooked)

U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin	Annotation
			Business Class, First Class	
1	<=1	<=2	*Business Class*	
2	<=1	(2..8]	*Business Class*	
3	<=1	>8	*First Class*	*Start Filling First Class when heavily overbooked*
4	(1..5]	<=2	*Business Class*	
5	(1..5]	(2..8]	*Business Class*	
6	(1..5]	>8	*First Class*	
7	>5	<=2	*Business Class*	
8	>5	(2..8]	*Business Class*	
9	>5	>8	*First Class*	*Reward Frequent Flyers*

Click on a cell to select it, and click again to edit it.

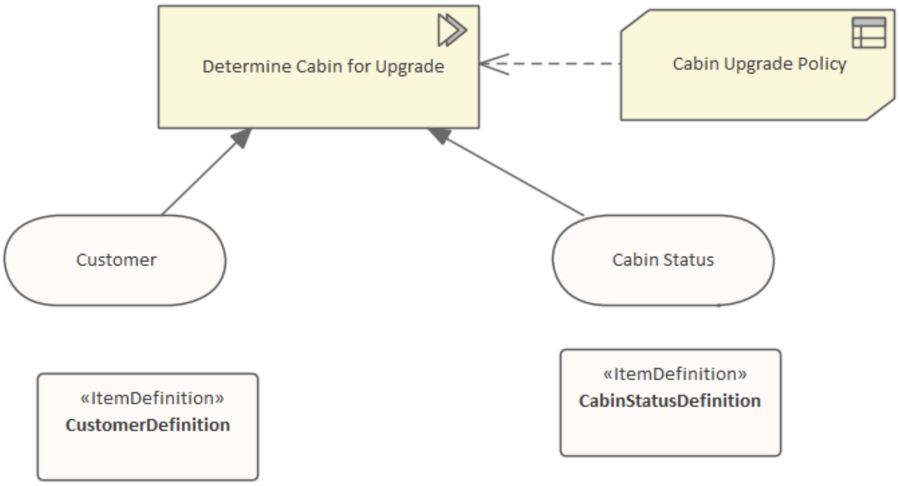
You can copy and paste existing rules by selecting the rows to copy (Shift+click adds to the selection), right-click and choose 'Copy', then right-click and choose 'Append'.

Once you have finished editing the rules, click on the Save button 💾.

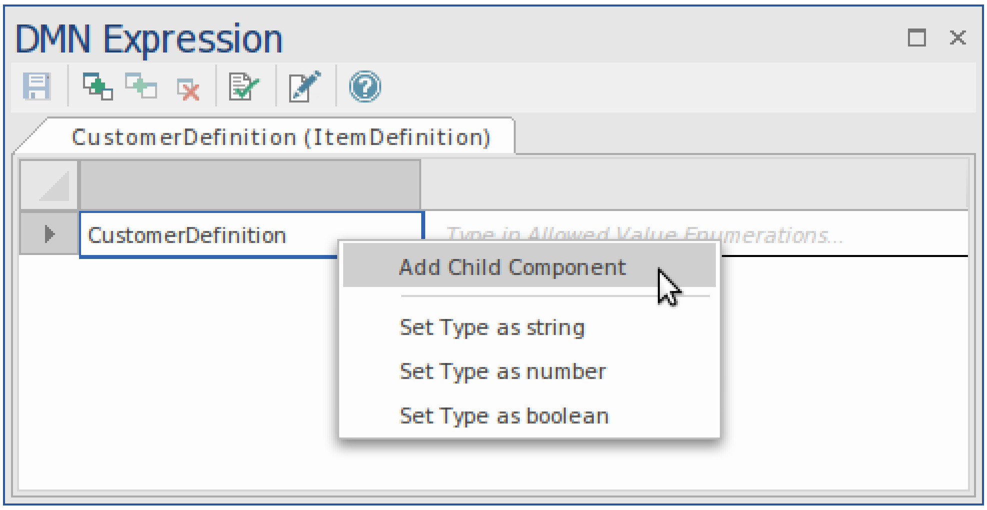
Finally, click the Validate button ✓, to check for errors in the table of rules.

Create ItemDefinition Elements

Add two ItemDefinition elements to the diagram, one for each of the InputData elements. Name one element 'CustomerDefinition' and the other 'CabinStatusDefinition'.



Double-click the ItemDefinition named 'CustomerDefinition' to edit the definition. The DMN Expression window is displayed.

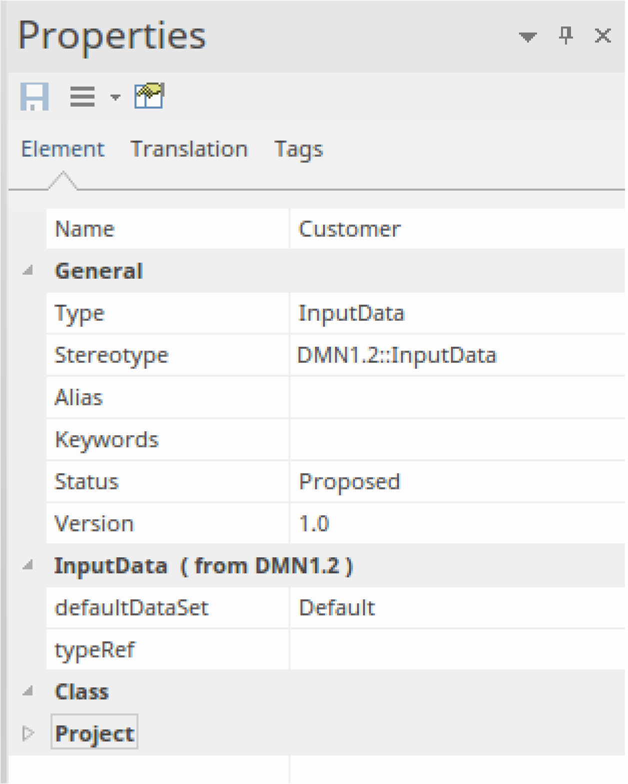


Right-click on the cell 'CustomerDefinition' and choose 'Add Child Component'. Overtype the name of the child component with 'Num of Flights in Last Month' and overtype its datatype with 'number'. Click the 'Save' button to save the changes, and close the window.

Similarly, double-click on the ItemDefinition named 'CabinStatusDefinition', add a child component named 'Num of Pax Overbooked' and set its data type to 'number'. Save the changes and close the window.

Specify the Data Type For Each InputData Element

Select the InputData element 'Customer'. In the Properties window, select the property 'typeRef' and click on the ... button.



Name	Customer
General	
Type	InputData
Stereotype	DMN1.2::InputData
Alias	
Keywords	
Status	Proposed
Version	1.0
InputData (from DMN1.2)	
defaultDataSet	Default
typeRef	
Class	
Project	

Select the ItemDefinition 'Customer Definition' as the type. Click on 'OK'.

Similarly, specify 'Cabin Status Definition' as the type for 'Cabin Status'.

Specify the Inputs to the Decision Element

Double-click on the decision element 'Determine Cabin for Upgrade'

In the DMN Expression window, locate the table row containing the text 'Num of Pax Overbooked' in the first column. Click in the cell in the second column of this row, and press the Spacebar. A list of possible input values is displayed. Choose 'Cabin Status . Num of Pax Overbooked' and press 'Enter'. The selection is written into the cell.

Repeat this process for the second table row 'Num of Flights in Last Month', choosing 'Customer . Num of Flights in Last Month'.

Click on the Save button.

Click on the Validate button.

Define Data Sets

The 'correctness' of your decision model can be tested, by running simulations using a range of representative data sets to verify that the model produces the correct result in all situations.

You can create numerous Data Sets with various names, using a range of data values. You can set one of the data sets as the *default value*.

We will now create a Data Set for each of our InputData elements.

Step	Description
1	Double-click on the InputData element 'Customer'.

	The DMN Expression window displays.
2	In the DMN Expression window, click on the 'Edit Data Set' button . The 'Edit Data Set' window is displayed.
3	Click on the button. A new data set is created.
4	Overwrite the name of the data set if you wish. Leave the Type as 'number'. Enter a value of, for example, 3. Click on the Save icon and the OK button.
5	Repeat for the InputData 'Cabin Status'. Enter a value of, for example, 4.

Add a DMNSimConfiguration Artifact

Locate the DMN 'Simulation Configuration' Artifact in the Diagram Toolbox. Drop one of these onto the diagram as well.

Double-click on it to open the DMN Simulation window at the 'Simulate' tab.

From the DMN Simulation window, you can run simulations of the completed Decision Model. You can also perform validation, generate code and generate test modules.

Step	Description
1	Locate the edit field in the toolbar of this window.
2	Click on the drop-down arrow in this field. A list displays, showing all of the Decision Services and Decision elements in the Package associated with the DMNSim Configuration Artifact. In this case, 'Determine Cabin for Upgrade' is the only item in the list.
3	Click on 'Determine Cabin for Upgrade'.
4	The body of the window now displays the InputData elements and the decision results that are available as inputs to the selected decision. Click on the Save button.
5	Use the 'Value' column to select one of the predefined DataSets for the InputValues, then you can click on the 'Run' button in the lower toolbar to run a simulation, using the selected data sets.

Decision Requirements Diagrams

The elements modeled in Decision Requirements graphs (DRGs) and Decision Requirements diagrams (DRDs) are Decision, Business Knowledge Model, Input Data, Knowledge Source and Decision Service. The dependencies between these elements express three kinds of requirement - Information, Knowledge and Authority.

Components of Decision Requirements Diagrams

This table summarizes the notation for all components of a Decision Requirements diagram.

Component	Description
Decision	A Decision element denotes the act of determining an output from a number of inputs (Input Data or Decision), using decision logic expressed as Literal Expressions, Decision Tables, Invocations or Boxed Context.
Business Knowledge Model	A Business Knowledge Model denotes a reusable module of decision logic represented by a function, which includes zero, one or more than one parameter.
Decision Service (expanded)	A Decision Service can enclose a set of reusable decisions that are invoked internally - for example, by another Decision or Business Knowledge Model - or externally - for example, by a BPMN Process. A good practice is to use a diagram to describe a single expanded Decision Service.
Decision Service (collapsed)	If a Decision Service element serves as an invocable element, connected with knowledge requirements to other elements with invocation logic, we can hide the details of the Decision Service to focus on the decision hierarchies in the big picture.
Input Data	An Input Data element denotes information used as an input to one or more Decisions.
Item Definition	An Item Definition is used to define the type and structure of data items used in the decision model. It is primarily referenced by Input Data elements as a basis for the type and structure of data expected to be input. It can also be referenced for setting the structure for an output. The Item Definition contains Data Sets that provide sets of values useful when performing varied simulations.
Knowledge Source	A Knowledge Source element denotes an authority for a Business Knowledge Model or Decision.
Information Requirement	An Information Requirement denotes Input Data or Decision output being used as input to a Decision.
Knowledge Requirement	A Knowledge Requirement denotes the invocation of a Business Knowledge Model or Decision Service.
Authority Requirement	An Authority Requirement denotes the dependence of a DRG element on another DRG element that acts as a source of guidance or knowledge.

Decision Expression Editor

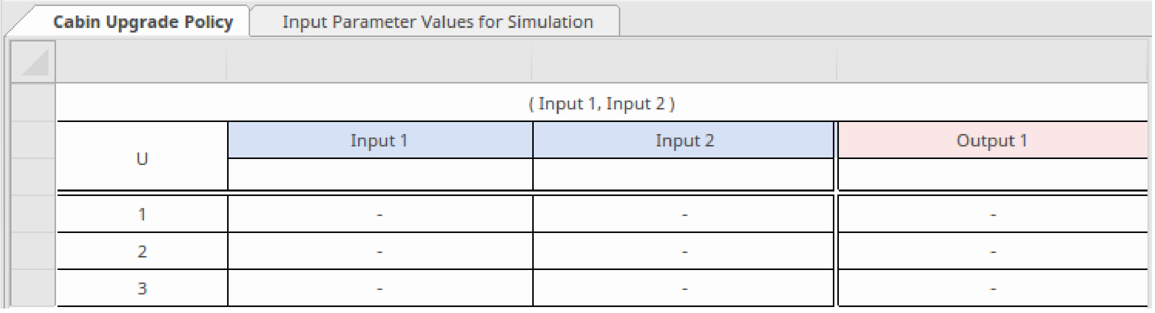
The DMN Expression Editor is the window in which you will define, review and update the details of most of the different types of DMN element within your model. Primarily, it is used for editing the Value Expressions of Decision elements and BusinessKnowledgeModel (BKM) elements.

A different version of the DMN Expression Editor is displayed for each of the four types of value expression used by Decision elements and BKM elements. For BKM elements a second window tab is also presented, for defining the input and output parameters used in calling the BKM.

Two additional versions of the DMN Expression Editor also exist to support editing of ItemDefinition and InputData elements.

The toolbar that is displayed, and the layout of the window content, are dependent upon the type of DMN element that is currently selected and, where applicable, the type of Value Expression being defined.

This image shows the version of the DMN Expression Editor used for defining a Decision Table. In this case, the underlying element is a BusinessKnowledgeModel, and so the decision logic is 'invoked' by other elements, with input and output passed via parameters.



(Input 1, Input 2)

U	Input 1	Input 2	Output 1
1	-	-	-
2	-	-	-
3	-	-	-

Detailed explanations of the DMN Expression Editor's features for each element and expression type are provided in the child Help topics of this topic.

Access

Diagram	Double-click a DMN element on a diagram. The DMN Expression editor window corresponding to the element and its expression type is displayed.

Value Expressions

Summarized in this table are four distinct types of value expression with references to the Help topics detailing each of them.

Type and Icon	Description
Decision Table	A Decision Table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.
Literal Expression	A Literal Expression specifies the decision logic as a textual expression that describes how an output value is derived from its input values. To support simulation and execution, the Literal Expression can use JavaScript functions.

Boxed Context	A boxed context is a collection of context entries, consisting of (name, value) pairs, each with a result value. The context entries provide a means of decomposing a complex expression into a series of simple expressions, providing intermediate results that can be used in subsequent context entries.
Invocation	An invocation calls on another model element (a BusinessKnowledgeModel or a Decision Service) to provide a decision result. The invocation defines parameters that are passed into the 'invoked' element, providing context for evaluation of its decision logic. The decision result is then passed back to the 'invoking' element.

ItemDefinition and InputData Elements

Element	Description
ItemDefinition	ItemDefinition elements are used to define data structures and, optionally, to restrict the range of allowable values of the data. ItemDefinitions can range from a simple single type through to a complex structured type. ItemDefinitions are used to specify the type of InputData elements as well as input parameters.
InputData	InputData elements are used to provide input to Decision elements. The data type of an InputData element is defined using an ItemDefinition element. Data Sets can also be defined as part of an ItemDefinition and an InputData element can then specify a Data Set to be used when running a simulation.

Decision Table

A Decision Table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.

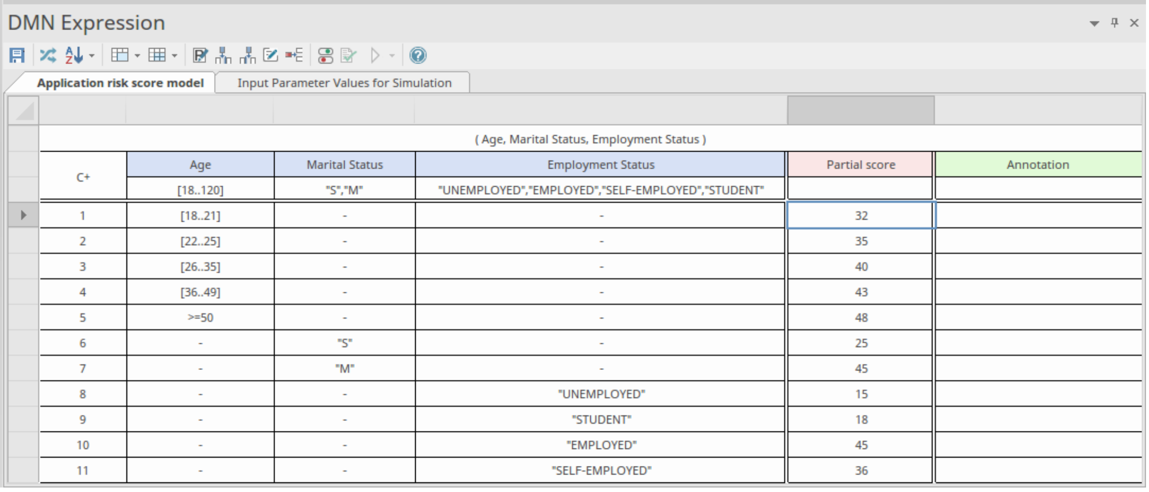
Decision Tables are supported by both the *Decision* and the *Business Knowledge Model* element types. They are denoted by the icon in the top right corner of the element on a diagram.

Access

Diagram	On a diagram, double-click on a Decision element or BusinessKnowledgeModel element. The DMN Expression window is displayed, showing details of the selected element.

Overview

This image shows the DMN Expression window as it appears for a Decision Table.


DMN Expression
Application risk score model
Input Parameter Values for Simulation

(Age, Marital Status, Employment Status)

C+	Age	Marital Status	Employment Status	Partial score	Annotation
	[18..120]	"S","M"	"UNEMPLOYED","EMPLOYED","SELF-EMPLOYED","STUDENT"		
1	[18..21]	-	-	32	
2	[22..25]	-	-	35	
3	[26..35]	-	-	40	
4	[36..49]	-	-	43	
5	>=50	-	-	48	
6	-	"S"	-	25	
7	-	"M"	-	45	
8	-	-	"UNEMPLOYED"	15	
9	-	-	"STUDENT"	18	
10	-	-	"EMPLOYED"	45	
11	-	-	"SELF-EMPLOYED"	36	

A Decision Table consists of:

- The Table Hit Policy (C+, U, A, P and so on) that specifies how the rules are applied
- A list of rules (1, 2, 3, 4 and so on), where each rule row contains specific input entries and corresponding output entries
- A list of Input Clauses (under the blue headings), defined as expressions that generally involve one or more input values
- A list of Output Clauses (under the pink heading), defining the output corresponding to a specific set of inputs
- Optional annotations for each rule (under the green heading) which you can add through the window Toolbar

An Input Clause consists of an expression and an optional list of allowed values (the row just underneath the column headings). Very often, the expression is simply an unmodified input value; however, it could also be an expression involving more than one input value or perhaps a conditional statement such as 'Application Risk Score > 100'. The allowable values apply to the *expression result* rather than the input values used.

Each Output Clause consists of an identifier (a name) and again an optional list of allowed values for that clause.

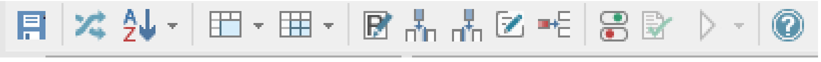
The Decision Table should contain all the inputs - and only those inputs - required to determine an output.

In determining which rules are applied, the expressions defined in the Input Clauses are evaluated for the given inputs and the *expression results* are then used to find rules with matching input entries.

Where the DMN Expression window is not wide or deep enough to display all columns and rows, you can use scroll bars to access the hidden content, or drag the borders out to increase every column width. The 'Input' and 'Output' column widths are initially the same, but you can adjust each column width independently of the others, by dragging the column border either within the table or in the gray bar just below the tab names.

Toolbar for Decision Table Editor

When a Decision Table is selected, the features available in the DMN Expression window are accessed via the Toolbar at the top of the window, as shown:

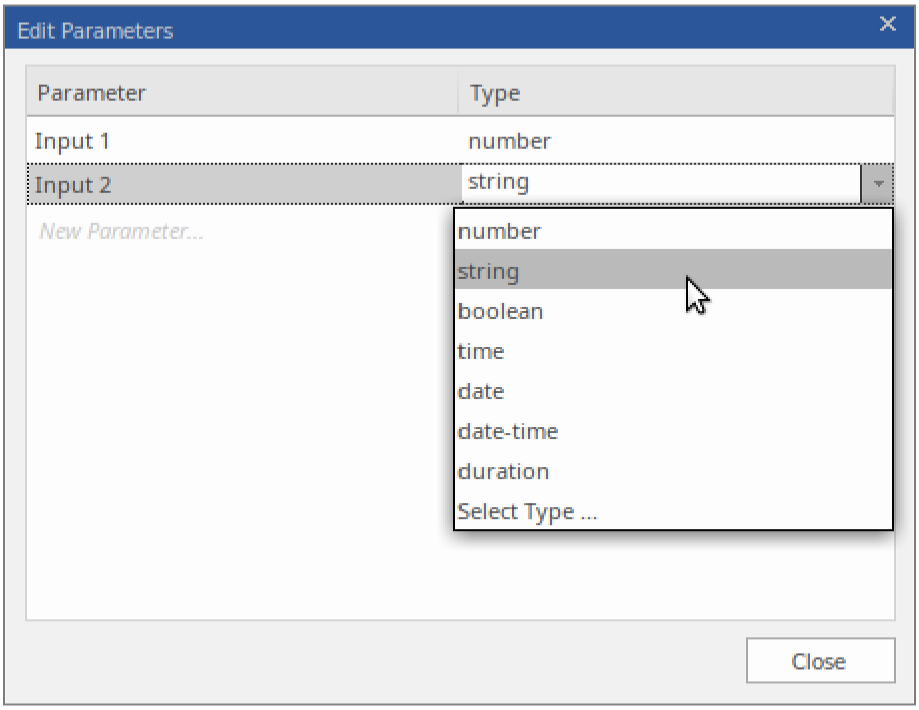


For more details refer to the *Toolbar for Decision Table Editor* Help topic.

Parameters

In the case of Business Knowledge Model (BKM) elements, parameters are used to pass input values supplied by the invoking element. The BKM's decision logic is evaluated using the input parameters and the result is returned to the invoking element. By default, a BKM element is created with two input parameters, 'Input 1' and 'Input 2'.

Click on the 📝 icon in the toolbar of the DMN Expression window to display the 'Edit Parameters' dialog.



Edit Parameters

Parameter	Type
Input 1	number
Input 2	string
New Parameter...	

Here you can change the parameter names, re-arrange the sequence, set their data types, create additional parameters or delete existing ones.

Hit Policy

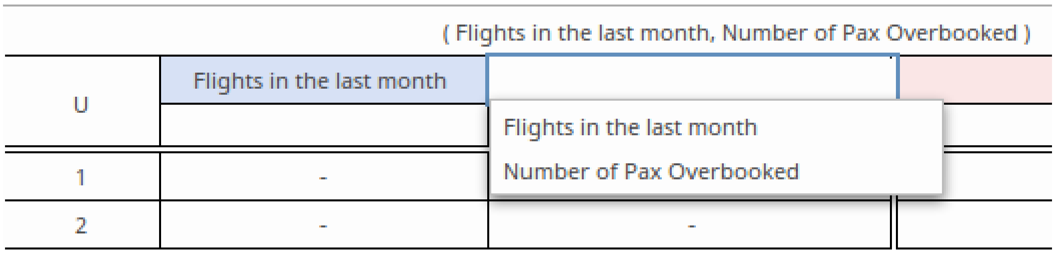
Right-click on the 'Hit Policy Indicator', then choose the desired Hit Policy from the pop-up menu. The various Table Hit Policies are described in detail in the *Decision Table Hit Policy* Help topic.

Input Clauses

An Input Clause of a Decision Table is defined as an expression. Very often, the expression is simply an unmodified input value; however, it could also be an expression involving more than one input value or it could be defined as a conditional statement, such as 'Application Risk Score > 100'. The allowable values apply to the *expression result* rather than the input values used and, as such, the type of the values should match the type of the expression result.

Decision Tables are created with two default Input Clauses, 'Input 1' and 'Input 2'. The data type for both of these clauses is 'number'. In the DMN Expression window, the Input Clauses are displayed as column headings on the Decision Table. To modify an Input Clause, click on the column heading to select the cell, then click again or press F2 to edit.

Auto-completion is supported when editing Input Clauses. That means, for Decision elements, any inputs that are connected to the Decision element are made available for selection from a list. Similarly, for Business Knowledge Model elements, the invocation parameters are made available for selection from a list. See the *DMN Expression Auto-completion* Help topic for further information.



(Flights in the last month, Number of Pax Overbooked)

U	Flights in the last month		
1	-		
2	-	-	

To add additional columns of input entries to the Decision Table, click on the icon on the toolbar of the DMN Expression window.

To remove input columns from the table, right-click within the unwanted input column, then select the option 'Delete Input Column' from the pop-up menu.

The order of the input columns in the table can be re-arranged by dragging and dropping columns to new positions. (Drag the unlabelled cell at the very top of the table column to the required position.)

Allowed Values

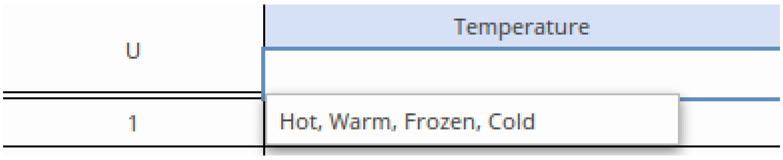
When defining an 'Input' or an 'Output' column, the second row of the column defines the Allowed Values. This is an optional cell in the column, but useful for clarifying the entries in the rows beneath it. When running a validation, each of the cells below the Allowed Values cell are checked to make sure they conform to the expression in this cell.

The expressions used in this cell depend on how the 'Input' or 'Output' column is typed. For example:

- Number - [18 ..35]
- String - 'High', 'Low', 'Medium'
- Boolean - true, false
- Undefined - '-'

Fast Fill Allowed Values

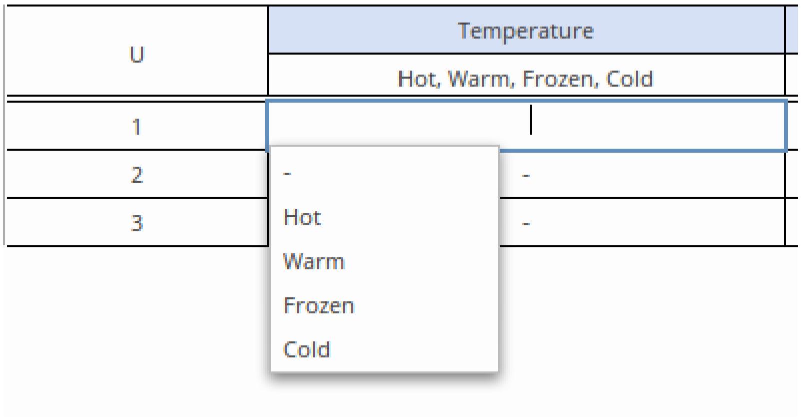
The Input/Output Expression that this references can be a simple value or a complex FEEL expression; however, if it is directly related to an ItemDefinition's 'Allowed Values' field then pressing the Spacebar will enable a fast-fill option to set the 'Allowed Values' as defined in the ItemDefinition (usually referenced via an InputData element).



U	Temperature
1	Hot, Warm, Frozen, Cold

Fast Fill Rows

Once the 'Allowed Values' field is defined, as well as restricting the values that can be used when defining the rules in the table, the 'Allowed Values' field also provides the user with a fast fill option. This is invoked, in a rule cell, by pressing the Spacebar and selecting the required item:



U	Temperature
	Hot, Warm, Frozen, Cold
1	
2	-
3	-

For more details see the Help topic *DMN Expression Auto Completion*.

Output Clauses

An Output Clause consists of a name, a data type and an optional list of allowed values. To modify an Output Clause, click on the column heading cell to select the cell, then click again or press F2 to edit.

To add additional columns of output entries to the Decision Table, click on the icon on the toolbar of the Expression editor window.

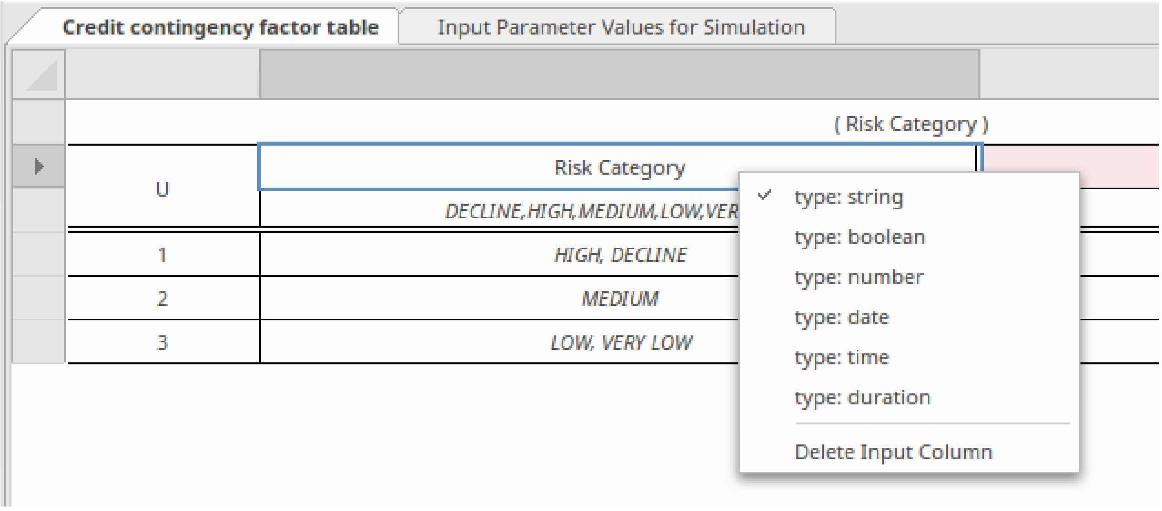
To remove Output columns from the table, right-click within the unwanted Output column, then select the option 'Delete Output Column' from the pop-up menu.

The order of the columns in the table can be re-arranged by dragging and dropping columns to new positions. (Drag the unlabelled cell at the very top of the table column to the desired position.)

Data Type for Input/Output Clauses

For the simulation to work it is critical to set the data type for all Input and Output Clauses. Range, gap and overlap validations are supported for clauses of type 'number', but validation cannot be performed if the type has not been specified. Code Generation for typed languages such as C++, C# and Java requires that the data types are specified. When the data type is specified as 'string', there is no need to enclose each string literal within quotes. String values are displayed using italic font if the type has been declared.

To set the data type, right-click on the Input or Output column header and select the required type from the list.



	(Risk Category)
U	Risk Category DECLINE,HIGH,MEDIUM,LOW,VER
1	HIGH, DECLINE
2	MEDIUM
3	LOW, VERY LOW

Defining Decision Table Rules

Decision Table rules are defined by specifying input entries and corresponding output entries within the cells of a table row. For 'number' data types, input entries can be specified as a single value, or as a number range, such as '<10', '>100' or '[2..8)'. (When defining number ranges, the use of round brackets indicates that the bounding number is NOT included; use of square brackets indicates the bounding number is included.) Output entries should specify a single value per cell.

Additional rules can be appended to the list of rules by clicking on the icon in the toolbar. Unwanted rules can be deleted from the table by right-clicking on the rule and selecting the option 'Delete Rule Row' from the pop-up menu.

Existing rules can be copied and pasted within the table by first selecting the rules, (use 'Ctrl+Click' to add/remove from selection), then using the menu options 'Copy Rules to Clipboard' and 'Paste Rules from Clipboard' to perform the copy and paste. The copied rules can then be modified by selecting and editing individual cell entries.

If the 'Allowed Values' field is set for a string or Boolean expression, the Spacebar can be used to display a list of values to select from, as shown in the earlier *Allowed Values - Fast Fill Rows* section.

Rules can also be sorted within the table, either by:

- Clicking the icon on the toolbar, then choosing to either 'Sort By Input' or 'Sort By Output', or
- Right-clicking on individual rules within the table and selecting the 'Move Rule Up' or 'Move Rule Down' option from the pop-up menu

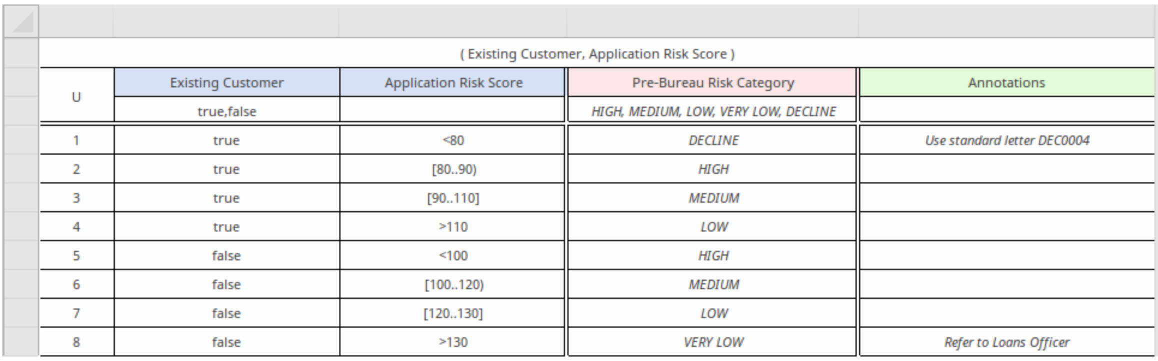
To determine which table rows are selected for output, the *expressions* that are defined by the Input Clauses are evaluated for the given inputs and the *results* of the expressions are then compared against the input entries of the table rows. Where the expression results match the input entries of a table row, that row is selected for output.

The Decision Table's 'Hit Policy' determines how the table's matching rows are then used to produce its output.

Rule Formats

You can select - using a Toolbar icon - to display the Decision Table in one of three formats, as shown here.

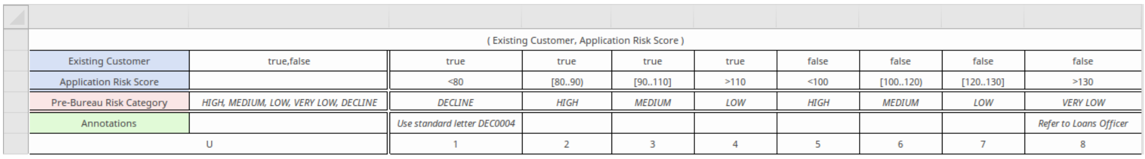
Rule-as-Row format, where the rule is developed along rows with the inputs, outputs and annotations set in columns:



(Existing Customer, Application Risk Score)

U	Existing Customer	Application Risk Score	Pre-Bureau Risk Category	Annotations
	true,false		HIGH, MEDIUM, LOW, VERY LOW, DECLINE	
1	true	<80	DECLINE	Use standard letter DEC0004
2	true	[80..90)	HIGH	
3	true	[90..110]	MEDIUM	
4	true	>110	LOW	
5	false	<100	HIGH	
6	false	[100..120)	MEDIUM	
7	false	[120..130]	LOW	
8	false	>130	VERY LOW	Refer to Loans Officer

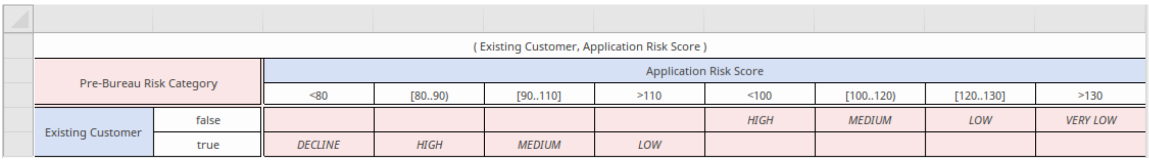
Rule-as-Column format, where the rules are developed down columns with the inputs, outputs and annotations set along the rows:



(Existing Customer, Application Risk Score)

Existing Customer	true,false	true	true	true	true	false	false	false	false
Application Risk Score		<80	[80..90)	[90..110]	>110	<100	[100..120)	[120..130]	>130
Pre-Bureau Risk Category	HIGH, MEDIUM, LOW, VERY LOW, DECLINE	DECLINE	HIGH	MEDIUM	LOW	HIGH	MEDIUM	LOW	VERY LOW
Annotations		Use standard letter DEC0004							Refer to Loans Officer
U		1	2	3	4	5	6	7	8

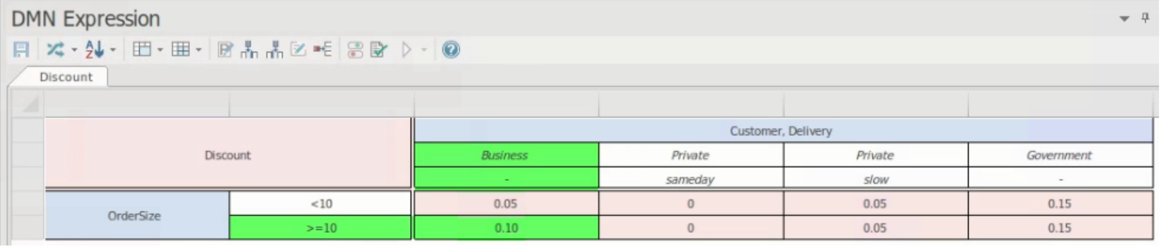
Rule-as-Crosstab format, where the rules are formed from inputs defined as a set of rows AND a combination of columns, with outputs set in the intersecting cells. (Note that this format hides the 'Annotation' fields):



(Existing Customer, Application Risk Score)

Pre-Bureau Risk Category		Application Risk Score							
		<80	[80..90)	[90..110]	>110	<100	[100..120)	[120..130]	>130
Existing Customer	false					HIGH	MEDIUM	LOW	VERY LOW
	true	DECLINE	HIGH	MEDIUM	LOW				

At the end of a simulation, in a Crosstab Decision Table, related input entries and output entries are highlighted. For example, in this simulation processing resulted in the output of a 0.10 discount for a Business Customer where Order Size was less than or equal to 10 and delivery was not applicable.



Discount		Customer, Delivery			
		Business	Private	Private	Government
		-	sameday	slow	-
OrderSize	<10	0.05	0	0.05	0.15
	>=10	0.10	0	0.05	0.15

Crosstab Settings

In Rule-as-Crosstab format, as the inputs form both rows and columns and the outputs are at the intersections, the steps for setting values are slightly different from those for the other two formats.

1. To add another type of input, right-click on the input column header and select the 'Add Input' option. You are prompted to enter the input name; the input is added as a set of fields under the current column fields.
   To delete an input type from the columns, right-click on its set of fields and select the 'Delete Input' option. The name of the input and its set of fields are removed from the column headings.

2. To add another type of output, right-click on the Output block in the top left of the window and select the 'Add Output' option. You are prompted to enter the Output name; the name is added to the Output block and a new row is added to each cell in the body of the window.
   To delete an Output type, right-click on the type name in the Output block in the top left of the window and select the 'Delete Output' option. The Output name and its fields in the grid are removed.

3. You can rotate between the input types to select one for the row headers. Right-click on the row and click on the 'Select Input as Row Header' option. This displays a list of the input types; click on the type to use as the row header; the other types are combined in the columns.

4. To add a value entry row or column to the inputs, right-click on a current row or column and select the 'Add Input Entry Row' or 'Add Input Entry Column' option, as appropriate. A prompt displays for the name of the input entry; when you enter this, the appropriate row or column is added to the Decision Table.
   To delete a value entry row or column, right-click on it and select the 'Delete Input Entry Row' or 'Delete Input Entry Column' option. The selected row or column is deleted from the table.

5. In the Input columns, each row matches a type of input. If you want to move the row for one type of input above or below another, right-click on it and select the 'Move Input Up' or 'Move Input Down' option. The context menu only provides the options that can be actioned, so as it is not possible to move, say, the last row downwards, the 'Move Input Down' option is not listed.

Toolbar for Decision Table Editor

This table provides descriptions of the features accessible in the DMN Expression window when a Decision Table is selected.

Toolbar Options

Icon	Description
	Save changes to the currently selected Decision or BusinessKnowledgeModel element.
	Toggle the view for the Decision Table between Rule-as-Row, Rule-as-Column and Rule-as-Crosstab. Alternatively, click on the drop-down arrow and select the format you require.
	Click on 'Sort By Input' to sort the rules by input columns; click on 'Sort By Output' to sort the rules by output columns. The columns can be dragged and dropped to organize the sorting order.
	Merge cells of adjacent rules, where the content of the input entries is the same. You can edit the content of the merged cells. During simulation, the merged items are highlighted.
	Split input entry cells that have previously been merged.
	Display the 'Edit Parameters' window, where you can specify the names and data types of the parameters that are passed when invoking the decision logic of a BusinessKnowledgeModel element.
	Append an input column to the Decision Table.
	Append an output column to the Decision Table.
	Append an annotation column (with a green heading cell) to the table, in which you can record short notes or comments on the rule. (See the illustrations in the Rule-as-Row/Rule-as-Column row earlier.) You can add more than one annotation column if required, typing in an appropriate column title in each heading cell. To remove an annotation column, right-click on it and select the 'Delete Annotation Column' option.
	Append a rule to the Decision Table.
	Show or hide the allowed values fields for the 'Input' and 'Output' columns. The allowed values defined for an input or output will be used for validation and auto completion editing.
	Perform validation of the Decision Table. Enterprise Architect will perform a series of validations to help you discover any errors in the Decision Table.

▷	This button is enabled when a Decision Table is defined for a BusinessKnowledgeModel element. Select the 'Input Parameter Values for Simulation' tab, complete the fields, then click on this button. The test result will be presented on the Decision Table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a BusinessKnowledgeModel element, without specifying its context. A number of menu options are available for this tool bar button. For more information, see the Help topic *Simulate DMN Model.*

Decision Table Hit Policy

The Hit Policy specifies the result of the Decision Table in cases of overlapping rules. The single character in a particular Decision Table cell indicates the table type and unambiguously reflects the decision logic.

Single Hit Policies:

- **U**nique: no overlap is possible and all rules are disjoint; only a single rule can be matched (this is the default)
- **A**ny: there might be overlap, but all the matching rules show equal output entries for each output, so any match can be used
- **P**riority: multiple rules can match, with different output entries; this policy returns the matching rule with the highest output priority
- **F**irst: multiple (overlapping) rules can match, with different output entries; the first hit by rule order is returned

Multiple Hit Policies:

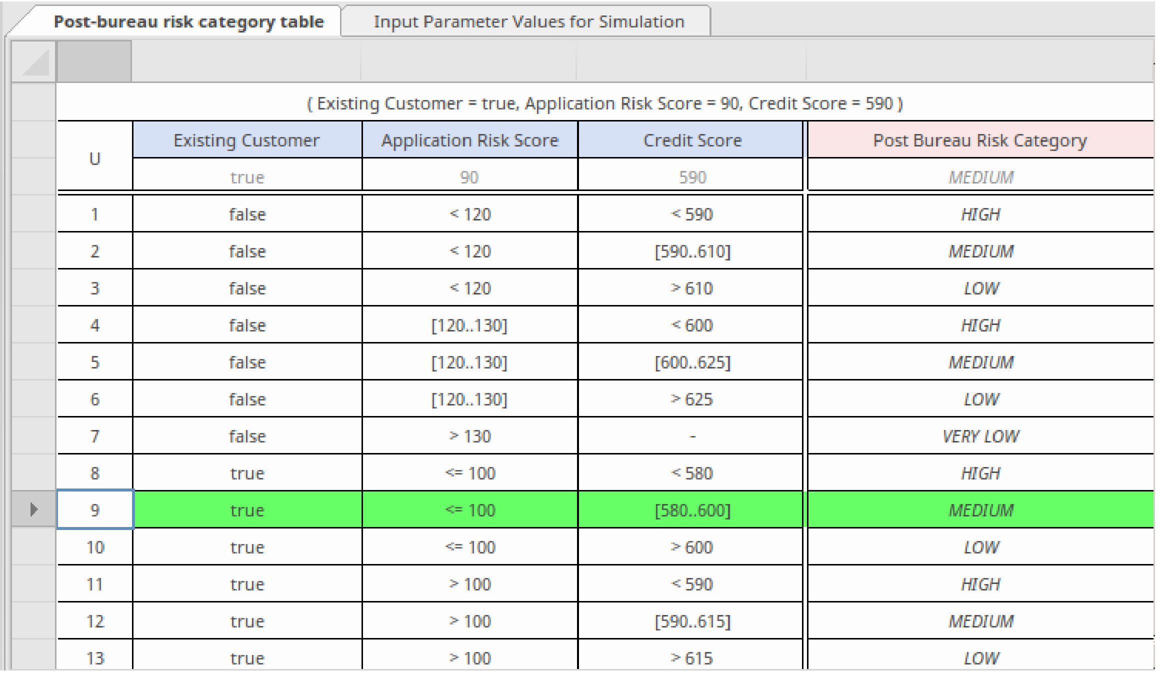
- **O**utput order: returns all hits in decreasing output priority order
- **R**ule order: returns all hits in rule order
- **C**ollect: returns all hits in arbitrary order; an operator (‘+’, ‘<’, ‘>’, ‘#’) can be added to apply a simple function to the outputs

Collect operators are:

- + (sum): the result of the Decision Table is the sum of all the distinct outputs
- < (min): the result of the Decision Table is the smallest value of all the outputs
- > (max): the result of the Decision Table is the largest value of all the outputs
- # (count): the result of the Decision Table is the number of distinct outputs

Example of Unique Hit Policy

The 'Unique' Hit Policy is the most popular type for a Decision Table and all rules are disjoint.

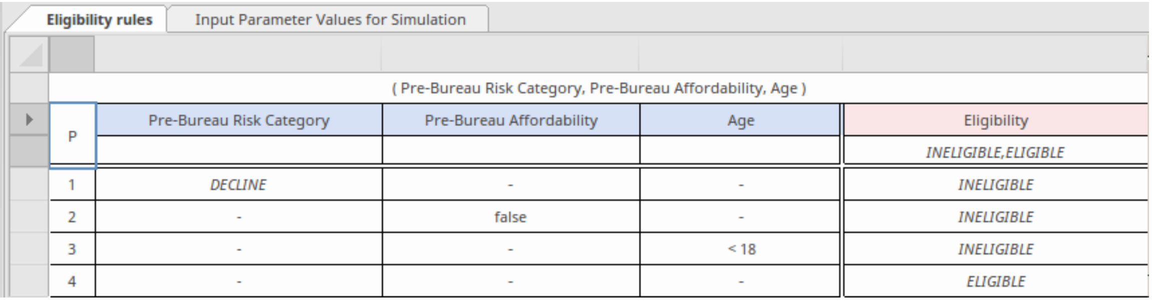

Post-bureau risk category table
Input Parameter Values for Simulation

(Existing Customer = true, Application Risk Score = 90, Credit Score = 590)

U	Existing Customer	Application Risk Score	Credit Score	Post Bureau Risk Category
	true	90	590	*MEDIUM*
1	false	< 120	< 590	*HIGH*
2	false	< 120	[590..610]	*MEDIUM*
3	false	< 120	> 610	*LOW*
4	false	[120..130]	< 600	*HIGH*
5	false	[120..130]	[600..625]	*MEDIUM*
6	false	[120..130]	> 625	*LOW*
7	false	> 130	-	*VERY LOW*
8	true	<= 100	< 580	*HIGH*
9	true	<= 100	[580..600]	*MEDIUM*
10	true	<= 100	> 600	*LOW*
11	true	> 100	< 590	*HIGH*
12	true	> 100	[590..615]	*MEDIUM*
13	true	> 100	> 615	*LOW*

Example of Priority Hit Policy

In a table with the 'Priority' Hit Policy, multiple rules can match, with different output entries. This policy returns the matching rule with the highest output priority.

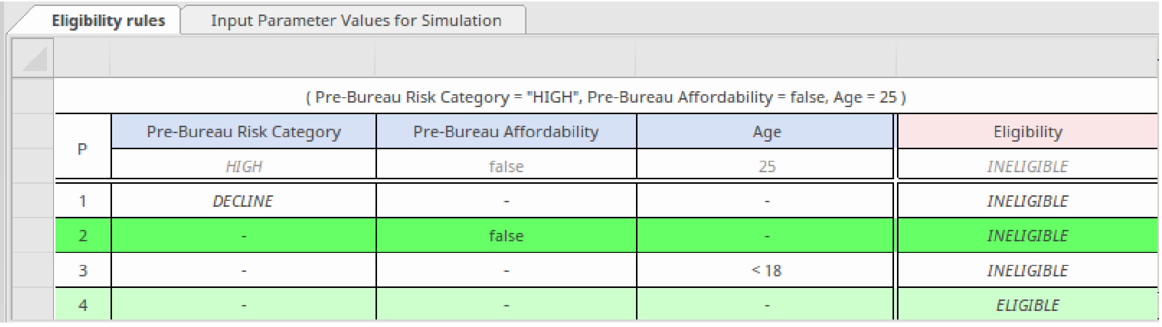


(Pre-Bureau Risk Category, Pre-Bureau Affordability, Age)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
				INELIGIBLE,ELIGIBLE
1	DECLINE	-	-	INELIGIBLE
2	-	false	-	INELIGIBLE
3	-	-	< 18	INELIGIBLE
4	-	-	-	ELIGIBLE

Note: The list of allowable values is used to define the output priority. Here, the allowable values are listed as INELIGIBLE and ELIGIBLE; INELIGIBLE is defined as having a higher priority than ELIGIBLE.

One possible simulation result might resemble this:



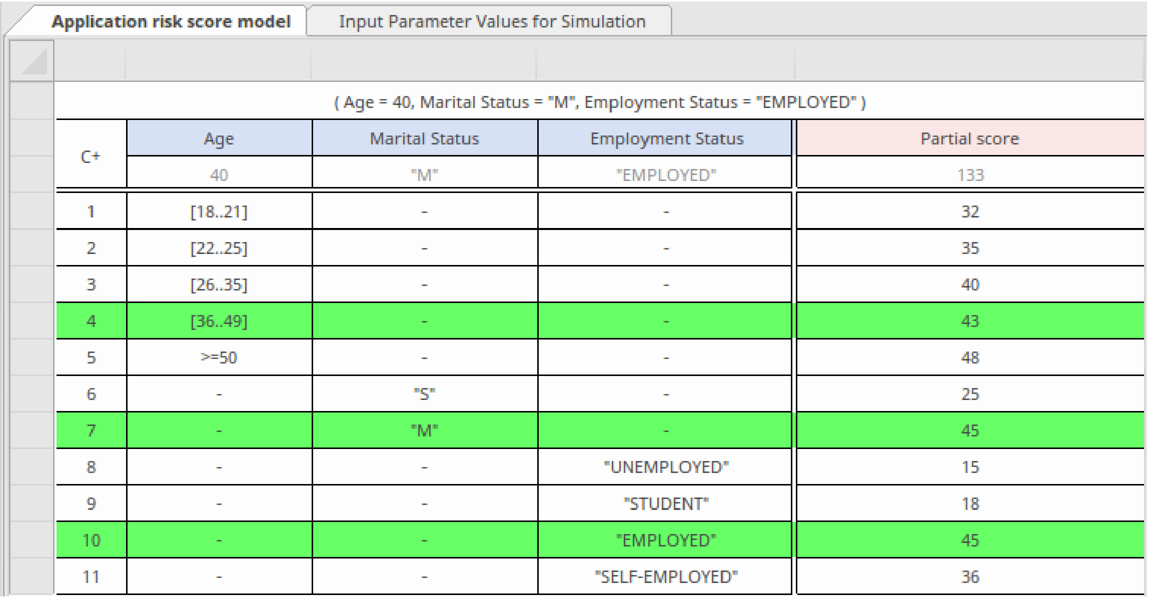
(Pre-Bureau Risk Category = "HIGH", Pre-Bureau Affordability = false, Age = 25)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	HIGH	false	25	INELIGIBLE
1	DECLINE	-	-	INELIGIBLE
2	-	false	-	INELIGIBLE
3	-	-	< 18	INELIGIBLE
4	-	-	-	ELIGIBLE

The matching rules are highlighted, but the output from rule 2 is chosen because INELIGIBLE has higher priority than ELIGIBLE.

Example of Collection-Sum Hit Policy

For a Decision Table with the 'Collect-Sum' (C+) Hit Policy, the result of the Decision Table is the sum of all the distinct outputs.


Application risk score model
Input Parameter Values for Simulation

(Age = 40, Marital Status = "M", Employment Status = "EMPLOYED")

C+	Age	Marital Status	Employment Status	Partial score
	40	"M"	"EMPLOYED"	133
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	"S"	-	25
7	-	"M"	-	45
8	-	-	"UNEMPLOYED"	15
9	-	-	"STUDENT"	18
10	-	-	"EMPLOYED"	45
11	-	-	"SELF-EMPLOYED"	36

In this example, the output Partial Score is calculated as 43 + 45 + 45 = 133

Decision Table Validation

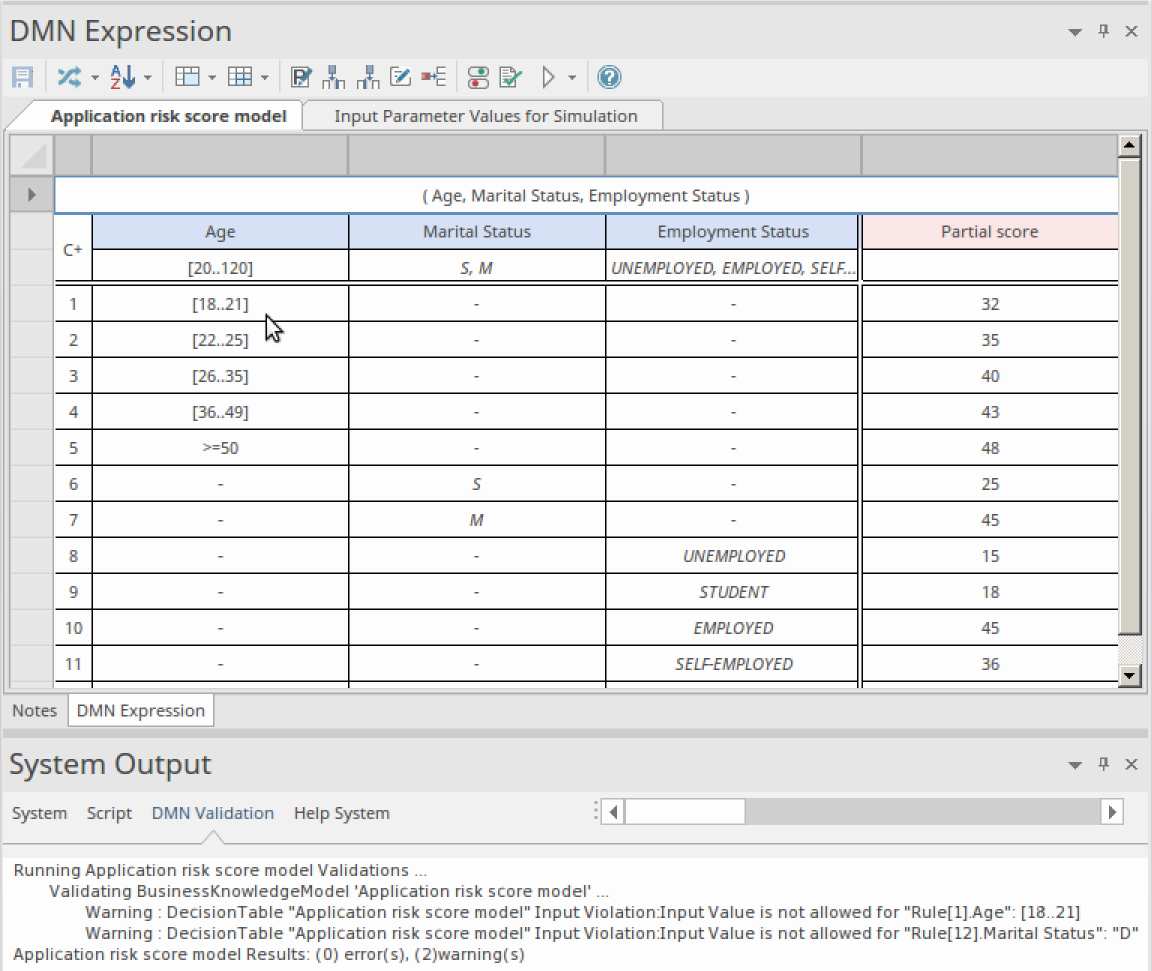
A Decision Table is one of the most common and useful DMN Expressions used to express decision logic. However, modeling a Decision Table can also be complicated, especially if multiple Input Clauses are used in combination for many Decision Table rules. Enterprise Architect provides the facility to validate Decision Tables, as explained in this topic.

Access

DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Open DMN Simulation > Configure : Validate button

Entries out of range detection

It is good practice to define 'allowed values' for the Input Clauses and Output Clauses of a Decision Table. The 'allowed values' list is used to perform range-checking of the input and output entry values for the table rules.



(Age, Marital Status, Employment Status)

C+	Age	Marital Status	Employment Status	Partial score
	[20..120]	S, M	UNEMPLOYED, EMPLOYED, SELF...	
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	S	-	25
7	-	M	-	45
8	-	-	UNEMPLOYED	15
9	-	-	STUDENT	18
10	-	-	EMPLOYED	45
11	-	-	SELF-EMPLOYED	36

```
Running Application risk score model Validations ...
      Validating BusinessKnowledgeModel 'Application risk score model' ...
            Warning : DecisionTable "Application risk score model" Input Violation:Input Value is not allowed for "Rule[1].Age": [18..21]
            Warning : DecisionTable "Application risk score model" Input Violation:Input Value is not allowed for "Rule[12].Marital Status": "D"
Application risk score model Results: (0) error(s), (2)warning(s)
```

In this example:

- The 'Age' Input Clause defines a range of [20..120]; however, the input entry for rule 1 specifies a range of [18..21]; this is outside the range of allowed values, so rule 1 is reported as invalid
- The 'Marital Status' clause defines its allowed values as an enumeration of 'S, M'; rule 12 specifies a value of 'D', hence that rule is also reported as invalid

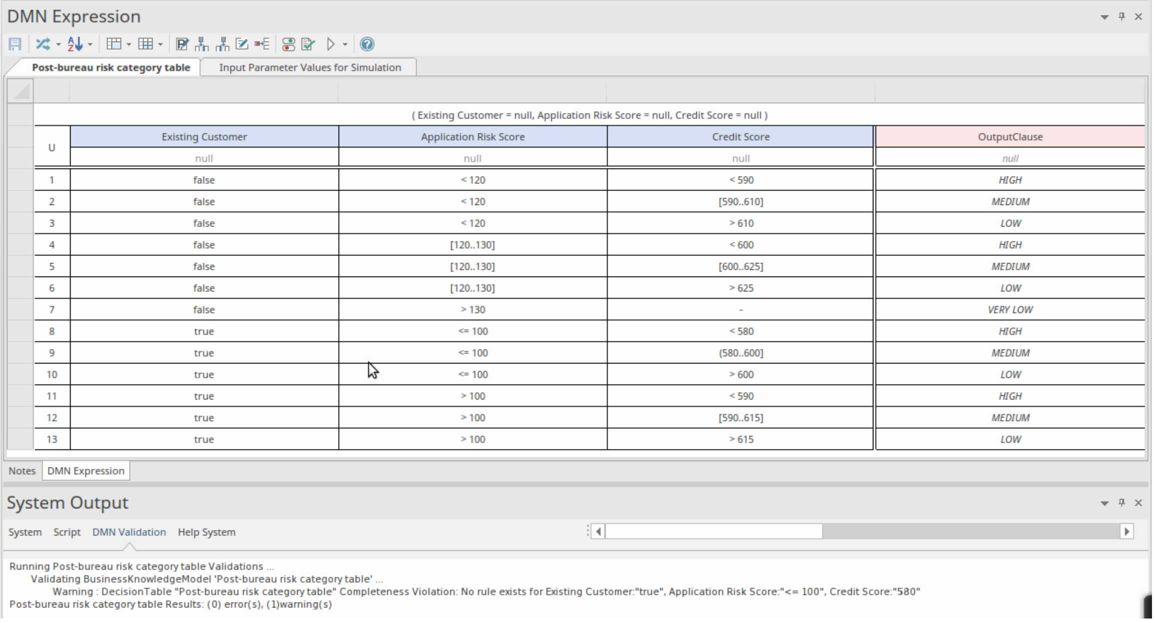
These issues can be corrected, either by updating the 'allowed values' or by modifying the input entries for the invalid rules, depending on the actual business rules.

Completeness detection - report gaps in the rules

The gaps in rules for a Decision Table mean that, given a combination of input values, no rule is matched. This indicates that some logic or rule might be missing (unless a default output is defined).

When the Decision Table contains many rules that specify number ranges, it becomes difficult to visually detect gaps and quite time-consuming to compose and run exhaustive test cases.

For example:



(Existing Customer = null, Application Risk Score = null, Credit Score = null)

U	Existing Customer	Application Risk Score	Credit Score	OutputClause
	null	null	null	*null*
1	false	< 120	< 590	*HIGH*
2	false	< 120	[590..610]	*MEDIUM*
3	false	< 120	> 610	*LOW*
4	false	[120..130]	< 600	*HIGH*
5	false	[120..130]	[600..625]	*MEDIUM*
6	false	[120..130]	> 625	*LOW*
7	false	> 130	-	*VERY LOW*
8	true	<= 100	< 580	*HIGH*
9	true	<= 100	(580..600]	*MEDIUM*
10	true	<= 100	> 600	*LOW*
11	true	> 100	< 590	*HIGH*
12	true	> 100	[590..615]	*MEDIUM*
13	true	> 100	> 615	*LOW*

```
Running Post-bureau risk category table Validations ...
    Validating BusinessKnowledgeModel 'Post-bureau risk category table' ...
        Warning : DecisionTable "Post-bureau risk category table" Completeness Violation: No rule exists for Existing Customer:"true", Application Risk Score:"<= 100", Credit Score:"580"
Post-bureau risk category table Results: (0) error(s), (1)warning(s)
```

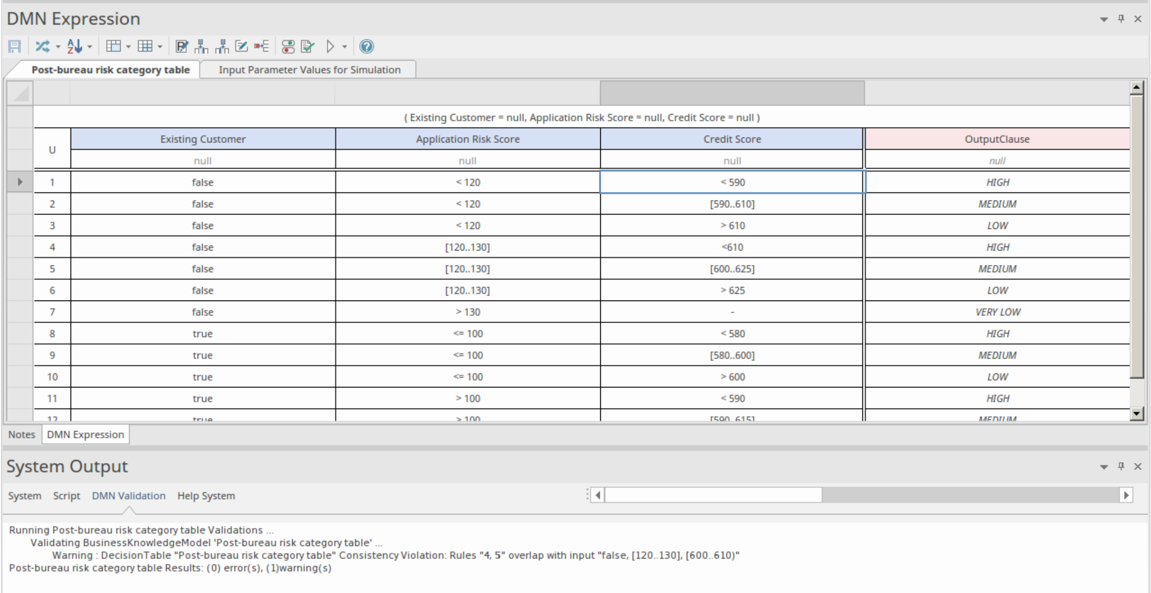
The validation reports a gap in the rules. Closer inspection reveals an error in rule 9. The input entry **(**580..600], should be **[**580..600].

Rule Overlap detection for Unique Hit Policy

When rules overlap, for a given combination of input values, multiple rules are matched. This is a violation if the Decision Table specifies its Hit Policy as 'Unique'.

When the Decision Table contains many rules that specify number ranges, it becomes difficult to visually detect gaps and quite time-consuming to compose and run exhaustive test cases.

For example:



(Existing Customer = null, Application Risk Score = null, Credit Score = null)

U	Existing Customer	Application Risk Score	Credit Score	OutputClause
	null	null	null	*null*
1	false	< 120	< 590	*HIGH*
2	false	< 120	[590..610]	*MEDIUM*
3	false	< 120	> 610	*LOW*
4	false	[120..130]	<610	*HIGH*
5	false	[120..130]	[600..625]	*MEDIUM*
6	false	[120..130]	> 625	*LOW*
7	false	> 130	-	*VERY LOW*
8	true	<= 100	< 580	*HIGH*
9	true	<= 100	[580..600]	*MEDIUM*
10	true	<= 100	> 600	*LOW*
11	true	> 100	< 590	*HIGH*
12	true	> 100	[590..615]	*MEDIUM*

```
Running Post-bureau risk category table Validations ...
    Validating BusinessKnowledgeModel 'Post-bureau risk category table' ...
        Warning : DecisionTable "Post-bureau risk category table" Consistency Violation: Rules "4, 5" overlap with input "false, [120..130], [600..610)"
Post-bureau risk category table Results: (0) error(s), (1)warning(s)
```

The validation reports an overlap in the rules, involving rules 4 & 5. Closer inspection reveals the overlap exists in the third input 'Credit Score', where '<610' overlaps with '[600..625]'. You could correct this issue either by changing rule 4 to '<600' or by changing rule 5 to '[610..625]', to reflect the actual business rules.

Literal Expression

A Literal Expression is the simplest form of DMN expression; it is commonly defined as a single-line statement, or an if-else conditional block. The Literal Expression is a type of value expression used in both Decision elements and Business Knowledge Model (BKM) elements. As the expression becomes more complex, you might prefer a Boxed Context or, in order to improve the readability, you can encapsulate some of the logic as a function in the DMN Library.

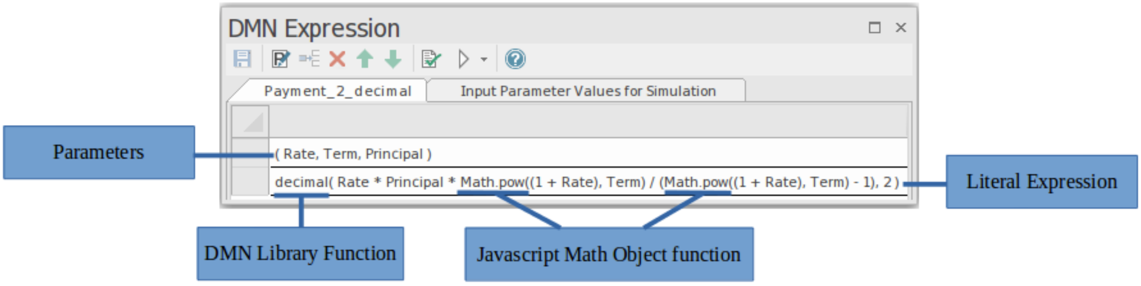
The =X icon on the top right corner of the Decision or BKM element indicates that it is implemented as a *Literal Expression*.

Access

Diagram	On a diagram, double-click on a Decision element or BusinessKnowledgeModel element. The DMN Expression editor window displays showing details of the selected element.

Overview

This image shows the DMN Expression editor window, as it appears for a Literal Expression.



```
( Rate, Term, Principal )
decimal( Rate * Principal * Math.pow((1 + Rate), Term) / (Math.pow((1 + Rate), Term) - 1), 2 )
```

The Literal Expression is a textual representation of the decision logic. It describes how an output value is derived from its input values, using mathematical and logical operations.

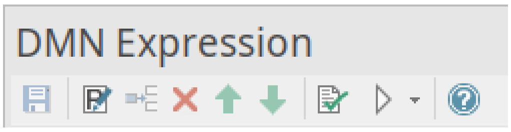
The expression editor window presents the Literal Expression as a table, with two key rows:

- Parameters: defines the input parameters used in the expression
- Literal Expression: where the formula for the expression is defined - this defines the output of the Decision

In order to support simulation and execution, the literal expression can use JavaScript global functions or JavaScript object functions. Users can also create DMN Library functions for use within the expressions.

Toolbar for Literal Expression Editor

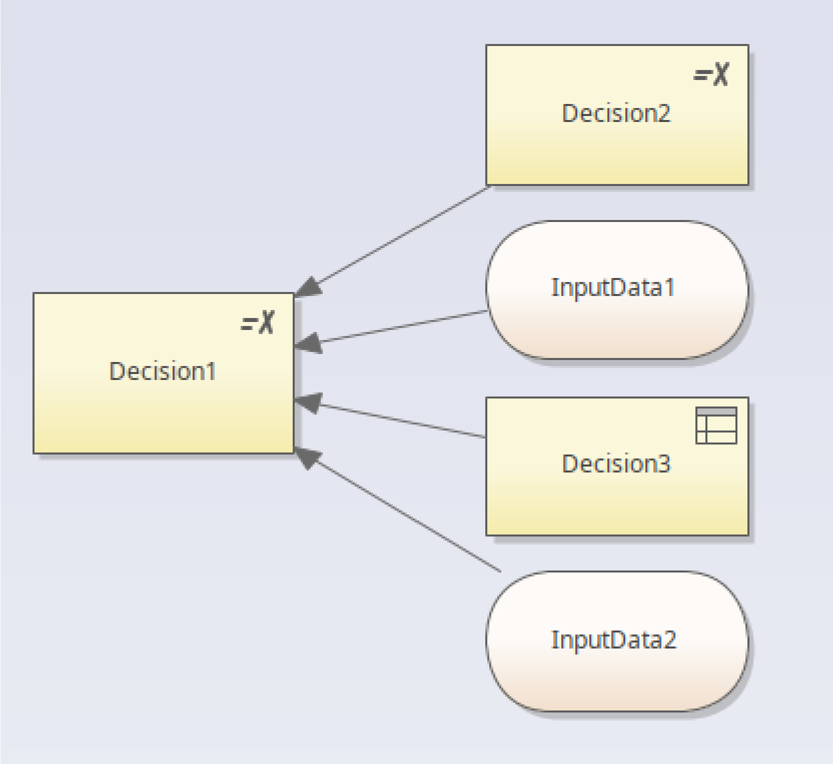
When a Literal Expression is selected, the layout of features accessible in the DMN Expression window is:



For more details refer to the Help topic *Toolbar for Literal Expression Editor*.

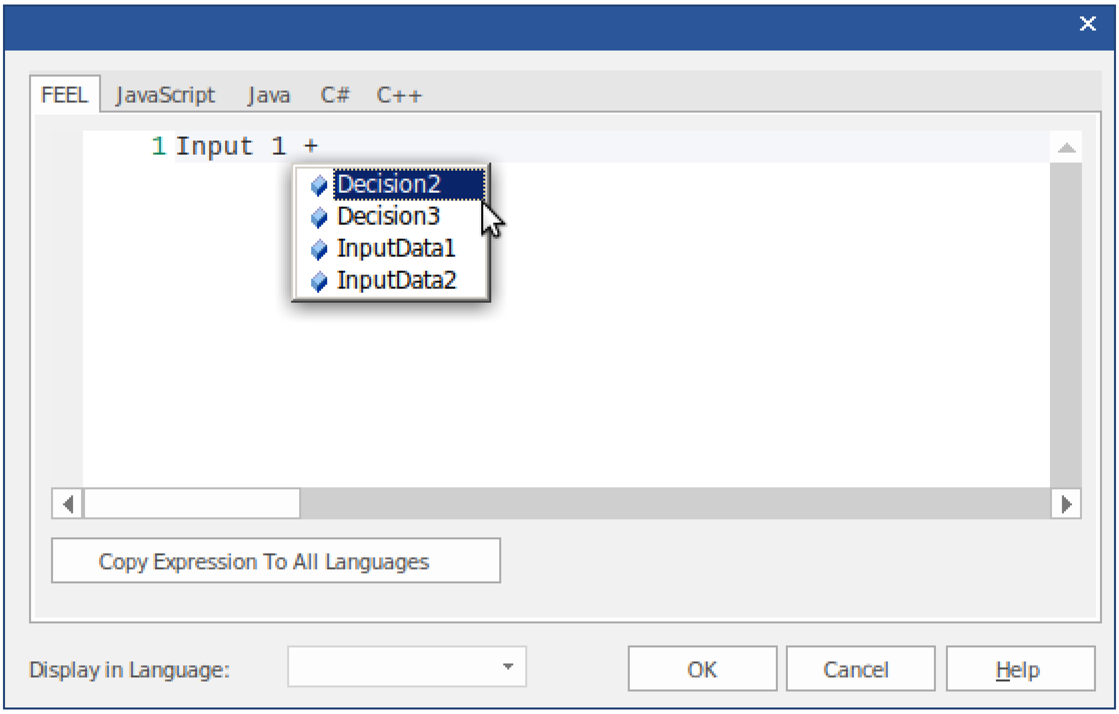
Expression Editor and Intelli-sense support

In accordance with the Friendly Enough Expression Language (FEEL) language specification, parameter names can contain spaces, which makes the expression easier to read. Enterprise Architect also provides Intelli-sense support for editing the expressions, allowing for minimal typing and fewer mistakes.



Given a decision hierarchy such as the one shown, when editing the expression for 'Decision1', the inputs to 'Decision1' - namely 'Decision2', 'Decision3', 'InputData1' and 'InputData2' - will be available through Intelli-sense in the editor.

By right-clicking on the 'Expression' row of the DMN Expression window, then choosing the menu option 'Edit Expressions...', the expression code editor dialog is displayed. Pressing Ctrl+Space displays the Intelli-sense menu:



```
Input 1 +
```

- For 'Decision' elements, all of the inputs to the decision will be displayed
- For Business Knowledge Model (BKM) elements, all of the input parameters will be displayed

The DMN Model can be generated as source code in JavaScript, Java, C# or C++; since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

In the generated code, the space inside a variable name will be replaced by an underscore.

Toolbar for Literal Expression Editor

When a Literal Expression is selected, the DMN Expression window displays a toolbar specific to that type of expression.

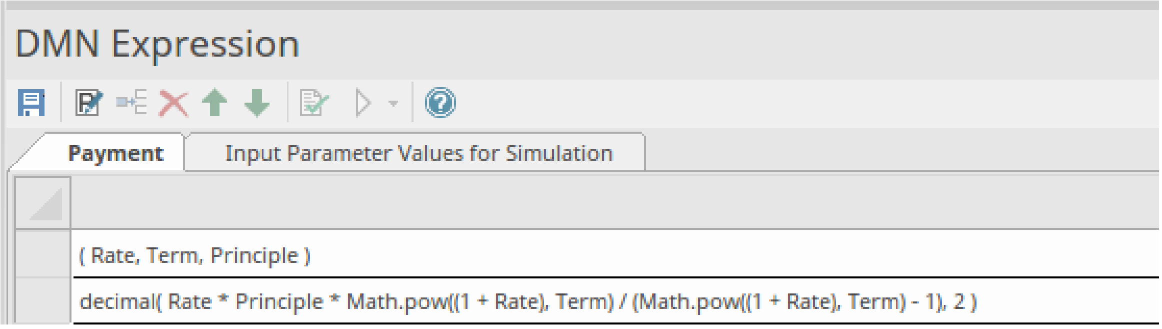
Toolbar Options

This table provides descriptions of the features accessible from the toolbar in the DMN Expression window when a Literal Expression is selected.

Options	Description
💾	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
✎	Click on this button to edit parameters for the Business Knowledge Model.
⇶	This option is disabled for Literal Expressions.
✕	This option is disabled for Literal Expressions.
⬆	This option is disabled for Literal Expressions.
⬇	This option is disabled for Literal Expressions.
✓	Click on this button to perform validation of the Literal Expression. Enterprise Architect will perform a series of validations to help you locate any errors in the Expression.
▷	This button is enabled when the Literal Expression is defined for a BusinessKnowledgeModel element.

Example - Loan Repayment

This Business Knowledge Model (BKM) *Payment* is implemented as a Literal Expression.

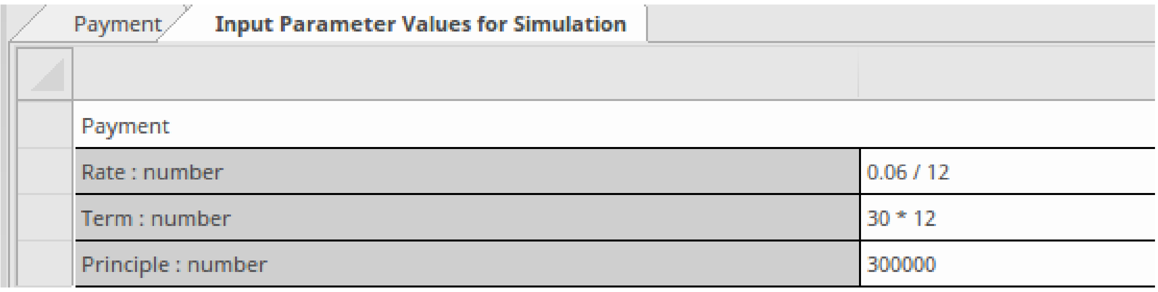


(Rate, Term, Principle)

decimal(Rate * Principle * Math.pow((1 + Rate), Term) / (Math.pow((1 + Rate), Term) - 1), 2)

- The BKM defines three parameters: Rate, Term and Principle

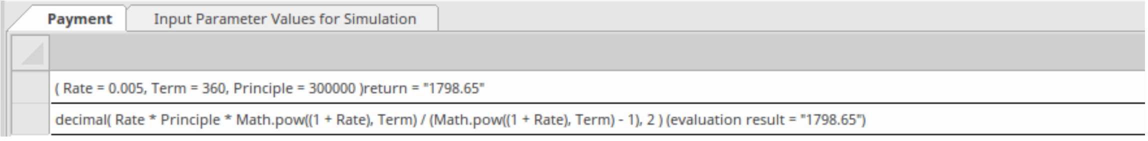
Set the values for the Input Parameters and evaluate the model:



Payment

Rate : number	0.06 / 12
Term : number	30 * 12
Principle : number	300000

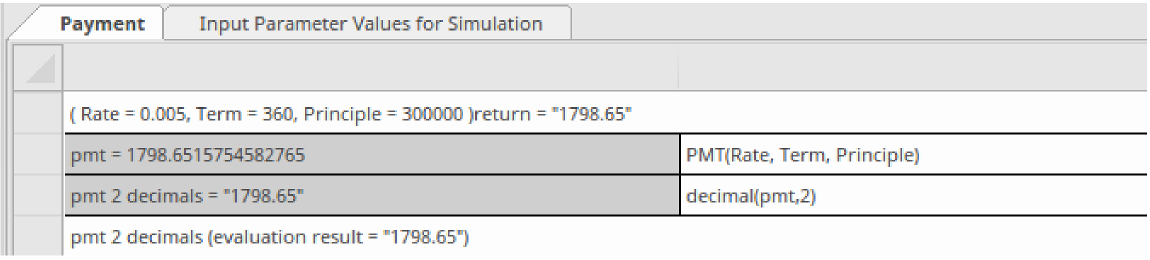
- The runtime parameter value will be displayed; for example, Rate = 00.005
- The BKM's result will be evaluated by the literal expression and the value is displayed on the declaration line; for example, return = 1798.65



(Rate = 0.005, Term = 360, Principle = 300000)return = "1798.65"

decimal(Rate * Principle * Math.pow((1 + Rate), Term) / (Math.pow((1 + Rate), Term) - 1), 2) (evaluation result = "1798.65")

Although the formula for this can be written in one line, it is quite complicated. We can re-factor this model with Built-In function and Boxed Context to improve readability:



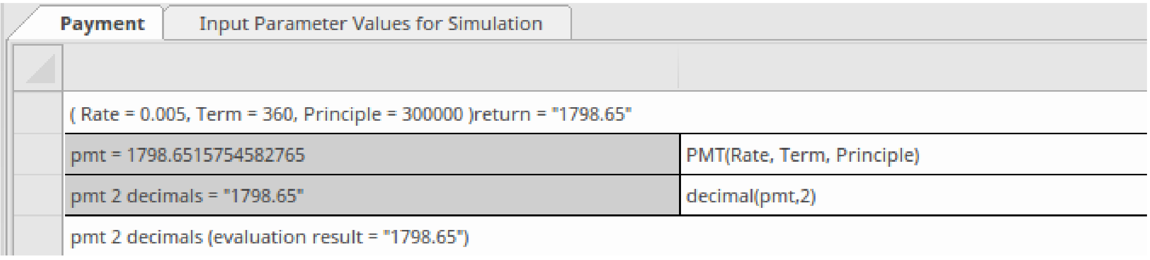
(Rate = 0.005, Term = 360, Principle = 300000)return = "1798.65"

pmt = 1798.6515754582765	PMT(Rate, Term, Principle)
pmt 2 decimals = "1798.65"	decimal(pmt,2)

pmt 2 decimals (evaluation result = "1798.65")

- The Boxed Context defines two variable-expression paired entries; these variables serve as 'local variables', which can be used in later expressions
- Return value: the expression can use the value of 'local variables'
- Any expressions in a Boxed Context can use built-in functions that are defined in the customizable Template — *DMN Library*; for example, functions PMT(...) and decimal(...) are used in this example

The simulation result is exactly the same as a Literal Expression:



(Rate = 0.005, Term = 360, Principle = 300000)return = "1798.65"	
pmt = 1798.6515754582765	PMT(Rate, Term, Principle)
pmt 2 decimals = "1798.65"	decimal(pmt,2)
pmt 2 decimals (evaluation result = "1798.65")	

Boxed Context

A Boxed Context is a collection of context entries, presented in the form of a table, followed by a final result expression.

These context entries consist of a variable paired with a value expression and can be thought of as intermediate results. This allows for complex expressions to be decomposed into a series of simple expressions, with the final result being evaluated in a much simpler form.

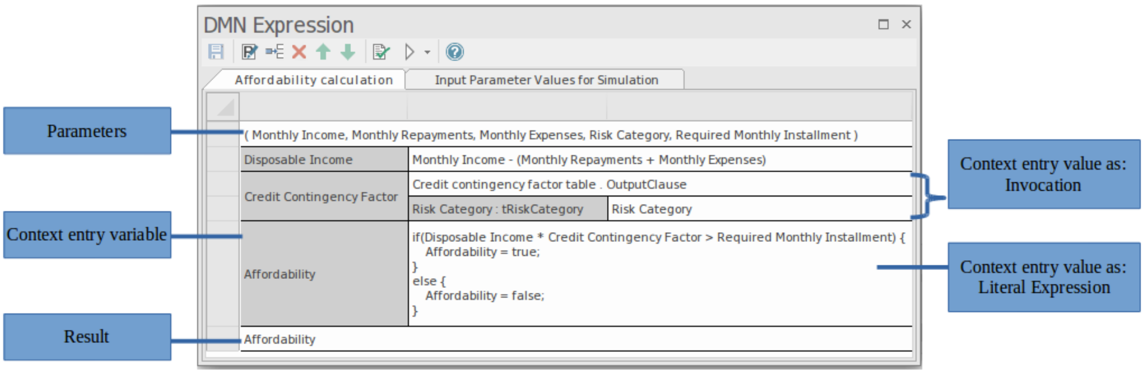
The Boxed Context type is supported in both the *Decision* and the *Business Knowledge Model* element types. It is denoted by the icon.

Access

Diagram	On a diagram, double-click on a Decision element or BKM element. The DMN Expression editor window is displayed, showing details for the selected element.

Overview

This image shows the DMN Expression editor window as it appears for a Boxed Context expression.


DMN Expression
Parameters
Context entry variable
Result
Context entry value as: Invocation
Context entry value as: Literal Expression

Affordability calculation | Input Parameter Values for Simulation

(Monthly Income, Monthly Repayments, Monthly Expenses, Risk Category, Required Monthly Installment)		
Disposable Income	Monthly Income - (Monthly Repayments + Monthly Expenses)	
Credit Contingency Factor	Credit contingency factor table . OutputClause	
	Risk Category : tRiskCategory	Risk Category

Affordability	if(Disposable Income * Credit Contingency Factor > Required Monthly Installment) {
	Affordability = true;
	}
	else {
	Affordability = false;
	}	

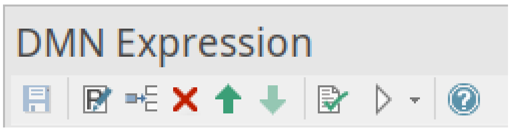
Affordability		

A Boxed Context is a collection of context entries, presented in the form of a table, followed by a final result expression. Each context entry consists of a variable and a value expression. The variable can be considered as an intermediate result, and it can be used within the value expression of any subsequent context entry. The value expression of a context entry can be either a Literal Expression or an Invocation, and can make use of any available inputs such as parameters (to a BKM element), InputData or decision results, as well as any previously defined context variables.

The final result of a Boxed Context expression is determined by working through each context entry in turn, evaluating the value expression and assigning its result to the variable, then finally evaluating the result expression. The result expression can also make use of any input or local variable, but must evaluate to provide a result.

Toolbar for Boxed Context Editor

When a Boxed Context expression is selected, the layout of features accessible in the DMN Expression window is:

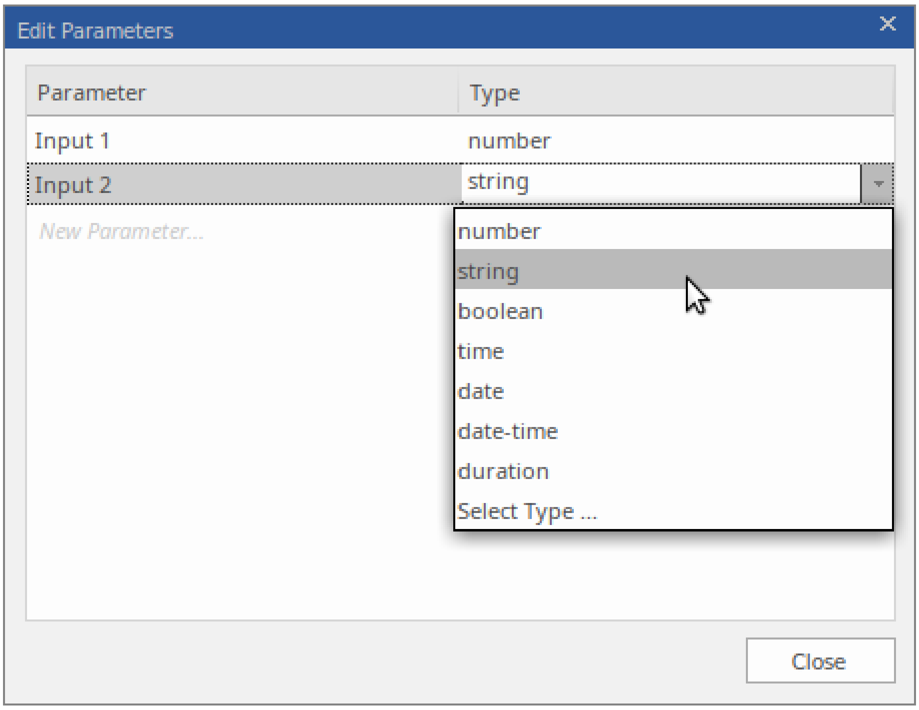


For more details refer to the Help topic '*Toolbar for Boxed Context Editor*'.

Specifying Parameters

In the case of BusinessKnowledgeModel elements, parameters are used to pass input values supplied by the invoking element. The BKM's decision logic is evaluated using the input parameters and the result is returned to the invoking element. By default, a BKM element is created with two input parameters, 'Input 1' and 'Input 2'.

Click on the icon in the toolbar of the DMN Expression window to display the 'Edit Parameters' window.



Here you can change the parameter names, set their data types, create additional parameters or delete existing ones.

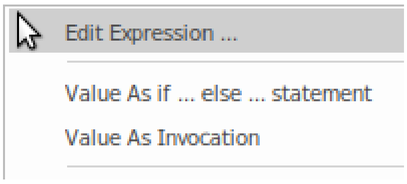
Specifying Context Entries

Each context entry consists of a variable-expression pair.

The variable name can be any text that you like and can even contain spaces. To edit the variable name, click on the cell to select it, then click again or press F2 to enter edit mode. To exit edit mode, click elsewhere or press the Enter key.

In general, it is not necessary to specify a data type for the expression or variables - the type will be inferred from the value. However, if you intend to generate code for compiled languages such as Java, C++ or C#, you will have to specify the type of all context entry variables.

The value expression of a context entry can be either a Literal Expression or an Invocation and can make use of any available inputs, such as parameters (to a Business Knowledge Model element), InputData or decision results, as well as any previously defined context variables. Right-clicking on the expression cell displays a pop-up menu that provides options for displaying an expression code editor, or for setting the value expression as an If-Else statement or an Invocation.



You can also edit the value expression by entering text directly into the expression cell.

For further information on how to specify Literal Expressions or Invocations, please see the Help topics covering those subjects.

Toolbar for Boxed Context Editor

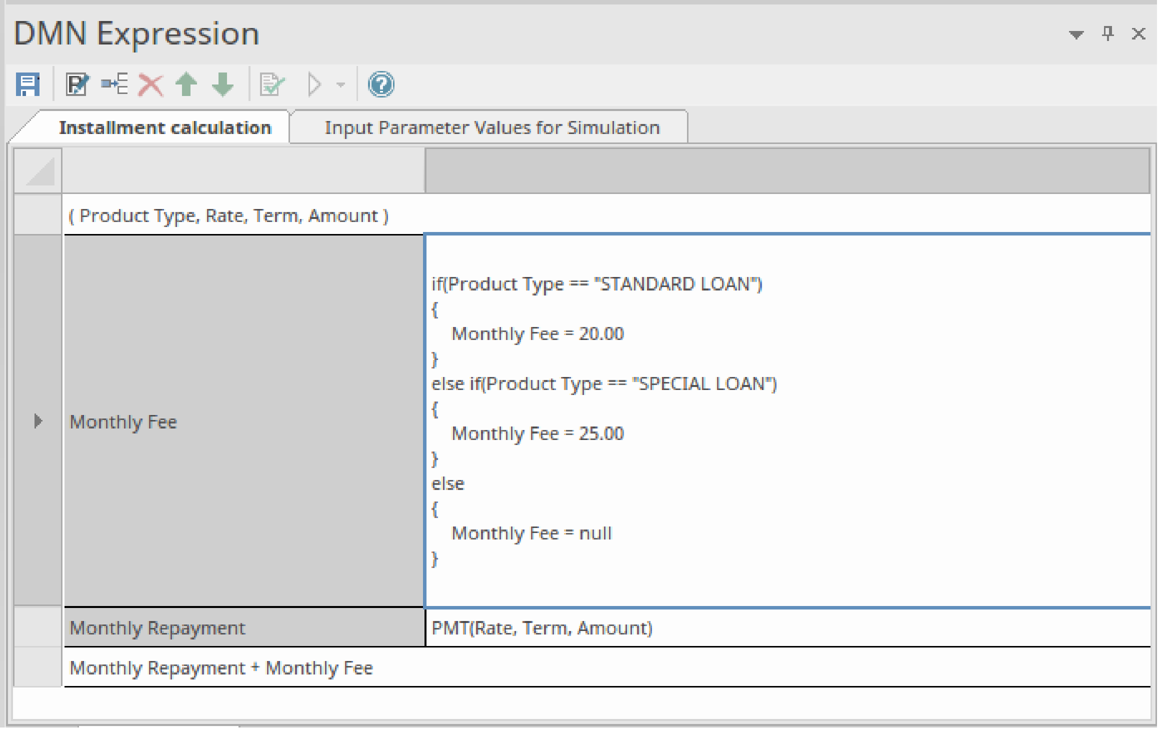
This table provides descriptions of the features accessible in the DMN Expression window when a Boxed Context is selected.

Toolbar Options

This toolbar is for Boxed Context.

Options	Description
💾	Save changes to the currently selected Decision or BusinessKnowledgeModel element.
✎	Display the 'Edit Parameters' window, where you can specify the name and data type of each parameter that is passed when invoking the decision logic of a BusinessKnowledgeModel element.
⇥	Create a new context entry and append it to the list of context entries.
✕	Delete the currently selected context entry.
⬆	Move the currently selected context entry up one position in the list.
⬇	Move the currently selected context entry down one position in the list.
✓	Perform validation of the BoxedContext. Enterprise Architect will perform a series of validations to help you discover any errors in the BoxedContext definition.
▷	This button is enabled when a Decision Table is defined for a Business Knowledge Model element. Select the 'Input Parameter Values for Simulation' tab, complete the fields, then click on this button. The test result will be presented on the Decision Table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a BusinessKnowledgeModel element, without specifying its context. A number of menu options are available for this tool bar button. For more information, see the Help topic *Simulate DMN Model.*

Example - Loan Installment Calculation



(Product Type, Rate, Term, Amount)

Monthly Fee	if(Product Type == "STANDARD LOAN")
	{
	Monthly Fee = 20.00
	}
	else if(Product Type == "SPECIAL LOAN")
	{
	Monthly Fee = 25.00
	}
	else
	{
	Monthly Fee = null
	}

Monthly Repayment	PMT(Rate, Term, Amount)
Monthly Repayment + Monthly Fee	

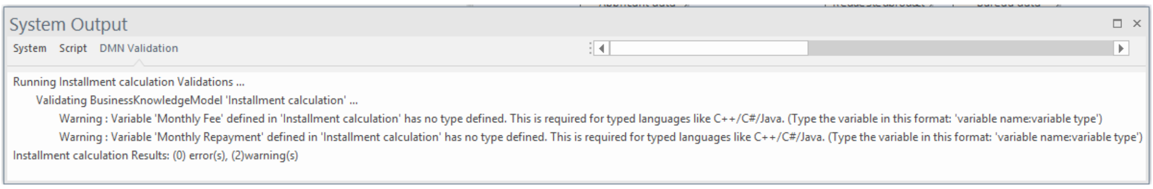
The Business Knowledge Model (BKM) *Installment calculation* is implemented as Boxed Context.

- The BKM defines four parameters: Product Type, Rate, Term and Amount
- The Boxed Context defines two variable-expression pair entries; these variables serve as 'local variables' that can be used in later expressions
- Return value: The expression can use the value of 'local variables'
- Any expressions in a Boxed Context can use built-in functions, which are defined in the customizable Template — *DMN Library*; the functions PMT(...) and decimal(...) are used in this example

Specify Type to Context Entry Variable

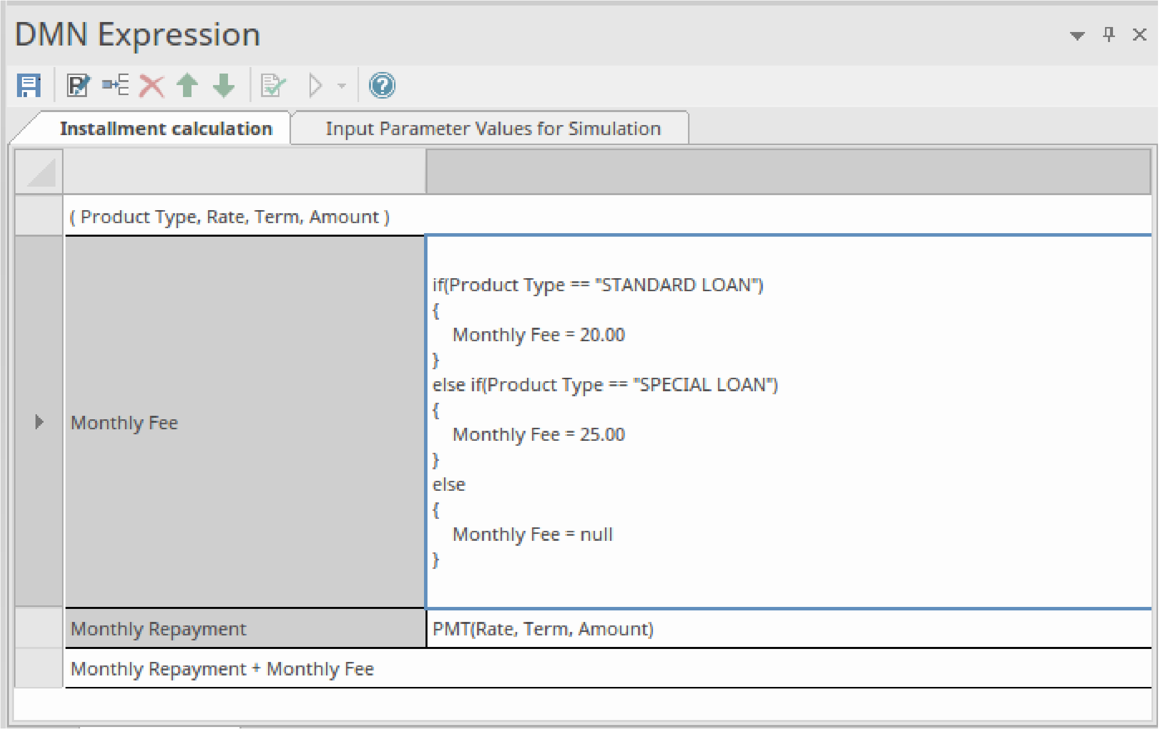
In general, the expression and variables do not have to specify a type, which is inferred from the value provided. This feature is supported generically by JavaScript, which is used for Enterprise Architect's DMN Simulation.

However, if you want to generate code from a DMN model to compiled languages such as Java, C++ or C#, you will have to specify the type for each Context Entry Variable. Otherwise, if you validate the model, you will see warnings such as:



```
Running Installment calculation Validations ...
    Validating BusinessKnowledgeModel 'Installment calculation' ...
        Warning : Variable 'Monthly Fee' defined in 'Installment calculation' has no type defined. This is required for typed languages like C++/C#/Java. (Type the variable in this format: 'variable name:variable type')
        Warning : Variable 'Monthly Repayment' defined in 'Installment calculation' has no type defined. This is required for typed languages like C++/C#/Java. (Type the variable in this format: 'variable name:variable type')
Installment calculation Results: (0) error(s), (2)warning(s)
```

Right-click on the Context Entry Variable (Monthly Fee, Monthly Repayment) in this model.



(Product Type, Rate, Term, Amount)

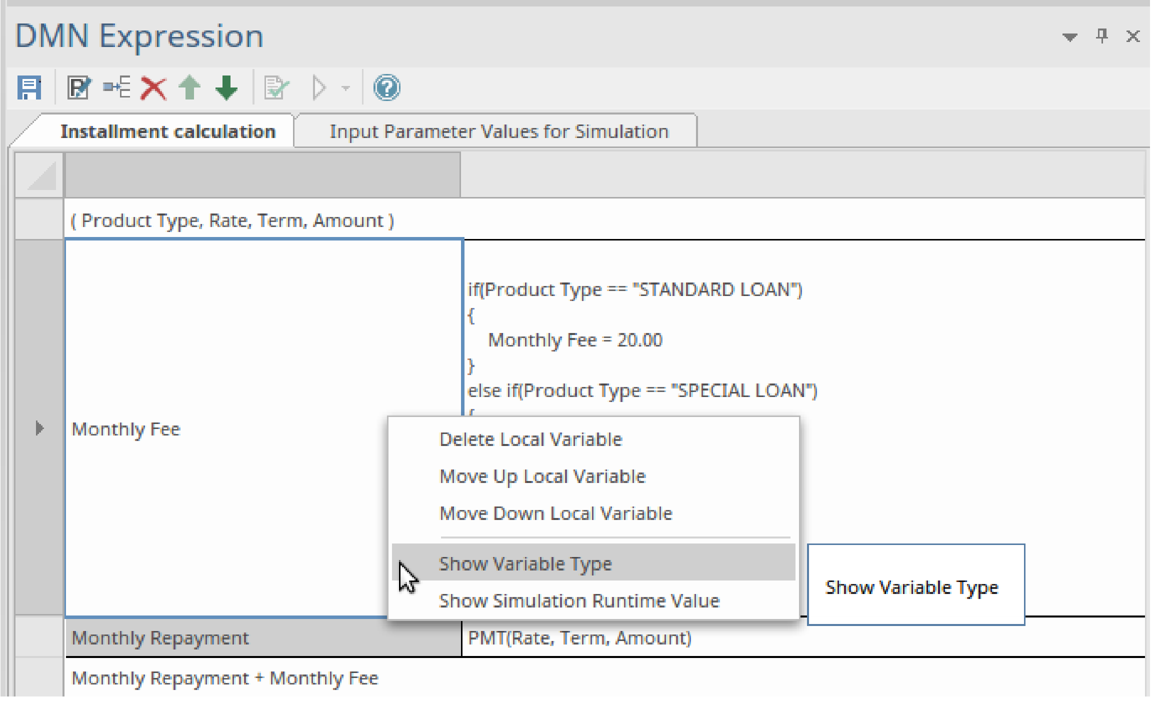
Monthly Fee

```
if(Product Type == "STANDARD LOAN")
{
    Monthly Fee = 20.00
}
else if(Product Type == "SPECIAL LOAN")
{
    Monthly Fee = 25.00
}
else
{
    Monthly Fee = null
}
```

Monthly Repayment: PMT(Rate, Term, Amount)

Monthly Repayment + Monthly Fee

Select the 'Show Variable Type' option.



(Product Type, Rate, Term, Amount)

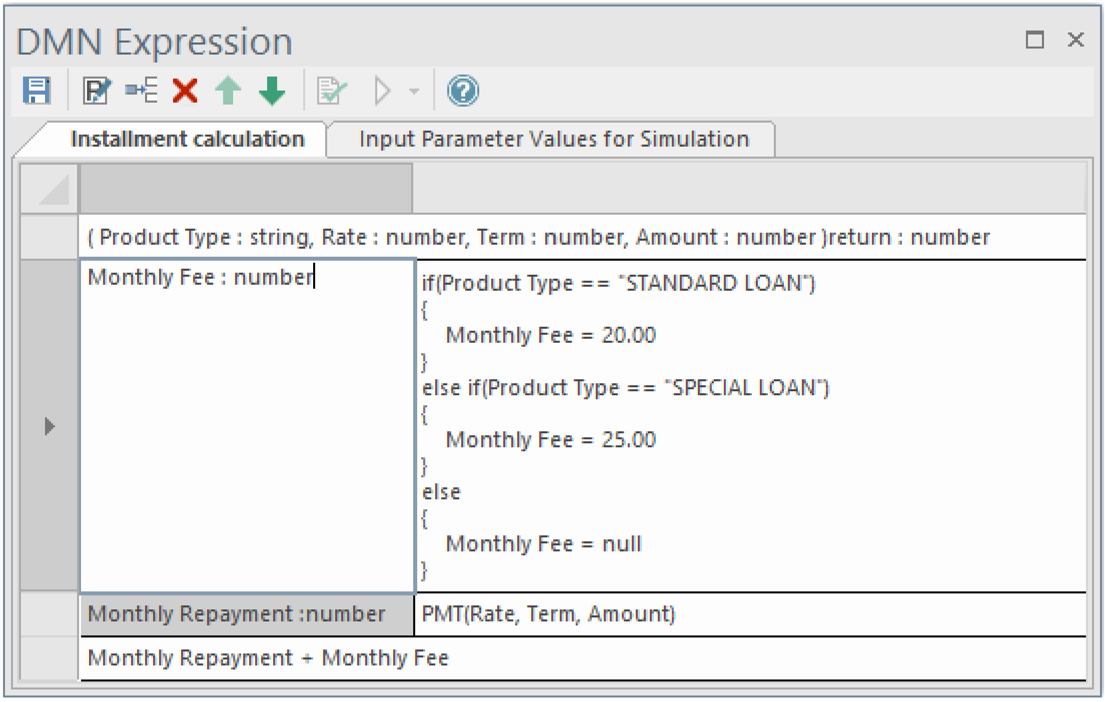
Monthly Fee

```
if(Product Type == "STANDARD LOAN")
{
    Monthly Fee = 20.00
}
else if(Product Type == "SPECIAL LOAN")
{
```

Monthly Repayment: PMT(Rate, Term, Amount)

Monthly Repayment + Monthly Fee

Now type in the variable type, appending it to the variable name and separated by a colon, as shown here.



(Product Type : string, Rate : number, Term : number, Amount : number)return : number	

Monthly Fee : number	if(Product Type == "STANDARD LOAN")
	{
	Monthly Fee = 20.00
	}
	else if(Product Type == "SPECIAL LOAN")
	{
	Monthly Fee = 25.00
	}
	else
	{
	Monthly Fee = null
	}

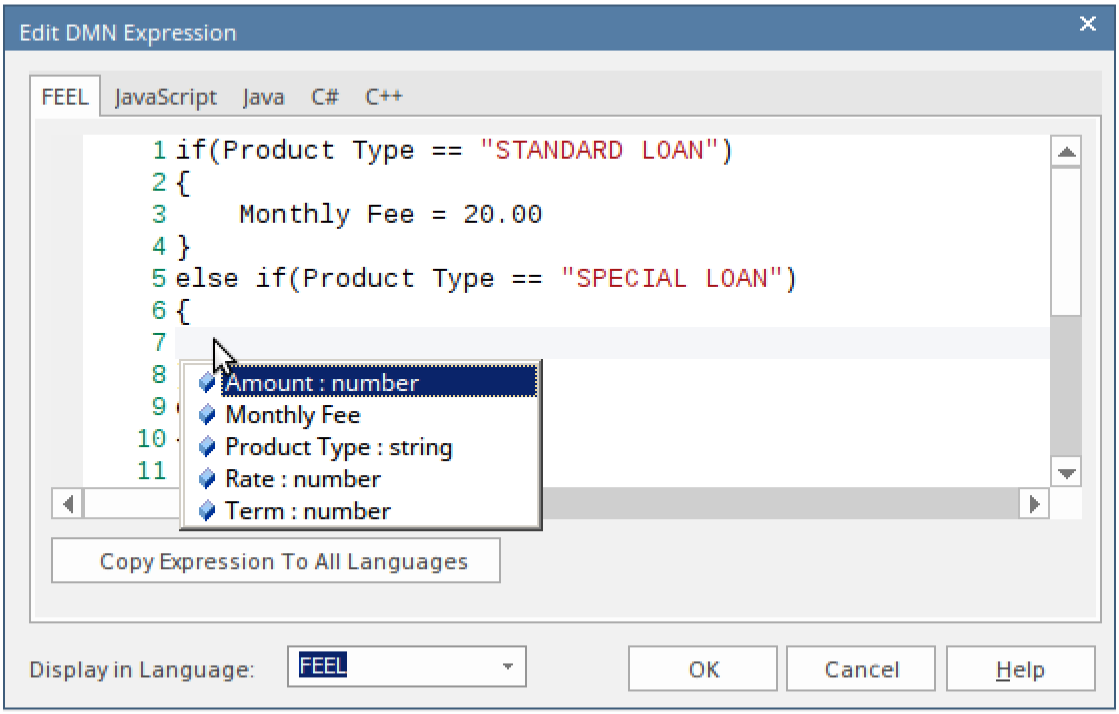
Monthly Repayment :number	PMT(Rate, Term, Amount)
Monthly Repayment + Monthly Fee	

Then click on the Save button on the toolbar to save the expression, and click on the button to validate the model again.

Expression Editor and Intelli-sense Support

The parameter and Context Entry's variable name can contain spaces, according to the FEEL language specification. This feature makes the expression easy to read. In order to help you edit the expressions with less typing and making fewer mistakes, Enterprise Architect provides Intelli-sense support for editing expressions:

To edit an expression, right-click on the expression (in the right-hand field) and select the 'Edit Expressions' menu option. The 'Expression' dialog displays. Click on the required line and press Ctrl+Space to show the Intelli-sense menu:



```
if(Product Type == "STANDARD LOAN")
{
    Monthly Fee = 20.00
}
else if(Product Type == "SPECIAL LOAN")
{
```

Display in Language: FEEL

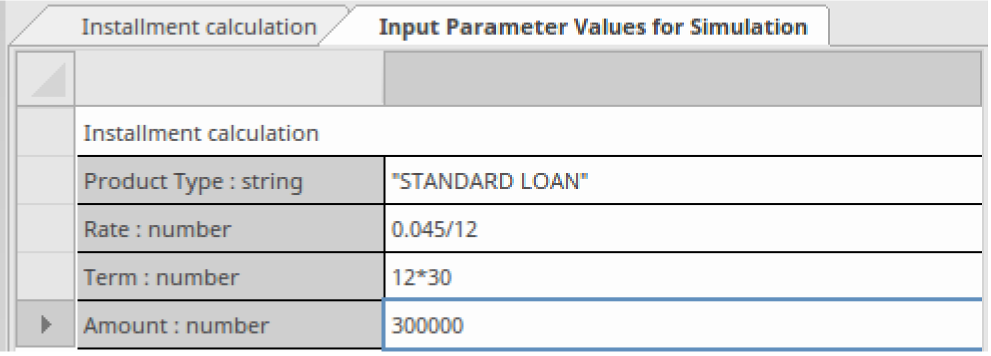
- All the Context Entry Variables earlier than the current one will be included (the Context Entry Variables later than the current one are excluded)
- For a Business Knowledge Model (BKM), all the parameters will be included
- For a Decision, all the required Decisions will be included

The DMN model can be generated as source code for JavaScript, Java, C# and C++. Since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

In the generated code, the space inside a variable name will be replaced by an underscore.

Simulation of Business Knowledge Model

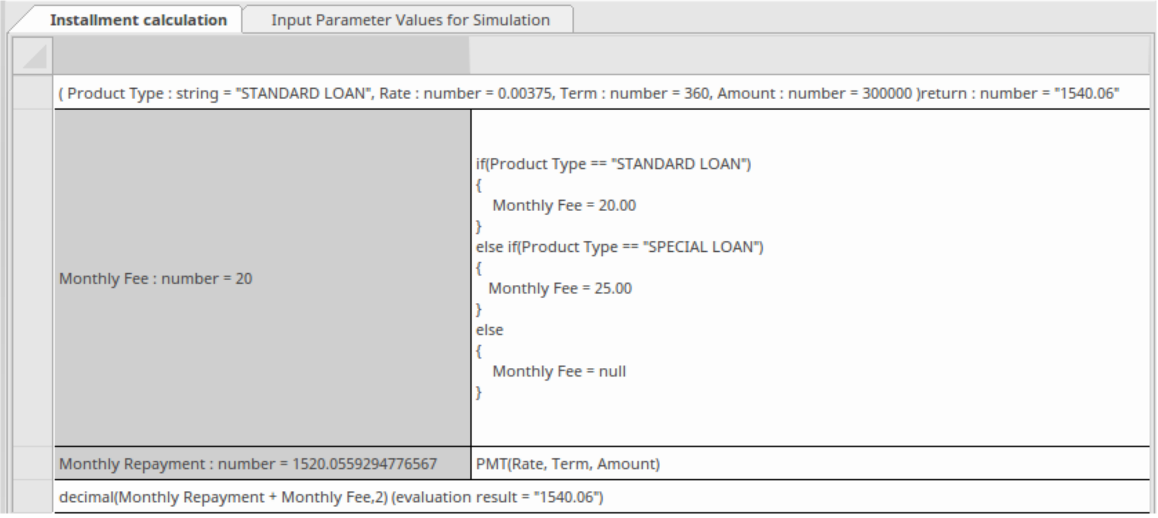
Select the 'Input Parameter Values for Simulation' tab and complete each field.



Installment calculation

Product Type : string	"STANDARD LOAN"
Rate : number	0.045/12
Term : number	12*30
Amount : number	300000

Click on the Save button and then on the Simulation button on the toolbar; the test result will be presented in the Boxed Context expression.



(Product Type : string = "STANDARD LOAN", Rate : number = 0.00375, Term : number = 360, Amount : number = 300000)return : number = "1540.06"

Monthly Fee : number = 20	if(Product Type == "STANDARD LOAN")
	{
	Monthly Fee = 20.00
	}
	else if(Product Type == "SPECIAL LOAN")
	{
	Monthly Fee = 25.00
	}
	else
	{
	Monthly Fee = null
	}

Monthly Repayment : number = 1520.0559294776567	PMT(Rate, Term, Amount)

decimal(Monthly Repayment + Monthly Fee,2) (evaluation result = "1540.06")

- The runtime parameter value will be displayed; for example, 'Rate = 0.00375'
- The 'Context Entry' variable's runtime value will be displayed; for example, 'Monthly Repayment = 1520.06'
- The Business Knowledge Model (BKM)'s result will be evaluated by the last entry and the values displayed on the declaration line; for example, 'return = 1540.06'

You can use this functionality to unit test a BKM without knowing the context so that later on it can be invoked by a Decision or another BKM.

Boxed List

A DMN Boxed List is a Decision element that contains a list of boxed expressions. These items are arranged as a vertical list in the DMN Expression window.

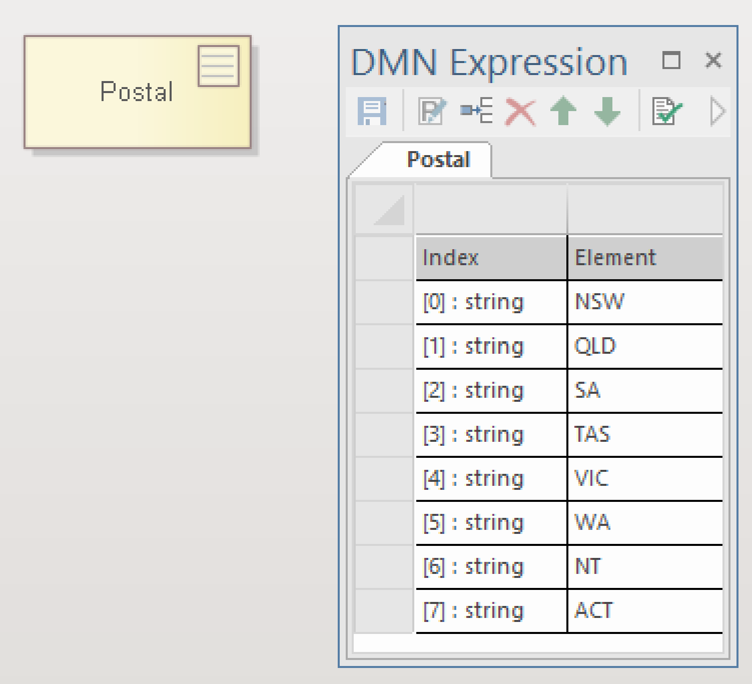
A Boxed List is often used in conjunction with a *for loop* expression that is contained in a related Decision element. The *for loop* expression is used to iterate over each row in the Boxed List, binding the List's Element-field to the corresponding variable, and evaluating the expression in the scope. The output of the *for loop* is a list containing the evaluation of the expression for each individual iteration.

Access

Diagram Toolbox	Drag a Decision element or BKM element from the Toolbox onto a DMN diagram, and select 'List' from the pop-up expression menu. Double-click on the DMN element; the DMN Expression window is displayed, showing details of the selected element.
Properties	Right-click on a DMN Decision or BKM element on the diagram, and select the 'Properties | Properties' menu option. On the General page, select the 'Tags' tab, and in the 'expressionType' value field click on the drop-down arrow and select 'List'. Click on the OK button. Double click on the DMN element; the DMN Expression window is displayed, showing details for the selected element.

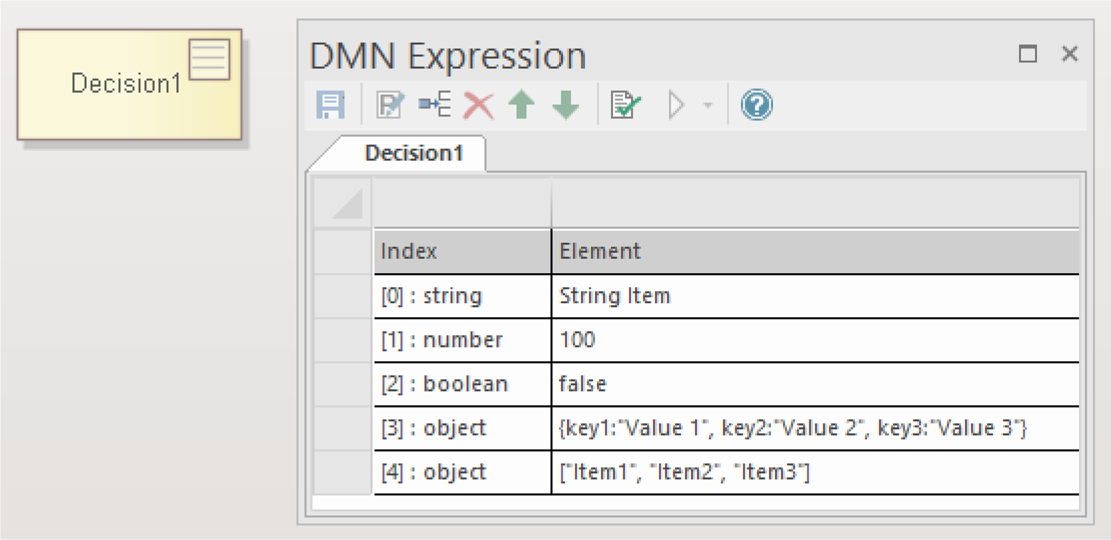
Overview

It is common for Boxed Lists to be used as Enumerations, where all List items are of the same type.



Index	Element
[0] : string	NSW
[1] : string	QLD
[2] : string	SA
[3] : string	TAS
[4] : string	VIC
[5] : string	WA
[6] : string	NT
[7] : string	ACT

It is also common for Boxed Lists to be used as a collection of data, where each item might have a different type.



Index	Element
[0] : string	String Item
[1] : number	100
[2] : boolean	false
[3] : object	{key1:"Value 1", key2:"Value 2", key3:"Value 3"}
[4] : object	["Item1", "Item2", "Item3"]

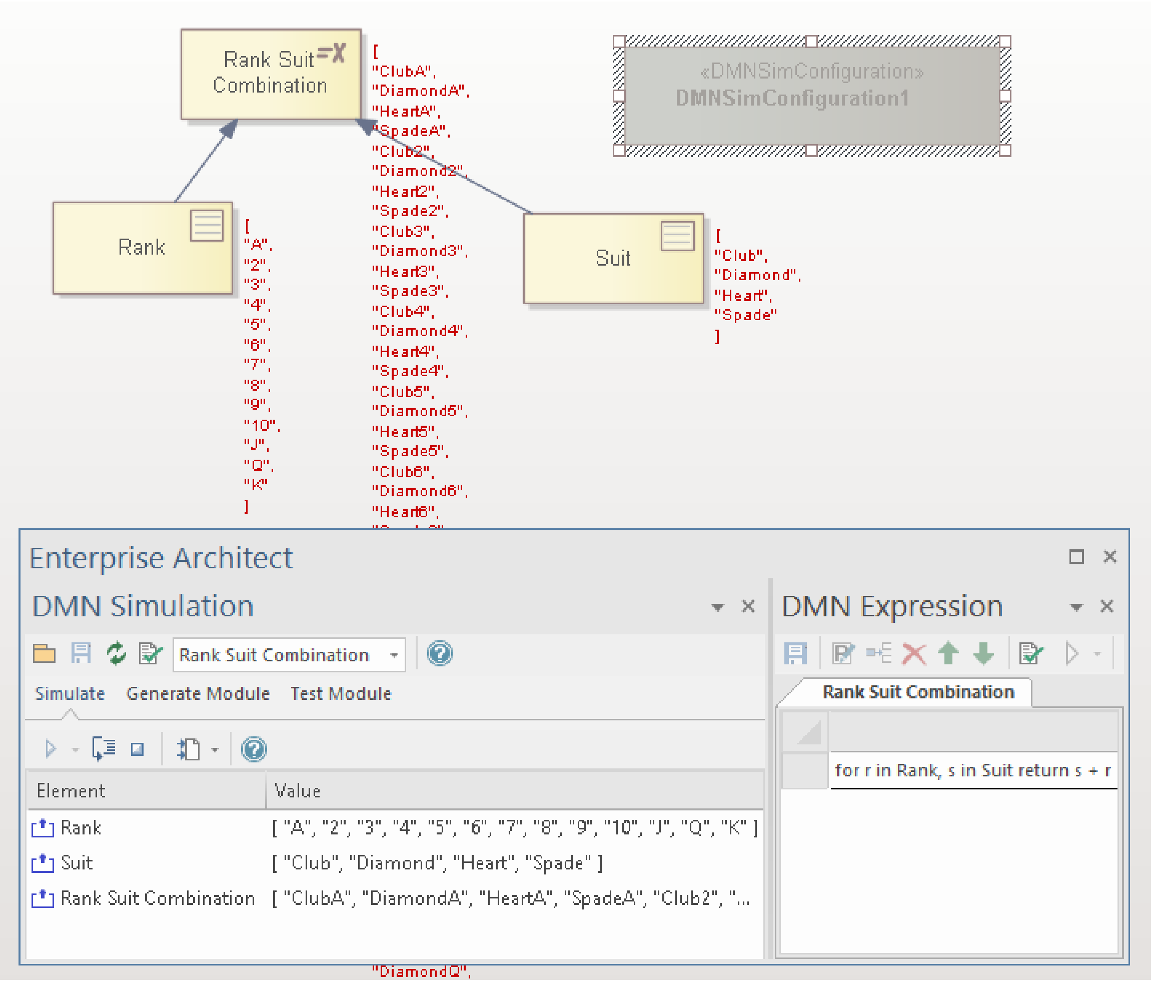
Editing Boxed Lists

The DMN Expression window has a toolbar providing the options 'Add New List Item', 'Delete Existing List Item' and 'Move Item Up or Down'.

Right-click on a List item to display context menu options for setting the List item's type - string, number, boolean or object.

Example - Poker Rank and Suit

In this example, we have three Decisions: Rank, Suit and Rank Suit Combination


Rank Suit Combination
Rank
Suit
«DMNSimConfiguration»
DMNSimConfiguration1
["ClubA", "DiamondA", "HeartA", "SpadeA", "Club2", "Diamond2", "Heart2", "Spade2", "Club3", "Diamond3", "Heart3", "Spade3", "Club4", "Diamond4", "Heart4", "Spade4", "Club5", "Diamond5", "Heart5", "Spade5", "Club6", "Diamond6", "Heart6",
"DiamondQ",
["A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
["Club", "Diamond", "Heart", "Spade"]
Enterprise Architect
DMN Simulation
Rank Suit Combination
Simulate
Generate Module
Test Module
DMN Expression
Rank Suit Combination

Element	Value
Rank	["A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
Suit	["Club", "Diamond", "Heart", "Spade"]
Rank Suit Combination	["ClubA", "DiamondA", "HeartA", "SpadeA", "Club2", "...

for r in Rank, s in Suit return s + r

- Decision Rank is represented as a Boxed List with 13 Items from 'A' through to 'K'
- Decision Suit is represented as a Boxed List with 4 Items: 'Club', 'Diamond', 'Heart' and 'Spade'
- Decision Rank Suit Combination is represented as a Literal Expression with a *for* loop: *for r in Rank, s in Suit return s + r*

When multiple iteration contexts are defined in the same *for* loop expression, the resulting iteration is a cross-product of the elements of the iteration contexts. The iteration order is from the inner iteration context to the outer iteration context.

In this example, the cross-product of Rank (13 items) and Suit (4 items) is a list of 13 * 4 = 52 elements.

Relation

A DMN Decision Relation element provides a convenient shorthand method for defining a list of related values in a DMN diagram. A Decision Relation is like a relation table with columns and rows. The header of the grid displays the name of each column. Each row displays the set of values for the corresponding columns.

Access

Toolbox	To create a Decision Relation: • Ensure the Perspective is set to: Requirements > Decision Modeling • From the DMN Components page of the Diagram Toolbox, drag a Decision element or BKM element onto a DMN diagram • Select 'Relation' from the pop-up expression menu • Double-click on the DMN element to display the DMN Expression window.

Alternatively, you can change an existing DMN Decision or BKM element to a Decision-Relation type. To do this:

- Right-click on a DMN Decision or BKM element on the diagram, and select the 'Properties | Properties' menu option
- On the General page, select the 'Tags' tab, and in the 'expressionType' value field click on the drop-down arrow and select 'Relation'; click on the OK button

Overview

The DMN *Relation* Decision-type is a vertical list containing rows of values. A key to using the Decision Relation is the means to iterate through the rows of values using a *For Loop*. The For Loop can be defined in, say, a related Literal Expression Decision as a formula to process the rows in the Decision-Relation element.

Editing DMN Relations

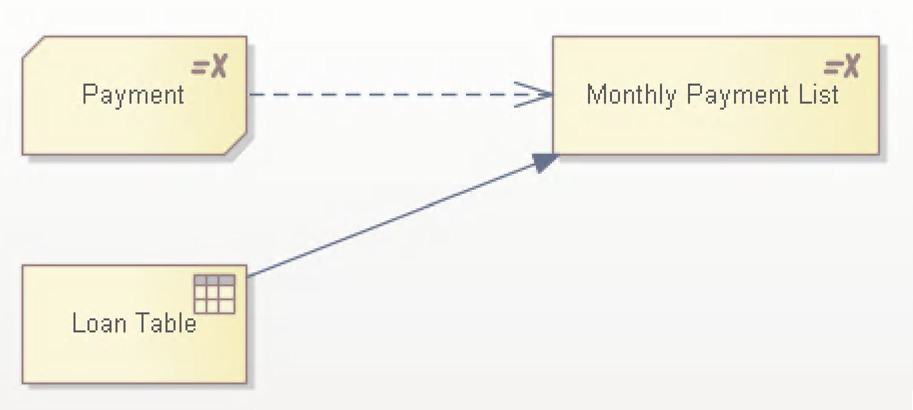
The DMN Expression window has a toolbar providing options to add a new row, delete an existing row and move the selected row up or down.

You can also:

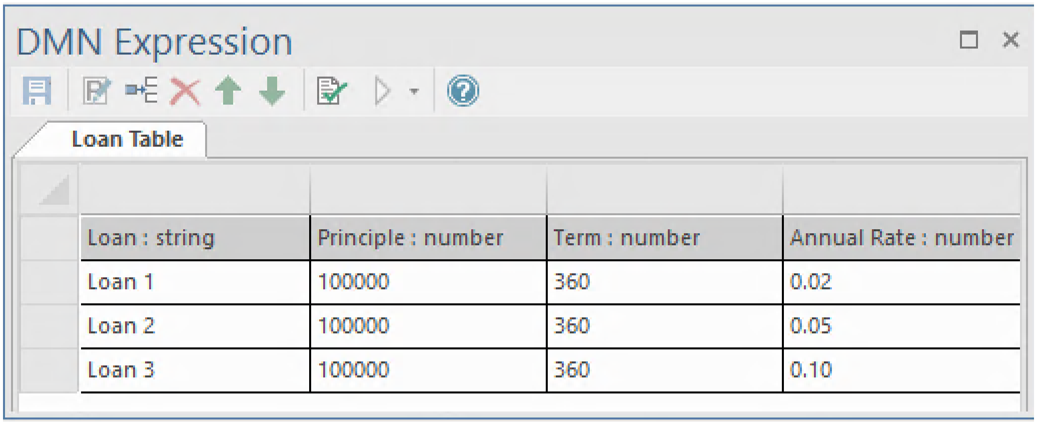
- Drag the grid header to reposition columns.
- Right-click on the header cells to display the context menu options for setting the column's Type to: 'string', 'number', 'boolean' or 'object'.

Example - Loan Table

In this example, we have two Decisions - 'Loan Table' and 'Monthly Payment List' - and one Business Knowledge Model - 'Payment'.

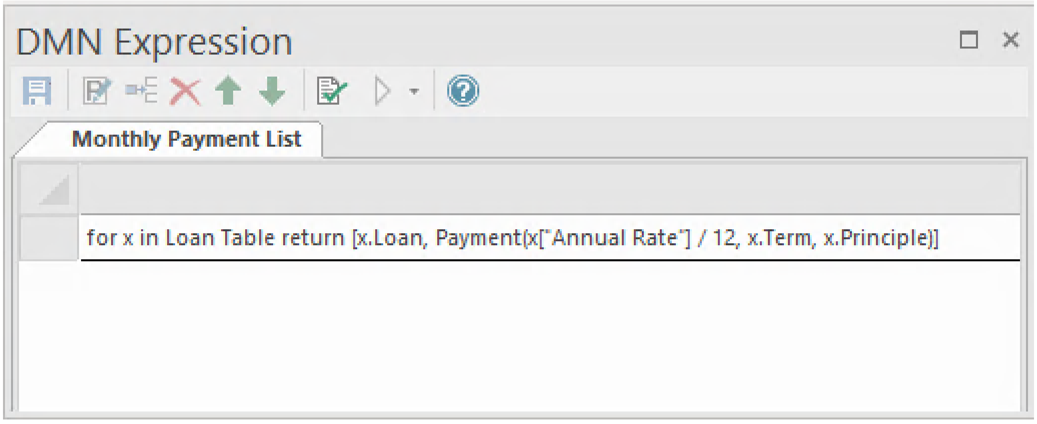


'Loan Table' is a Decision implemented as a Relation with four columns: 'Loan', 'Principle', 'Term' and 'Annual Rate'.



Loan : string	Principle : number	Term : number	Annual Rate : number
Loan 1	100000	360	0.02
Loan 2	100000	360	0.05
Loan 3	100000	360	0.10

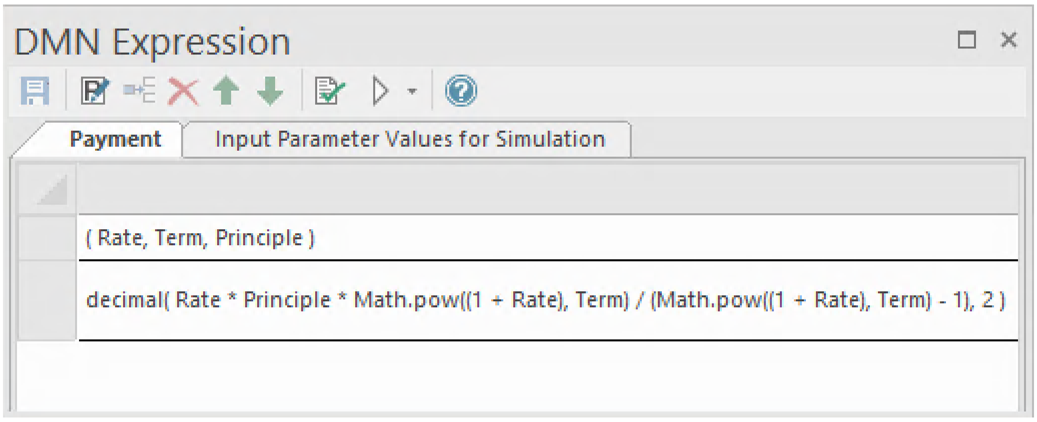
'Monthly Payment List' is a Decision implemented as a Literal Expression with a *for* loop:



```
for x in Loan Table return [x.Loan, Payment(x["Annual Rate"] / 12, x.Term, x.Principle)]
```

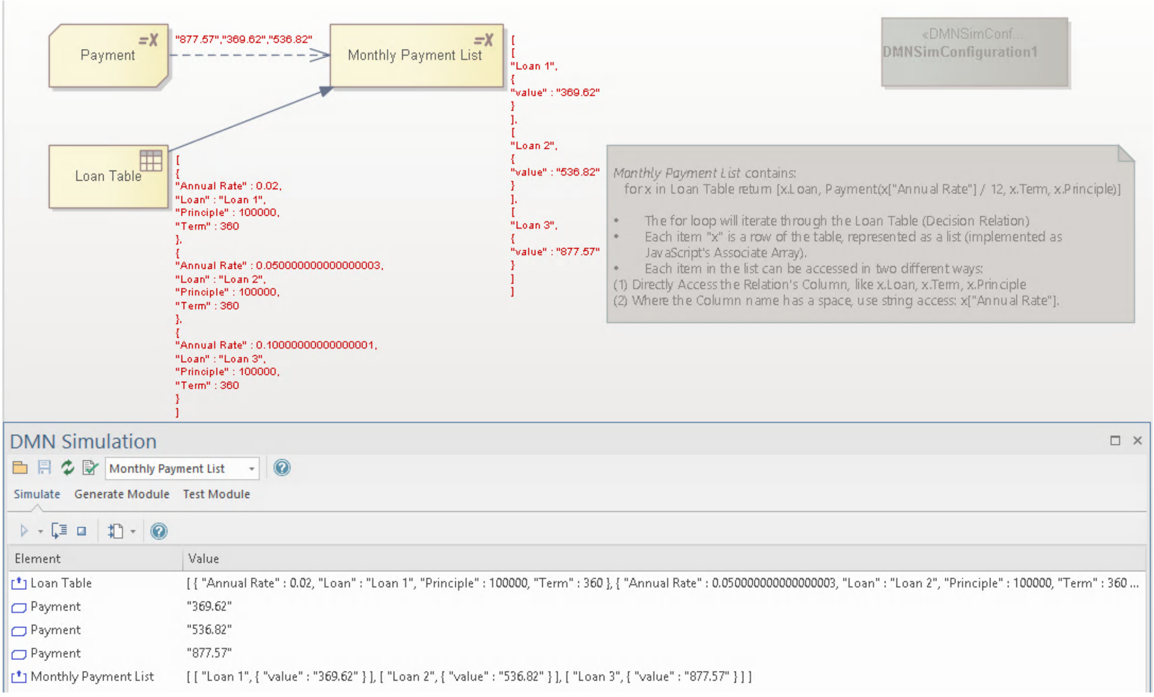
The *for* loop will iterate through 'Loan Table':

- Each item 'x' is a row of the table, represented as a list
- With each row item 'x', invoke Business Knowledge Model 'Payment' with the items in the row list
- Each item in the list can be accessed in two different ways:
  (1) Directly access the Relation's column, like x.Loan, x.Term, x.Principle
  (2) Where the column name has a space, use string access: x["Annual Rate"]



```
( Rate, Term, Principle )
decimal( Rate * Principle * Math.pow((1 + Rate), Term) / (Math.pow((1 + Rate), Term) - 1), 2 )
```

On simulation, the runtime values show in the Simulation window and on the diagram beside the element; you can click on a step to see the simulation process.



```
"877.57","369.62","536.82"
```

```
[
[
"Loan 1",
{
"value" : "369.62"
}
],
[
"Loan 2",
{
"value" : "536.82"
}
],
[
"Loan 3",
{
"value" : "877.57"
}
]
]
```

```
[
{
"Annual Rate" : 0.02,
"Loan" : "Loan 1",
"Principle" : 100000,
"Term" : 360
},
{
"Annual Rate" : 0.050000000000000003,
"Loan" : "Loan 2",
"Principle" : 100000,
"Term" : 360
},
{
"Annual Rate" : 0.10000000000000001,
"Loan" : "Loan 3",
"Principle" : 100000,
"Term" : 360
}
]
```

Monthly Payment List contains:
for x in Loan Table return [x.Loan, Payment(x["Annual Rate"] / 12, x.Term, x.Principle)]

- The for loop will iterate through the Loan Table (Decision Relation)
- Each item "x" is a row of the table, represented as a list (implemented as JavaScript's Associate Array).
- Each item in the list can be accessed in two different ways:

(1) Directly Access the Relation's Column, like x.Loan, x.Term, x.Principle
(2) Where the Column name has a space, use string access: x["Annual Rate"].

Element	Value
Loan Table	[{ "Annual Rate" : 0.02, "Loan" : "Loan 1", "Principle" : 100000, "Term" : 360 }, { "Annual Rate" : 0.050000000000000003, "Loan" : "Loan 2", "Principle" : 100000, "Term" : 360 ...
Payment	"369.62"
Payment	"536.82"
Payment	"877.57"
Monthly Payment List	[["Loan 1", { "value" : "369.62" }], ["Loan 2", { "value" : "536.82" }], ["Loan 3", { "value" : "877.57" }]]

Invocation

An invocation is a container for the parameter bindings that provide the context for the evaluation of the body of a Business Knowledge Model. There are two common use cases for an Invocation:

- Bind an Input Data to the Business Knowledge Model
- Bind parameters or context entry variables to the Business Knowledge Model

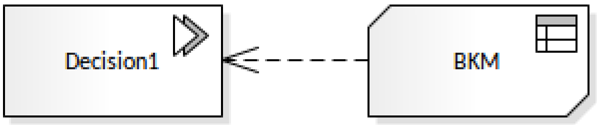
An example of each is provided in the sub-topics of this Help topic.

Access

Diagram	Double-click on the appropriate Decision element or BKM element. The DMN Expression window displays, showing details for the selected element.

Overview

An Invocation is a type of value expression applicable to both Decision elements and Business Knowledge Model elements. It is a tabular representation of how decision logic defined within an invocable element (a Business Knowledge Model or a Decision Service) is invoked by a Decision or by another Business Knowledge Model.



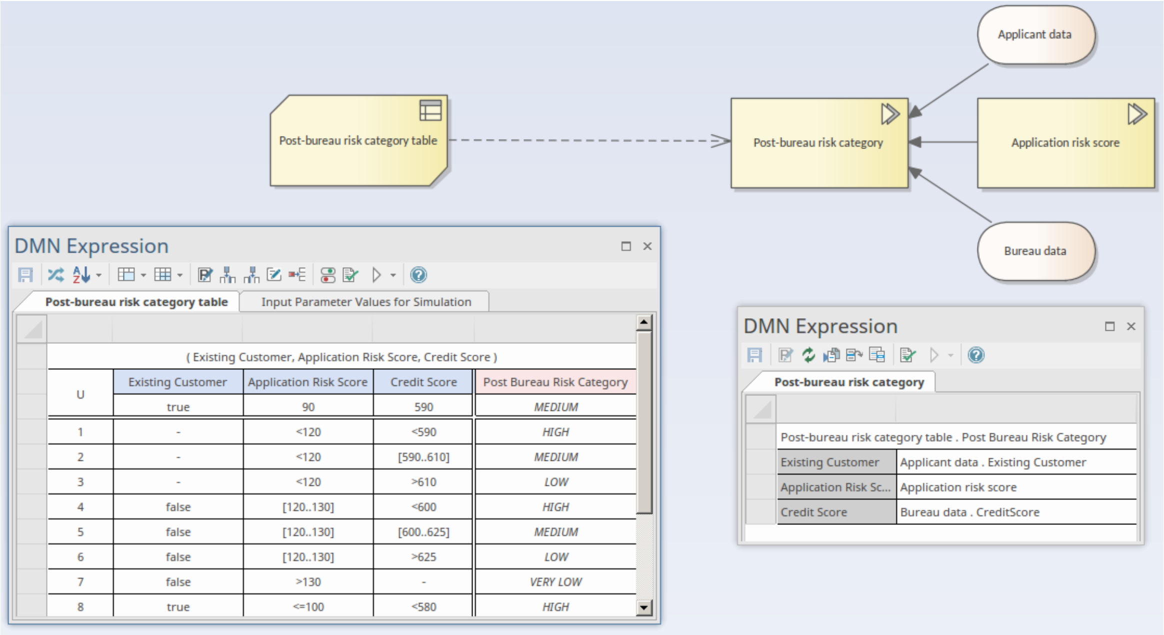
Toolbar for Invocation Editor

When an Invocation is selected, a number of facilities for working on it are accessible from the toolbar of the DMN Expression window:

For more details refer to the Help topic '*Toolbar for Invocation Editor*'.

Bindings

The parameter bindings of an Invocation provide the context for evaluation of the body of the invocable element.


Applicant data
Post-bureau risk category table
Post-bureau risk category
Application risk score
Bureau data
DMN Expression
Post-bureau risk category table
Input Parameter Values for Simulation
DMN Expression
Post-bureau risk category

(Existing Customer, Application Risk Score, Credit Score)

U	Existing Customer	Application Risk Score	Credit Score	Post Bureau Risk Category
	true	90	590	*MEDIUM*
1	-	<120	<590	*HIGH*
2	-	<120	[590..610]	*MEDIUM*
3	-	<120	>610	*LOW*
4	false	[120..130]	<600	*HIGH*
5	false	[120..130]	[600..625]	*MEDIUM*
6	false	[120..130]	>625	*LOW*
7	false	>130	-	*VERY LOW*
8	true	<=100	<580	*HIGH*

Post-bureau risk category table . Post Bureau Risk Category

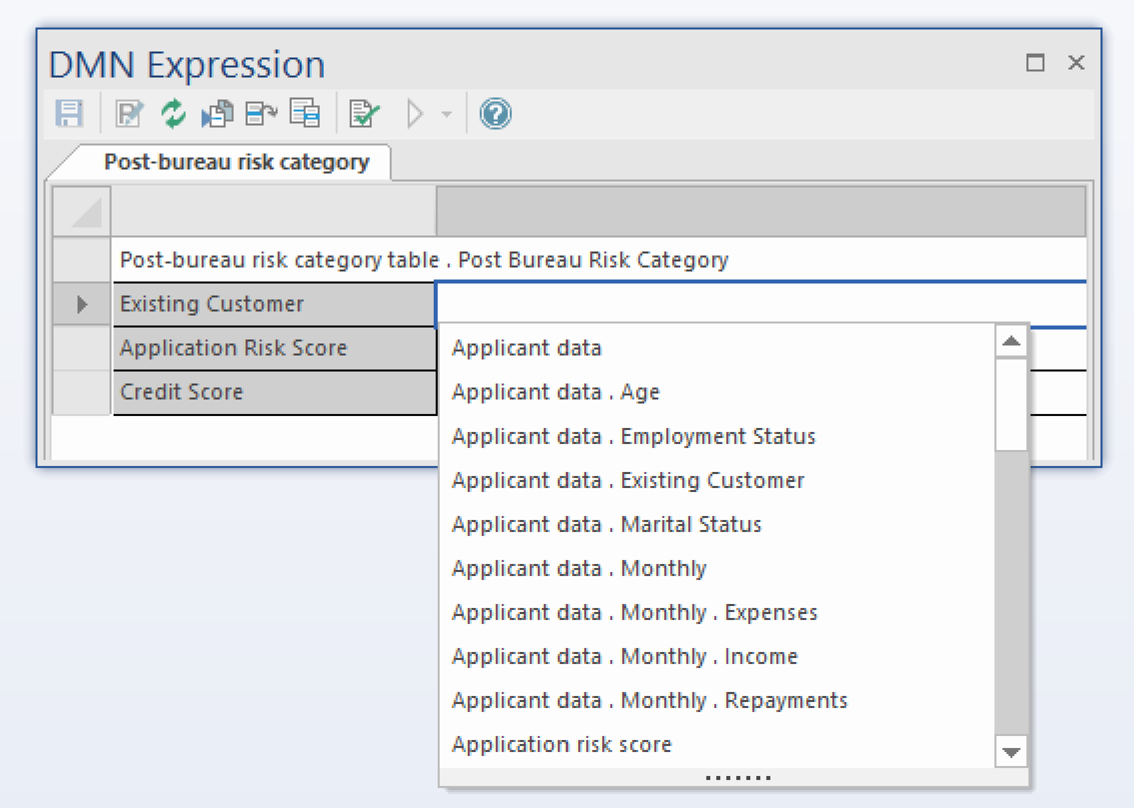
Existing Customer	Applicant data . Existing Customer
Application Risk Sc...	Application risk score
Credit Score	Bureau data . CreditScore

In this example:

- The Decision 'Post-bureau risk category' is represented as an Invocation connecting to the Business Knowledge Model 'Post-bureau risk category table', implemented as a Decision Table
- The Decision 'Post-bureau risk category' is the target of three Information Requirement connectors from two Input Data elements and one Decision element
- The binding list binds the input values to the Business Knowledge Model's parameters
- The Invocation also specifies the requested 'OutputClause'; in the case where a Decision Table has multiple Output Clauses defined, the Invocation must explicitly request an Output Clause as the result of the expression

Inputs

Inputs from other Decisions and InputData elements can be set by pressing the Spacebar in the field:



Post-bureau risk category table . Post Bureau Risk Category

Existing Customer

Application Risk Score

Credit Score

Applicant data

Applicant data . Age

Applicant data . Employment Status

Applicant data . Existing Customer

Applicant data . Marital Status

Applicant data . Monthly

Applicant data . Monthly . Expenses

Applicant data . Monthly . Income

Applicant data . Monthly . Repayments

Application risk score

Output

As an Invocation can only invoke one Business Knowledge Model, the output is defined by the Business Knowledge Model output.

Toolbar for Invocation Editor

When an Invocation expression is selected, the DMN Expression window toolbar provides options specific to that expression type.

Toolbar Options

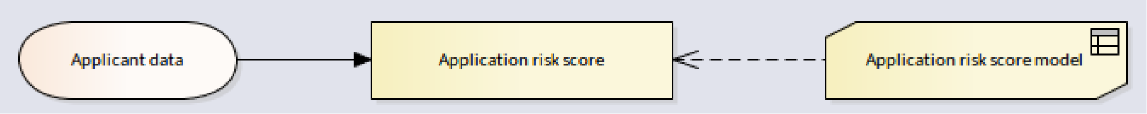
This table provides descriptions of the features accessible in the DMN Expression window when an Invocation is selected.

Options	Description
💾	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
✎	Click on this button to edit parameters for the Business Knowledge Model.
🔄	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. Click on this button to synchronize with the invoked BKM. For example, if the BKM changes name, parameters, outputs or types, click on this button to synchronize these changes.
📋	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. Click on this button to set or change a BKM as an invocation.
↪	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. Click on this button to open the invoked BKM in the DMN Expression window.
☰	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. When a BKM is implemented as a Decision Table, it could define multiple Output Clauses; the invocation on this BKM might have to specify which output is requested. Click on this button to list all the available outputs in a context menu; the currently configured output is checked.
✓	Perform validation of the Invocation. Enterprise Architect will perform a series of validations to help you locate any errors in the Invocation definition.
▷	This button is enabled when the Invocation is defined for a Business Knowledge Model. Select the 'Input Parameter Values for Simulation' tab, complete the fields and click on this button. The test result will be presented on the Decision Table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a Business Knowledge Model without knowing the context and later on invoked by a Decision or other Business Knowledge Model. Menu options are available for this toolbar button. For more information, see the

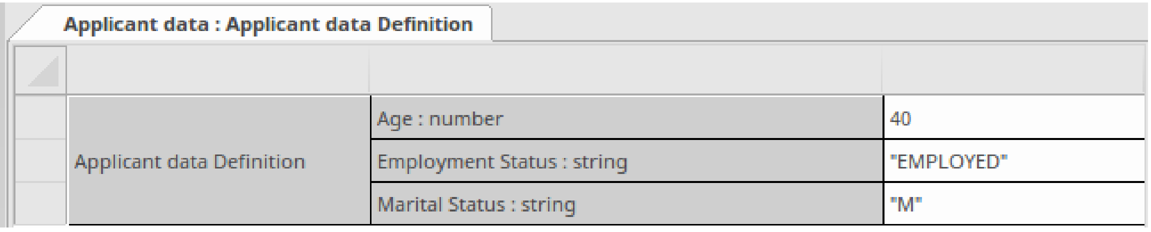
	Simulate DMN Model Help topic.

Example 1 - Bind Input Data to Business Knowledge Model

A full example can be created with a Model Pattern (in the ribbon, select 'Simulate > Decision Analysis > DMN > Apply Perspective > DMN Decision > Decision With BKM : Create Model(s)').



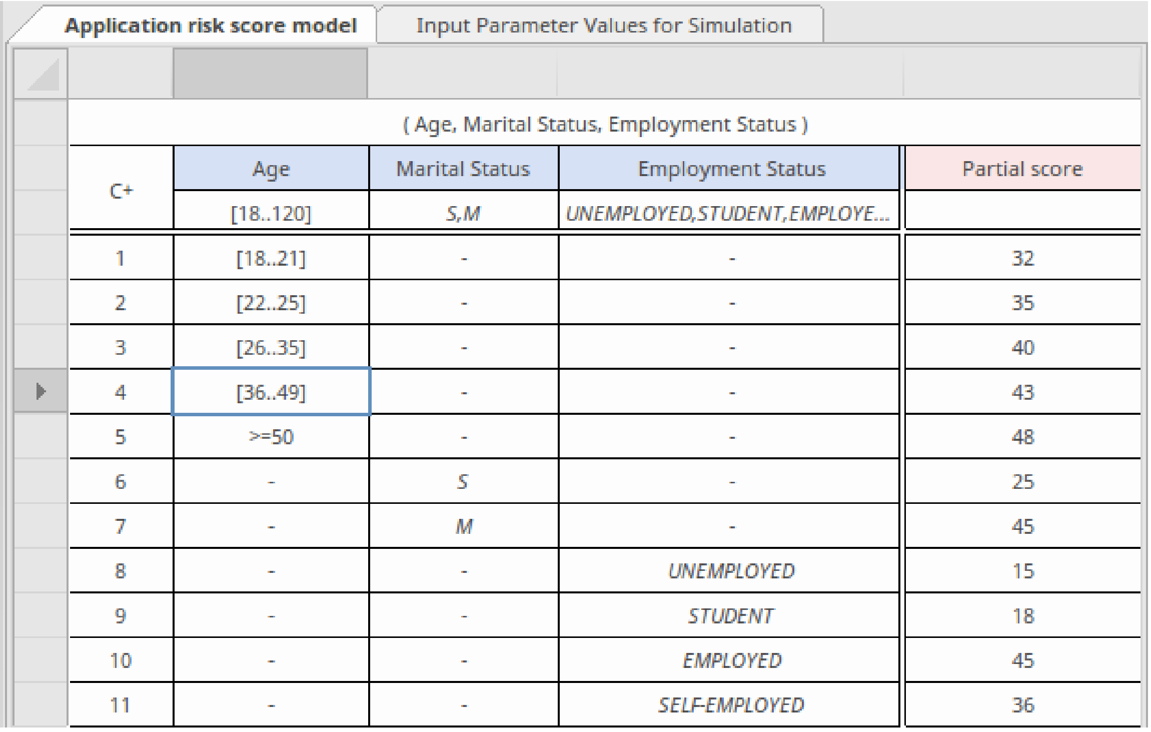
In this example, Input Data *Applicant Data* is typed to *Applicant data Definition,* which has three components.



Applicant data : Applicant data Definition

Applicant data Definition	Age : number	40
	Employment Status : string	"EMPLOYED"
	Marital Status : string	"M"

The Business Knowledge Model *Application risk score model* is implemented as a Decision Table with three inputs and one output.

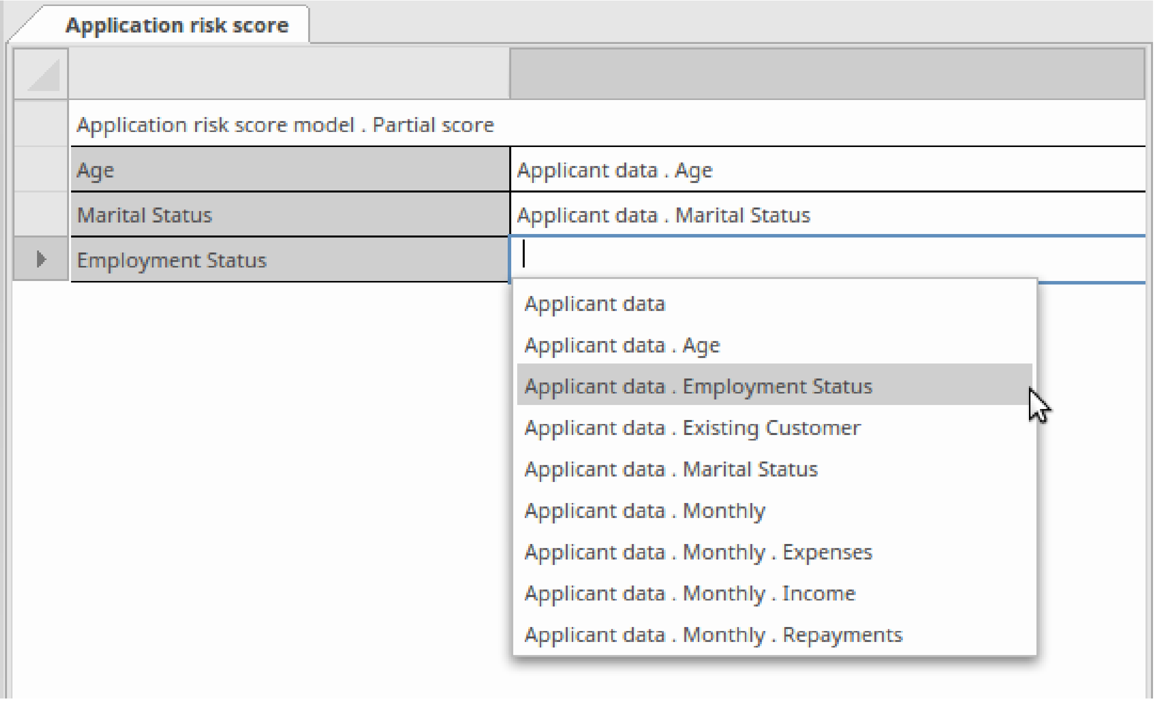


Application risk score model | Input Parameter Values for Simulation

(Age, Marital Status, Employment Status)

C+	Age	Marital Status	Employment Status	Partial score
	[18..120]	S,M	UNEMPLOYED,STUDENT,EMPLOYE...	
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	S	-	25
7	-	M	-	45
8	-	-	UNEMPLOYED	15
9	-	-	STUDENT	18
10	-	-	EMPLOYED	45
11	-	-	SELF-EMPLOYED	36

The Decision *Application risk score* is implemented as an Invocation to bind the Input Data's 'leaf' components to the BKM's parameters.



Application risk score model . Partial score	
Age	Applicant data . Age
Marital Status	Applicant data . Marital Status
Employment Status	

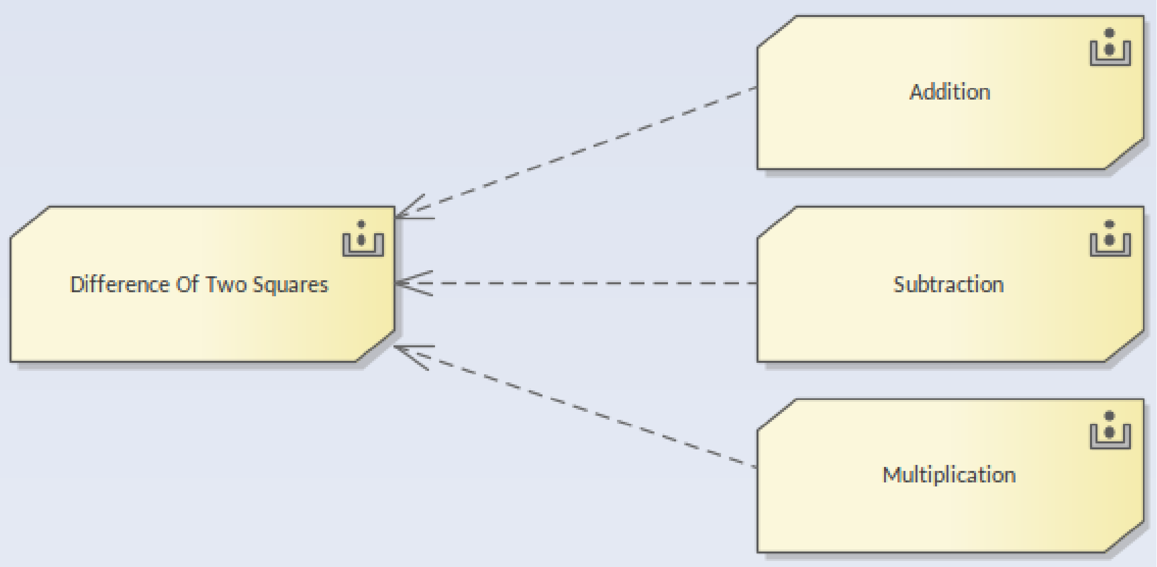
Applicant data
Applicant data . Age
Applicant data . Employment Status
Applicant data . Existing Customer
Applicant data . Marital Status
Applicant data . Monthly
Applicant data . Monthly . Expenses
Applicant data . Monthly . Income
Applicant data . Monthly . Repayments

In order to make the binding easier, Auto-Completion is supported for the binding expression.

The full modeling and simulation instructions are available in the Pattern's documentation.

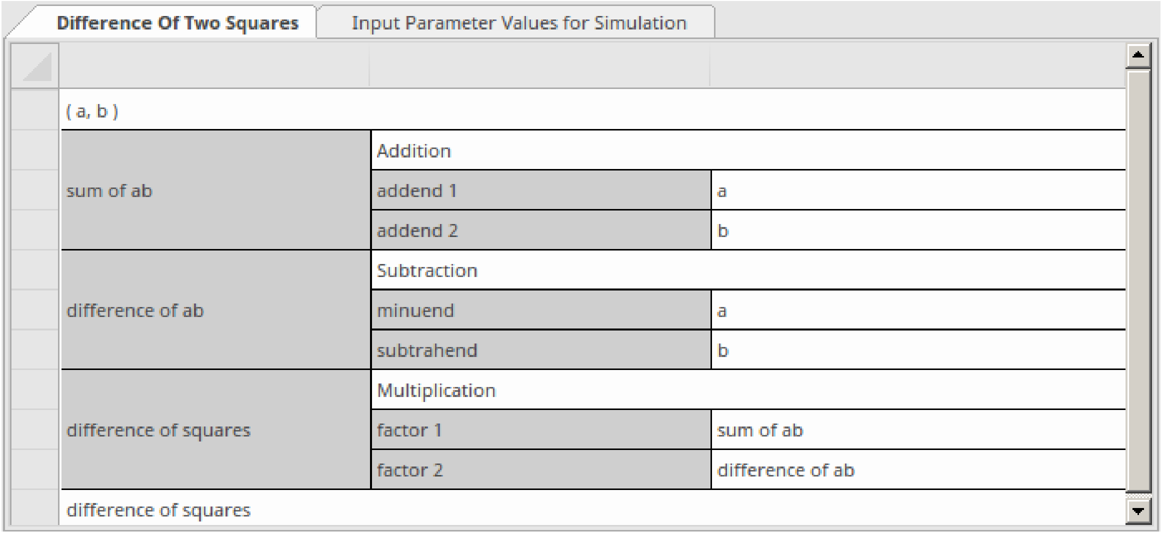
Example 2 - Bind Context Entry variables to Business Knowledge Model

A full example can be created with a Model Pattern (in the ribbon, select 'Simulate > Decision Analysis > DMN > Apply Perspective > DMN Business Knowledge Model Examples > Business Knowledge Model Invocation : Create Model(s)).



In this example, the Business Knowledge Model (BKM) *Difference Of Two Squares* is implemented as Boxed Context:

- The variable *sum of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Addition*
- The variable *difference of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Subtraction*
- The variable *difference of squares* is implemented as an invocation by binding local variables *sum of ab* and *difference of ab* to BKM *Multiplication*


Difference Of Two Squares
Input Parameter Values for Simulation

(a, b)		
sum of ab	Addition	
	addend 1	a
	addend 2	b
difference of ab	Subtraction	
	minuend	a
	subtrahend	b
difference of squares	Multiplication	
	factor 1	sum of ab
	factor 2	difference of ab
difference of squares		

In order to make the binding easier, auto-completion is supported for the binding expression.

The full modeling and simulation instructions are available in the Pattern's documentation.

Edit DMN Expression Dialog

The 'Edit DMN Expression' dialog is used for setting expressions in the Boxed Content, Invocation and Literal Expression element types. It provides Intelli-sense support for constructing expressions based on the FEEL grammar, as well as the code languages that can be used for code generation of the model.

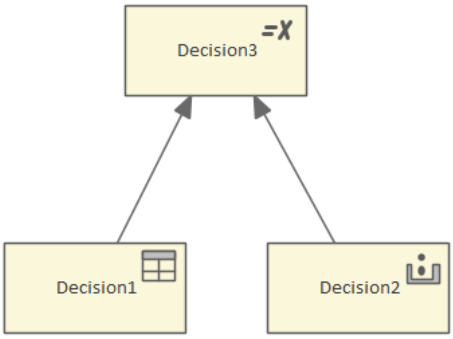
DMN Expression Editor and Intelli-sense support

To help you edit expressions with less typing and fewer mistakes, Enterprise Architect provides Intelli-sense support for editing the expressions.

Note that the parameter and Context Entry variable names can contain spaces, according to the FEEL language specification. This feature is intended to make each expression easy to read.

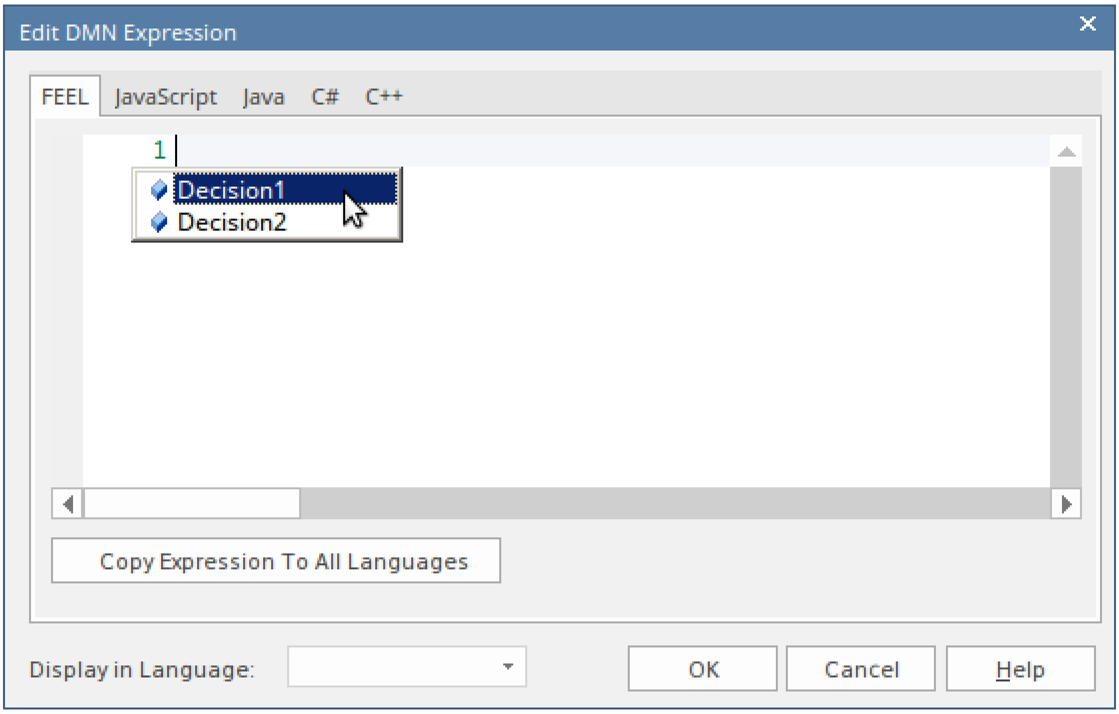
Examples

Given this decision hierarchy, the expression in 'Decision3' is able to use the outputs from the two referenced Decisions.



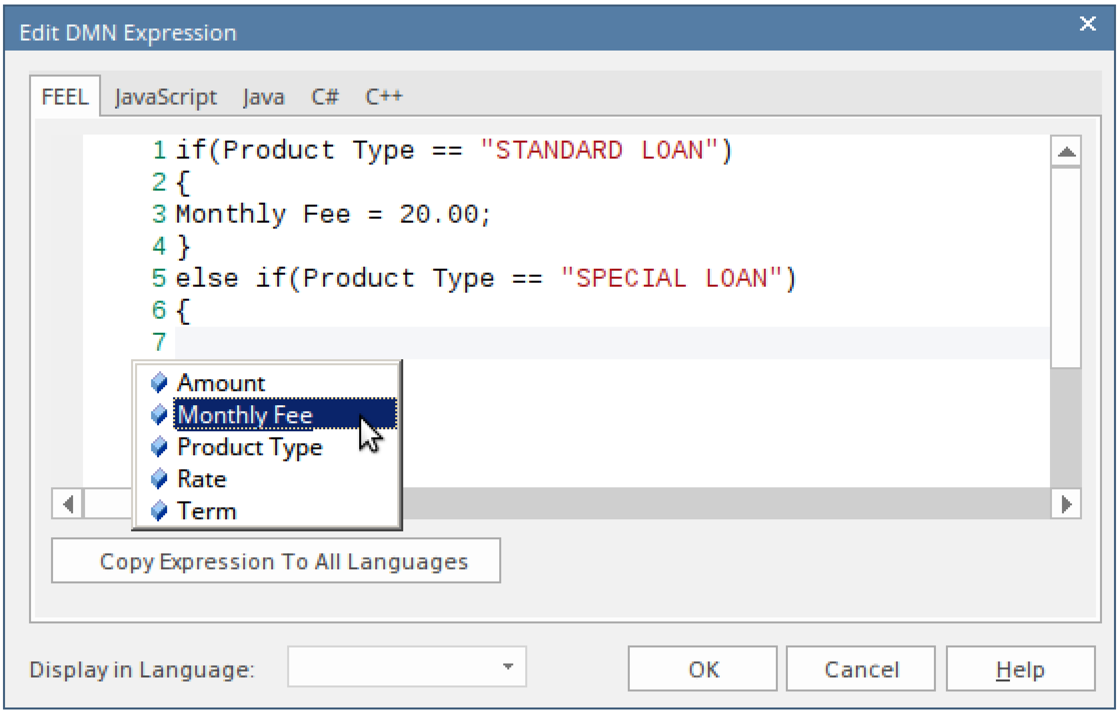
To open the 'Edit DMN Expression' dialog:

1. Double-click on the Decision element in the diagram, to display the DMN Expression window.
2. Right-click on the expression line and select the menu option 'Edit Expression'. The 'Edit DMN Expression' dialog displays.
3. Click on a line and press Ctrl+Spacebar to show the Intelli-sense menu:



- For a BusinessKnowledgeModel expression all the parameters will be included
- For Decision expression all the required Decisions will be included
- All Context Entry Variables earlier than the current one will be included (Context Entry Variables later than the current one are excluded)

In this example, editing a BKM Boxed Context expression, the Input Parameters are shown in the Intelli-sense menu:



```
if(Product Type == "STANDARD LOAN")
{
Monthly Fee = 20.00;
}
else if(Product Type == "SPECIAL LOAN")
{

```

Language selection

The DMN Model can be generated as source code in JavaScript, Java, C# or C++. As the syntax differs between the languages, Enterprise Architect provides language-override pages for each language. If no override code is specified for a language, the expression that is defined for the FEEL language will be used.

Note: In the generated code, the space inside a variable name will be replaced by an underscore.

DMN Expression Validation

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of these expressions are implemented largely by 'text'.

To make modeling easier and more reliable, Enterprise Architect provides two features: Auto Completion and Validation.

- Validation: Identifies modeling errors caused by typos, logic incompleteness, inconsistency, and so on
- Auto Completion: You can select a text string from a list of enumerations rather than type the text in

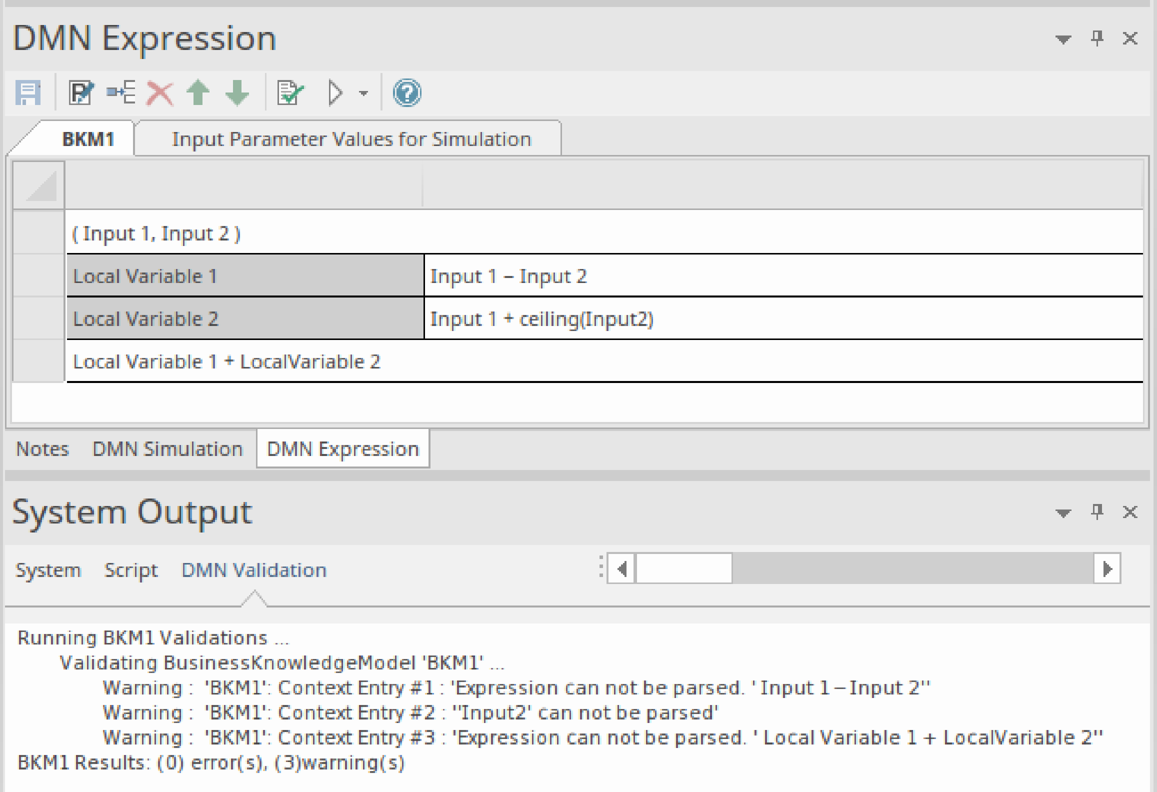
In this topic, we will show you how to validate a DMN Expression.

Access

DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Open DMN Simulation > Simulate : Validate icon

Common Validations

Variable Name Validation



(Input 1, Input 2)	
Local Variable 1	Input 1 – Input 2
Local Variable 2	Input 1 + ceiling(Input2)
Local Variable 1 + LocalVariable 2	

```
Running BKM1 Validations ...
    Validating BusinessKnowledgeModel 'BKM1' ...
        Warning :  'BKM1': Context Entry #1 : 'Expression can not be parsed. ' Input 1 – Input 2''
        Warning :  'BKM1': Context Entry #2 : ''Input2' can not be parsed'
        Warning :  'BKM1': Context Entry #3 : 'Expression can not be parsed. ' Local Variable 1 + LocalVariable 2''
BKM1 Results: (0) error(s), (3)warning(s)
```

In this example, the Boxed Context Business Knowledge Model BKM1 defines two parameters, 'Input 1' and 'Input 2', and two local variables, 'Local Variable 1' and 'Local Variable 2'. The expression has been validated, and the results output to the 'DMN Validation' tab of the System Output window.

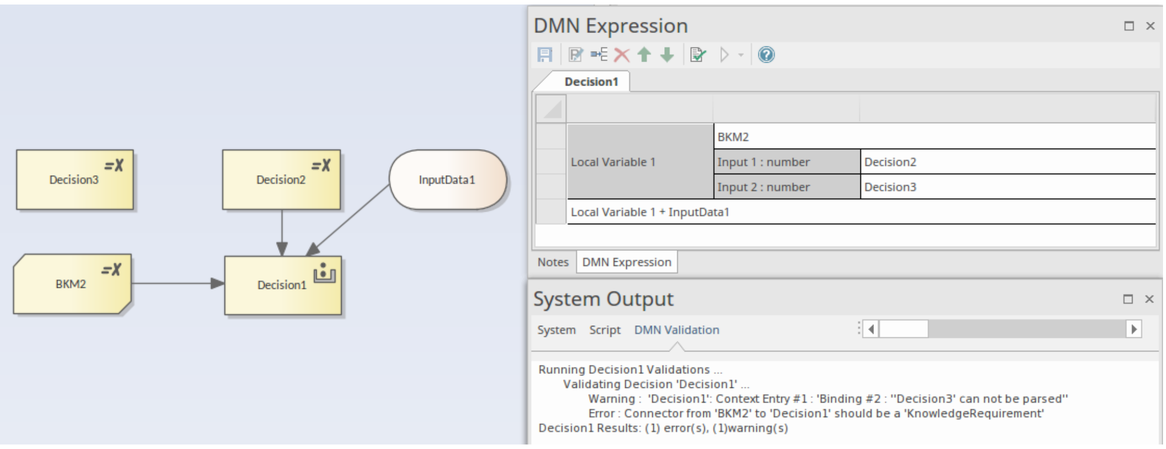
- Context Entry #1 failed because there is a typographic error; it should be operator '-', but the user typed or copied in '_'
- Context Entry #2 failed because there is no space between 'Input' and the number 2; note that the function 'ceiling()' is defined in the DMN Library so it can be successfully parsed
- Context Entry #3 failed because there is no space between 'Local' and 'Variable'

It is hard to visually identify these kinds of error. Running validation can help identify errors and then you can easily perform a correction.

Dependency Validation

A decision might require other decisions, input data and business knowledge models; these relationships are identified by InformationRequirement and KnowledgeRequirement connectors.

When the graph is getting complex, it is quite possible that some connectors are missing or the wrong connector type is being used.



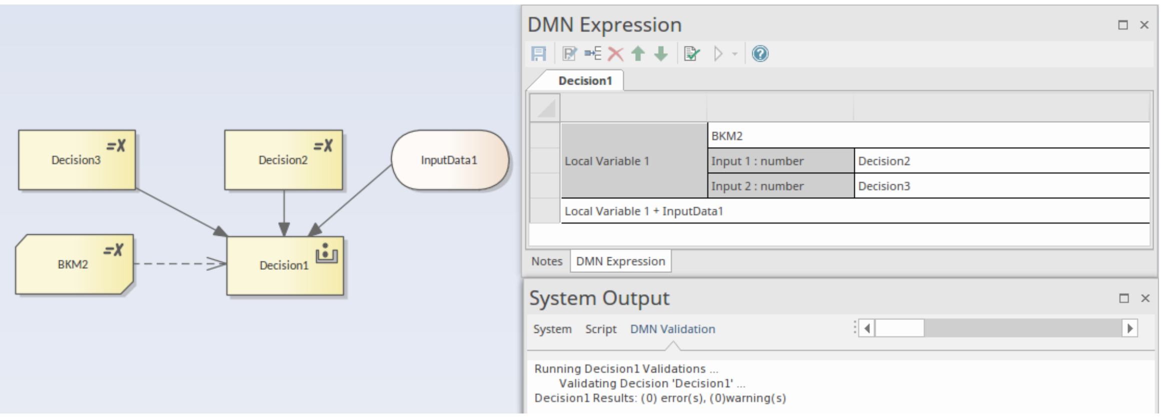
Local Variable 1	BKM2	
	Input 1 : number	Decision2
	Input 2 : number	Decision3
Local Variable 1 + InputData1		

```
Running Decision1 Validations ...
      Validating Decision 'Decision1' ...
            Warning :  'Decision1': Context Entry #1 : 'Binding #2 : ''Decision3' can not be parsed''
            Error : Connector from 'BKM2' to 'Decision1' should be a 'KnowledgeRequirement'
Decision1 Results: (1) error(s), (1)warning(s)
```

In this example, click on the Validate button, Enterprise Architect will show that:

- 'Decision3' is used by 'Decision1' by binding to a parameter of the called BKM2; however, it is not defined - an InformationRequirement connector is missing
- The Invocation defined in 'Decision1' is not valid; the connector type from 'BKM2' to 'Decision1' should be a KnowledgeRequirement

After fixing these problems, run the validation again:



Local Variable 1	BKM2	
	Input 1 : number	Decision2
	Input 2 : number	Decision3
Local Variable 1 + InputData1		

```
Running Decision1 Validations ...
      Validating Decision 'Decision1' ...
Decision1 Results: (0) error(s), (0)warning(s)
```

DMN Expression Auto Completion

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of these expressions are implemented largely by text.

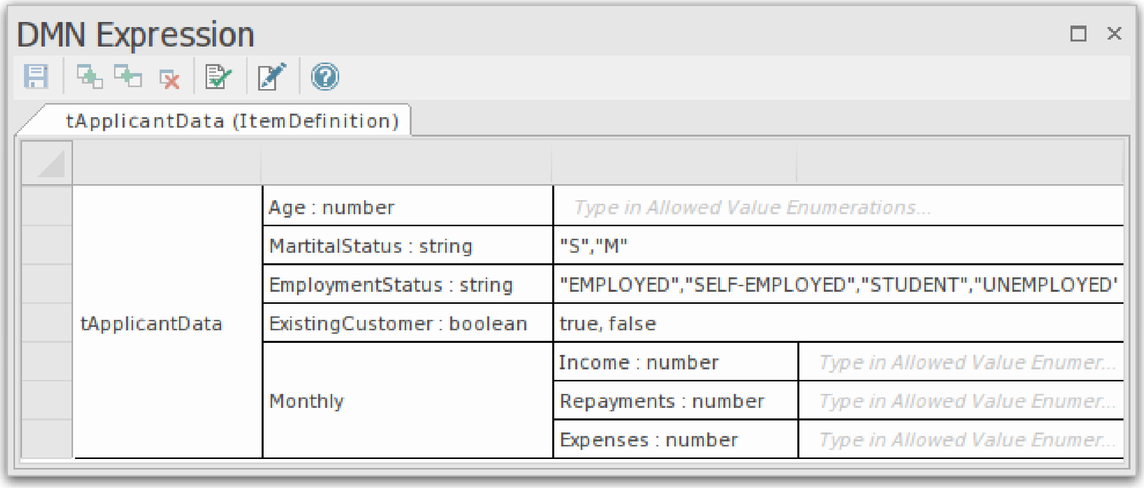
To make modeling easy and reliable, Enterprise Architect provides an Auto Completion facility, helping provide the:

- Allowed Values of ItemDefinition
- Input/Output Entries of a Decision Table
- InformationRequirement

Allowed Values of ItemDefinition

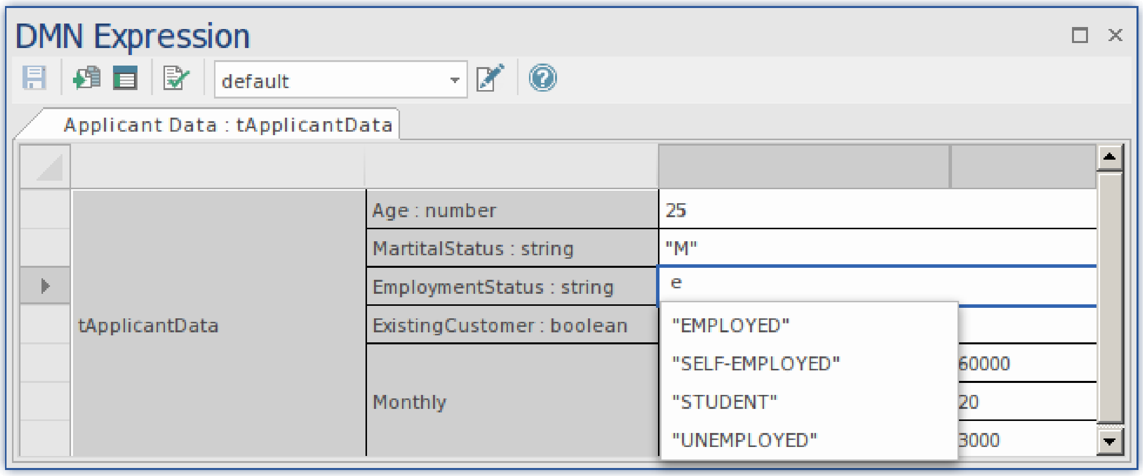
The idea is to define allowed value enumerations in ItemDefinition, then compose a list for selection whenever these values are requested.

In this example, ItemDefinition 'Applicant data . Employment Status' defines an enumeration of allowed values.



tApplicantData	Age : number	Type in Allowed Value Enumerations...	
	MartitalStatus : string	"S","M"	
	EmploymentStatus : string	"EMPLOYED","SELF-EMPLOYED","STUDENT","UNEMPLOYED'	
	ExistingCustomer : boolean	true, false	
	Monthly	Income : number	Type in Allowed Value Enumer...
		Repayments : number	Type in Allowed Value Enumer...
		Expenses : number	Type in Allowed Value Enumer...

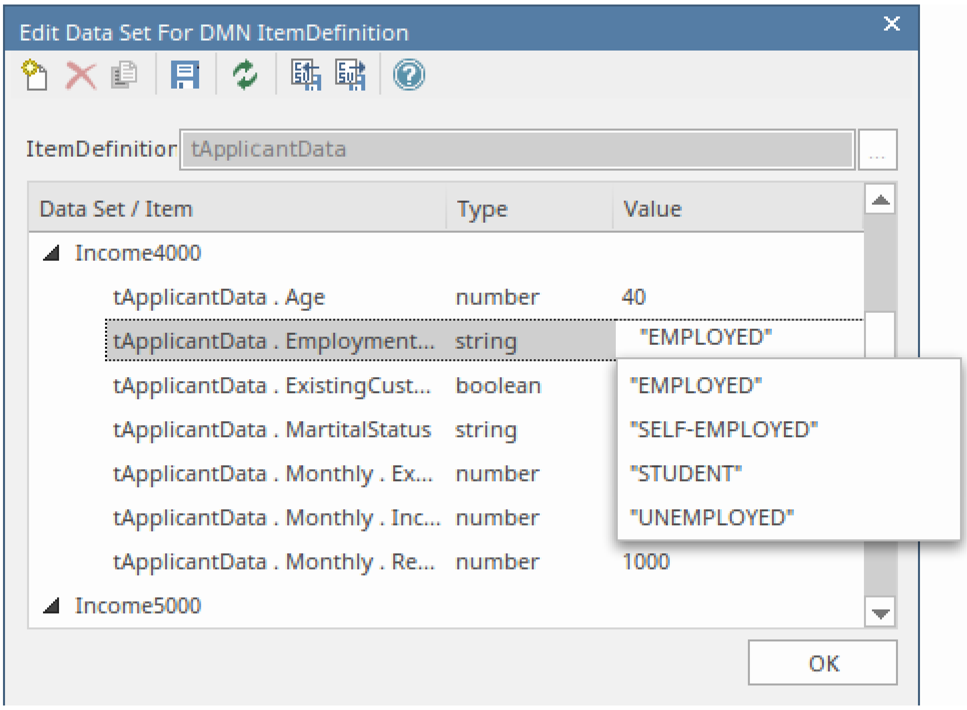
When editing values for the InputData typed to this ItemDefinition, press the Spacebar on the keyboard to display a list of values to select from.



tApplicantData	Age : number	25	
	MartitalStatus : string	"M"	
	EmploymentStatus : string	e	
	ExistingCustomer : boolean		
	Monthly		60000
			20
			3000

"EMPLOYED"
"SELF-EMPLOYED"
"STUDENT"
"UNEMPLOYED"

We could also define multiple data sets for the InputData, as the Auto Completion feature is available on this dialog.

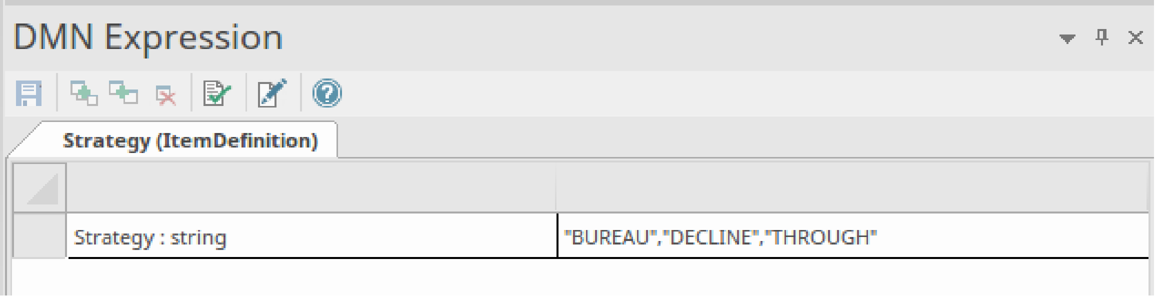


ItemDefinitior: tApplicantData

Data Set / Item	Type	Value
Income4000		
tApplicantData . Age	number	40
tApplicantData . Employment...	string	"EMPLOYED"
tApplicantData . ExistingCust...	boolean	
tApplicantData . MartitalStatus	string	
tApplicantData . Monthly . Ex...	number	
tApplicantData . Monthly . Inc...	number	
tApplicantData . Monthly . Re...	number	1000
Income5000		

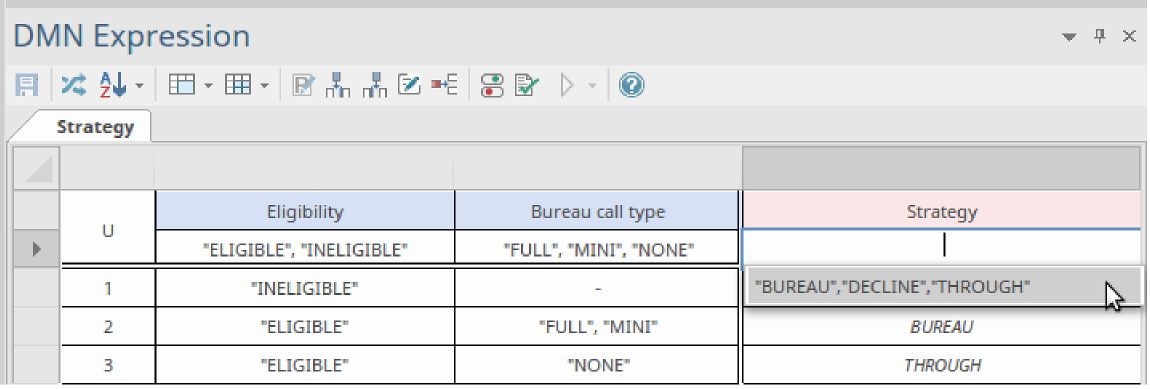
Input/Output Entries of a Decision Table

Take the 'Strategy' ItemDefinition as an example:



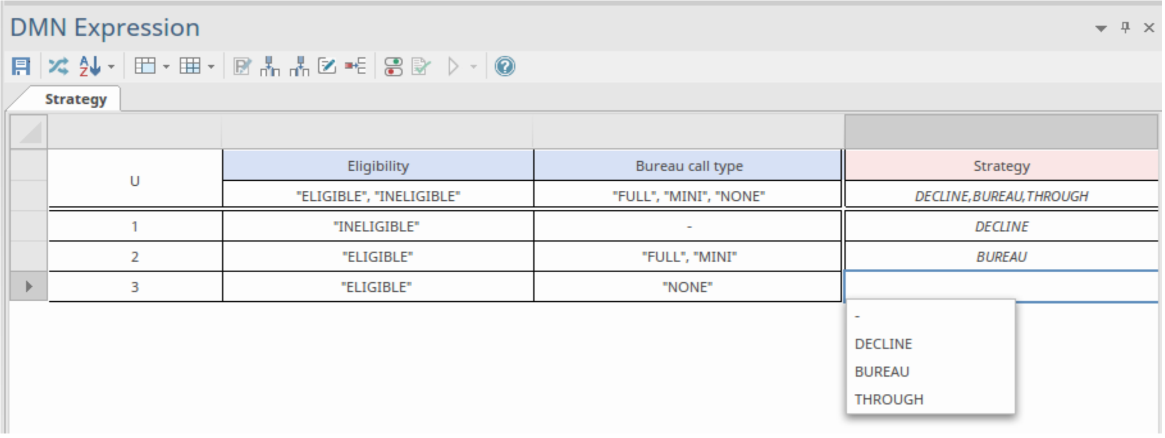
Strategy : string	"BUREAU","DECLINE","THROUGH"

We can quickly fill the 'Allowed Values' field for a Decision Table by selection:



U	Eligibility	Bureau call type	Strategy
	"ELIGIBLE", "INELIGIBLE"	"FULL", "MINI", "NONE"	
1	"INELIGIBLE"	-	
2	"ELIGIBLE"	"FULL", "MINI"	*BUREAU*
3	"ELIGIBLE"	"NONE"	*THROUGH*

Then we can quickly fill the Decision Table rules by selection:

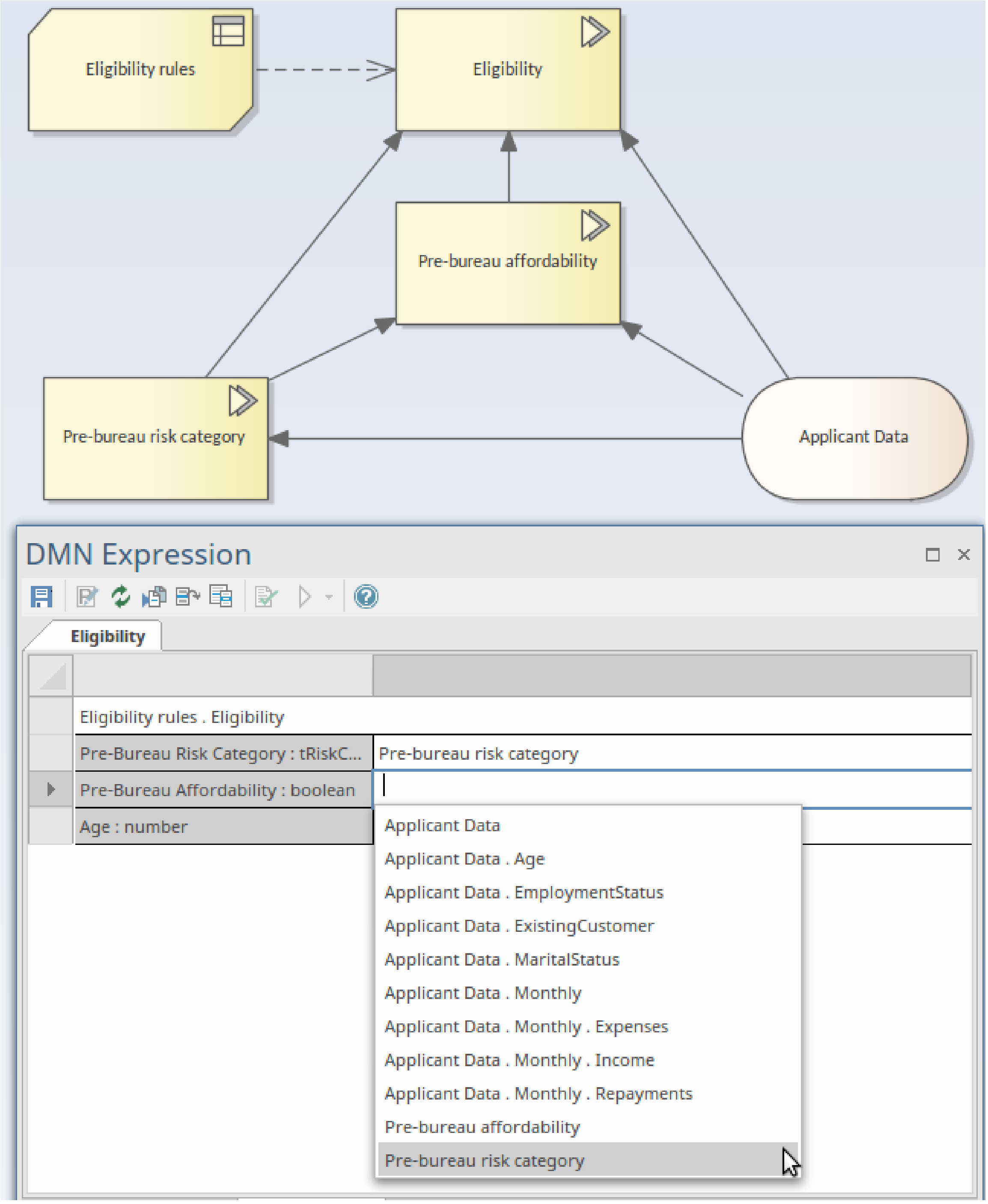


U	Eligibility	Bureau call type	Strategy
	"ELIGIBLE", "INELIGIBLE"	"FULL", "MINI", "NONE"	*DECLINE,BUREAU,THROUGH*
1	"INELIGIBLE"	-	*DECLINE*
2	"ELIGIBLE"	"FULL", "MINI"	*BUREAU*
3	"ELIGIBLE"	"NONE"	

Note: the default '-' denotes 'Undefined'.

Information Requirement

On a decision hierarchy, a decision might access required decisions and input data; these required elements form a list of variables that can be used by the decision.



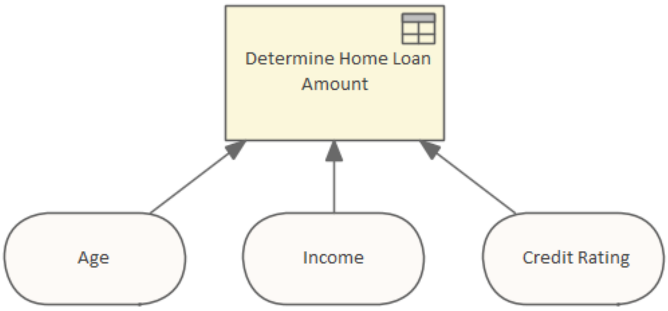
Eligibility rules . Eligibility	
Pre-Bureau Risk Category : tRiskC...	Pre-bureau risk category
Pre-Bureau Affordability : boolean	
Age : number	

In this example, Decision 'Eligibility' requires two decisions - 'Pre-bureau risk category' and 'Pre-bureau affordability' - and one Input Data item 'Applicant data'.

When setting the binding values for the invoked BusinessKnowledgeModel 'Eligibility rules', an Auto Completion list will prompt for selection. In this list, there are sub-decision names - leaf components of the input data. With this feature, you can easily set up an invocation.

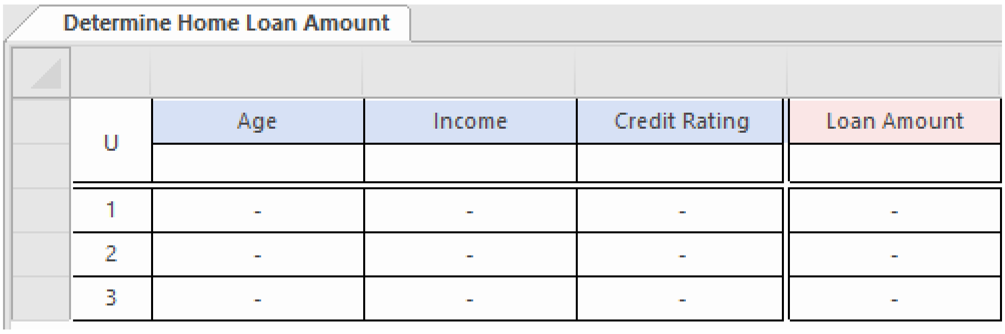
Modeling with DMN

This topic introduces you to the most important elements that you need to create Decision models. As discussed in earlier topics there are two fundamental parts of a decision model namely the Decision Requirements diagram and the decision logic. Creating a decision diagram is straight forward and probably the most onerous part of the exercise will be unraveling the way an organization makes decisions and what the inputs to these decision are. The diagrams will typically contain decisions that are chained together describing the fact that one decision can provide input to another decision and so on.



Simple Decision Requirements diagram showing three inputs into a Decision using a Decision Table.

The logic of the decision is described using a number of devices but the most commonly used and accessible form is the Decision Table. The Decision Table contains rows and columns like a spreadsheet and covers all the possible combinations of inputs to produce a number of outputs. For example if an applicant is older than 21 and younger than 65 and earns $60,000 per year with a good credit rating the bank will lend them $300,000 for a home loan.



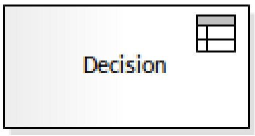
Determine Home Loan Amount

U	Age	Income	Credit Rating	Loan Amount
1	-	-	-	-
2	-	-	-	-
3	-	-	-	-

A Decision Table showing three inputs and one output, rows would be added to define the rules.

Decision

A Decision element is used to evaluate an output based on one or more inputs. The logic that determines the output is either defined within that Decision element or it invokes the decision logic contained in a Business Knowledge Model that is connected to the Decision.



Inputs

A Decision can have any number of inputs, including the option to define the input values in the element. The most common input is to use an Input Data Element.

Output

A Decision can have zero or one output. The output can be a complex data set.

Value Expressions

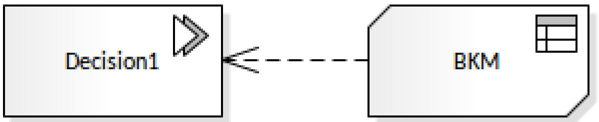
The output of a Decision element is determined using a Value Expression. The Value Expression contains the element's decision logic and can take one of four forms: Decision Table, Literal Expression, Invocation or Boxed Context. Value Expressions are defined and edited using the DMN Expression editor, which displays one of four formats according to the type of expression being used.

When displayed on a diagram, the Decision element shows an icon in the top-right corner that indicates which type of value expression it is using.

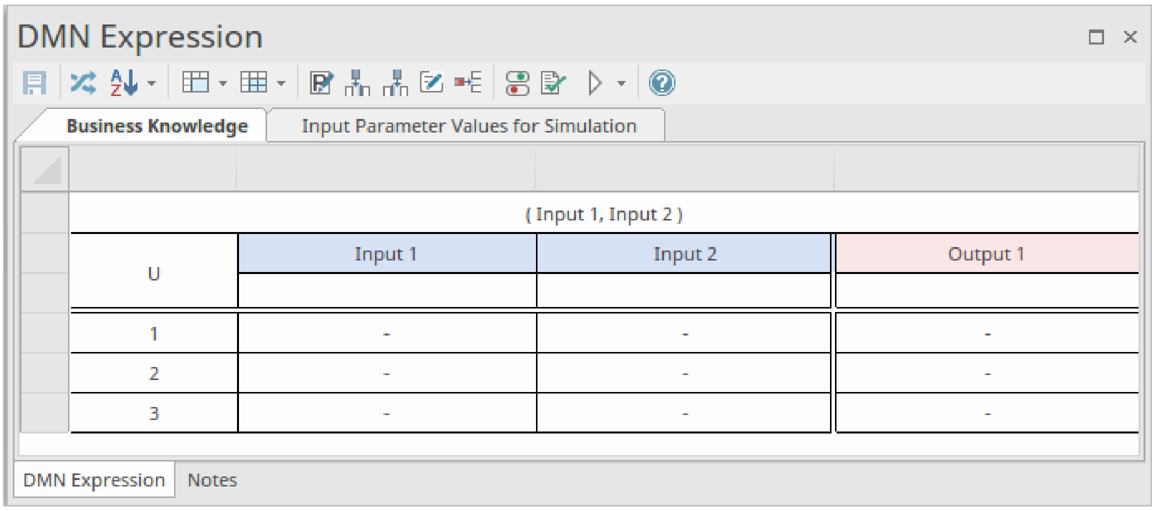
Type	Description
	A Decision Table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.
	A Literal Expression is the simplest form of DMN expression. It is commonly defined as a single-line statement or an if-else conditional block.
	A Decision Invocation requires that a Business Knowledge Model element is referenced using a Knowledge Requirement connector. The Decision element simply contains the parameters that provide the context for evaluating the Business Knowledge Model (BKM). Part or all of the result returned from the BKM can be set to be passed as the output of the Decision.
	A Boxed Context is a collection of context entries. Each context entry consists of a variable and an expression. The Context also has a result value.

Business Knowledge Model

A Business Knowledge Model (BKM) element represents a reusable piece of decision logic. Typically, it is connected to a Decision element that invokes the BKM and passes on a set of inputs. The BKM, using its internal logic, evaluates an output that is passed back to the Decision.



Unless a BKM is working on fixed values, it usually requires defining a set of input parameters, as well as the definition of an output. The parameters and the decision logic are defined using the DMN Expression window.



	(Input 1, Input 2)		
U	Input 1	Input 2	Output 1
1	-	-	-
2	-	-	-
3	-	-	-

Inputs and Output

When used in a decision model, a BKM must be connected via a KnowledgeRequirement to a Decision or another BKM, through which it receives its inputs . The input parameters are defined using the icon. These can be set as a simple type or a complex type defined using an ItemDefinition. The naming of the input parameters influences the naming within the Value Expression.

Output

A BKM output is via a KnowledgeRequirement which must be an input to a Decision or to another BKM. The output is defined using:

- The icon for a Literal Expression
- Output column(s) in the DMN Expression table for a Decision Table, Boxed Content and Invocation.

An output can be a simple type or a complex type defined using an ItemDefinition.

Value Expressions

To define a means for evaluating an output, based on the decision logic, a Business Knowledge Model (BKM) element contains a Value Expression. This is defined and edited using the DMN Expression window, which has four formats, the format being determined by the type of Value Expression that you want to use.

The BKM element can be set with these structures for the Value Expression. Each is shown in the model with an icon.

Type	Description
	A Decision Table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.
	A Literal Expression is the simplest form of DMN expression. It is commonly defined as a single-line statement or an if-else conditional block.
	A Decision Invocation requires that a Business knowledge model element is referenced using a Knowledge Requirement connector. It simply contains the parameters that provide the context for the evaluating a business knowledge model.
	A Boxed Context is a collection of context entries. Each context entry consists of a variable and an expression. The Context also has a result value.

Validation and Testing

To ensure a BKM element is able to produce a correct output it can be validated using the Validation icon ✎. A BKM can also be tested as a unit to ensure it is operative using the Simulation ▷ button. For more details see the *Input Parameter Values for Simulation* Help topic.

BKM Parameters

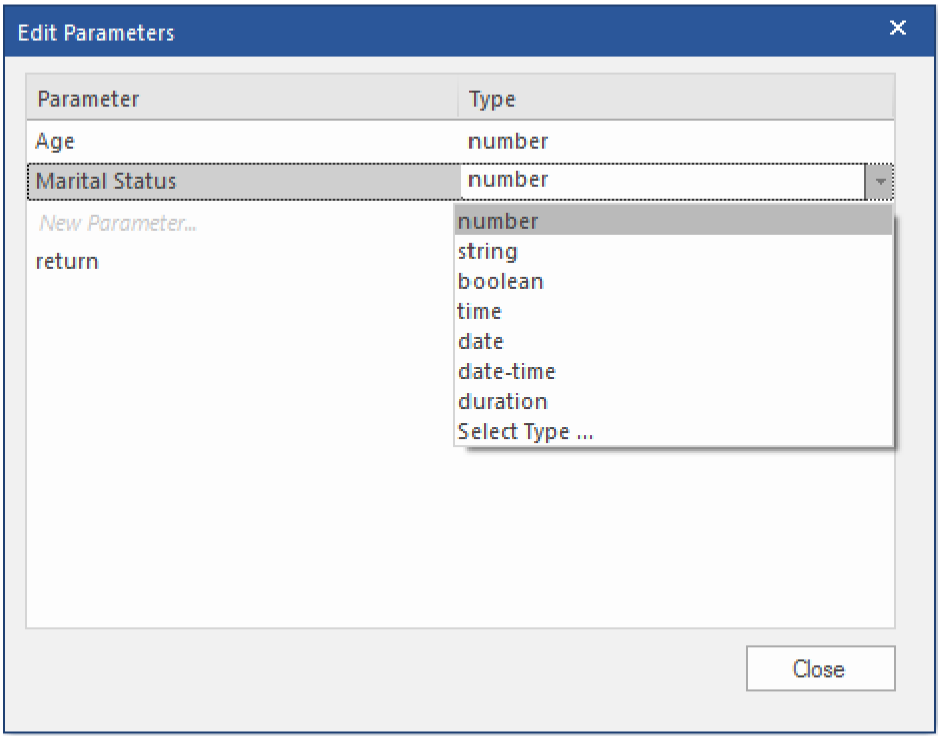
A Business Knowledge Model (BKM) is implemented as a function definition, with parameters and a DMN expression as its body (such as Decision Table, Boxed Context or Literal Expressions).

As a BKM is intended to function stand-alone, and be called by Decisions or other BKMs, it is necessary to define any input parameters. Also, for Literal Expressions, you must define the output parameter.

When defining any input Parameters you can set them with default values for testing. After creating a BKM, to verify that it functions correctly, you can run a simulation based on these default values.

Parameters of a Business Knowledge Model

To open the 'Edit Parameters' dialog, in the DMN Expression window, click on the Edit Parameters button :



Note: this is an example for a Literal Expression that includes a *return* type.

Edit Parameters

You can perform these actions on the parameters:

Action	Description
	Add a new parameter by typing in the 'New Parameter...' row.
	Modify the name of the existing parameter by in-place editing in the cell.
	Delete an existing parameter using the context menu.
	Click on the Type to enable a drop-down. Select a type for the parameter from the

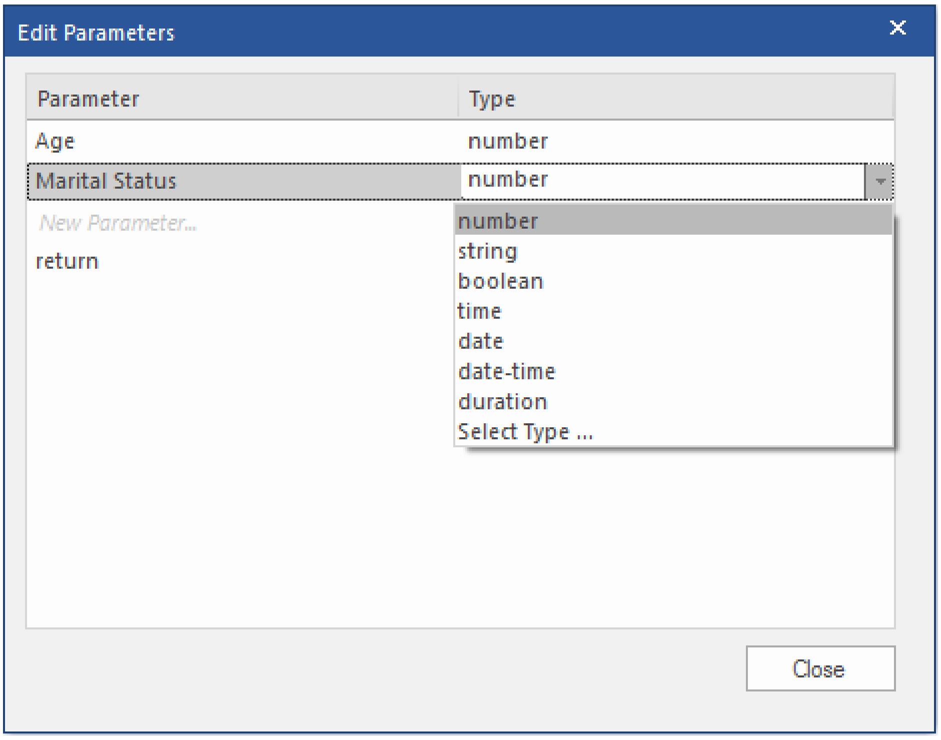
	drop-down. **Set an Item Definition Type** When changing the type of Parameter there is an option to select a pre-defined type from an ItemDefinition. The option for this is 'Select Type ...'. When this option is selected it will open a dialog for selecting an ItemDefinition.

Input Parameter Values for Simulation

As a Business Knowledge Model is self-contained, it is possible to perform a simulation 'Unit Test' by providing a default set of values as an input for its parameters. These values can be defined in the *Input Parameter Values for Simulation* tab in the DMN Expression window.

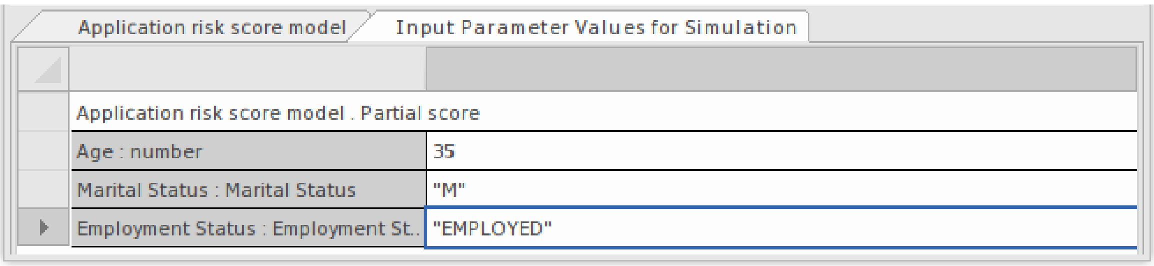
Parameters of a Business Knowledge Model (BKM)

Parameters for a BKM are accessed from the DMN Expression window, using the Edit Parameters button on the toolbar:



Parameter	Type
Age	number
Marital Status	number
return	

A default set of values for these parameters, that can be used in a simulation of the BKM, are defined in the 'Input Parameter Values for Simulation' tab on the DMN Expression window:


Application risk score model
Input Parameter Values for Simulation

Application risk score model . Partial score

Age : number	35
Marital Status : Marital Status	"M"
Employment Status : Employment St..	"EMPLOYED"

With these parameters set the BKM can be tested using the Simulation ▷ button.

Simulation Examples

These are two examples of using the *Input Parameter Values for Simulation*.

Type	Description

Decision Table	An example simulation of a BKM Decision Table element based on values set in the *Input Parameter Values for Simulation* tab.
Literal Expression	An example simulation of a BKM Literal Expression element based on values set in the *Input Parameter Values for Simulation* tab.

Decision Table Simulation Example

The example Business Knowledge Model (BKM) described in this section is available from the Model Wizard (Ctrl+Shift+M). Select a host Package in your model, invoke the Model Wizard and - from the Perspectives drop-down menu - select 'Requirements | Decision Modeling'.

To access the example used in this section:

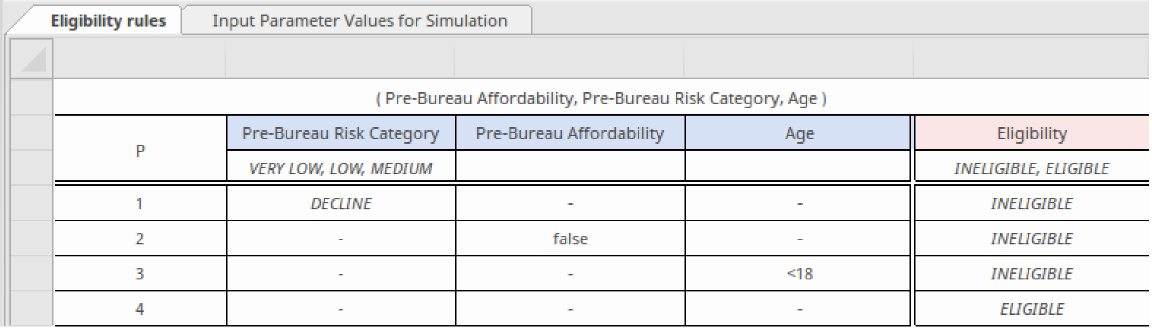
- Create a pattern for 'DMN Decision | A Complete Example'
- Navigate in the Browser window to 'A Complete Example | Business Knowledge Models'

It is also available in the Enterprise Architect Example model (EAExample):

- Navigate in the Browser window to 'Analysis and Business Modeling > DMN Examples > A Complete Example > Business Knowledge Models'

Double-click on the 'Eligibility rules' element to open the BKM in the DMN Expression window

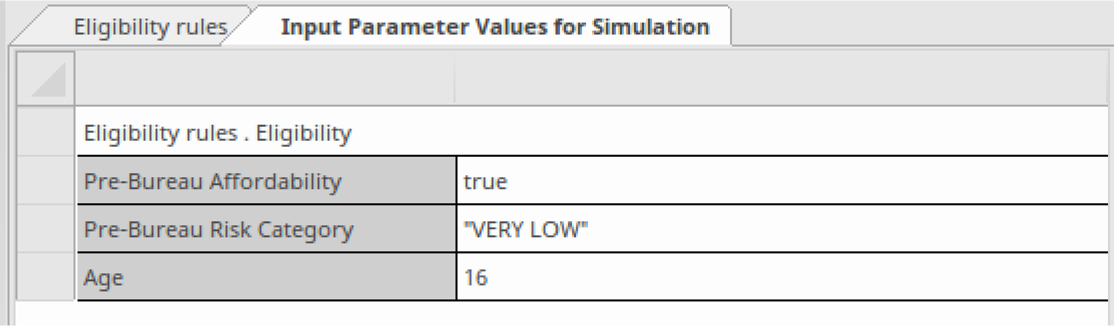
When a Decision Table is created for a Business Knowledge Model, we can test this BKM by binding some values:



(Pre-Bureau Affordability, Pre-Bureau Risk Category, Age)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	VERY LOW, LOW, MEDIUM			*INELIGIBLE, ELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

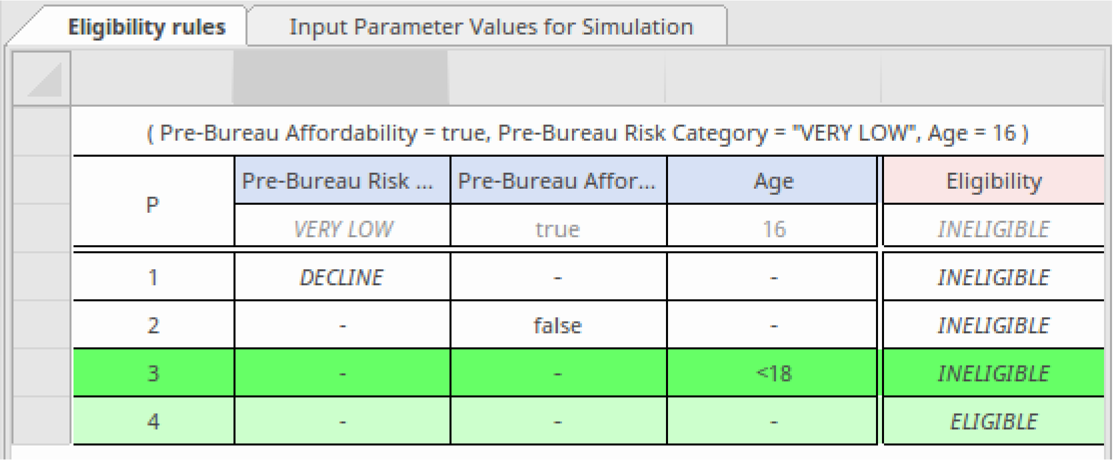
We can provide test values such as these:



Eligibility rules . Eligibility

Pre-Bureau Affordability	true
Pre-Bureau Risk Category	"VERY LOW"
Age	16

Click on the Simulation button ▷ on the tool bar to obtain this result:



(Pre-Bureau Affordability = true, Pre-Bureau Risk Category = "VERY LOW", Age = 16)

P	Pre-Bureau Risk ...	Pre-Bureau Affor...	Age	Eligibility
	VERY LOW	true	16	*INELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

- The runtime parameter value will take the place of 'Allowed Values' in simulation mode
- Valid rule(s) are highlighted
- Since this Decision Table's Hit Policy is P (Priority) the final result is determined by the order of Output Values; since 'INELIGIBLE' and 'ELIGIBLE' are the output values and 'INELIGIBLE' comes ahead of 'ELIGIBLE', rule #3 will give the final result and this applicant is 'INELIGIBLE'.

Literal Expression Simulation Example

The Business Knowledge Model (BKM) described in this section is available from the Model Wizard (Ctrl+Shift+M). Select a host Package in your model, invoke the Model Wizard and - from the Perspectives drop-down menu - select 'Requirements | Decision Modeling'.

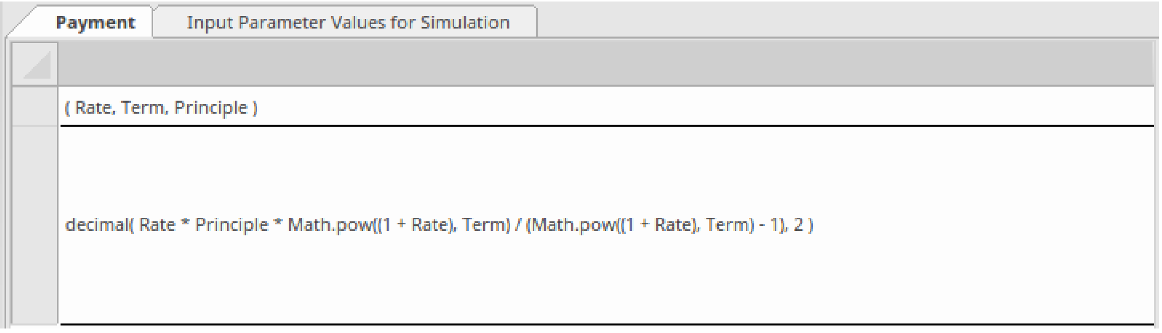
To access the example used in this section:

- Create a pattern for 'DMN Business Knowledge Model > Business Knowledge Model Literal Expression'
- Navigate in the Browser window to 'Business Knowledge Model Literal Expression > Payment'

It is also available in the Enterprise Architect Example model (EAExample):

- Navigate in the Browser window to 'Model Simulation > DMN Models > Business Knowledge Model > Business Knowledge Model Literal Expression'

Double-click on the 'Payment' element to open the BKM in the DMN Expression window.



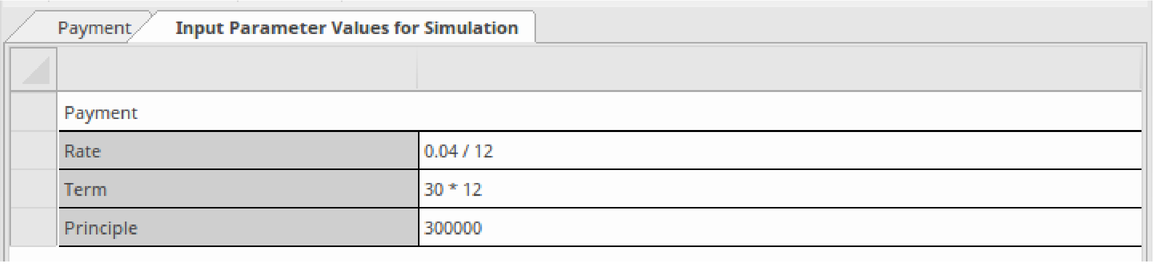
(Rate, Term, Principle)

decimal(Rate * Principle * Math.pow((1 + Rate), Term) / (Math.pow((1 + Rate), Term) - 1), 2)

Similar to a Decision Table, the Business Knowledge Model implemented as a Boxed Expression can be tested as well.

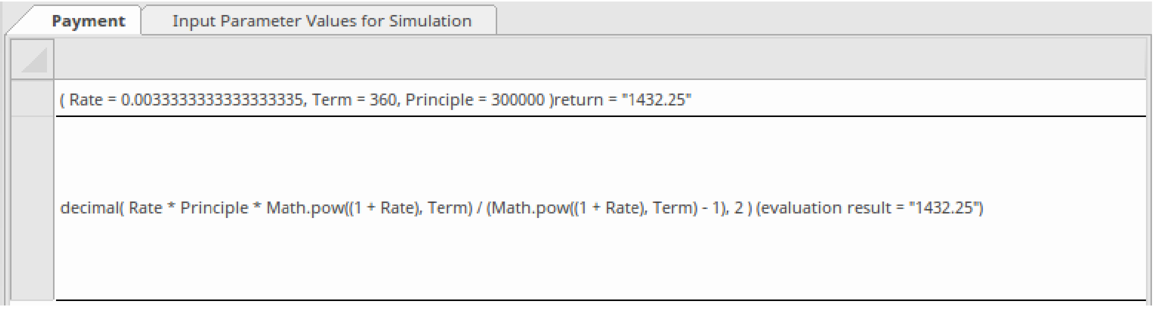
Take the 'Payment' element as an example. This BKM will calculate the monthly repayment based on interest rate, number of terms and principal amount.

We could provide test values such as these:



Payment	
Rate	0.04 / 12
Term	30 * 12
Principle	300000

Click on the Simulation ▷ button on the tool bar; this result is obtained:



(Rate = 0.0033333333333333335, Term = 360, Principle = 300000)return = "1432.25"

decimal(Rate * Principle * Math.pow((1 + Rate), Term) / (Math.pow((1 + Rate), Term) - 1), 2) (evaluation result = "1432.25")

The runtime parameter and return values will be displayed with an equals sign '=' followed by the runtime value. This value is also displayed as a label against the element on its parent diagram.

In this example, given an annual Rate of 4% for 30 years and a principal of $300,000, the monthly repayment is $1,432.25.

Note: The DMN Library already has a PMT function defined; this example mainly demonstrates how Literal Expression works and how to test it with a set of arguments.

InputData

An InputData element is used to input into Decisions a set of values that originate outside the model. That set of values is used for evaluating Decisions. It derives its type and a set of values from an ItemDefinition.

Overview

InputData elements are created by dragging an Input Data icon from the Toolbox onto a DMN diagram.

Applicant data

The name of the InputData element must be unique and not duplicate the name of any other Decision, InputData, Business Knowledge Model, Decision Service, or Import in the decision model.

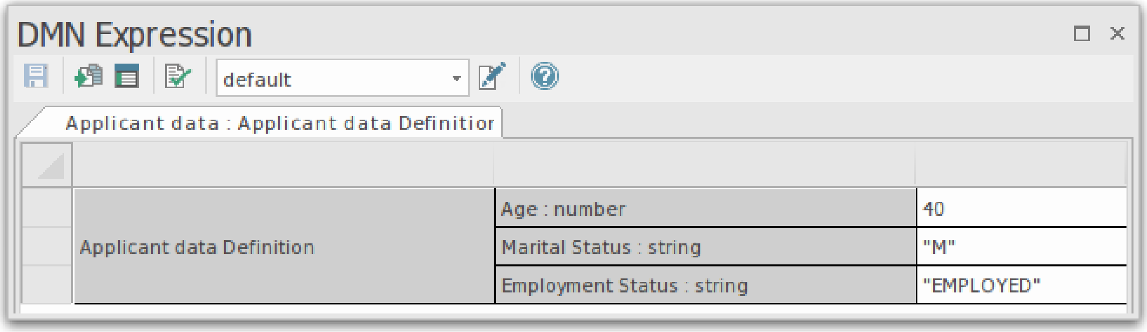
Referencing an ItemDefinition

The structure of the data, as well as sets of values for an InputData element, are defined in an ItemDefinition element. A DMN InputData element must be referenced (typed) by an ItemDefinition by either:

- Clicking on the icon on the DMN Expression window of the InputData element or
- Selecting the InputData element and pressing Ctrl+L to select the ItemDefinition from the dialog

InputData properties

The properties of an InputData element are accessible via the DMN Expression window. Double-click on the InputData element to open this window.

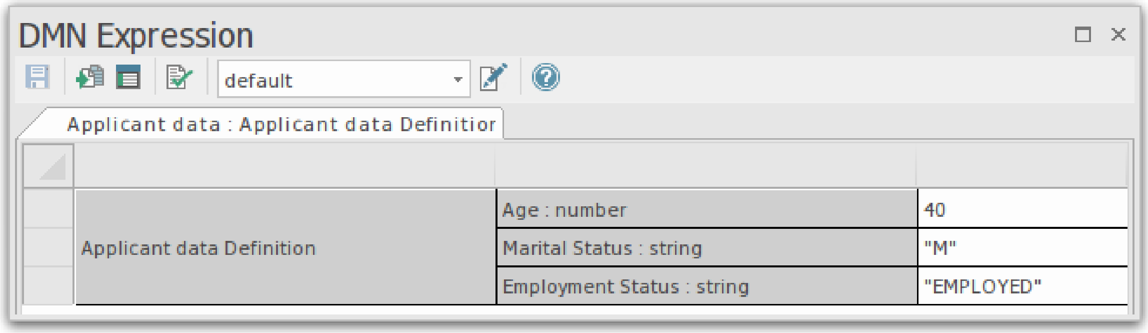


Applicant data Definition	Age : number	40
	Marital Status : string	"M"
	Employment Status : string	"EMPLOYED"

The DMN Expression window provides a view of the data structure as well as access to Data Sets that can be used in simulations.

InputData DMN Expression

The DMN Expression window provides a view of an InputData's data structure, options to alter the value of Items, and access to Data Sets that can be used in simulations.



Applicant data Definition	Age : number	40
	Marital Status : string	"M"
	Employment Status : string	"EMPLOYED"

Access

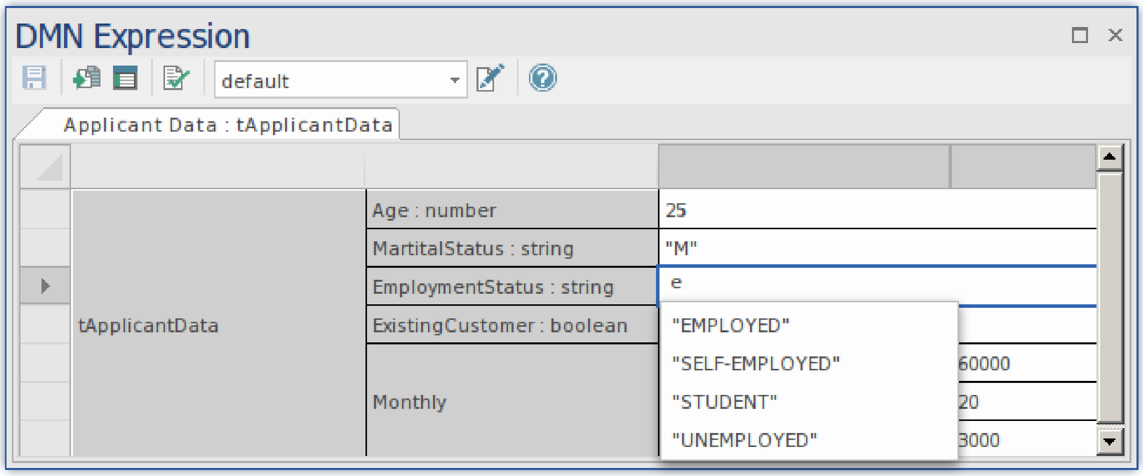
Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select / create an InputData
Other	Double-click on a DMN InputData element

Toolbar Options

Option	Description
💾	Saves the configuration to the current InputData element.
⊕	Sets the InputData's type by selecting a reference to an ItemDefinition.
▤	Opens the ItemDefinition element that is referenced by this InputData as its type definition.
✓	Runs a validation of the InputData. Enterprise Architect will perform a series of validations to help you identify errors in the InputData.
	Option to select a Data Set as defined in the ItemDefinition that references this InputData.
✎	Opens the dialog for editing data sets for this input data. Each InputData can define multiple data sets. With this feature, the DMN Simulation can quickly test the results of a decision by choosing different data sets.

Auto Completion

If the InputData has a field with 'Allowed Value' defined, then the field can be populated by selecting the field, pressing on the Spacebar, then selecting an option from the drop-down.



tApplicantData	Age : number	25	
	MartitalStatus : string	"M"	
	EmploymentStatus : string	e	
	ExistingCustomer : boolean		
	Monthly		60000
			20
			3000

Data Sets

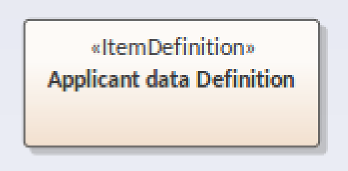
Data Sets are defined in the ItemDefinition referenced by the InputData element. Using the toolbar drop-down you can select a data set from the ItemDefinition. Once a set is selected you can alter the values of the items. You can also add new Data Sets by opening the Edit Data Set window using the ✎ icon.

ItemDefinition

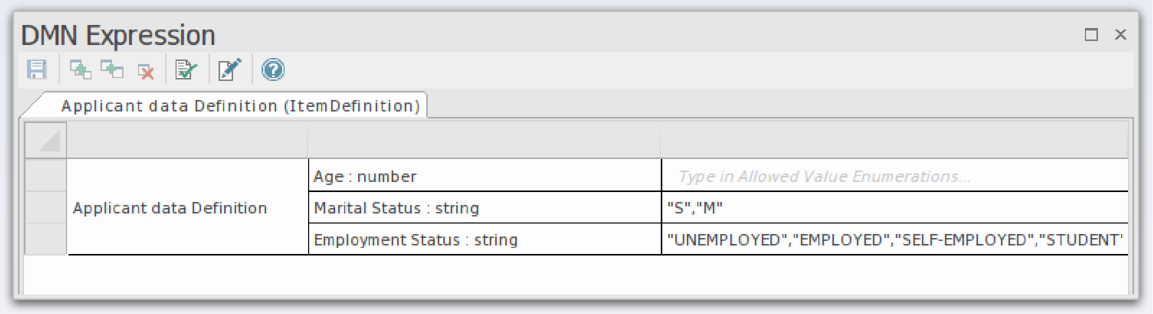
Fundamental to creating Decision Models is the definition of the structure of data items used within the model. An ItemDefinition is used to define the structure of the input data and, optionally, to restrict the range of allowable values of the data. ItemDefinitions can range from a simple single type through to a complex structured type.

Overview

ItemDefinition elements are created by dragging a Item Definition icon from the DMN Toolbox page onto a DMN diagram.



The core properties of an ItemDefinition element are accessed via the DMN Expression window.



Applicant data Definition	Age : number	Type in Allowed Value Enumerations...
	Marital Status : string	"S","M"
	Employment Status : string	"UNEMPLOYED","EMPLOYED","SELF-EMPLOYED","STUDENT'

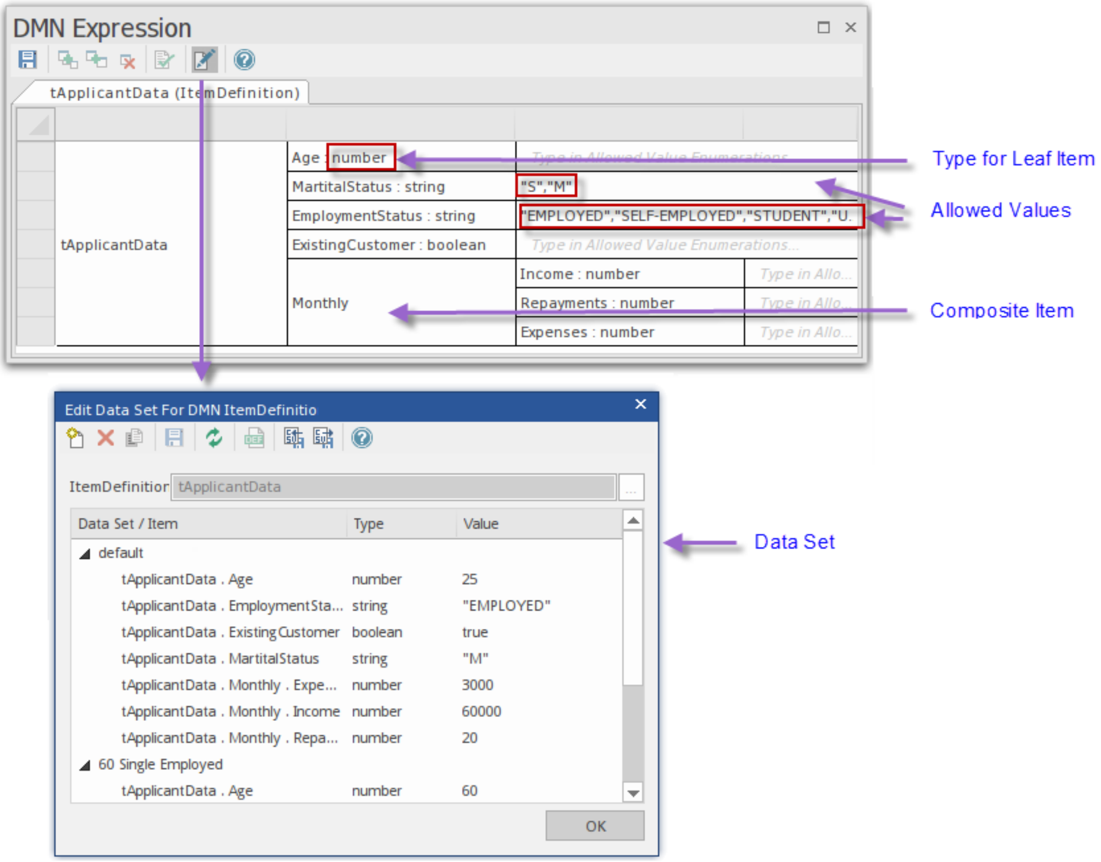
Access

To open the DMN Expression window for an ItemDefinition Element:

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select or create an ItemDefinition
Other	Double-click on a DMN ItemDefinition

DMN Expression and Data set

This image is an overview of the DMN Expression window, showing a complex data item and the layout of the key fields used in the definition of the data. Included is a view of a Data Set defined using this ItemDefinition. A Data Set is an 'instance' of data conforming to an ItemDefinition, which contains a set of values to be used in the DMN simulation.



tApplicantData	Age : number	Type in Allowed Value Enumerations...
	MartitalStatus : string	"S","M"
	EmploymentStatus : string	"EMPLOYED","SELF-EMPLOYED","STUDENT","U...
	ExistingCustomer : boolean	Type in Allowed Value Enumerations...
	Monthly	Income : number — Type in Allo...
		Repayments : number — Type in Allo...
		Expenses : number — Type in Allo...

ItemDefinitior tApplicantData

Data Set / Item	Type	Value
◢ default		
tApplicantData . Age	number	25
tApplicantData . EmploymentSta...	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	true
tApplicantData . MartitalStatus	string	"M"
tApplicantData . Monthly . Expe...	number	3000
tApplicantData . Monthly . Income	number	60000
tApplicantData . Monthly . Repa...	number	20
◢ 60 Single Employed		
tApplicantData . Age	number	60

As ItemDefinitions are foundation elements in the model, it is recommended that they are validated before going on to use them in the model. This will ensure that any issues are resolved early on in the process of creating a complex model.

For more details on setting up ItemDefinitions, see these Help topics:

- *DMN Item Definition, Data Set and Input Data*
- *Types of Component*
- *ItemDefinition Allowed Value*
- *DMN Expression Auto Completion*
- *DMN Expression Validation*

Item Definition Toolbar

This table provides descriptions of the features accessible in the DMN Expression window when an ItemDefinition element is selected.

Toolbar Options

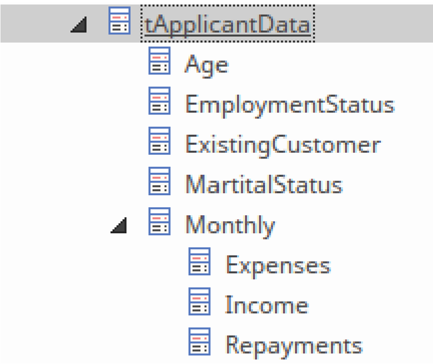
Option	Description
💾	Saves the configuration of the current ItemDefinition.
➕	Creates a new data component as a child of the selected component.
➕	Creates a new data component as a sibling of the selected component.
✗	Deletes the selected data component.
✓	Validates the ItemDefinition; Enterprise Architect will perform a series of validations to help you identify any errors in the ItemDefinition.
✎	Opens the 'Edit Data Set' dialog, in which you can create and edit 'instances' of the ItemDefinition for use by InputData elements.

ItemDefinitions and Data Sets

An ItemDefinition describes the types and structures of data items used in a Decision model. It serves as the data type definition for InputData elements, Decision elements and Business Knowledge Model parameters. An ItemDefinition can also define data sets that provide sets of values for use in DMN Simulations. Switching between different data sets provides the ability to do 'what-if' analysis using the Decision model.

ItemDefinition Structure

A complex ItemDefinition consists of nested elements. For example, *tApplicantData* is structured as:

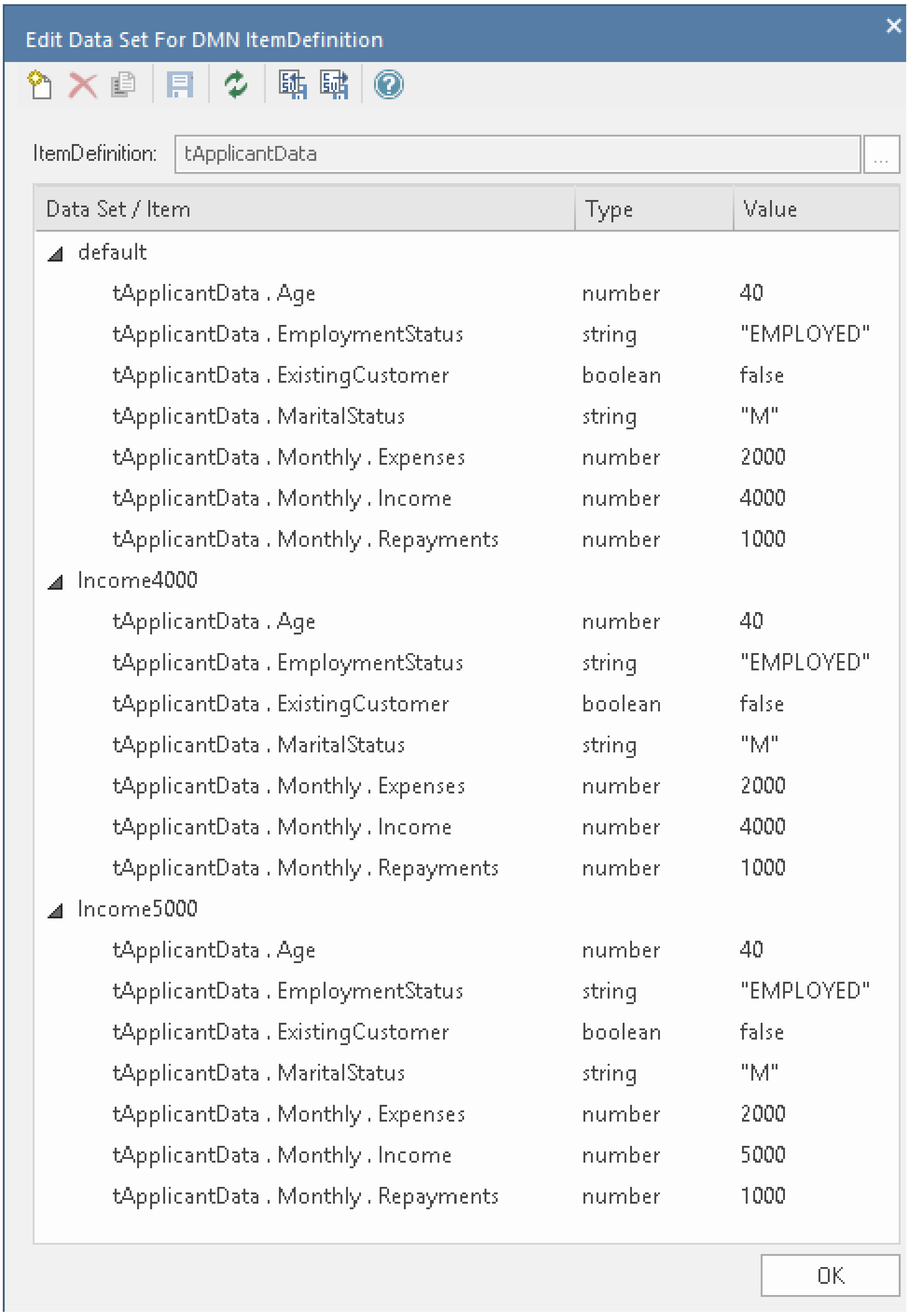


```
◢ tApplicantData
    Age
    EmploymentStatus
    ExistingCustomer
    MartitalStatus
  ◢ Monthly
      Expenses
      Income
      Repayments
```

The tApplicantData ItemDefinition example is a composite type of five child items. 'Monthly' is composed of three children (Expenses, Income and Repayments). The Leaf components (non-composite), will have a primitive type such as number, string or Boolean.

Data Set

The ItemDefinition's Data Set can be viewed and edited using the ✎ icon on the Toolbar. With the 'Edit Data Set' dialog, you can add, delete and duplicate the data sets. There is also support for CSV import and export of data sets.



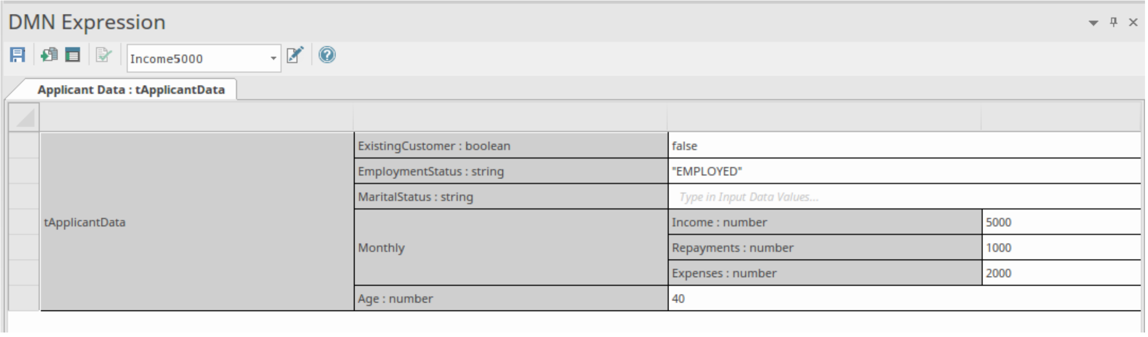
ItemDefinition: tApplicantData

Data Set / Item	Type	Value
default		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income4000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income5000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	5000
tApplicantData . Monthly . Repayments	number	1000

As shown in the example, the ItemDefinition for *tApplicantData* defines three data sets:

- default
- Income4000
- Income5000

Each data set can be viewed in an InputData element that is typed to the ItemDefinition. For example the 'Applicant Data' InputData element is typed to the 'tApplicantData' ItemDefinition. The DMN Expression window for 'Applicant Data', illustrated here, shows the data values according to the data set selected in the drop-down list in the window toolbar (*Income5000* in this case).



Applicant Data : tApplicantData

tApplicantData	ExistingCustomer : boolean	false	
	EmploymentStatus : string	"EMPLOYED"	
	MaritalStatus : string	Type in Input Data Values...	
	Monthly	Income : number	5000
		Repayments : number	1000
		Expenses : number	2000
	Age : number	40	

Setting a Reference to an ItemDefinition

A DMN InputData element is set to be referenced (typed) by an ItemDefinition using either:

- The icon on the DMN Expression window of the InputData element or
- Selecting the InputData element and pressing Ctrl+L to select the ItemDefinition from the dialog

There are other cases of using ItemDefinitions; for instance, when setting the type for an Input Parameter in a BKM or an output parameter in a Decision Table.

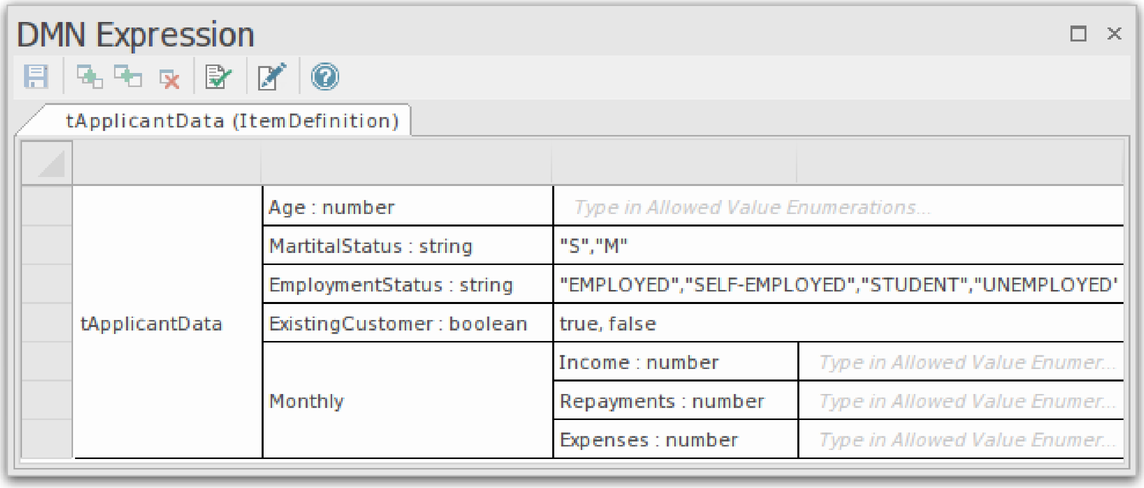
Types of Component

An ItemDefinition element can be defined as a tree of components that consists of only one of either:

- A built-in type or
- A Composition of ItemDefinition elements

In this tree of components, if a component is a 'leaf' that has no child components, it must be set as a built-in type. If an ItemDefinition has child components, it is those child/leaf components that are set as a built-in type.

For example *Applicant Data* and *Monthly* are compositions, whereas *Age* and *Expenses* are leaves set to a built-in type:



DMN Expression

tApplicantData (ItemDefinition)

tApplicantData	Age : number	Type in Allowed Value Enumerations...	
	MartitalStatus : string	"S","M"	
	EmploymentStatus : string	"EMPLOYED","SELF-EMPLOYED","STUDENT","UNEMPLOYED'	
	ExistingCustomer : boolean	true, false	
	Monthly	Income : number	Type in Allowed Value Enumer...
		Repayments : number	Type in Allowed Value Enumer...
		Expenses : number	Type in Allowed Value Enumer...

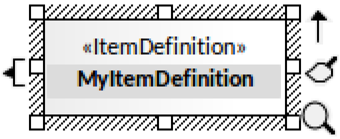
The FEEL language has these built-in types:

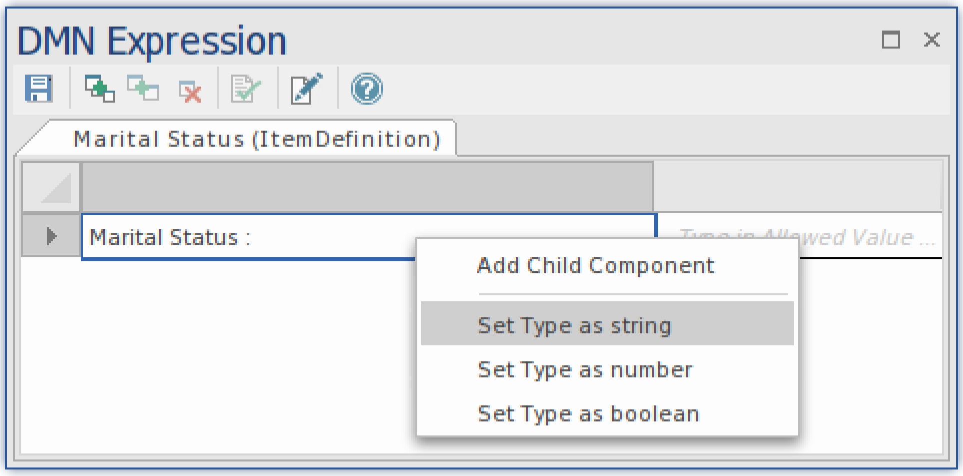
- **number**
- **string**
- **boolean**
- days and time duration
- years and months duration
- time
- date and time

Note: 'number', 'string' and 'boolean' are supported by Enterprise Architect for simulation.

To set a type for a 'leaf' ItemDefinition, you can use one of three methods:

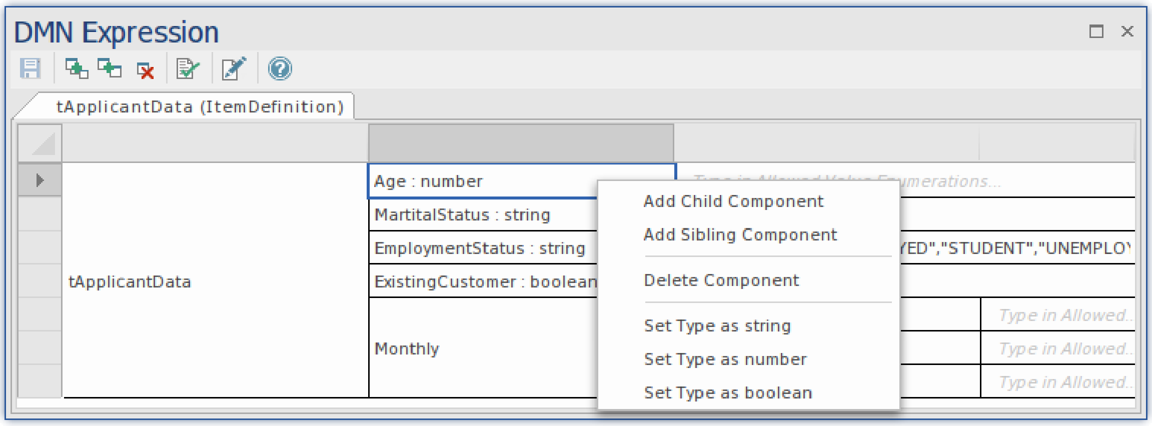
- Select the appropriate context menu option in the DMN Expression window (Recommended)





- Type ': string', ': boolean' or ': number' after the name in the cell in the DMN Expression window
- Type 'string', 'boolean' or 'number' as the value of the tag 'Type' in the Properties window for the ItemDefinition

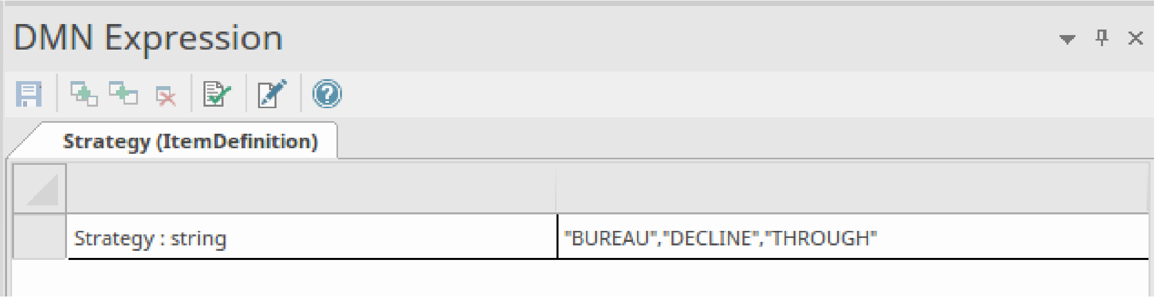
For composite ItemDefinitions, the context menu also offers options to create a child or a sibling component, or to delete the selected item:



Allowed Value Enumerations

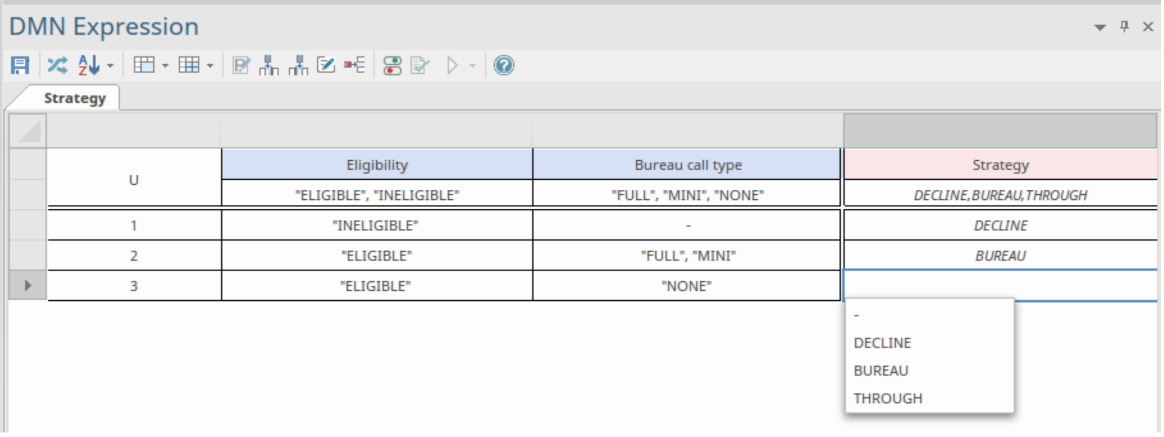
When defining data inputs for a Decision, it is common to want to restrict the set of allowable values for an input. For example, you might want to restrict the allowed values for Marital Status to just two options, 'Single' and 'Married'.

You can specify the allowed values for any leaf component of an ItemDefinition. Initially, the data field for a leaf component contains the text *Type in Allowed Value Enumerations*. You simply type over this text with the allowed values. For example, the ItemDefinition *Strategy* has three allowed values - BUREAU, DECLINE and THROUGH.



Strategy : string	"BUREAU","DECLINE","THROUGH"

Allowed Value Enumerations are also used to support Auto Completion. When specifying values for an InputData element or an input parameter that references an ItemDefinition in which Allowed Values have been defined, the user can simply press the Spacebar and choose a value from the list.

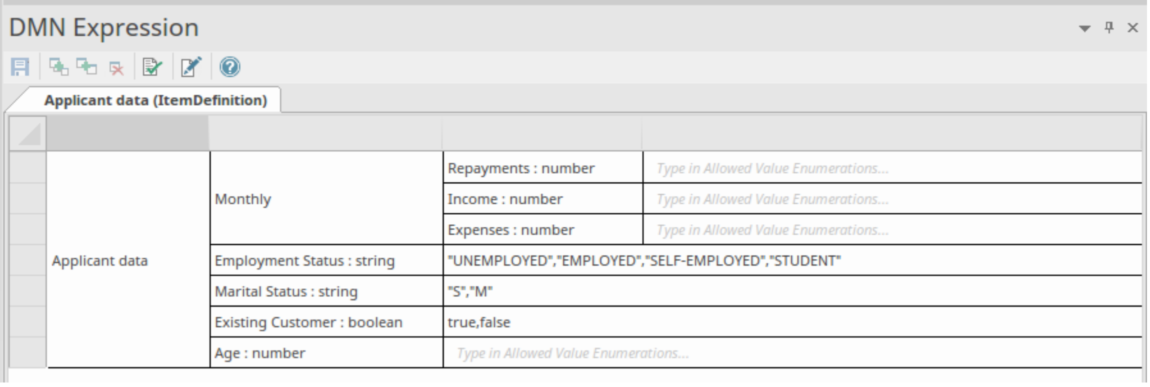


U	Eligibility	Bureau call type	Strategy
	"ELIGIBLE", "INELIGIBLE"	"FULL", "MINI", "NONE"	DECLINE,BUREAU,THROUGH
1	"INELIGIBLE"	-	DECLINE
2	"ELIGIBLE"	"FULL", "MINI"	BUREAU
3	"ELIGIBLE"	"NONE"	

You can also autocomplete by typing the first letter of the option you want to enter.

The input parameters and Output Clauses of Decision Tables also support the specification of allowable values. This restricts the values that can be used when defining the rules in the table, but also allows the user to fast fill the rules by pressing the Spacebar then selecting the required item.

A more complex ItemDefinition can include a number of Allowed Value Enumerations; for example:



Applicant data	Monthly	Repayments : number	Type in Allowed Value Enumerations...
		Income : number	Type in Allowed Value Enumerations...
		Expenses : number	Type in Allowed Value Enumerations...
	Employment Status : string	"UNEMPLOYED","EMPLOYED","SELF-EMPLOYED","STUDENT"	
	Marital Status : string	"S","M"	
	Existing Customer : boolean	true,false	
	Age : number	Type in Allowed Value Enumerations...	

Data Sets

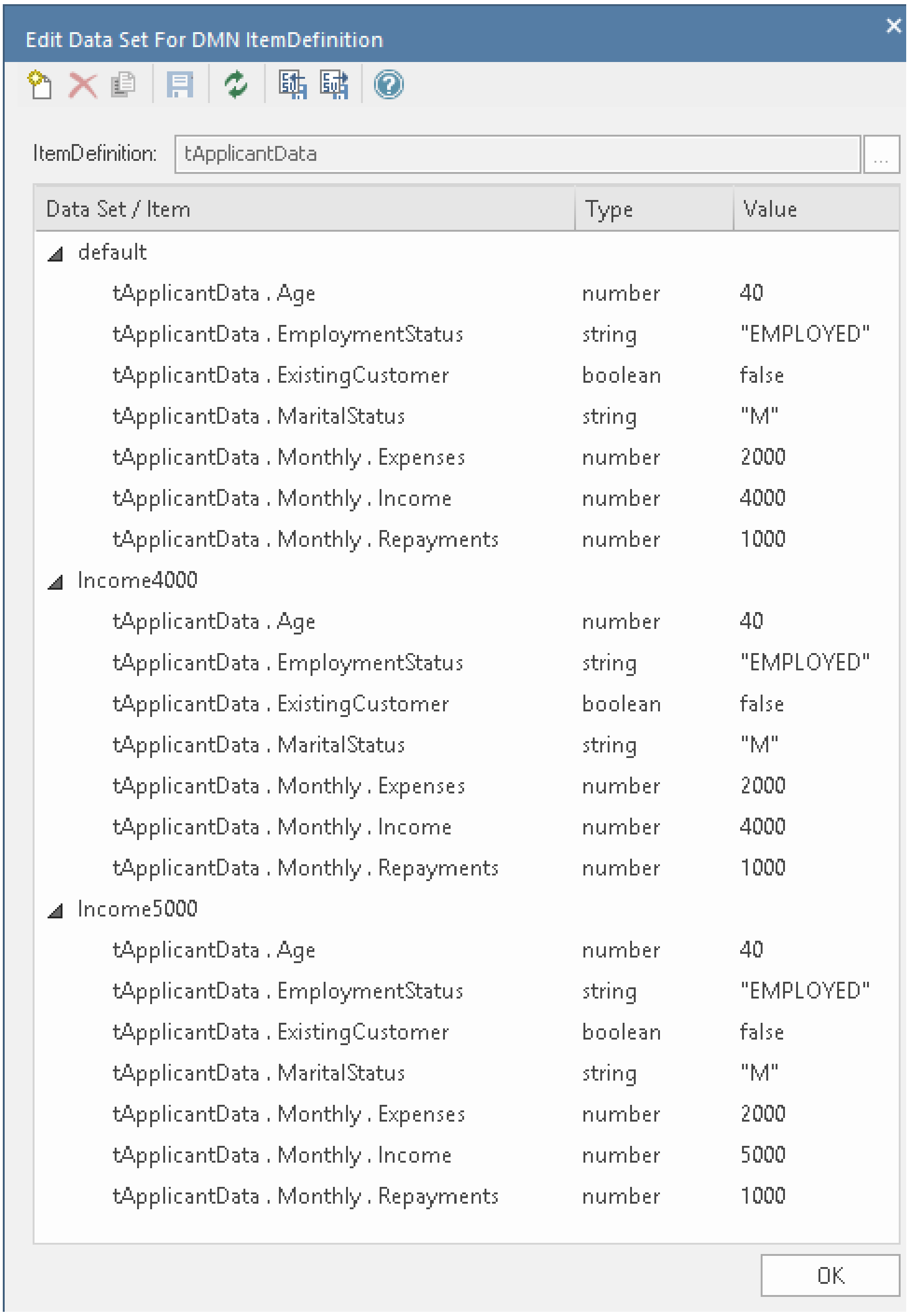
Each InputData element typed by an ItemDefinition has a set of components, and multiple data sets can be defined to provide different sets of values for those components. With this feature, a user performing a DMN Simulation can quickly test the result of a decision by choosing different data sets. The data sets are associated with and based on the ItemDefinition, but you can also work on them via the InputData element.

You add or update data sets using the 'Edit Data Set' dialog, which you invoke from the DMN Expression window for either the ItemDefinition or the InputData element. Initially, the 'Edit Data Set' dialog shows a single set of components with no values, under the set name 'default'. You can either leave this set with no values, or provide values; either way, you can use this as a template to duplicate for new data sets. You cannot delete the 'default' data set.

When you access an InputData element in the DMN Expression window, the values in the 'default' data set are shown against the components of the element. You can then click on the drop-down arrow in the toolbar and select any other data set from the list. Note that if you leave the 'default' data set untouched, you can create a duplicate 'default' data set and assign values to it, and that 'default' set will provide the values when you initially access the InputData element.

You can duplicate and delete any other data set that you create, export the data sets to a CSV file and import them from a CSV file.

Note that if you create a data set and do not enter values, it is discarded when you close the dialog.



ItemDefinition: tApplicantData

Data Set / Item	Type	Value
default		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income4000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income5000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	5000
tApplicantData . Monthly . Repayments	number	1000

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression > click on InputData item : ✎ icon
Other	In a diagram, double-click on the DMN InputData element : ✎ icon.

Toolbar Options

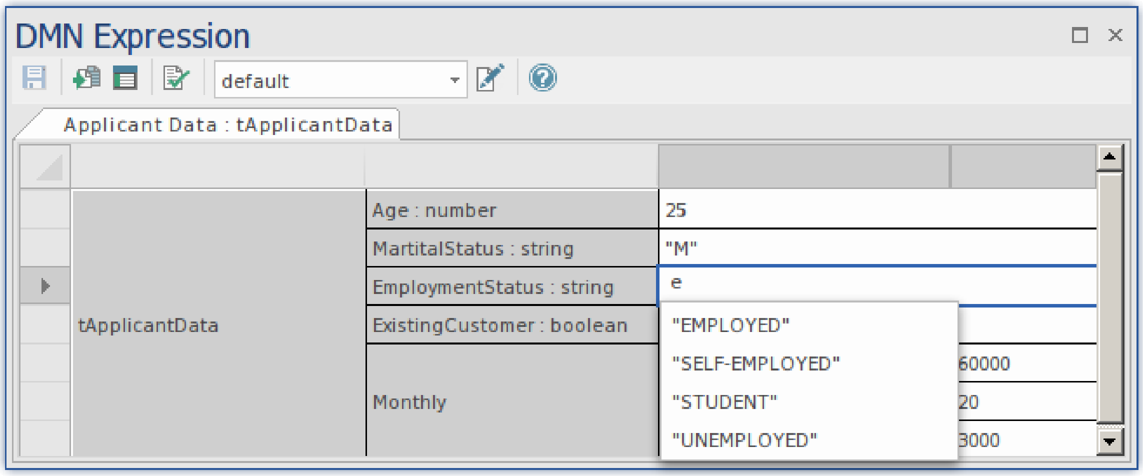
Option	Description
	Click on this button to create a new data set.
✕	Click on this button to delete the selected data set.
	Click this button to duplicate the selected data set.
	Click on this button to save the data sets to the InputData.
🔄	Click on this button to reload the data sets for the InputData.
	Click on this button to import data sets from a CSV file.
	Click on this button to export the datasets to a CSV file.

Exchange Data Sets using DataObjects

When testing code generated from a DMN model, or when simulating Business Process Model and Notation (BPMN) models that call DMN models, you need a means of exchanging data sets. For example, in a BPMN call of a DMN model, a BPMN DataObject is used to store the set of variables that will be passed on to the DMN model that it is calling. This DataObject needs to be populated with data fitting the DMN InputData's data structure ready to be passed to that InputData object. This same BPMN DataObject is used when testing the code generated from a DMN model.

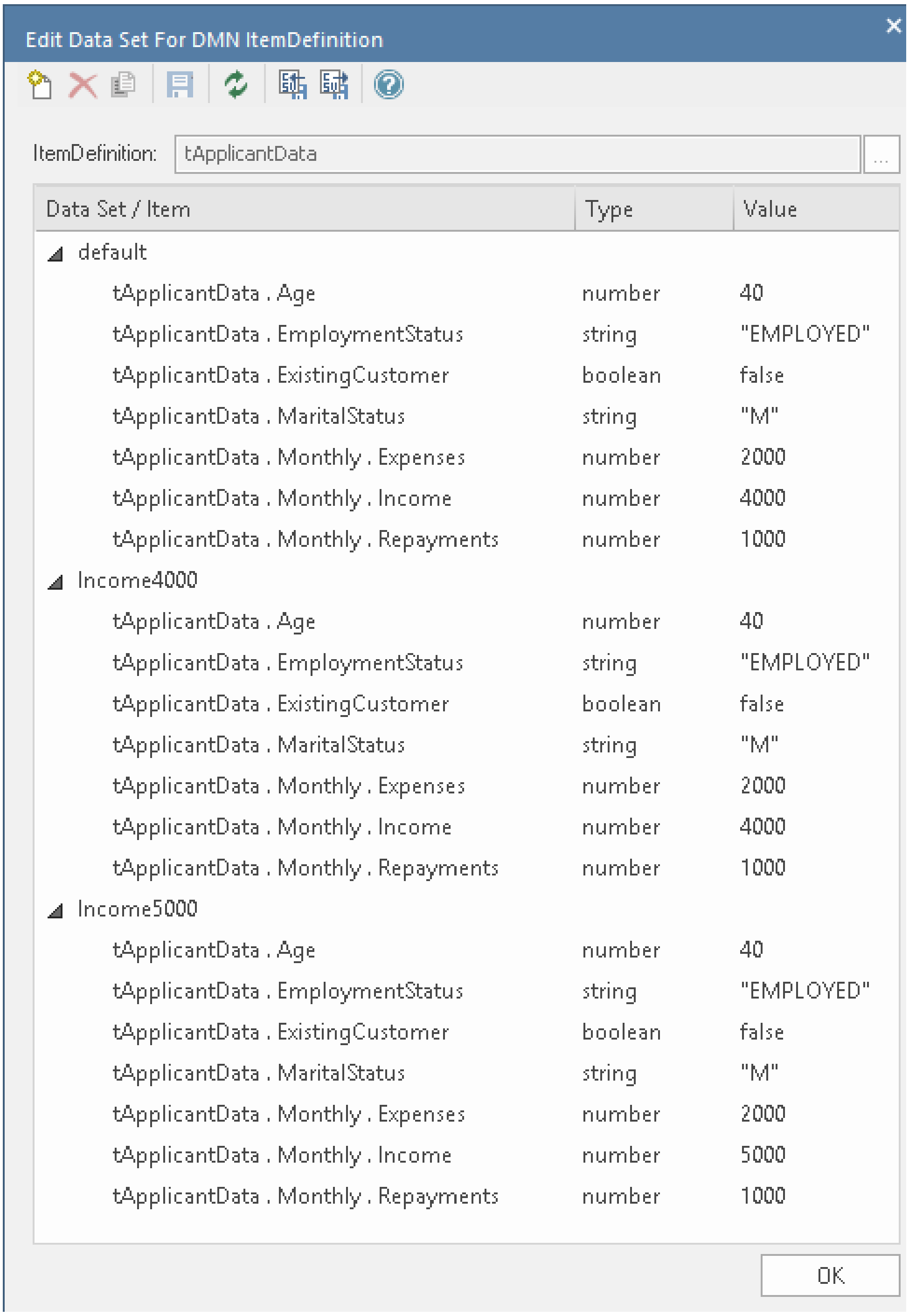
This topic describes the process of creating BPMN DataObjects from DMN Data Sets.

A Data Set is stored in a DMN InputData element and can be accessed using the ✎ icon on the DMN Expression window.



tApplicantData	Age : number	25
	MartitalStatus : string	"M"
	EmploymentStatus : string	e
	ExistingCustomer : boolean	
	Monthly	60000
		20
		3000

This opens the InputData's 'Edit Data Set' dialog, which can contain multiple sets of values:



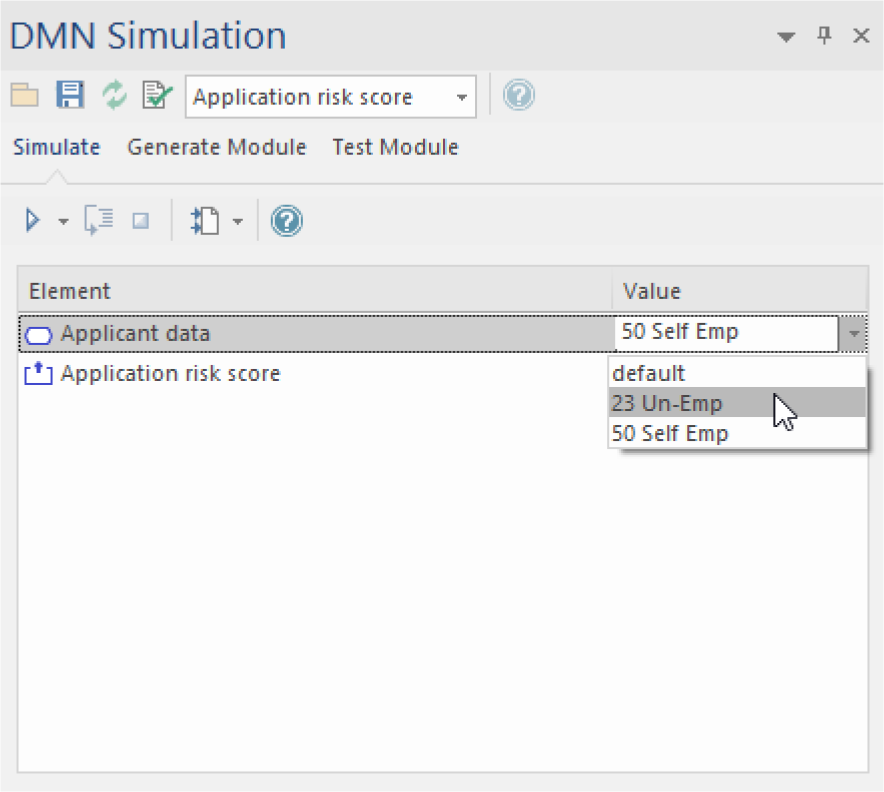
ItemDefinition: tApplicantData

Data Set / Item	Type	Value
default		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income4000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income5000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	5000
tApplicantData . Monthly . Repayments	number	1000

There are two options to transfer the Data Set to a DataObject:

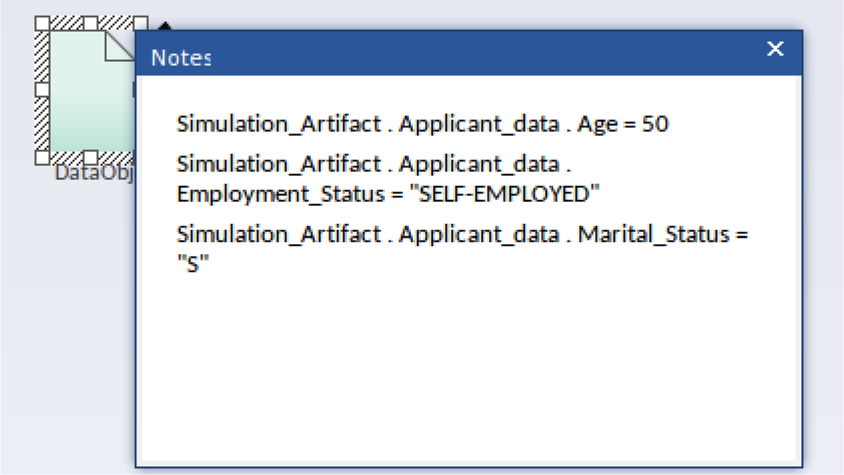
1. Direct

- Create a BPMN DataObject under a Package in the Browser window.
- Open the DMN Simulation window



- Select a Data Set from the 'Value' drop-down
- Click on the icon on the DMN Simulation window; this opens the 'Select Element' dialog
- Select the BPMN DataObject element
- Click on the OK button

The Data Set is now viewable in the Notes of the DataObject.



Simulation_Artifact . Applicant_data . Age = 50

Simulation_Artifact . Applicant_data . Employment_Status = "SELF-EMPLOYED"

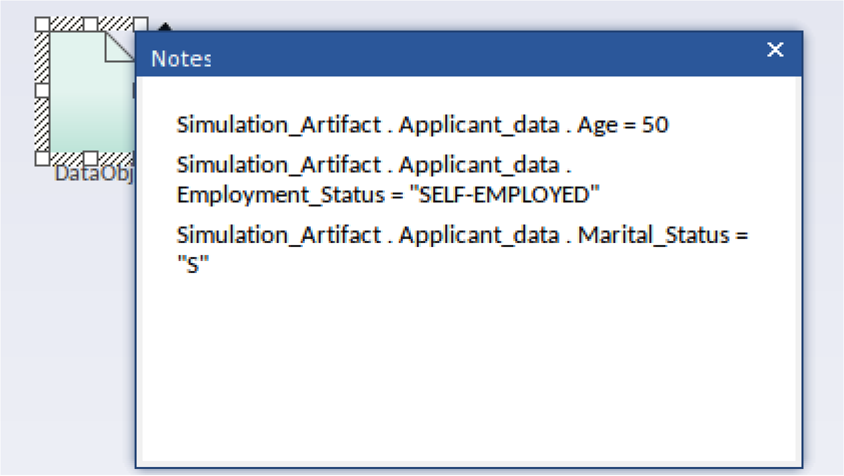
Simulation_Artifact . Applicant_data . Marital_Status = "S"

2. Manual

To manually exchange this Dataset:

- Open the DMN Expression window for the InputData element
- Click on the Edit DataSet icon ; this opens the 'Edit Data Set' dialog
- Use the CSV Export icon to export these details to a file

The text in the CSV file can be added as text in the Notes of a BPMN DataObject element.



Simulation_Artifact . Applicant_data . Age = 50

Simulation_Artifact . Applicant_data . Employment_Status = "SELF-EMPLOYED"

Simulation_Artifact . Applicant_data . Marital_Status = "S"

Decision Service

Portions of this topic have been used verbatim or are freely adapted from the DMN Specification, which is available at: https://www.omg.org/spec/DMN. This site contains a full description of the DMN and its capabilities.

A Decision Service exposes one or more decisions from a Decision model as a reusable element, which might be invoked internally by another decision in the Decision model, or externally by a task in a BPMN process model.

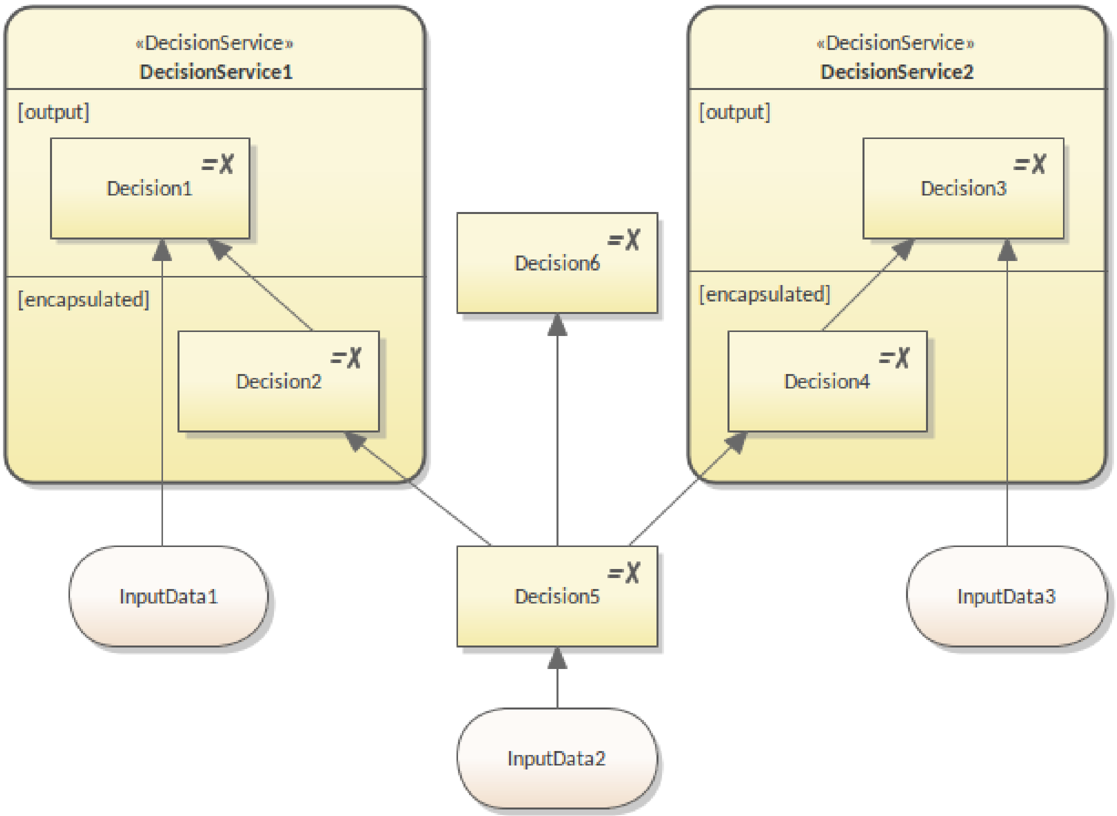
When the Decision Service is called with the necessary input data and input decisions, it returns the outputs of the exposed decisions.

The Interface of a Decision Service

The interface to the Decision Service consists of:

- Input data - instances of all the input data required by the encapsulated decisions
- Input decisions - instances of the results of all the input decisions
- Output decisions - the results of evaluating (at least) all the output decisions, using the provided input decisions and input data

When the Decision Service is called with the necessary input data and input decisions, it returns the outputs of the exposed decisions.



This figure shows a Decision model that includes six decisions and three items of input data.

For DecisionService1, the:

- Output decision is {Decision1}
- Input decision is {Decision5}, and
- Input data is {InputData1}

As Decision1 requires Decision2, which is not provided to the service as input, the service must also encapsulate

Decision2; therefore the encapsulated decisions are {Decision1, Decision2}.

It is obvious from the figure that Decision6, Decision3, Decision4 and InputData3 are not required by any decisions from DecisionService1. What about InputData2? Although it is required by Decision5, which is required by DecisionService1, InputData2 is actually not required by DecisionService1. This is because Decision5 is defined as the Input Decision. From the point of view of a Decision Service, we ignore any decisions or input data required by an Input Decision.

For DecisionService2, the:

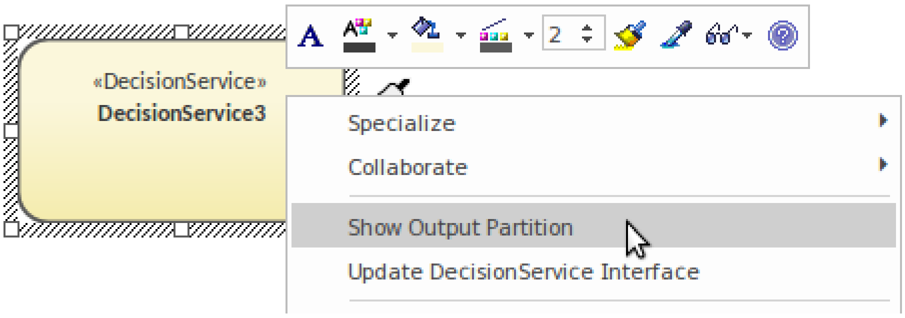
- Output decision is {Decision3}
- Input decision is {Decision5}, and
- Input data is {InputData3}

As Decision3 requires Decision4, which is not provided to the service as input, the service must also encapsulate Decision4; therefore the encapsulated decisions are {Decision3, Decision4}.

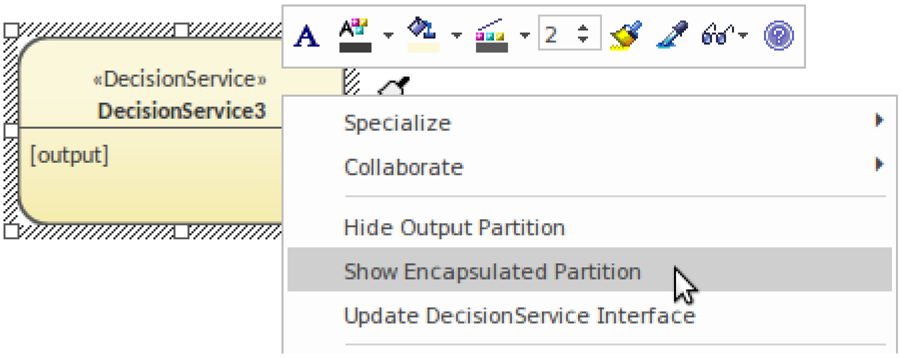
It is good practice to create a separate diagram for each Decision Service. In this way, the diagram will only contain the interface elements and encapsulated decisions for the Decision Service; the elements that are not relevant will not appear on the diagram.

Modeling a Decision Service

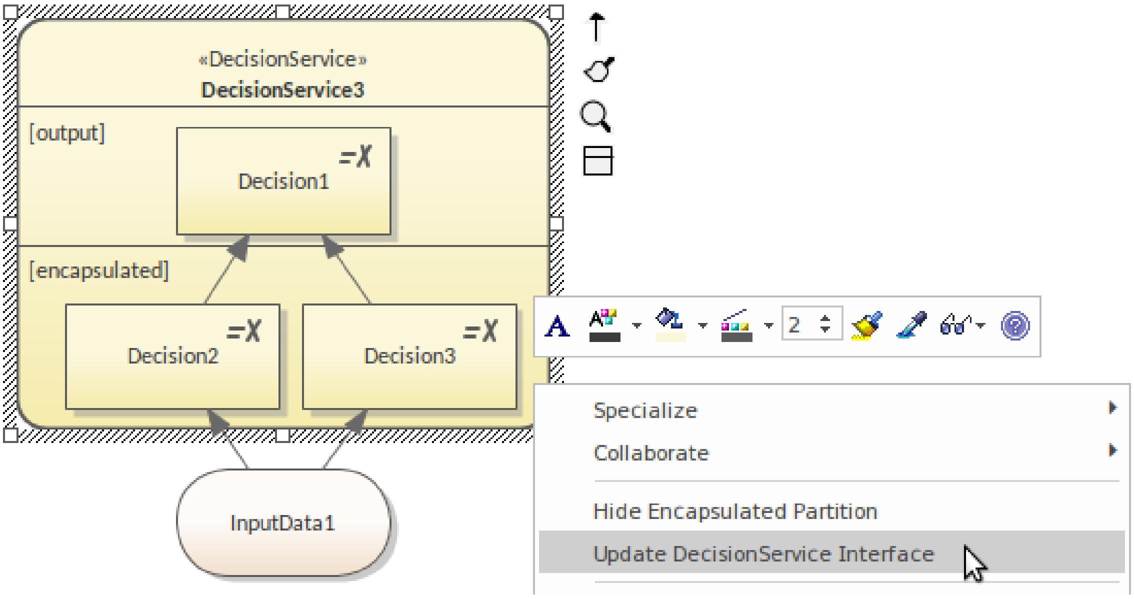
We can create a Decision Service element from the DMN pages of the Diagram Toolbox, and toggle [output] and [encapsulated] partitions from the context menu.



You can only show an [encapsulated] partition when an [output] partition is shown.



Once the decisions and input data are put in the correct partition(s), you must run the 'Update DecisionService Interface" command from the context menu to update the model.



Important: in order for the DMN simulation to work properly, please update the Decision Service interface whenever you:

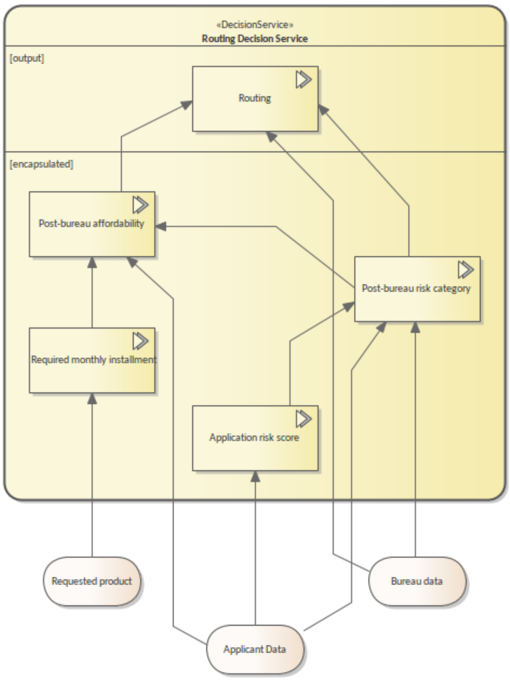
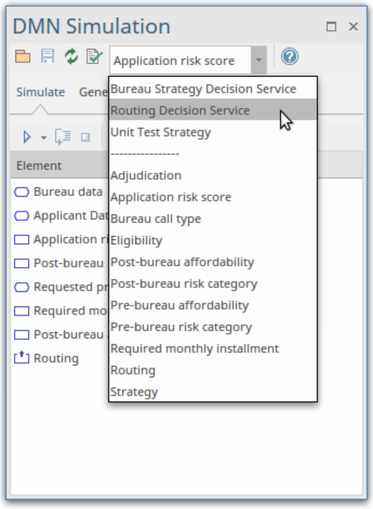
- Show/Hide the decision service partition(s)
- Add a decision to the decision service
- Remove a decision from the decision service
- Move a decision between partitions
- Add/Remove Decision Service Inputs: Input Data or Input Decisions

Simulating a Decision Service

It is possible to perform a model simulation on a Decision Service.

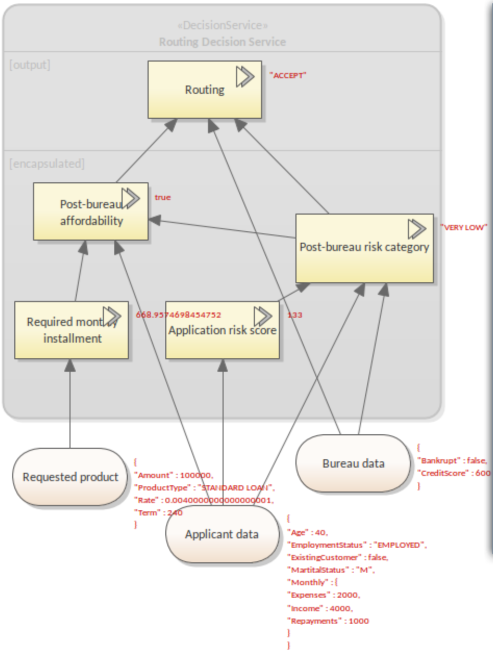
Decision Service Simulation

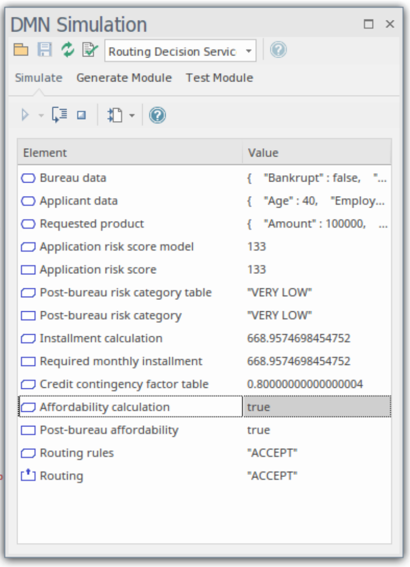
To perform a model simulation on the Decision Service, work through these steps:

Step	Description
1	Drag a Simulation Configuration Artifact element onto a diagram from the 'DMN Components' page of the Toolbox, and double-click on it to open it in the DMN Simulation window.  «DecisionService» Routing Decision Service [output] Routing [encapsulated] Post-bureau affordability Post-bureau risk category Required monthly installment Application risk score Requested product Bureau data Applicant Data  DMN Simulation Application risk score Simulate Gene Element Bureau data Applicant Dat Application r Post-bureau Requested pr Required mo Post-bureau Routing Bureau Strategy Decision Service Routing Decision Service Unit Test Strategy --------------- Adjudication Application risk score Bureau call type Eligibility Post-bureau affordability Post-bureau risk category Pre-bureau affordability Pre-bureau risk category Required monthly installment Routing Strategy By default, all Decision Service elements and each single decision are listed for selection in the drop-down field in the dialog toolbar.
2	Select a Decision Service element on which to run the simulation. In the example we chose 'Routing Decision Service', so three input data items and five encapsulated decisions (including one output decision) are loaded in the simulation list. Important: This list is drawn from the internal data of the Decision Service; make sure you run the 'Update DecisionService Interface' command from the context menu whenever the Decision Service model diagram is changed. Reload the Decision Model by clicking the 'Refresh' icon (third from the left) on the DMN Simulation window toolbar.
3	The input data and decisions are in the correct execution order. For example, 'Application risk score' will be executed before 'Post-bureau risk category', 'Post bureau affordability' and 'Routing'. For each Input Data element, click on the drop-down arrow in the 'Value' field and select the Data Sets to provide input data values. Validate the input data and decisions, and make any necessary corrections using the

<table>
<tr>
<td></td>
<td>DMN Expression window.

On the DMN Simulation window, click on the Save icon and on the ▷ ▾ button on the toolbar.</td>
</tr>
<tr>
<td>4</td>
<td>
<p>The runtime execution result is shown both in the list and on the diagram. You can also click on the 'Step-through' icon on the toolbar to debug the DMN model.</p>





<table>
<tr><th>Element</th><th>Value</th></tr>
<tr><td>Bureau data</td><td>{ "Bankrupt" : false, "...</td></tr>
<tr><td>Applicant data</td><td>{ "Age" : 40, "Employ...</td></tr>
<tr><td>Requested product</td><td>{ "Amount" : 100000, ...</td></tr>
<tr><td>Application risk score model</td><td>133</td></tr>
<tr><td>Application risk score</td><td>133</td></tr>
<tr><td>Post-bureau risk category table</td><td>"VERY LOW"</td></tr>
<tr><td>Post-bureau risk category</td><td>"VERY LOW"</td></tr>
<tr><td>Installment calculation</td><td>668.9574698454752</td></tr>
<tr><td>Required monthly installment</td><td>668.9574698454752</td></tr>
<tr><td>Credit contingency factor table</td><td>0.8000000000000004</td></tr>
<tr><td>Affordability calculation</td><td>true</td></tr>
<tr><td>Post-bureau affordability</td><td>true</td></tr>
<tr><td>Routing rules</td><td>"ACCEPT"</td></tr>
<tr><td>Routing</td><td>"ACCEPT"</td></tr>
</table>

<p>A good practice is to keep the DMN Expression window open while debugging. The run time status of the expression (such as Decision Table, Boxed Context, Literal Expression or Invocation) will show the details of the logic encapsulated by the Decision or invoked Business Knowledge Model.</p>
</td>
</tr>
</table>

Code Generation and Test Module

After a Decision model is created and simulated, you can generate a DMN module in Java, JavaScript, C++ or C#. That DMN module can be used with the Enterprise Architect BPSim Execution Engine, Executable StateMachine, or your own project.

Enterprise Architect also provides a 'Test Module' page, which is a preprocess for integrating DMN with BPMN. The concept is to provide one or more BPMN2.0::DataObject elements, then test if a specified target Decision can be evaluated correctly or not.

If any error or exception occurs, you can create an Analyzer Script to debug the code of the DMN module and Test Client.

After this 'Test Module' process, Enterprise Architect guarantees that the BPMN2.0::DataObject elements will work well with the DMN Module.

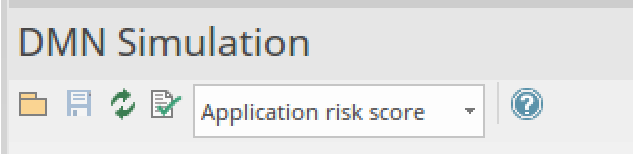
You then configure BPSim by loading DataObjects and assigning DMN module Decisions to BPSim Properties, which will be further used as conditions on the Sequence Flows outgoing from a Gateway.

Access

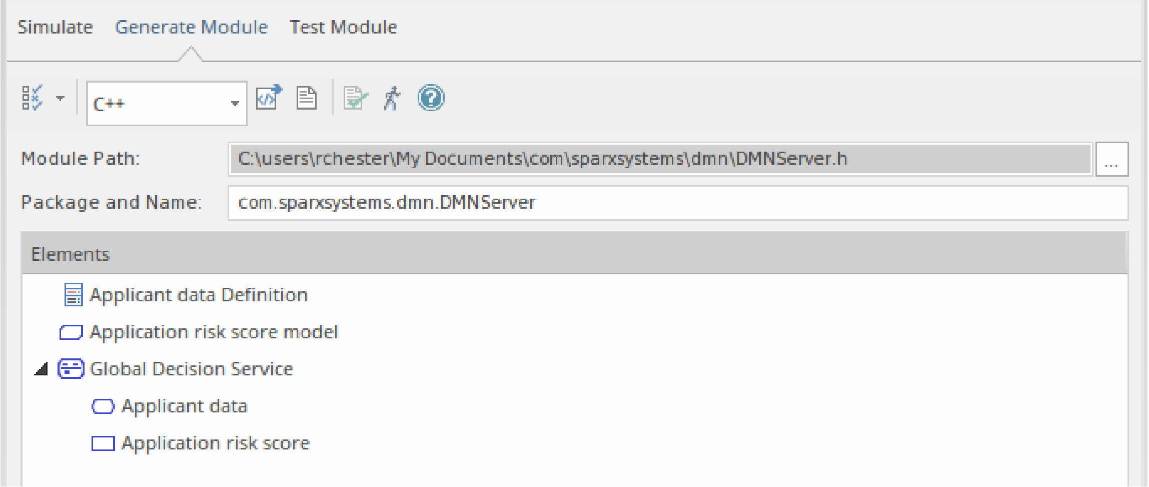
Ribbon	Simulate > Decision Analysis > DMN > Open DMN Simulation > Generate Module

DMN Module: Code Generation

On the DMN Simulation window, select the DMN structure you want to generate the module from, in the data entry field of the Toolbar.



Click on the 'Generate Module' tab, and then Ctrl+click on the names of the DMN elements you want to generate to the server.



C++

Module Path: C:\users\rchester\My Documents\com\sparxsystems\dmn\DMNServer.h

Package and Name: com.sparxsystems.dmn.DMNServer

In the data entry field in the tab toolbar, select the language to generate in, and in the 'Module Path' field click on the

... icon and browse to the path location to generate the module into (note, for Java the path has to match the Package structure).

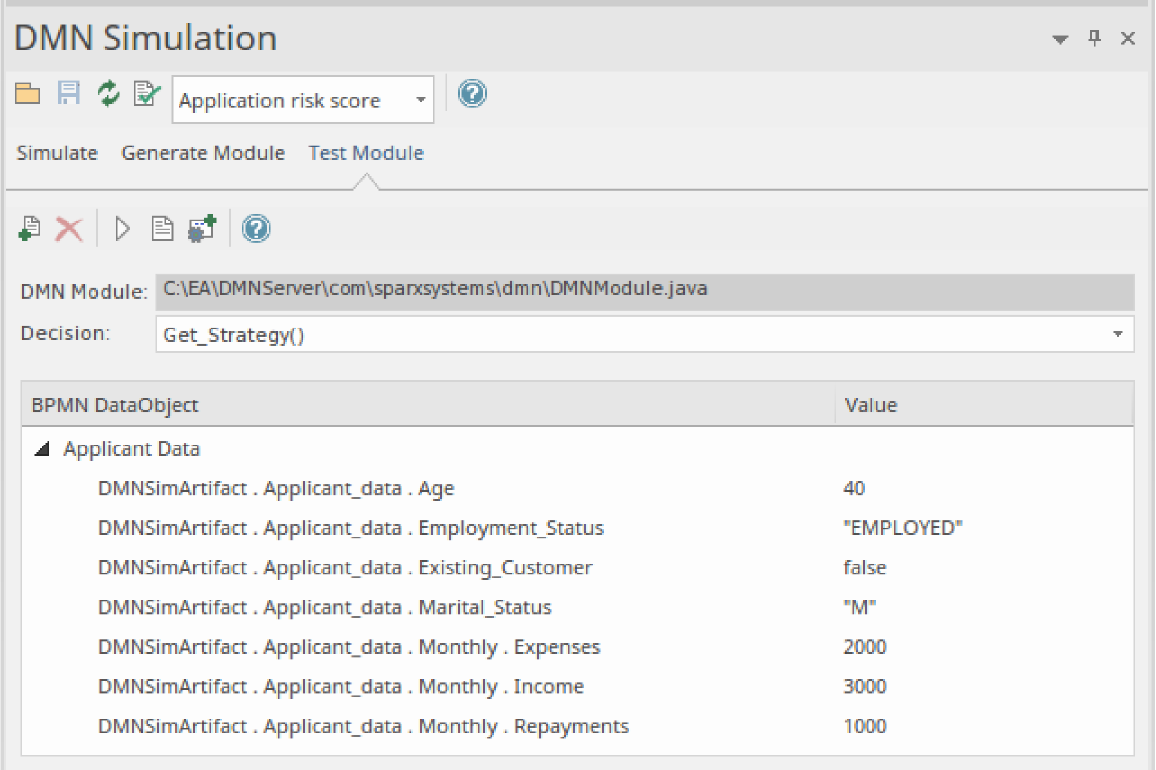
Click on the Generate button ( ).

When the generation is complete, click on the button to open the 'Test Module' tab for the generated module.

DMN Server: Test Module

When you use the button to select the 'Test Module' tab, the 'DMN Module' field will be filled automatically with the generated DMN Server path of the module you most recently generated on the 'Generate Module' tab. If necessary, on the 'Decision' field click on the drop-down arrow and select the required Decision.

Click on the Add BPMN DataObject button ( ) in the Toolbar and select one or more (Ctrl+Click) BPMN2.0 DataObject(s) to add to the list in the main panel.



DMN Module: C:\EA\DMNServer\com\sparxsystems\dmn\DMNModule.java

Decision: Get_Strategy()

BPMN DataObject	Value
◢ Applicant Data	
DMNSimArtifact . Applicant_data . Age	40
DMNSimArtifact . Applicant_data . Employment_Status	"EMPLOYED"
DMNSimArtifact . Applicant_data . Existing_Customer	false
DMNSimArtifact . Applicant_data . Marital_Status	"M"
DMNSimArtifact . Applicant_data . Monthly . Expenses	2000
DMNSimArtifact . Applicant_data . Monthly . Income	3000
DMNSimArtifact . Applicant_data . Monthly . Repayments	1000

Now click the Run button on the toolbar. In the System Output window, this message indicates the DMN Server and BPMN2.0 DataObject can work well with each other to evaluate the selected decision:

```
Running Test Client for DMN Server...
   dmnServer.Application_risk_score: 133.0
Result : 133.0
The Running completed successfully.
```

If there are errors, create an Analyzer script by clicking the toolbar button and use the script to fix the issue.

Important: This 'Test Module' step is recommended before integrating DMNServer.java with the Enterprise Architect BPSim Execution Engine. See the *Integrate a DMN Module Into BPSim for Simulation* Help topic.

Code Generation & Connect to BPMN

- Generate the DMN Server in Java, JavaScript, C++, or C#
- Run/Debug tests of the Java version of the DMN Server
- Connect the DMN Server to the Enterprise Architect BPSim Execution Engine

Common Errors & Solutions

- Variable Types: as DMN models use the FEEL language (Simulate with JavaScript), typing variables is not compulsory; however, when generating code to languages that are compiled, you do have to type a variable - there are context menu options and tag values for setting the type of a variable
- Since a DMN expression allows for spaces, in order to clarify the composite Input Data there must be a space before and after the '.' in the expression; for example, 'Applicant data . Age' is valid, whereas 'Applicant data.Age' is not valid
Note that when using the Auto Completion feature this issue will not arise
- Running validation will help you locate most of the modeling issues; do this before simulation and code generation

Notes

- Compiling with Java requires full read-write access to the target directory; compilation will fail if the module path is set to just 'C:' or 'C:\Program Files (x86)'

Integrate Into BPSim for Simulation

The strength of DMN is its ability to describe business requirements through the Decision Requirement diagram and to encapsulate the complicated logic in versatile expressions such as the Decision Table and Boxed Context.

Equally, the strength of BPMN is its ability to describe business processes with a Sequence Flow of tasks and events, or to describe collaborations of processes with Message Flows.

The Decision Requirements diagram forms a bridge between Business Process models and decision logic models:

- Business Process models define tasks within business processes, where decision-making is required
- Decision Requirements diagrams define the decisions to be made in those tasks, their interrelationships, and their requirements for decision logic
- Decision logic defines the required decisions in sufficient detail to allow validation and/or automation

DMN provides a complete Decision model that complements a Business Process model by specifying in detail the decision-making carried out in process tasks.

The two examples demonstrated in this topic can be accessed from:

- EAExample Model | Model Simulation | BPSim Models
- Perspective | Business Modeling | BPSim | BPSim Case Studies

There are two ways in which BPSim expressions use a DMN model:

- DMN's Decision Service - demonstrated by the Loan Application Process
- DMN's BusinessKnowledgeModel - demonstrated by the Delivery Cost Calculation

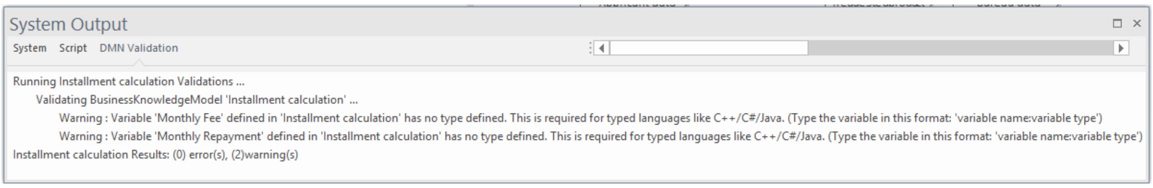
The process of integrating a DMN model with a BPSim model includes:

- DMN Model Validation, Simulation, Code Generation and Testing on the generated module
- Set up a usage dependency from the BPSim Artifact to the DMN Artifact
- Generate or update the BPMN DataObject from the DMN DataSet
- Create Property Parameters in BPSim to be used on tasks and Sequence Flows out going from Gateways
- Bind the DMN interface to BPSim Property Parameters

DMN Model Validation for Compiled languages such as Java

When you create a DMN model and simulate it in Enterprise Architect, the code driving the simulation is JavaScript; this means that the variables do not need to be explicitly typed (the variable type is inferred from the value assigned to it).

However, for languages such as C++, C# and Java, the compiler will report an error that a variable does not have a type.



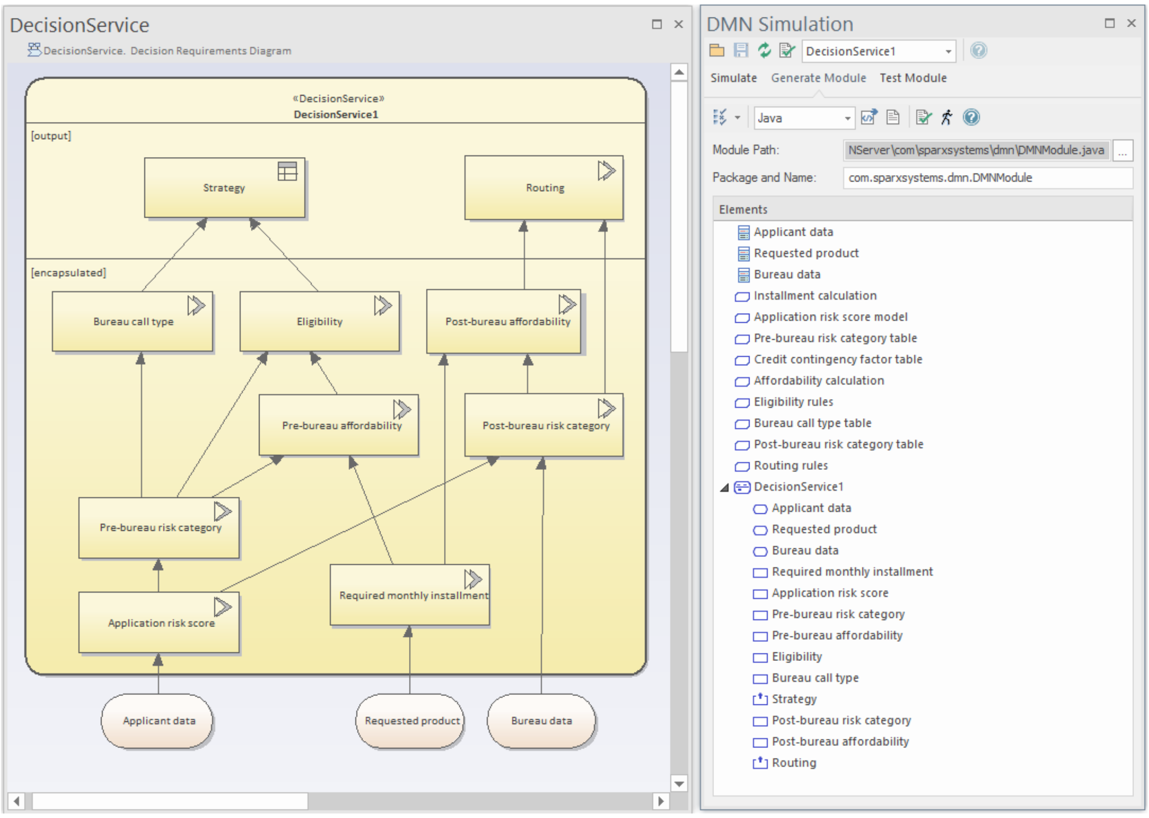
```
Running Installment calculation Validations ...
    Validating BusinessKnowledgeModel 'Installment calculation' ...
        Warning : Variable 'Monthly Fee' defined in 'Installment calculation' has no type defined. This is required for typed languages like C++/C#/Java. (Type the variable in this format: 'variable name:variable type')
        Warning : Variable 'Monthly Repayment' defined in 'Installment calculation' has no type defined. This is required for typed languages like C++/C#/Java. (Type the variable in this format: 'variable name:variable type')
Installment calculation Results: (0) error(s), (2)warning(s)
```

For generation to these languages you must run validation on the model and use the results to find variables that need their type set. For example:

- Business Knowledge Model parameter - select the BKM element to view in the DMN Expression window, click on the second button to open the 'Parameter' dialog, specify a type for the parameter
- Decision type - select the Decision element, open the Properties window, for the property 'variableType' select from the 'Value' field
- Decision Table Input/Output clauses - on the Decision Table Input/Output clause, right-click to display the context menu and choose the type
- Boxed Context variables - refer to the Boxed Context Help topic

DMN Code Generation In Java

After using validation to fix any variable type issues, we can proceed to the 'Generate Module' page in the DMN Simulation window.



Module Path: NServer\com\sparxsystems\dmn\DMNModule.java

Package and Name: com.sparxsystems.dmn.DMNModule

Elements

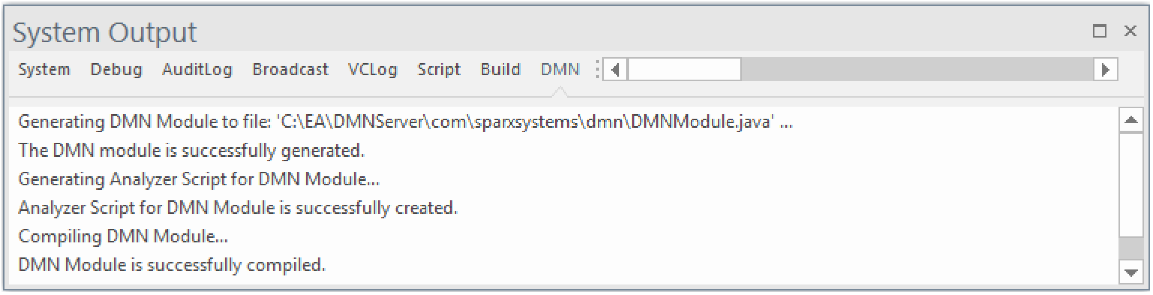
- Applicant data
- Requested product
- Bureau data
- Installment calculation
- Application risk score model
- Pre-bureau risk category table
- Credit contingency factor table
- Affordability calculation
- Eligibility rules
- Bureau call type table
- Post-bureau risk category table
- Routing rules
- DecisionService1
  - Applicant data
  - Requested product
  - Bureau data
  - Required monthly installment
  - Application risk score
  - Pre-bureau risk category
  - Pre-bureau affordability
  - Eligibility
  - Bureau call type
  - Strategy
  - Post-bureau risk category
  - Post-bureau affordability
  - Routing

- Select *DecisionService1* in the top toolbar data entry field; all the elements involved in *DecisionService1* will now be included in the list
- Item Definition and Business Knowledge Model are global elements
- Input Data and Decisions are encapsulated in the DecisionService element
- The supported languages are C++, C#, Java and JavaScript; note that for JavaScript the generated .js file is the same as the simulation script ('Simulate' tab | Run button drop down menu | Generate New Script (Scripting Window)) except that the simulation-related codes are omitted
- For Java, the 'Module Path' value must match the Package structure; in this example, the DMNModule.java must be generated to a directory to form a file path that ends with '\com\sparxsystems\dmn\DMNModule.java' - you have to manually create the directory structures for now

Click on the Generate Code button ( ) on the toolbar. This example will use Java; however, C++ and C# are the same. These actions are performed:

- The .java file is generated to the path specified
- An Analyzer Script (Build script) for this Artifact is created
- The Build Script for this Analyzer Script is executed
- Progress messages are reported in the System Output window

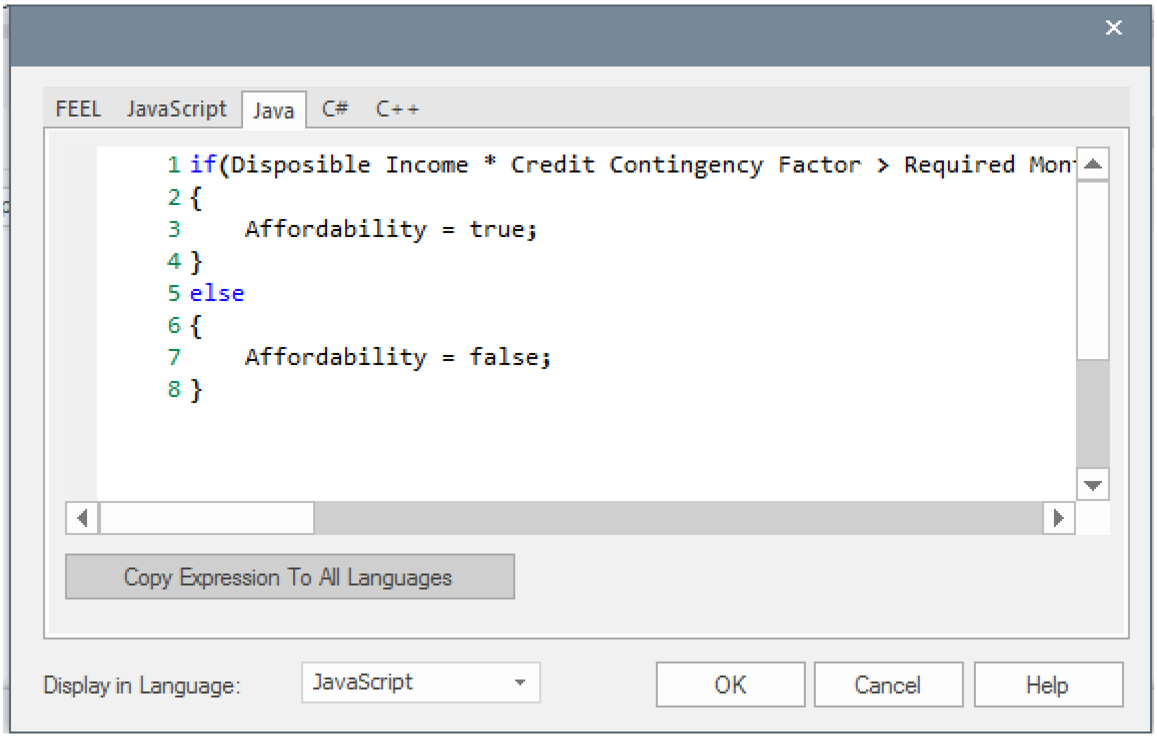
If the model is valid, this process will return the message:



```
Generating DMN Module to file: 'C:\EA\DMNServer\com\sparxsystems\dmn\DMNModule.java' ...
The DMN module is successfully generated.
Generating Analyzer Script for DMN Module...
Analyzer Script for DMN Module is successfully created.
Compiling DMN Module...
DMN Module is successfully compiled.
```

If there are compiling errors, you can open the generated .java file by clicking the 🗎 button next to the </> button on the toolbar, manually fix the issue, and compile with the generated script until you are successful.

One common reason for a compile failure is that languages can have different grammars for an expression. You might need to provide a value for a language to overwrite the default (right-click on a DMN Literal Expression | Edit Expression).



```
if(Disposible Income * Credit Contingency Factor > Required Mon
{
    Affordability = true;
}
else
{
    Affordability = false;
}
```

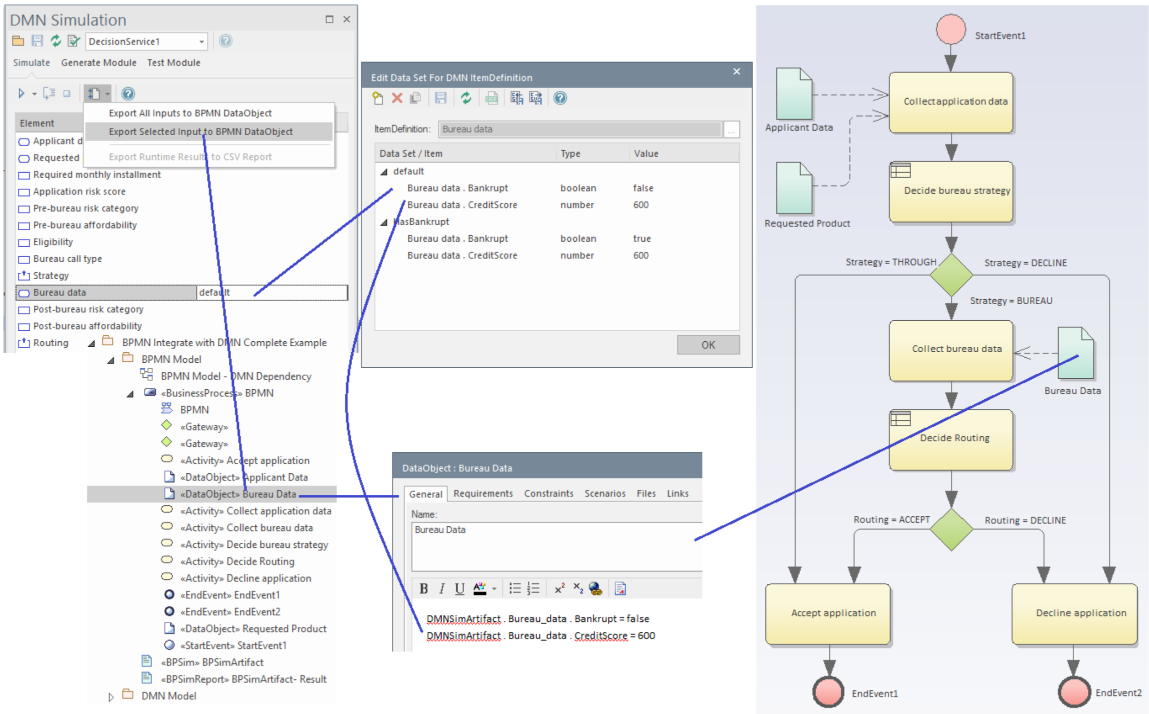
Display in Language: JavaScript

Testing DMN Modules before external Use

Having generated the model to java code and successfully compiled it we now want to:

- Test this module's correctness
- Provide it with inputs
- Get the output Decision values

Generate BPMN DataObject



ItemDefinition: Bureau data

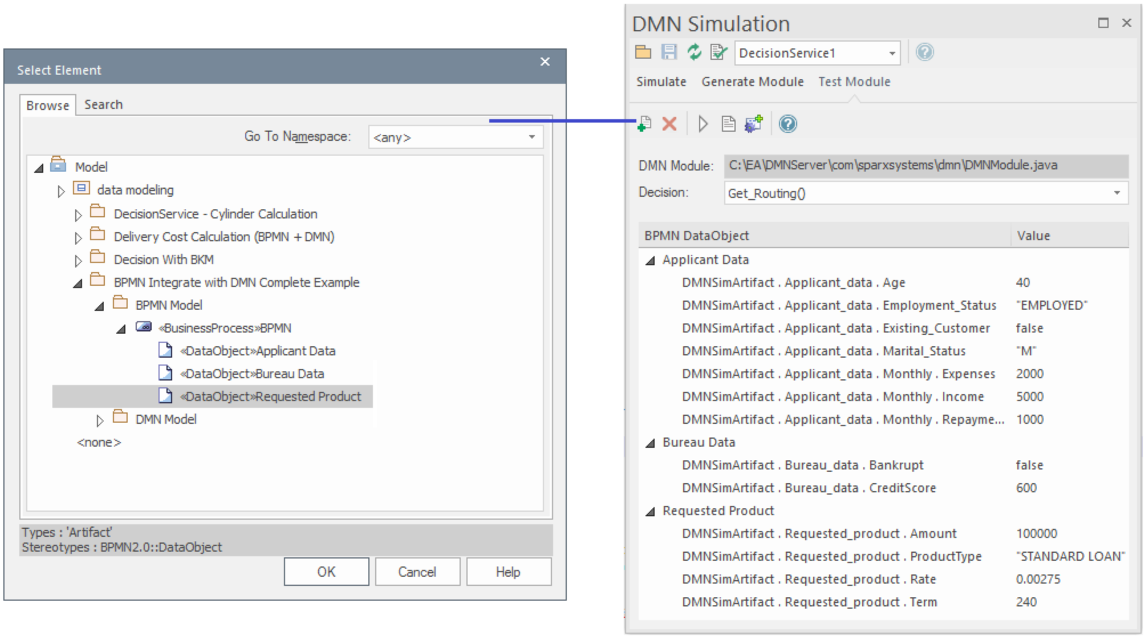
Data Set / Item	Type	Value
default		
Bureau data . Bankrupt	boolean	false
Bureau data . CreditScore	number	600
hasBankrupt		
Bureau data . Bankrupt	boolean	true
Bureau data . CreditScore	number	600

Name: Bureau Data

DMNSimArtifact . Bureau_data . Bankrupt = false
DMNSimArtifact . Bureau_data . CreditScore = 600

The data carried by the selected data set will be generated to the BPMN DataObject's 'Notes' field.

- Click the button (2nd to the right on the toolbar of the 'Generate Module' tab) to open the 'Test Module' tab



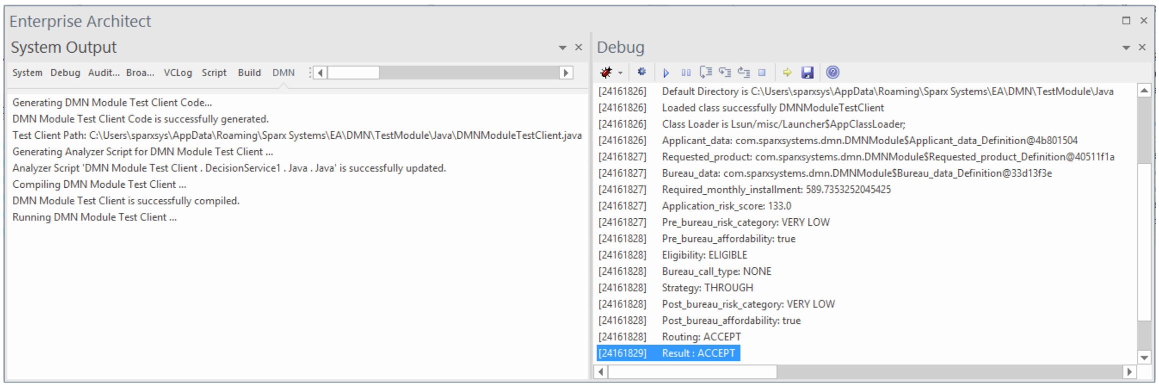
Go To Namespace: <any>

Types : 'Artifact'
Stereotypes : BPMN2.0::DataObject

DMN Module: C:\EA\DMNServer\com\sparxsystems\dmn\DMNModule.java
Decision: Get_Routing()

BPMN DataObject	Value
Applicant Data	
DMNSimArtifact . Applicant_data . Age	40
DMNSimArtifact . Applicant_data . Employment_Status	"EMPLOYED"
DMNSimArtifact . Applicant_data . Existing_Customer	false
DMNSimArtifact . Applicant_data . Marital_Status	"M"
DMNSimArtifact . Applicant_data . Monthly . Expenses	2000
DMNSimArtifact . Applicant_data . Monthly . Income	5000
DMNSimArtifact . Applicant_data . Monthly . Repayme...	1000
Bureau Data	
DMNSimArtifact . Bureau_data . Bankrupt	false
DMNSimArtifact . Bureau_data . CreditScore	600
Requested Product	
DMNSimArtifact . Requested_product . Amount	100000
DMNSimArtifact . Requested_product . ProductType	"STANDARD LOAN"
DMNSimArtifact . Requested_product . Rate	0.00275
DMNSimArtifact . Requested_product . Term	240

- Click the on the toolbar to select the input BPMN DataObject elements
- Select the available outputs from the 'Decision' combo box, such as Get_Routing(), and click on the Run button on the toolbar



```
Generating DMN Module Test Client Code...
DMN Module Test Client Code is successfully generated.
Test Client Path: C:\Users\sparxsys\AppData\Roaming\Sparx Systems\EA\DMN\TestModule\Java\DMNModuleTestClient.java
Generating Analyzer Script for DMN Module Test Client ...
Analyzer Script 'DMN Module Test Client . DecisionService1 . Java . Java' is successfully updated.
Compiling DMN Module Test Client ...
DMN Module Test Client is successfully compiled.
Running DMN Module Test Client ...
```

```
[24161826]    Default Directory is C:\Users\sparxsys\AppData\Roaming\Sparx Systems\EA\DMN\TestModule\Java
[24161826]    Loaded class successfully DMNModuleTestClient
[24161826]    Class Loader is Lsun/misc/Launcher$AppClassLoader;
[24161826]    Applicant_data: com.sparxsystems.dmn.DMNModule$Applicant_data_Definition@4b801504
[24161827]    Requested_product: com.sparxsystems.dmn.DMNModule$Requested_product_Definition@40511f1a
[24161827]    Bureau_data: com.sparxsystems.dmn.DMNModule$Bureau_data_Definition@33d13f3e
[24161827]    Required_monthly_installment: 589.7353252045425
[24161827]    Application_risk_score: 133.0
[24161827]    Pre_bureau_risk_category: VERY LOW
[24161828]    Pre_bureau_affordability: true
[24161828]    Eligibility: ELIGIBLE
[24161828]    Bureau_call_type: NONE
[24161828]    Strategy: THROUGH
[24161828]    Post_bureau_risk_category: VERY LOW
[24161828]    Post_bureau_affordability: true
[24161828]    Routing: ACCEPT
[24161829]    Result : ACCEPT
```

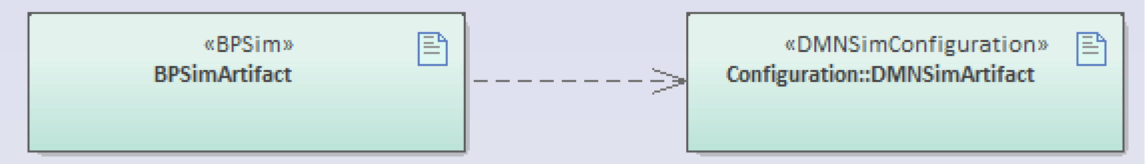
The execution result will be displayed in the Debug window. You can also open the test module file, set a breakpoint on the line and debug in the DMN Module to do line-level-debugging.

We highly recommend you test your DMN Module with this window to guarantee that the DMN Module is functional with the given inputs (from the BPMN DataObjects) and that it will successfully compute the result of the output.

Note: The DMN Module path is saved in the DMNSimConfiguration Artifact's 'Filepath' property.

Now, it is time to integrate the DMN module with the BPSim model.

The first step is to set up the usage dependency between the BPSim Artifact and the DMN Simulation Artifact.



Note: A BPSim Artifact can use multiple DMN modules if necessary. This is supported by simply putting all DMN Artifacts on this diagram and drawing a Dependency connector from the BPSim Artifact to each DMN Simulation Artifact.

These Help topics provide two examples of using these methods. See:

- *Example: Integrate DMN Decision Service into BPSim Data Object and Property Parameter*
- *Example: Integrate DMN Business Knowledge Model into BPSim Property Parameter*

Example: Integrate DMN Decision Service into BPSim Data Object and Property Parameter

An example of integrating a DMN Decision service into the BPSim model is provided in the Model Wizard for BPSim.

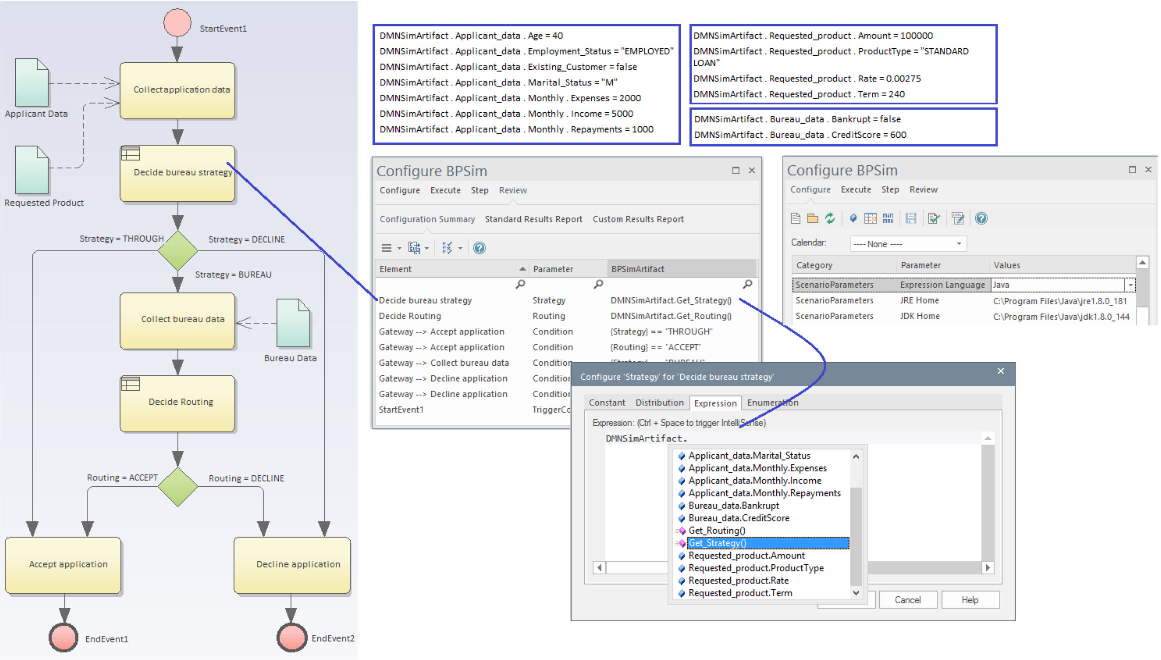
To access this:

- Set the *Perspective* to *Business Modeling > BPSim*
- Open the Model Wizard (Start Page 'Create from Pattern' tab)
- From the BPSim Case Studies group select *BPMN Integrate with DMN Complete Example*
- Click on the Create Model(s) button.

This will create BPMN and DMN models configured to simulate a call to a DMN model from the BPMN model.

Note: In order to integrate the DMN Module, the Expression Language must use Java and the JRE and JDK must be configured correctly (the minimum version of Java is 1.7). See *Install the BPSim Execution Engine* in the Help topic BPSim Business Simulations.

In this BPMN diagram there are three DataObjects (aqua) connected to BPMN Activities. These DataObject elements carry input data, generated from the DMN Simulation window.



DMNSimArtifact . Applicant_data . Age = 40
DMNSimArtifact . Applicant_data . Employment_Status = "EMPLOYED"
DMNSimArtifact . Applicant_data . Existing_Customer = false
DMNSimArtifact . Applicant_data . Marital_Status = "M"
DMNSimArtifact . Applicant_data . Monthly . Expenses = 2000
DMNSimArtifact . Applicant_data . Monthly . Income = 5000
DMNSimArtifact . Applicant_data . Monthly . Repayments = 1000

DMNSimArtifact . Requested_product . Amount = 100000
DMNSimArtifact . Requested_product . ProductType = "STANDARD LOAN"
DMNSimArtifact . Requested_product . Rate = 0.00275
DMNSimArtifact . Requested_product . Term = 240

DMNSimArtifact . Bureau_data . Bankrupt = false
DMNSimArtifact . Bureau_data . CreditScore = 600

Configure BPSim

Element	Parameter	BPSimArtifact
Decide bureau strategy	Strategy	DMNSimArtifact.Get_Strategy()
Decide Routing	Routing	DMNSimArtifact.Get_Routing()
Gateway --> Accept application	Condition	{Strategy} == "THROUGH"
Gateway --> Accept application	Condition	{Routing} == "ACCEPT"
Gateway --> Collect bureau data	Condition	
Gateway --> Decline application	Conditior	
Gateway --> Decline application	Conditior	
StartEvent1	TriggerCc	

Configure BPSim

Calendar: ---- None ----

Category	Parameter	Values
ScenarioParameters	Expression Language	Java
ScenarioParameters	JRE Home	C:\Program Files\Java\jre1.8.0_181
ScenarioParameters	JDK Home	C:\Program Files\Java\jdk1.8.0_144

Configure 'Strategy' for 'Decide bureau strategy'

Expression: (Ctrl + Space to trigger IntelliSense)

```
DMNSimArtifact.
```

Applicant_data.Marital_Status
Applicant_data.Monthly.Expenses
Applicant_data.Monthly.Income
Applicant_data.Monthly.Repayments
Bureau_data.Bankrupt
Bureau_data.CreditScore
Get_Routing()
Get_Strategy()
Requested_product.Amount
Requested_product.ProductType
Requested_product.Rate
Requested_product.Term
Cancel
Help

- When the simulation is running it will automatically load all DataObjects connecting to the task when the simulation token passes through
- The second Business Rules task 'Decide bureau strategy' is configured to set the property 'Strategy' to the value 'DMNSimArtifact.Get_Strategy()'; you don't need to type this in - press Ctrl+Space to help you edit the expression

When these are set, click on the 'Execute' tab and simulate the model. You can then view the report or go to the 'Step' page to do step debugging of the BPSim model.

Example: Integrate DMN Business Knowledge Model into BPSim Property Parameter

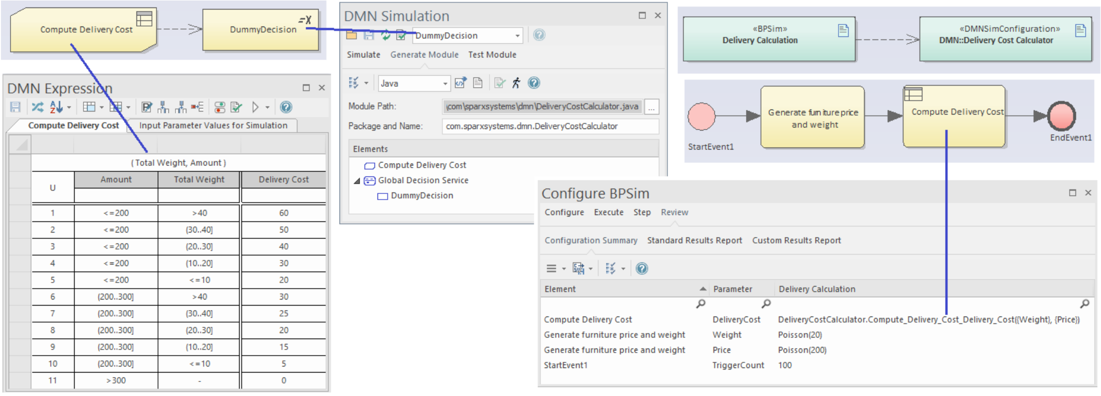
In some cases, you might want just to design a Decision Table to use in a BPMN model. If so, there is no need to go through the processes of creating a Decision Service, Decision, Input Data or even Item Definition, as a Business Knowledge Model (BKM) can be directly interfaced.

An example of integrating a DMN BKM into the BPSim model is provided in the Model Wizard for BPSim.

To access this:

- Set the *Perspective* to *Business Modeling > BPSim*
- Open the Model Wizard (Start Page 'Create from Pattern' tab)
- From the BPSim Case Studies group select *BPMN Integrate with DMN - Delivery Cost Calculation*
- Click on the Create Model(s) button

1. Create a simple Business Knowledge Model as a Decision Table (you can also create other expressions such as Boxed Context or Literal Expressions) with parameters, then model the logic (Input Clause, Output Clause, rules) and test it (the 'Input Parameter Values for Simulation' tab on the DMN Expression window).



(Total Weight, Amount)

U	Amount	Total Weight	Delivery Cost
1	<=200	>40	60
2	<=200	(30..40]	50
3	<=200	(20..30]	40
4	<=200	(10..20]	30
5	<=200	<=10	20
6	(200..300]	>40	30
7	(200..300]	(30..40]	25
8	(200..300]	(20..30]	20
9	(200..300]	(10..20]	15
10	(200..300]	<=10	5
11	>300	-	0

DummyDecision

Java

Module Path: \com\sparxsystems\dmn\DeliveryCostCalculator.java

Package and Name: com.sparxsystems.dmn.DeliveryCostCalculator

Element	Parameter	Delivery Calculation
Compute Delivery Cost	DeliveryCost	DeliveryCostCalculator.Compute_Delivery_Cost_Delivery_Cost({Weight}, {Price})
Generate furniture price and weight	Weight	Poisson(20)
Generate furniture price and weight	Price	Poisson(200)
StartEvent1	TriggerCount	100

2. Connect the BKM to a Decision with a Knowledge Requirement connector. This Decision serves as a group name for a number of BKM functions; you can simply input a number such as '10' to the expression. For example, if you want to generate Java code with only five BKMs (considering your model might have over one hundred BKMs), you can connect these five BKMs to a Decision and select this Decision in the DMN Simulation window, then all five BKMs will be included automatically.
3. Generate Java code and (assuming everything is correct) the compile will be successful.
4. In the BPSim configuration, we simply use Intelli-sense to construct the expression for task 'Compute Delivery cost'.

In this example, the 'Generate furniture price and weight' task will generate random values to the properties 'Weight' and 'Price', then the 'Compute Delivery cost' task will pass the value to the Business Knowledge Model and the result will be carried back to the property 'DeliveryCost'.

You can now execute the simulation, and step through the debug process to observe, for example, the attribute value changes.

Integrate Into UML Class Element

After a Decision Model is created and simulated, you can generate a DMN Module in Java, JavaScript, C++ or C# and test it.

The DMN Module can be integrated with a UML Class element, so the code generated from that Class element can reuse the DMN Module and be well-structured. Since a Class element can define a StateMachine, after integration with the DMN Module the Executable StateMachine simulation will generically be able to use the power of the DMN Module.

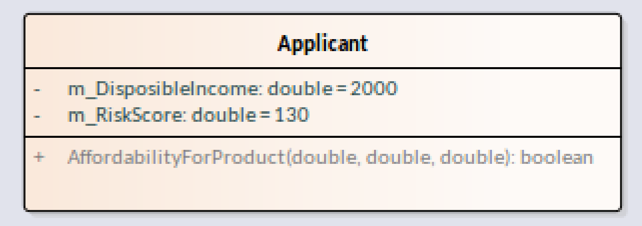
In this topic, we will explain the process of integrating a DMN Model with a UML Class element, considering the:

- Class element's requirement
- DMN Models
- DMN Binding to Class & Intelli-sense
- Code Generation on the Class element

Class Element's Requirement

Suppose we have a Class *Applicant* with an operation *AffordabilityForProduct* that evaluates whether the applicant can afford a loan product.

A simplified model resembles this:


Applicant
- m_DisposibleIncome: double = 2000
- m_RiskScore: double = 130
+ AffordabilityForProduct(double, double, double): boolean

The Class *Applicant* contains two attributes, which are actually calculated from more basic data such as the applicant's monthly income, expenses, existing repayments, age and employment status.

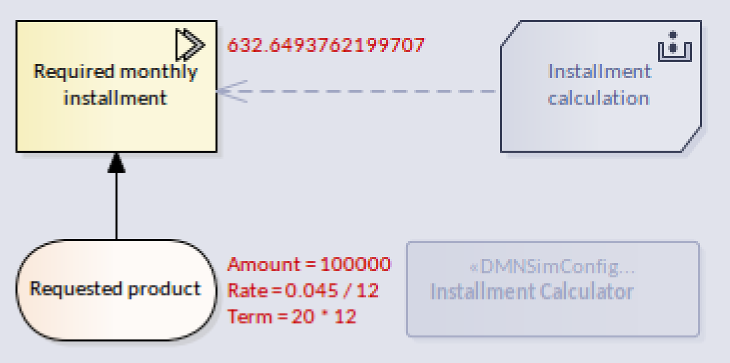
In this example, however, we simplify the model by skipping these steps and providing disposable income and risk score directly.

DMN Models

In this example, we have two disjoint DMN Models to show that a UML Class can integrate multiple DMN Models.

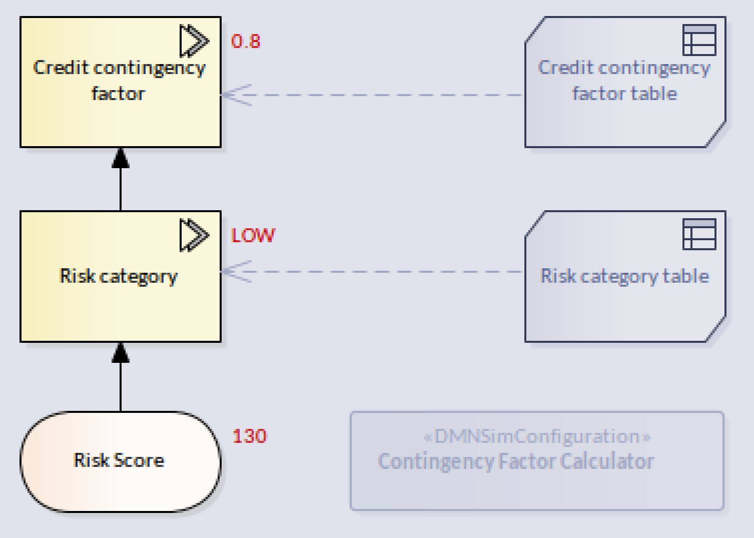
Installment Calculator

This DMN model computes the monthly repayment based on amount, rate and terms. It is composed of an InputData, a Decision and a Business Knowledge Model.



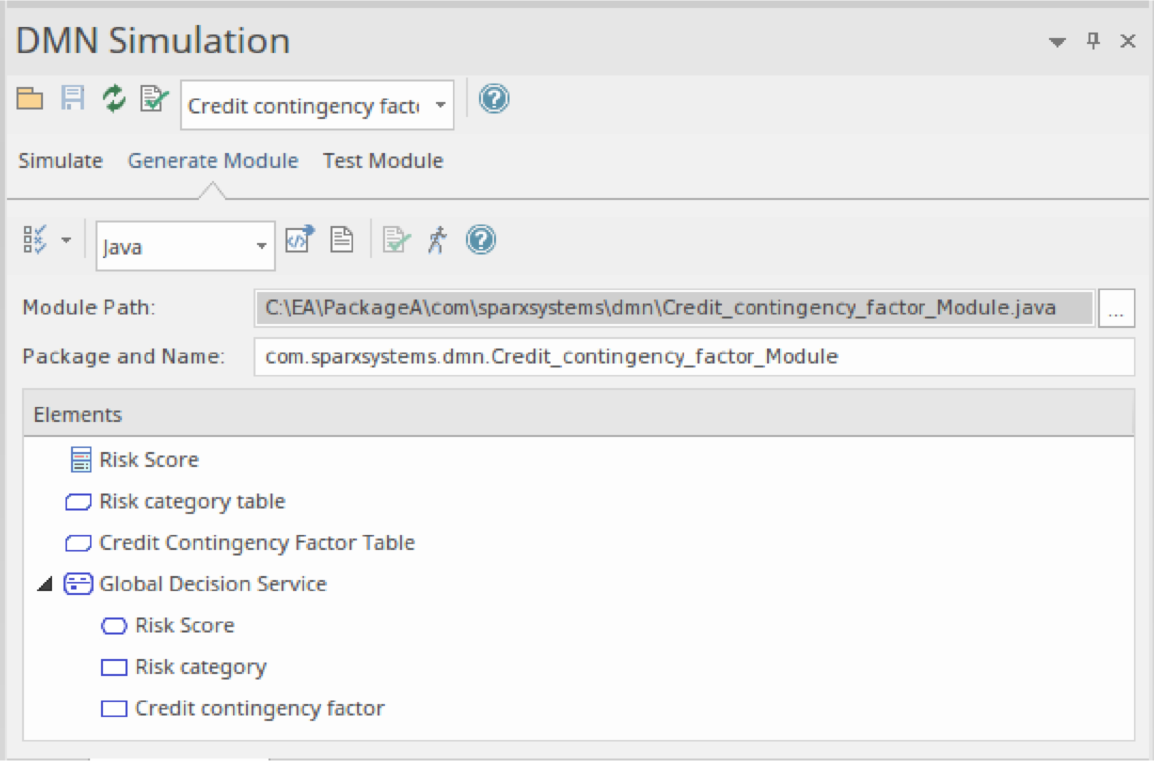
Credit Contingency Factor Calculator

This DMN model computes the credit contingency factor based on the applicant's risk score. It is composed of an InputData, two Decisions and two Business Knowledge Models.


Credit contingency factor
0.8
Credit contingency factor table
Risk category
LOW
Risk category table
Risk Score
130
«DMNSimConfiguration»
Contingency Factor Calculator

Note: In this example, we focus on how to integrate DMN modules into a Class element; the DMN elements' detail is not described here.

Generate code for both DMN Models

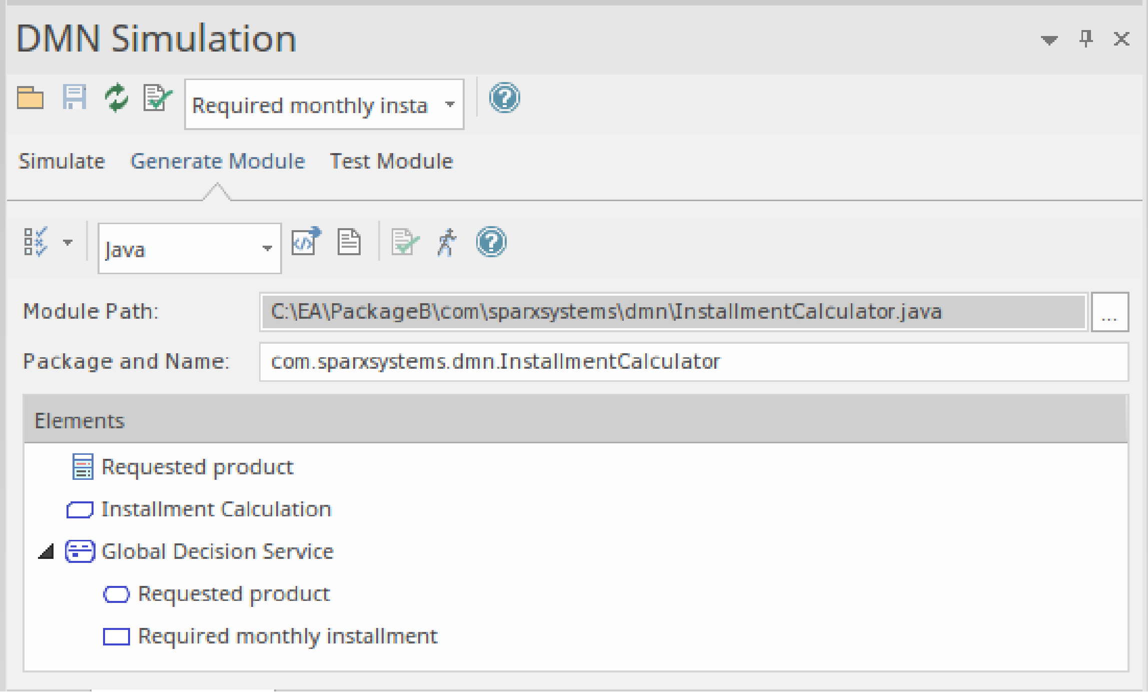


Module Path: C:\EA\PackageA\com\sparxsystems\dmn\Credit_contingency_factor_Module.java

Package and Name: com.sparxsystems.dmn.Credit_contingency_factor_Module

Click on the Generate Code icon, and check that you can see this string in the System Output window, 'DMN' tab:

DMN Module is successfully compiled.

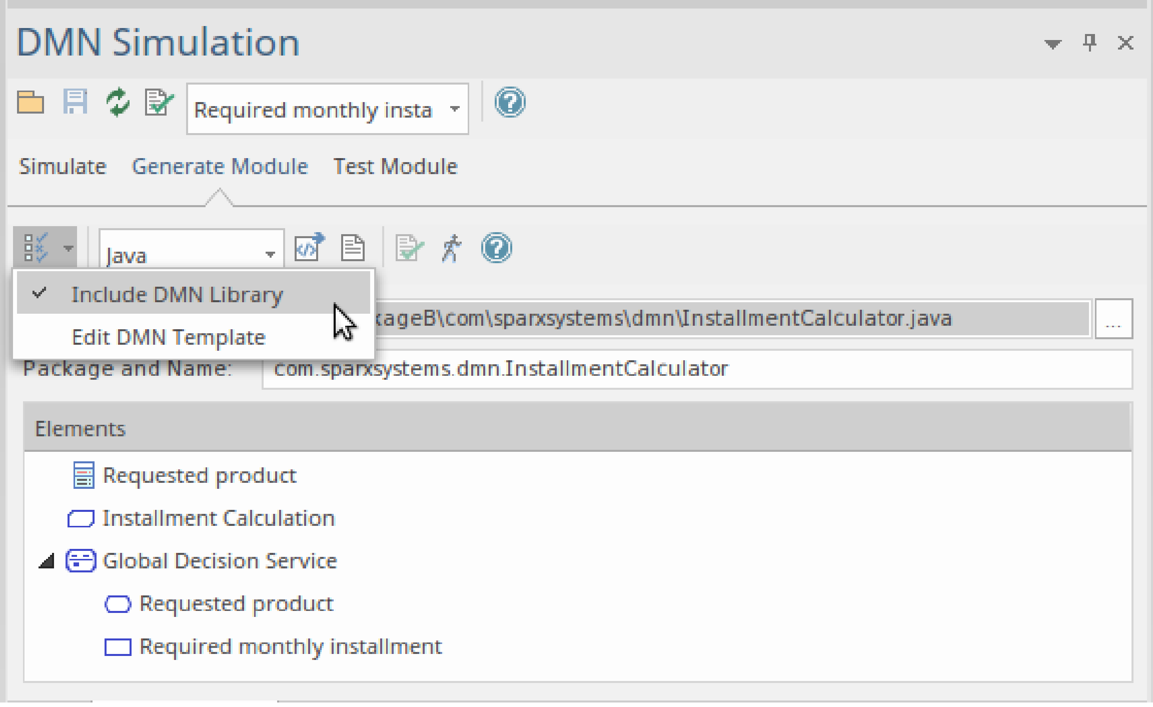


Module Path: C:\EA\PackageB\com\sparxsystems\dmn\InstallmentCalculator.java

Package and Name: com.sparxsystems.dmn.InstallmentCalculator

Elements
- Requested product
- Installment Calculation
- Global Decision Service
  - Requested product
  - Required monthly installment

Note: Since this model uses a built-in function PMT, the DMN Library has to be included:



Module Path: ...kageB\com\sparxsystems\dmn\InstallmentCalculator.java

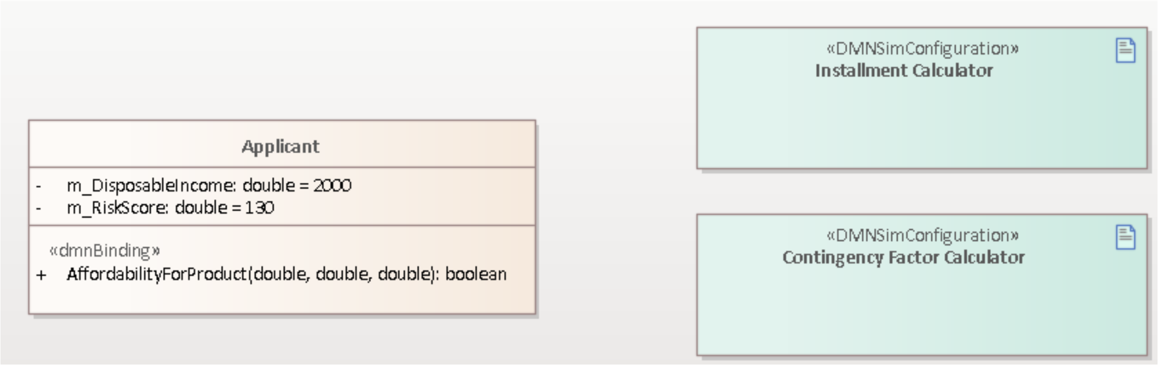
Package and Name: com.sparxsystems.dmn.InstallmentCalculator

Elements
- Requested product
- Installment Calculation
- Global Decision Service
  - Requested product
  - Required monthly installment

Click on the Generate Code icon, and check that you can see this string in the System Output window, 'DMN' page: *DMN Module is successfully compiled.*

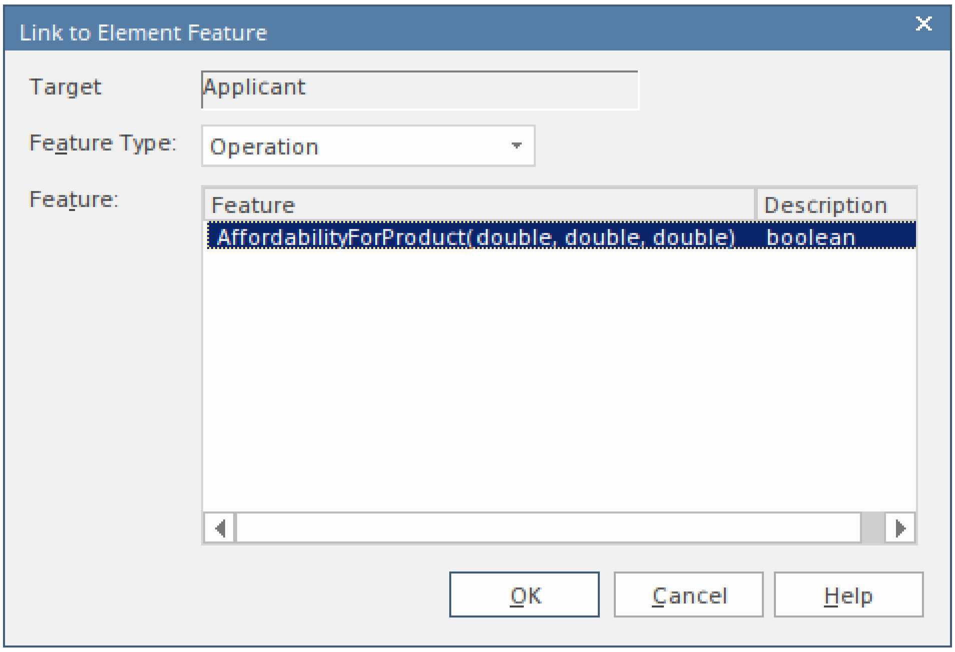
DMN Binding to Class & Intelli-sense

Put the two DMNSimConfiguration Artifacts on the Class diagram.


Applicant
- m_DisposableIncome: double = 2000
- m_RiskScore: double = 130
«dmnBinding»
+ AffordabilityForProduct(double, double, double): boolean
«DMNSimConfiguration»
Installment Calculator
«DMNSimConfiguration»
Contingency Factor Calculator

Use the Quick Linker to create a Dependency connector from the Class *Applicant* to each of the DMN Artifacts.

On creation of the connector, a dialog will prompt you to choose the operation to be bound to the DMN module.


Link to Element Feature

Target: Applicant

Feature Type: Operation

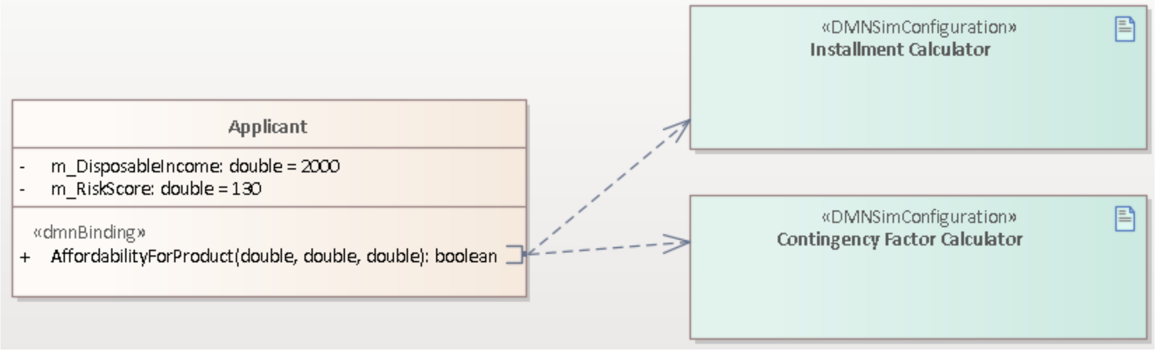
Feature:

Feature	Description
AffordabilityForProduct(double, double, double)	boolean

When the DMN module is bound to the operation:

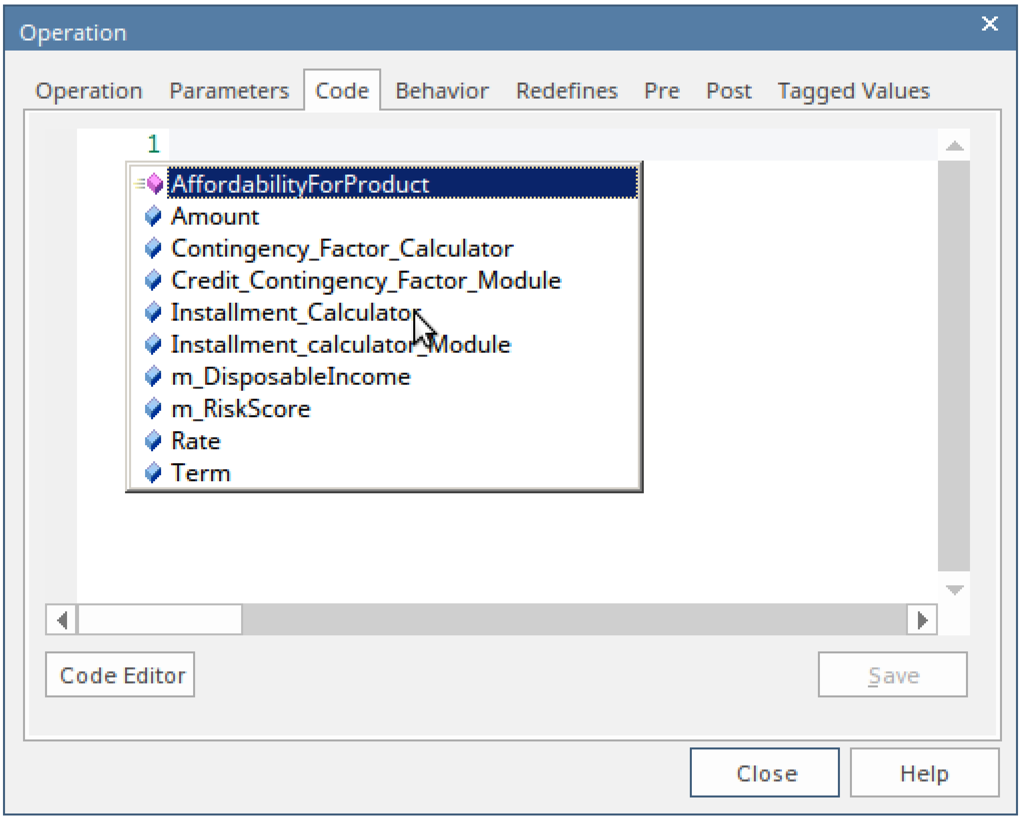
- The operation takes a stereotype <<dmnBinding>>
- The Dependency connector is linked to the operation

Multiple DMN Artifacts can be bound to the same operation.


Applicant
- m_DisposableIncome: double = 2000
- m_RiskScore: double = 130
«dmnBinding»
+ AffordabilityForProduct(double, double, double): boolean
«DMNSimConfiguration»
Installment Calculator
«DMNSimConfiguration»
Contingency Factor Calculator

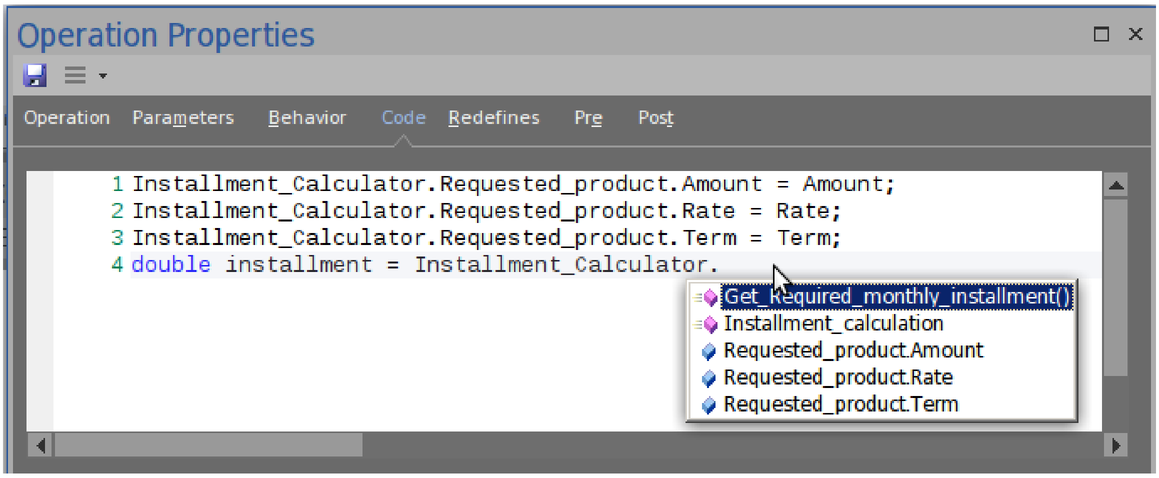
After DMN Bindings, Intelli-sense for the operation's code editor will support DMN Modules. To trigger the Intelli-sense, use these key combinations:

- Ctrl+Space - in most of the cases
- Ctrl+Shift+Space - when Ctrl+Space does not work after a parenthesis '('; for example, a function's arguments, or inside an 'If' condition's parentheses



- Class attributes will be listed - m_RiskScore, m_DisposableIncome
- Operation parameters will be listed - Amount, Rate, Term
- Operations will be listed - AffordabilityForProduct
- All bound DMN Modules will be listed - Contingency_Factor_Calculator, Installment_Calculator

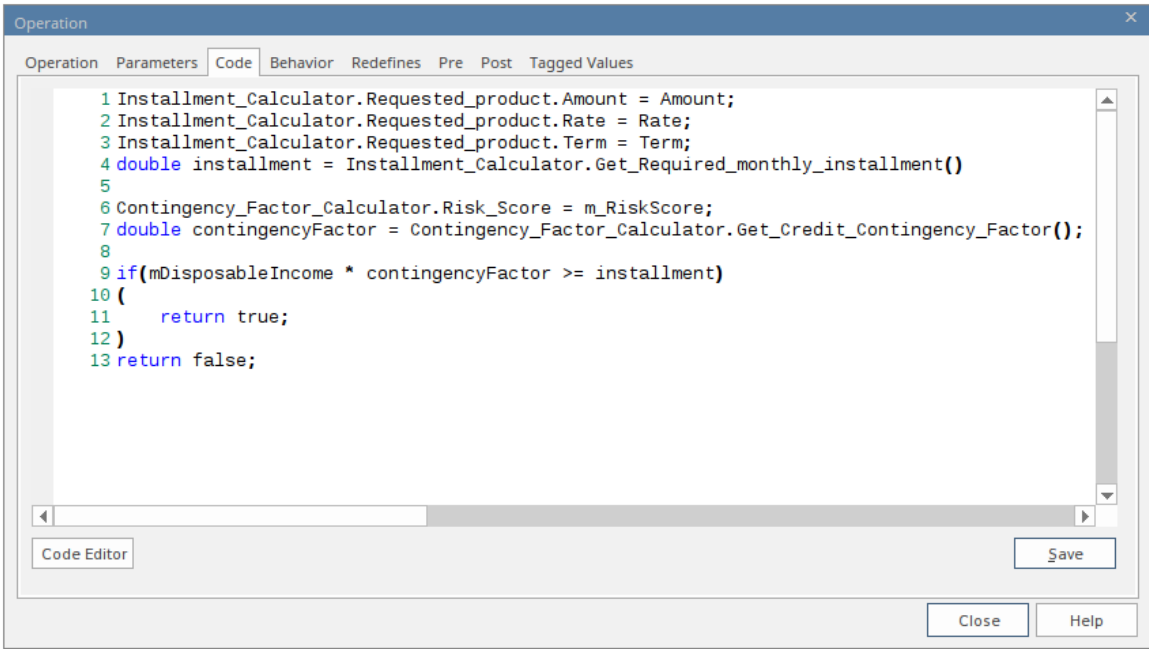
It is quite easy to compose the code with Intelli-sense support. On accessing the DMN Module, all the Input Datas, Decisions and Business Knowledge Models will be listed for selection.



```
Installment_Calculator.Requested_product.Amount = Amount;
Installment_Calculator.Requested_product.Rate = Rate;
Installment_Calculator.Requested_product.Term = Term;
double installment = Installment_Calculator.
```

This illustration shows that we are selecting 'Get_Required_monthly_installment()' from the Installment_Calculator.

This is the final implementation for the operation.



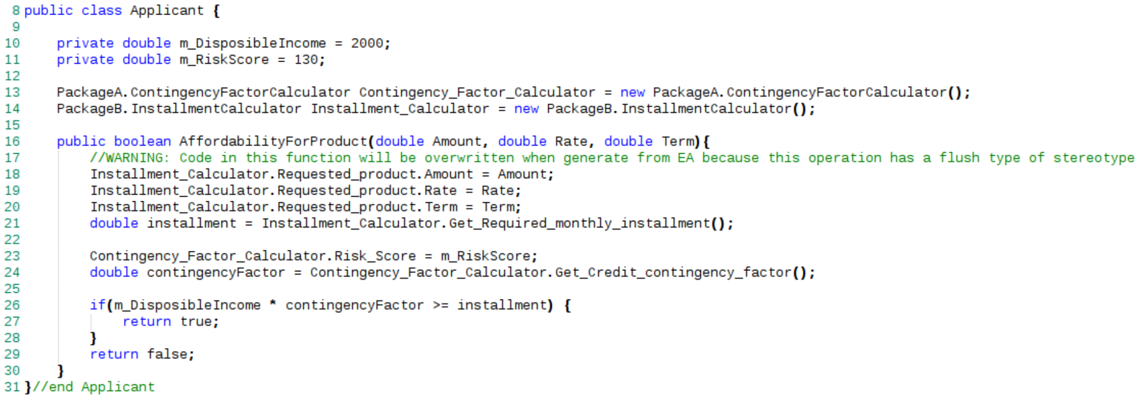
```
Installment_Calculator.Requested_product.Amount = Amount;
Installment_Calculator.Requested_product.Rate = Rate;
Installment_Calculator.Requested_product.Term = Term;
double installment = Installment_Calculator.Get_Required_monthly_installment()

Contingency_Factor_Calculator.Risk_Score = m_RiskScore;
double contingencyFactor = Contingency_Factor_Calculator.Get_Credit_Contingency_Factor();

if(mDisposableIncome * contingencyFactor >= installment)
{
    return true;
}
return false;
```

Code Generation for Class (With DMN Integration)

'Generate Code on Class Applicant' produces this code:



```
public class Applicant {

    private double m_DisposibleIncome = 2000;
    private double m_RiskScore = 130;

    PackageA.ContingencyFactorCalculator Contingency_Factor_Calculator = new PackageA.ContingencyFactorCalculator();
    PackageB.InstallmentCalculator Installment_Calculator = new PackageB.InstallmentCalculator();

    public boolean AffordabilityForProduct(double Amount, double Rate, double Term){
        //WARNING: Code in this function will be overwritten when generate from EA because this operation has a flush type of stereotype
        Installment_Calculator.Requested_product.Amount = Amount;
        Installment_Calculator.Requested_product.Rate = Rate;
        Installment_Calculator.Requested_product.Term = Term;
        double installment = Installment_Calculator.Get_Required_monthly_installment();

        Contingency_Factor_Calculator.Risk_Score = m_RiskScore;
        double contingencyFactor = Contingency_Factor_Calculator.Get_Credit_contingency_factor();

        if(m_DisposibleIncome * contingencyFactor >= installment) {
            return true;
        }
        return false;
    }
}//end Applicant
```

- The DMN Module(s) are generated as attributes of the Class
- The dmnBinding operation's code is updated

Note: Regardless of whether the generation option is 'Overwrite' or' Synchronize', the operation's code will be updated if it has the stereotype 'dmnBinding'.

Importing DMN XML

Enterprise Architect supports the import of a DMN 1.1 or 1.2 XML file into a project, with both model semantics and diagram-interchange information.

Access

In the Browser window, select the Package into which to import the XML file. Then use one of the methods outlined here to open the 'Import Package from DMN 1.1 XML' dialog.

Ribbon	Publish > Model Exchange > Import > DMN 1.1
Keyboard Shortcuts	Ctrl+Alt+I : Other XML Formats > DMN 1.1

Import DMN 1.1 XML

Step, Step	Action, Action
1	In the 'Filename' field, type in the source file path and name, or click on the ... icon to locate and select the file.
2	Click on the Import button to import the file into the Package.

Import the example from OMG

1. Download the zip file at this link and extract it to your file manager.
2. Browse for the folder *examples/Chapter 11/*.
3. Click on the file *Chapter 11 Example.dmn* and import it as a DMN 1.1 format file.

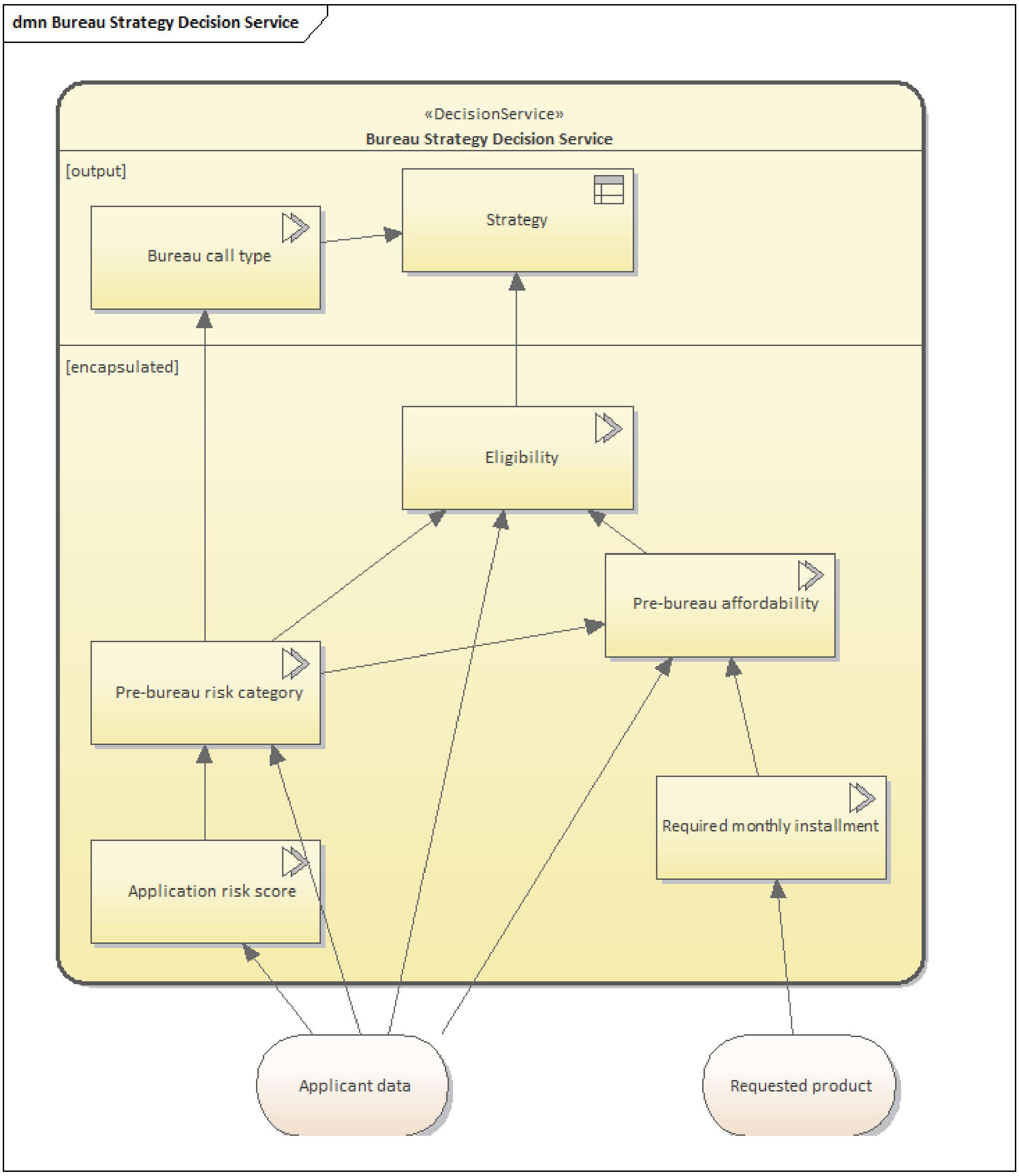
These diagrams are imported to show different perspectives of the model:

- DRD of all automated decision-making
- DRD for the Review Application decision point
- DRD for the Decide Routing decision point
- DRD for the Decide Bureau Strategy decision point

These diagrams are imported to define the Decision Services:

- Bureau Strategy Decision Service
- Routing Decision Service

The 'Bureau Strategy Decision Service' diagram is shown here. It has two Input Data elements (Applicant data, Requested product), two Output Decisions (Bureau call type, Strategy) and five Encapsulated Decisions. Note that the invoked Business Knowledge Models are not shown on the diagram.



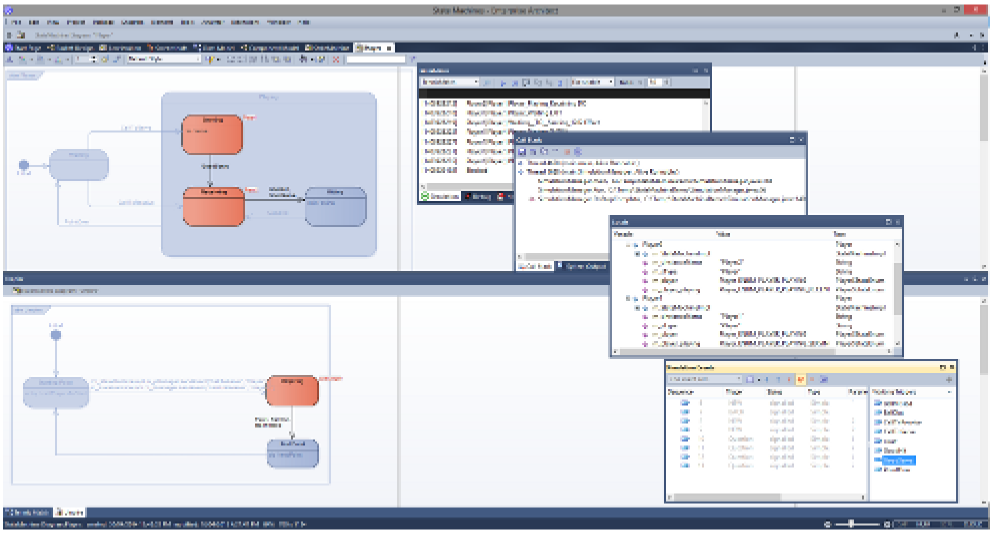
In order to generate production code from the model, you might have to run a validation and simulation to ensure that the imported model has the correct expressions.

1. Create a DMN Sim Configuration Artifact on any of the listed diagrams, and double-click on it to open it in the DMN Simulation window.
2. The Decision Services and Decisions are listed in the target drop-down field. Once you specify a target, all the required elements are listed in the window.
3. Click on the Validate button (4th on the toolbar). If any error or warning messages display, we suggest that you to fix the problems as directed by the error or warning descriptions, before performing the simulation.
4. Provide appropriate values for the inputs, and either run the simulation or step-debug the model.

Note: The 'Bureau Strategy Decision Service' example is also available in the EAExample Model. From the 'Getting Started' diagram, select 'Business Modelling > DMN Examples > Bureau Strategy Decision Service'.

More Information

Executable StateMachines



Executable StateMachines provide a means of rapidly generating, executing and simulating complex state models. In contrast to dynamic simulation of State Charts using Enterprise Architect's Simulation engine, Executable StateMachines provide a complete language-specific implementation that can form the behavioral 'engine' for multiple software products on multiple platforms. Visualization of the execution is based on a seamless integration with the Simulation capability. Evolution of the model now presents fewer coding challenges. The code generation, compilation and execution is taken care of by Enterprise Architect. For those having particular requirements, each language is provided with a set of code templates. Templates can be customized by you to tailor the generated code in any ways you see fit.

These topics introduce you to the basics of modeling Executable StateMachines and help you to understand how to generate and simulate them.

The creation and use of Executable StateMachines, and generating code from them, are supported by the Unified and Ultimate Editions of Enterprise Architect.

Overview of Building and Executing StateMachines

Building and using Executable StateMachines is quite straightforward, but does require a little planning and some knowledge of how to link the different components up to build an effective executing model. Luckily you do not have to spend hours getting the model right and fixing compilation errors before you can begin visualizing your design.

Having sketched out the broad mechanics of your model, you can generate the code to drive it, compile, execute and visualize it in a matter minutes. These points summarize what is required to start executing and simulating StateMachines.

Facility	Description
Build Class and State Models	The first task is to build the standard UML Class and State models that describe the entities and behavior to construct. Each Class of interest in your model should have its own StateMachine that describes the various states and transitions that govern its overall behavior.
Create an Executable StateMachine Artifact	Once you have modeled your Classes and State models, it is time to design the Executable StateMachine Artifact. This will describe the Classes and objects involved, and their initial properties and relationships. It is the binding script that links multiple objects together and it determines how these will communicate at runtime. Note that it is possible to have two or more objects in an Executable StateMachine Artifact as instances of a single Class. These will have their own state and behavior at run-time and can interact if necessary.

Generate Code and Compile	Whether you use JavaScript, C++, Java or C#, Enterprise Architect's engineering capabilities provide you with an effective tool, allowing you to regenerate the executable at any time, and without the loss of any customized code you might have made. This is a major advantage over a project's lifetime. It is probably also worth noting that the entire code base generated is independent and portable. In no way is the code coupled with any infrastructure used by the simulation engine.
Execute StateMachines	So how do we see how these StateMachines behave? One method is to build the code base for each platform, integrate it in one or more systems, examining the behaviors, 'in-situ', in perhaps several deployment scenarios. Or we can execute it with Enterprise Architect. Whether it is Java, JavaScript, C, C++ or C#, Enterprise Architect will take care of creating the runtime, the hosting of your model, the execution of its behaviors and the rendition of all StateMachines.
Visualize StateMachines	Executable StateMachine visualization integrates with Enterprise Architect's Simulation tools. Watch state transitions as they occur on your diagram and for which object(s). Easily identify objects sharing the same state. Importantly, these behaviors remain consistent across multiple platforms. You can also control the speed at which the machines operate to better understand the timeline of events.
Debug StateMachines	When states should change but do not, when a transition should not be enabled but is, when the behavior is - in short - undesirable and not immediately apparent from the model, we can turn to debugging. Enterprise Architect's Visual Execution Analyzer comes with debuggers for all the languages supported by ExecutableStateMachine code generation. Debugging provides many benefits, one of which might be to verify / corroborate the code attached to behaviors in a StateMachine to ensure it is actually reflected in the executing process.

Modeling Executable StateMachines

Most of the work required to model an Executable StateMachine is standard UML-based modeling of Classes and State models, although there are a couple of conventions that must be observed to ensure a well formed code base. The only novel construct is the use of a stereotyped Artifact element to form the configuration of an Executable StateMachine instance or scenario. The Artifact is used to specify details such as:

- The code language (JavaScript, C#, Java, C++ including C)
- The Classes and StateMachines involved in the scenario
- The instance specifications including run-state; note that this could include multiple instances of the same StateMachine, for example where a 'Player' Class is used twice in a Tennis Match simulation

Basic Modeling Tools and Objects for Executable StateMachines

These are the primary modeling elements used when building Executable StateMachines.

Element type	Description
Classes and Class Diagrams	Classes define the object types that are relevant to the StateMachine(s) being modeled. For example, in a simple Tennis Match scenario you might define a Class for each of a Player, a Match, a Hit and an Umpire. Each will have its own StateMachine(s) and at runtime will be represented by object instances for each involved entity. See the *UML Modeling Guide* for more information on Classes and Class diagrams.
StateMachines	For each Class you define that will have dynamic behavior within a scenario, you will typically define one or more UML StateMachines. Each StateMachine will determine the legal state-based behavior appropriate for one aspect of the owning Class. For example, it is possible to have a StateMachine that represents a Player's emotional state, one that tracks his current fitness and energy levels, and one that represents his winning or losing state. All these StateMachines will be initialized and started when the StateMachine scenario begins execution.
Executable StateMachine Artifact	This stereotyped Artifact is the core element used to specify the participants, configuration and starting conditions for an Executable StateMachine. From the scenario point of view it is used to determine which Instances (of Classes) are involved, what events they might Trigger and send to each other, and what starting conditions they operate under. From the configuration aspect, the Artifact is used to set up the link to an analyzer script that will determine output directory, code language, compilation script and similar. Right-clicking on the Artifact will allow you to generate, build, compile and visualize the real time execution of your StateMachines.

StateMachine Constructs Supported

This table details the StateMachine constructs supported and any limitations or general constraints relevant to each type.

Construct	Description
StateMachines	• Simple StateMachine: the StateMachine has one Region • Orthogonal StateMachine: the StateMachine contains multiple Regions

	Top level Region (owned by StateMachine) activation semantics: **Default Activation:** When the StateMachine starts executing. **Entry Point Entry:** Transitions from Entry Point to vertices in the contained Regions. • *Note 1: In each Region of the StateMachine owning the Entry Point, there is at most a single Transition from the entry point to a Vertex within that Region* • *Note 2: This StateMachine can be referenced by a Submachine State - the connection point references should be defined in the Submachine State as sources/targets of transitions; the connection point reference represents a usage of an Entry/Exit Point defined in the StateMachine and referenced by the Submachine State* **Multiple StateMachines:** The order of listing in the Browser window determines the execution order. • When a Submachine State is involved, there might be multiple StateMachines under the Class • Use the Move Up or Move Down arrows in the Browser window toolbar to adjust the order of StateMachines; the top one will be set as the main StateMachine **Not Supported** • Protocol StateMachine • StateMachine Redefinition
States	• Simple State: has no internal Vertices or Transitions • Composite State: contains exactly one Region • Orthogonal State: contains multiple Regions • Submachine State: refers to an entire StateMachine
Composite State Entry	• Default Entry • Explicit Entry • Shallow History Entry • Deep History Entry • Entry Point Entry
Substates	• Substates and Nested Substates Entry and Exit semantics, where the transition includes multiple nested levels of states, will obey the correct execution of nested behaviors (such as OnEntry and OnExit).
Transitions Support	• External Transition • Local Transition • Internal Transition (draw a self Transition and change Transition kind to Internal) • Completion Transition and Completion Events • Transition Guards • Compound Transitions • Firing priorities and selection algorithm For further details, refer to the *OMG UML Specification*.
Trigger and Events	An Executable StateMachine supports event handling for Signals only.

	To use Call, Timing or Change Event types you must define an outside mechanism to generate signals based on these events.

Signal	Attributes can be defined in Signals; the value of the attributes can be used as event arguments in Transition Guards and Effects.
	For example, this is the code set in the effect of a transition in C++:
	`if(signal->signalEnum == ENUM_SIGNAL2)`
	`{`
	`    int xVal = ((Signal2*)signal)->myVal;`
	`}`
	Signal2 is generated as this code:
	`class Signal2 : public Signal{`
	`public:`
	`    Signal2(){};`
	`    Signal2(std::vector<String>& lstArguments);`
	`    int myVal;`
	`};`
	Note: Further details can be found by generating an Executable StateMachine and referring to the generated 'EventProxy' file.

Initial	An Initial Pseudostate represents a starting point for a Region. It is the source for at most one Transition; there can be at most one Initial Vertex in a Region.
Regions	**Default Activation & Explicit Activation:** Transitions terminate on the containing State: • If an initial Pseudostate is defined in the Region: **Default activation** • If no initial Pseudostate is defined, the Region will remain inactive and the containing State is treated as a Simple State • If the transition terminates on one of the Region's contained vertices: **Explicit activation**, resulting in the default activation of all of its orthogonal Regions, unless those Regions are also entered explicitly (multiple orthogonal Regions can be entered explicitly in parallel through Transitions originating from the same Fork Pseudostate) For example, if there are three Regions defined for an Orthogonal State, and *RegionA* and *RegionB* have an Initial Pseudostate, then *RegionC* is explicitly activated. Default Activation applies to *RegionA* and *RegionB*; the containing State will have three active Regions.
Choice	Guard Constraints on all outgoing Transitions are evaluated dynamically, when the compound transition traversal reaches this Pseudostate.
Junction	Static conditional branch: guard constraints are evaluated before any compound transition is executed.
Fork / Join	Non-threaded, each active Region moves one step alternately, based on a completion event pool mechanism.
EntryPoint / ExitPoint Nodes	Non-threaded for orthogonal State or orthogonal StateMachine; each active Region moves one step alternately, based on a completion event pool mechanism.
History Nodes	• DeepHistory: represents the most recent active State configuration of its

	owning State - ShallowHistory: represents the most recent active Substate of its containing State, but not the Substates of that Substate
Deferred Events	Draw a self Transition, and change the Transition *kind* to Internal. Type 'defer();' in the 'Effect' field for the transition.
Connection Point References	A Connection Point Reference represents a usage (as part of a Submachine State) of an Entry/Exit Point defined in the StateMachine referenced by the Submachine State. Connection Point References of a Submachine State can be used as sources and targets of Transitions. They represent entries into or exits out of the StateMachine referenced by the Submachine State.
State behaviors	State 'entry', 'doActivity' and 'exit' behaviors are defined as operations on a State. By default, you type the code that will be used for each behavior into the 'Code' panel of the Properties window for the Behavior operation. Note that you can change this to type the code into the 'Behavior' panel, by customizing the generation template. The 'doActivity' behavior generated will be run to completion before proceeding. The code is not concurrent with other entry behavior; the 'doActivity' behavior is implemented as 'execute in sequence after entry' behavior.

References to Behaviors Within Other Contexts/Classes

If the Submachine State references behavior elements outside the current context or Class, you must add an <<import>> connector from the current context Class to the container context Class. For example:

Submachine State S1 in Class1 refers to StateMachine ST2 in Class2

Therefore, we add an <<import>> connector from Class1 to Class2 in order for Executable StateMachine code generation to generate code correctly for Submachine State S1. (On Class 1, click on the Quick Linker arrow and drag to Class 2, then select 'Import' from the menu of connector types.)

Reusing Executable StateMachine Artifacts

You can create multiple models or versions of a component using a single executable Artifact. An Artifact representing a resistor, for example, could be re-used to create both a foil resistor and a wire wound resistor. This is likely to be the case for similar objects that, although represented by the same classifier, typically exhibit different run states. A property named 'resistorType' taking the value 'wire' rather than 'foil' might be all that is required from a modeling point of view. The same StateMachines can then be re-used to test behavioral changes that might result due to variance in run-state. This is the procedure:

Step	Action
Create or open Component diagram	Open a Component diagram to work on. This might be the diagram that contains your original Artifact.
Select the Executable StateMachine to copy	Now find the original Executable StateMachine Artifact in the Browser window.
Create the New Component	Whilst holding the Ctrl key, drag the original Artifact on to your diagram. You will be prompted with two questions.

	The answer to the first is **Object** and to the second **All.** Rename the Artifact to differentiate it from the original and then proceed to alter its property values.

Executable StateMachine Artifact

An Executable StateMachine Artifact is key to generating StateMachines that can interact with each other. It specifies the objects that will be involved in a simulation, their state and how they connect. A big advantage in using Executable StateMachine Artifacts is that each of several parts in an Artifact can represent an instance of a StateMachine, so you can set up simulations using multiple instances of each StateMachine and observe how they interact. An example is provided in the *Example Executable StateMachine* Help topic.

Creating the Properties of an Executable StateMachine

Each Executable StateMachine scenario involves one or more StateMachines. The StateMachines included are specified by UML Property elements; each Property will have a UML Classifier (Class) that determines the StateMachine(s) included for that type. Multiple types included as multiple Properties can end up including many StateMachines, which are all created in code and initialized on execution.

Action	Description
Drop a Class from the Browser window on to the <<Executable StateMachine>> Artifact	The easiest way to define properties on an Executable StateMachine is to drop the Class onto the Executable StateMachine from the Browser window. On the dialog that is shown, select the option to create a Property. You can then specify a name describing how the Executable StateMachine will refer to this property. Note: Depending on your options, you might have to hold down the Ctrl key to choose to create a property. This behavior can be changed at any time using the 'Hold Ctrl to Show this dialog' checkbox.
Use and Connect Multiple UML Properties	An Executable StateMachine describes the interaction of multiple StateMachines. These can be different instances of the same StateMachine, different StateMachines for the same instance, or completely different StateMachines from different base types. To create multiple properties that will use the same StateMachine, drop the same Class onto the Artifact multiple times. To use different types, drop different Classes from the Browser window as required.

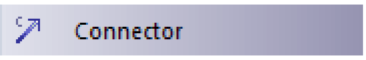
Defining the Initial State for Properties

The StateMachines run by an Executable StateMachine will all run in the context of their own Class instance. An Executable StateMachine allows you to define the initial state of each instance by assigning property values to various Class attributes. For example you might specify a Player's age, height, weight or similar if these properties have relevance to the scenario being run. By doing this it is possible to set up detailed initial conditions that will influence how the scenario plays out.

Action	Description
Set Property Values Dialog	The dialog for assigning property values can be opened by right-clicking on a Property and selecting 'Features | Set Property Values', or by using the keyboard shortcut Ctrl+Shift+R.
Assign a Value	The 'Set Property Values' dialog allows you to define values for any attribute defined in the original Class. To do this, select the variable, set the operator to '=' and enter the required value.

Defining Relationships Between Properties

In addition to describing the values to assign to variables owned by each property, an Executable StateMachine allows you to define how each property can reference others based on the Class model that they are instances of.

Action	Description
Create a Connector	Connect multiple properties using the Connector relationship from the Composite toolbox.  Alternatively, use the Quick Linker to create a relationship between two Properties and select 'Connector' as the relationship type.
Map to Class Model	Once a connector exists between two properties, you can map it back to the Association it represents in the Class model. To do this, select the connector and use the keyboard shortcut Ctrl+L. The 'Choose an Association' dialog displays, which allows the generated StateMachine to send signals to the instance filling the role specified in the relationship during execution.

Code Generation for Executable StateMachines

The code generated for an Executable StateMachine is based on its language property. This might be Java, C, C++, C# or JavaScript. Whichever language it is, Enterprise Architect generates the appropriate code, which is immediately ready to build and run. There are no manual interventions necessary before you run it. In fact after the initial generation, any Executable StateMachine can be generated, built and executed at the click of a button.

Language Supported

An Executable StateMachine supports code generation for these platform languages:

- Microsoft Native C/C++
- Microsoft .NET (C#)
- Scripting (JavaScript)
- Oracle Java (Java)

From Enterprise Architect Release 14.1, code generation is supported without dependency on the simulation environment (compilers). For example, if you don't have Visual Studio installed, you can still generate code from the model and use it in your own project. The compilers are still needed if you want to simulate models in Enterprise Architect.

Simulation Environment (Compiler Settings)

If you want to simulate the Executable StateMachine model in Enterprise Architect, these platforms or compilers are required for the languages:

Language Platform	Example of Framework Path
Microsoft Native (C/C++)	C:\Program Files (x86)\Microsoft Visual Studio 12.0 C:\Program Files (x86)\Microsoft Visual Studio\2017\Professional (or other editions)
Microsoft .NET (C#)	C:\Windows\Microsoft.NET\Framework\v3.5 (or higher)
Scripting (JavaScript)	N/A
Oracle Java (Java)	C:\Program Files (x86)\Java\jdk1.7.0_17 (or higher)

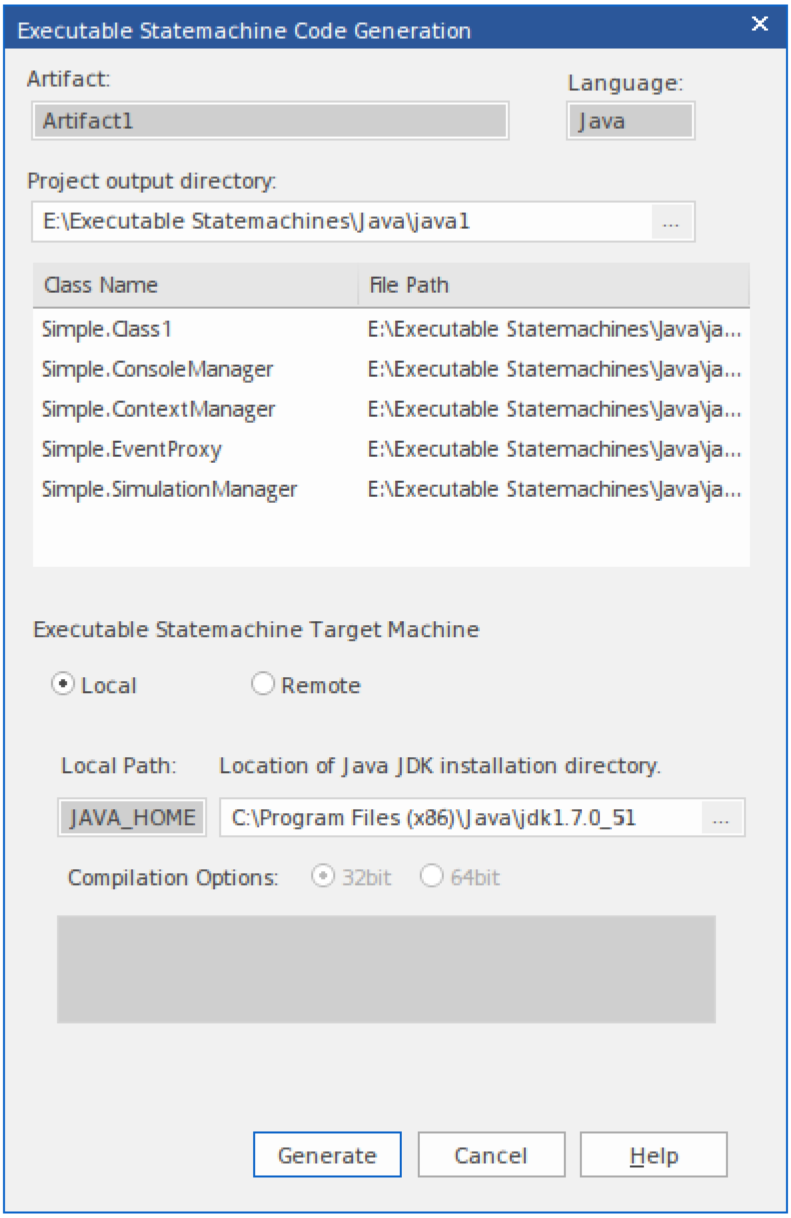
Access

Ribbon	Simulate > Executable States > Statemachine > Generate, Build and Run or Simulate > Executable States > Statemachine > Generate

Generating Code

The 'Simulate > Executable States > Statemachine' ribbon options provide commands for generating code for the StateMachine. Select the Executable StateMachine Artifact first, then use the ribbon option to generate the code. The 'Executable Statemachine Code Generation' dialog displayed depends on the code language.

Generating Code (Java on Windows)



Executable Statemachine Code Generation

Artifact: Artifact1

Language: Java

Project output directory: E:\Executable Statemachines\Java\java1

Class Name	File Path
Simple.Class1	E:\Executable Statemachines\Java\ja...
Simple.ConsoleManager	E:\Executable Statemachines\Java\ja...
Simple.ContextManager	E:\Executable Statemachines\Java\ja...
Simple.EventProxy	E:\Executable Statemachines\Java\ja...
Simple.SimulationManager	E:\Executable Statemachines\Java\ja...

Executable Statemachine Target Machine

● Local ○ Remote

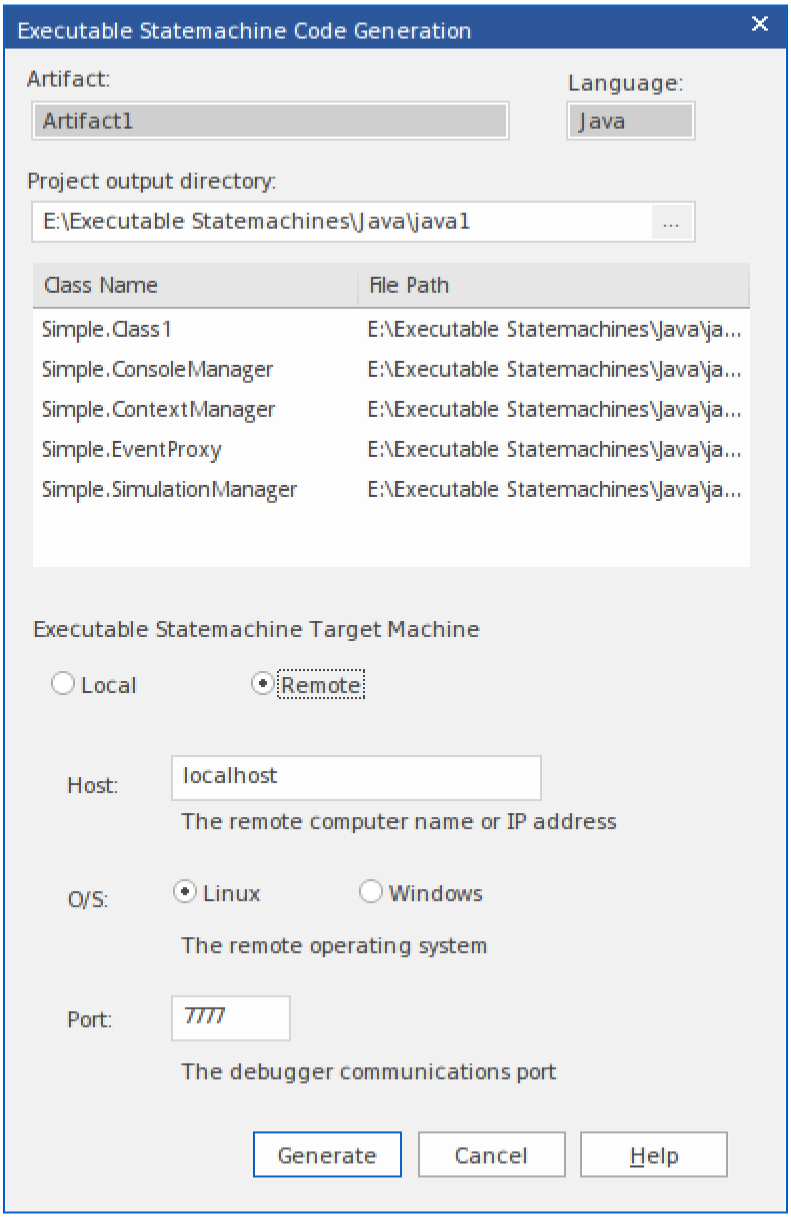
Local Path: Location of Java JDK installation directory.

JAVA_HOME C:\Program Files (x86)\Java\jdk1.7.0_51

Compilation Options: ● 32bit ○ 64bit

Project output directory	Displays the directory in which the generated code files will be stored. If necessary, click on the button at the right of the field to browse for and select a different directory. The names of the generated classes and their source file paths are displayed after this.
Executable Statemachine Target Machine	Select the 'Local' option.
Java JDK	Enter the installation directory of the Java JDK to be used.

Generating Code (Java on Linux)


Executable Statemachine Code Generation

Artifact: Artifact1

Language: Java

Project output directory: E:\Executable Statemachines\Java\java1

Class Name	File Path
Simple.Class1	E:\Executable Statemachines\Java\ja...
Simple.ConsoleManager	E:\Executable Statemachines\Java\ja...
Simple.ContextManager	E:\Executable Statemachines\Java\ja...
Simple.EventProxy	E:\Executable Statemachines\Java\ja...
Simple.SimulationManager	E:\Executable Statemachines\Java\ja...

Executable Statemachine Target Machine

○ Local ● Remote

Host: localhost

The remote computer name or IP address

O/S: ● Linux ○ Windows

The remote operating system

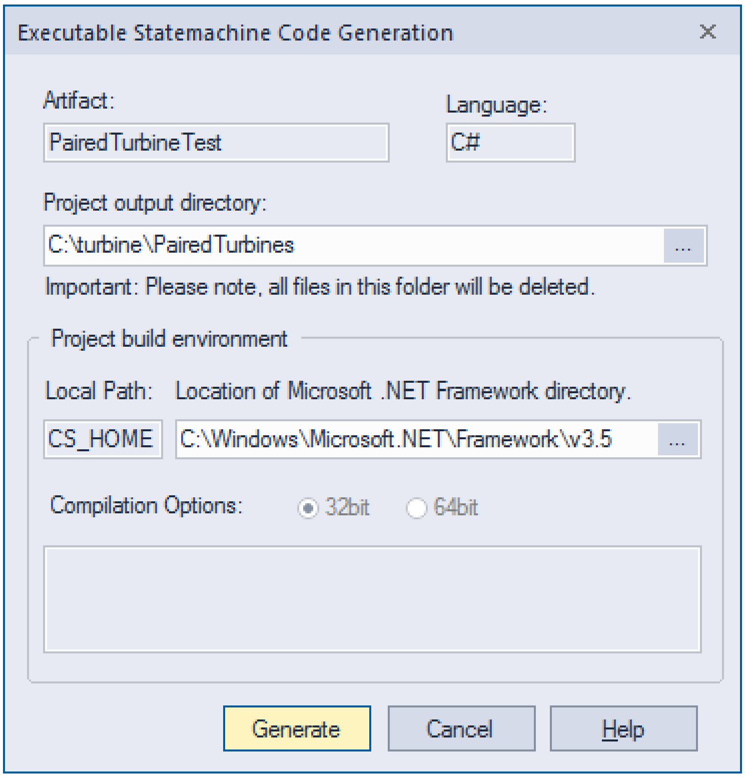
Port: 7777

The debugger communications port

Project output directory:	Displays the directory in which the generated code files will be stored. If necessary, click on the button at the right of the field to browse for and select a different directory. The names of the generated classes and their source file paths are displayed when the path is changed.
Executable Statemachine Target Machine	Select the 'Remote' option.
Operating System	Select Linux.
Port	This is the debugger Port to be used. You will find references to this Port number in

	the 'Debug' and 'DebugRun' sections of the Analyzer Script generated.

Generating Code (Other Languages)



Executable Statemachine Code Generation

Artifact: PairedTurbineTest

Language: C#

Project output directory: C:\turbine\PairedTurbines

Important: Please note, all files in this folder will be deleted.

Project build environment

Local Path: Location of Microsoft .NET Framework directory.

CS_HOME C:\Windows\Microsoft.NET\Framework\v3.5

Compilation Options: ● 32bit ○ 64bit

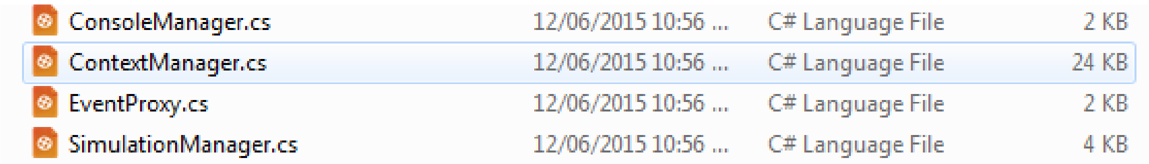
At the same time the System Output window opens at the 'Executable StateMachine Output' page, on which progress messages, warnings or errors are displayed during code generation.

On the 'Executable StateMachine Code Generation' dialog, the 'Artifact' field and 'Language' field display the element name and coding language as defined in the element's 'Properties' dialog.

Field/Option	Description
Project output directory	Displays the directory in which the generated code files will be stored. If necessary, click on the button at the right of the field to browse for and select a different directory.
Project build environment	The fields and information in this panel vary depending on the language defined in the Artifact element and in the script. However, each supported language provides an option to define the path to the target frameworks that are required to build and run the generated code; examples are shown in the *Languages Supported* section of this topic. This path, and its Local Paths ID, are defined in the 'Local Paths' dialog and shown here on the 'Executable StateMachine Code Generation' dialog.

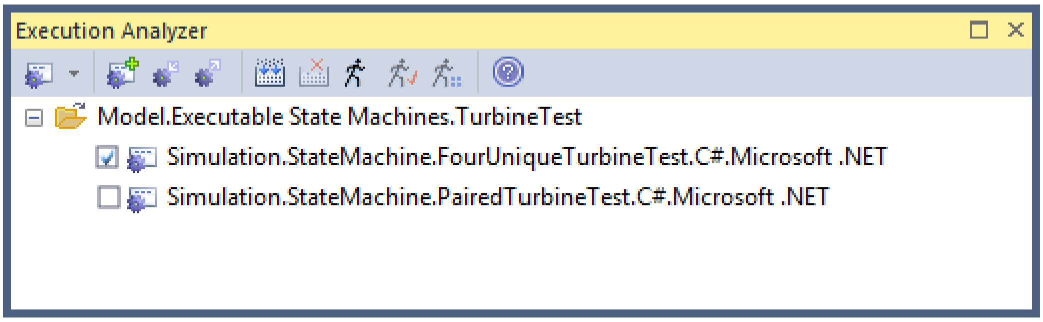
Generate

Click on this button to generate the StateMachine code. The code generation will overwrite any existing files in the project output directory. The set of files will include all required files including those for each Class referenced by the StateMachine.



ConsoleManager.cs	12/06/2015 10:56 ...	C# Language File	2 KB
ContextManager.cs	12/06/2015 10:56 ...	C# Language File	24 KB
EventProxy.cs	12/06/2015 10:56 ...	C# Language File	2 KB
SimulationManager.cs	12/06/2015 10:56 ...	C# Language File	4 KB

Each Executable StateMachine that is generated will also generate an Execution Analyzer script, which is the configuration script for building, running and debugging the Executable StateMachine.



Execution Analyzer

Model.Executable State Machines.TurbineTest

- ☑ Simulation.StateMachine.FourUniqueTurbineTest.C#.Microsoft .NET
- □ Simulation.StateMachine.PairedTurbineTest.C#.Microsoft .NET

Building Code

The code generated by an Executable StateMachine can be built by Enterprise Architect in one of three ways.

Method	Description
Ribbon Generate, Build and Run Command	For the selected Executable StateMachine, generates the entire code base again. The source code is then compiled and the simulation started.
Ribbon Build Command	Compiles the code that has been generated. This can be used directly after generating the code, if you have made changes to the build procedure (the Analyzer Script) or modified the generated code in some way.
Execution Analyzer Script	The generated Execution Analyzer script includes a command to build the source code. This means that when it is active, you can build directly using the built-in shortcut Ctrl+Shift+F12.
Build Output	When building, all output is shown on the 'Build' page of the System Output window. You can double-click on any compiler errors to open a source editor at the appropriate line.

Leveraging existing code

Executable StateMachines generated and executed by Enterprise Architect can leverage existing code for which no Class model exists. To do this you would create an abstract Class element naming only the operations to call in the external codebase. You would then create a generalization between this interface and the StateMachine Class, adding the required linkages manually in the Analyzer Script. For Java you might add .jar files to the Class path. For native code you might add a .dll to the linkage.

Debugging Execution of Executable StateMachines

Creation of Executable StateMachines provides benefits even after the generation of code. Using the Execution Analyzer, Enterprise Architect is able to connect to the generated code. As a result you are able to visually debug and verify the correct behavior of the code; the exact same code generated from your StateMachines, demonstrated by the simulation and ultimately incorporated in a real world system.

Debugging a StateMachine

Being able to debug an Executable StateMachine gives additional benefits, such as being able to:

- Interrupt the execution of the simulation and all executing StateMachines
- View the raw state of each StateMachine instance involved in the simulation
- View the source code and Call Stack at any point in time
- Trace additional information about the execution state through the placement of tracepoints on lines of source code
- Control the execution through use of actionpoints and breakpoints (break on error, for example)
- Diagnose changes in behavior, due to either code or modeling changes

If you have generated, built and run an Executable StateMachine successfully, you can debug it! The Analyzer Script created during the generation process is already configured to provide debugging. To start debugging, simply start running the Executable StateMachine using the Simulation Control. Depending on the nature of the behavior being debugged, however, we would probably set some breakpoints first.

Breaking execution at a state transition

Like any debugger we can use breakpoints to examine the executing StateMachine at a point in code. Locate a Class of interest in either the diagram or Browser window and press F12 to view the source code. It is easy to locate the code for State transitions from the naming conventions used during generation. If you want to break at a particular transition, locate the transition function in the editor and place a breakpoint marker by clicking in the left margin at a line within the function. When you run the Executable StateMachine, the debugger will halt at this transition and you will be able to view the raw state of variables for any StateMachines involved.

Breaking execution conditionally

Each breakpoint can take a condition and a trace statement. When the breakpoint is encountered and the condition evaluates to True the execution will halt. Otherwise the execution will continue as normal. You compose the condition using the names of the raw variables and comparing them using the standard equality operands: < > = >= <=. For example:

```
(this.m_nCount > 100) and (this.m_ntype == 1)
```

To add a condition to a breakpoint you have set, right-click on the breakpoint and select 'Properties'. By clicking on the breakpoint while pressing the Ctrl key, the properties can be quickly edited.

Tracing auxiliary information

It is possible to trace information from within the StateMachine itself using the TRACE clause in, for example, an *effect*. Debugging also provides trace features known as Tracepoints. These are simply breakpoints that, instead of breaking, print trace statements when they are encountered. The output is displayed in the Simulation Control window. They can be used as a diagnostic aid to show and prove the sequence of events and the order in which instances change state.

Viewing the Call Stack

Whenever a breakpoint is encountered, the Call Stack is available from the Analyzer menu. Use this to determine the sequence in which the execution is taking place.

Execution and Simulation of Executable StateMachines

One of the many features of Enterprise Architect is its ability to perform simulations. An Executable StateMachine generated and built in Enterprise Architect can hook into the Simulation facilities to visually demonstrate the live execution of the StateMachine Artifact.

Starting a Simulation

The Simulation Control toolbar provides a Search button that you use to select the Executable StateMachine Artifact to run. The control maintains a drop-down list of the most recent Executable StateMachines for you to choose from. You can also use the context menu on an Executable StateMachine Artifact itself to initiate the simulation.

Controlling Speed

The Simulation Control provides a speed setting. You can use this to adjust the rate at which the simulation executes. The speed is represented as a value between 0 and 100 (a higher value is faster). A value of zero will cause the simulation to halt after every step; this requires using the toolbar controls to manually step through the simulation.

Notation for Active States

As the Executable StateMachine executes, the relevant StateMachine diagrams are displayed. The display is updated at the end of every step-to-completion cycle. You will notice that only the active State for the instance completing a step is highlighted. The other States remain dimmed.

It is easy to identify which instance is in which State, as the States are labeled with the name of any instance currently in that particular state. If two or more Artifact properties of the same type share the same State, the State will have a separate label for each property name.

Generate Timing Diagram

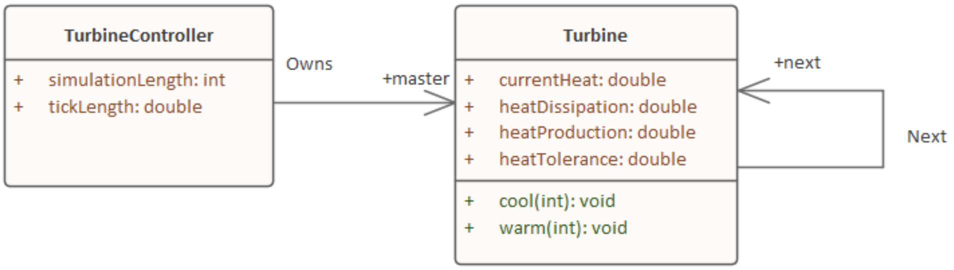
After completing the simulation of an Executable StateMachine, you can generate a Timing diagram from the output. To do this:

In the Simulation window toolbar, click on 'Tools | Generate Timing Diagram'.

Example: Executable StateMachine

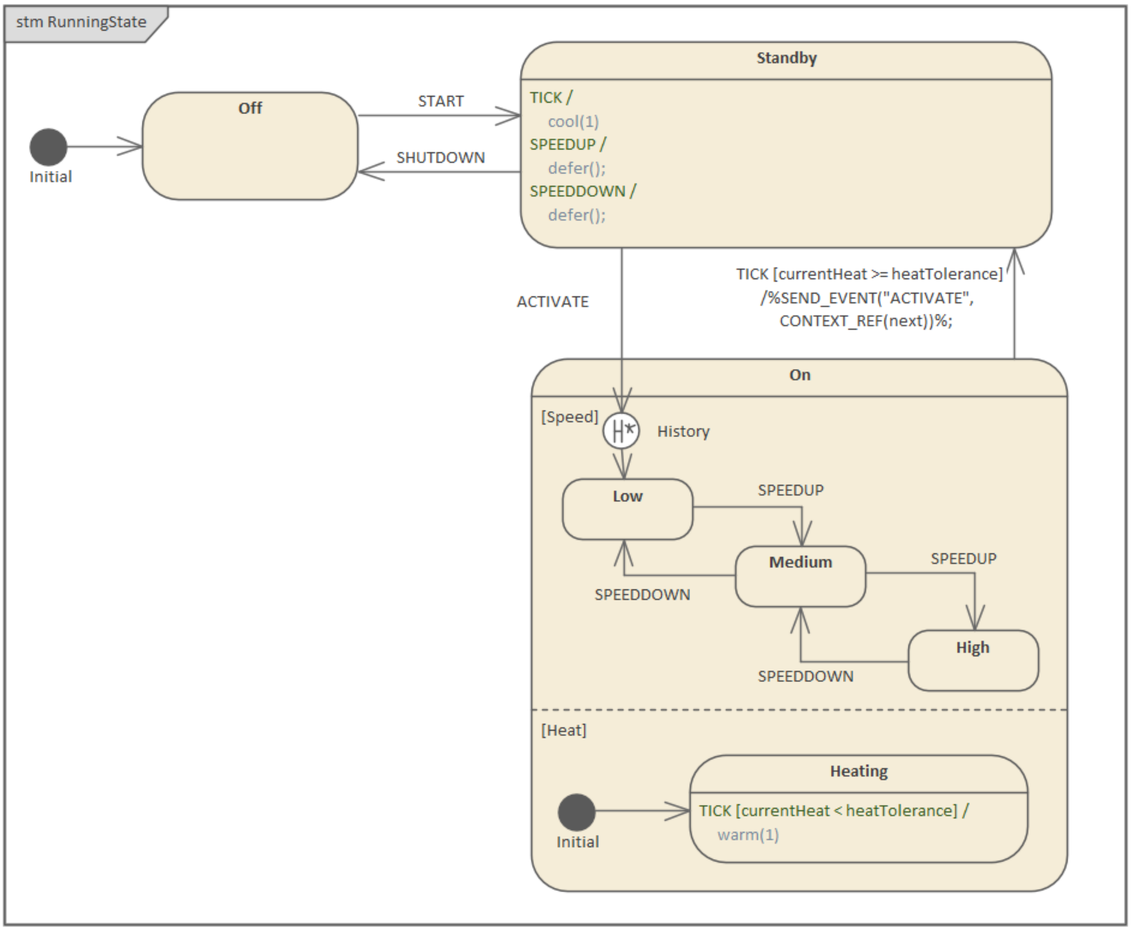
Example Class Model

This image shows a sample Class model that is used by the StateMachines described in this topic.

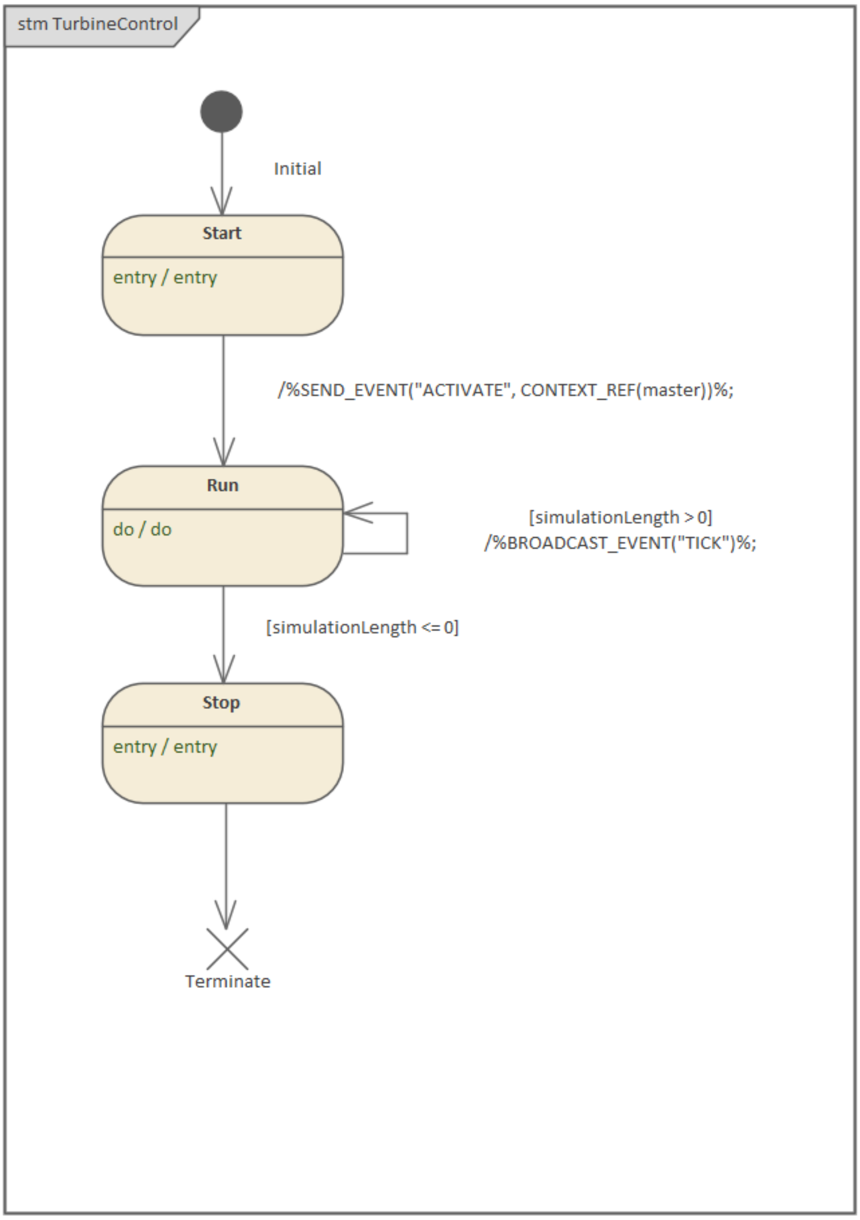


Example StateMachines

These two diagrams show the definitions of two StateMachines. The first references another StateMachine of the same type, while the second drives any instances of the first that exist.

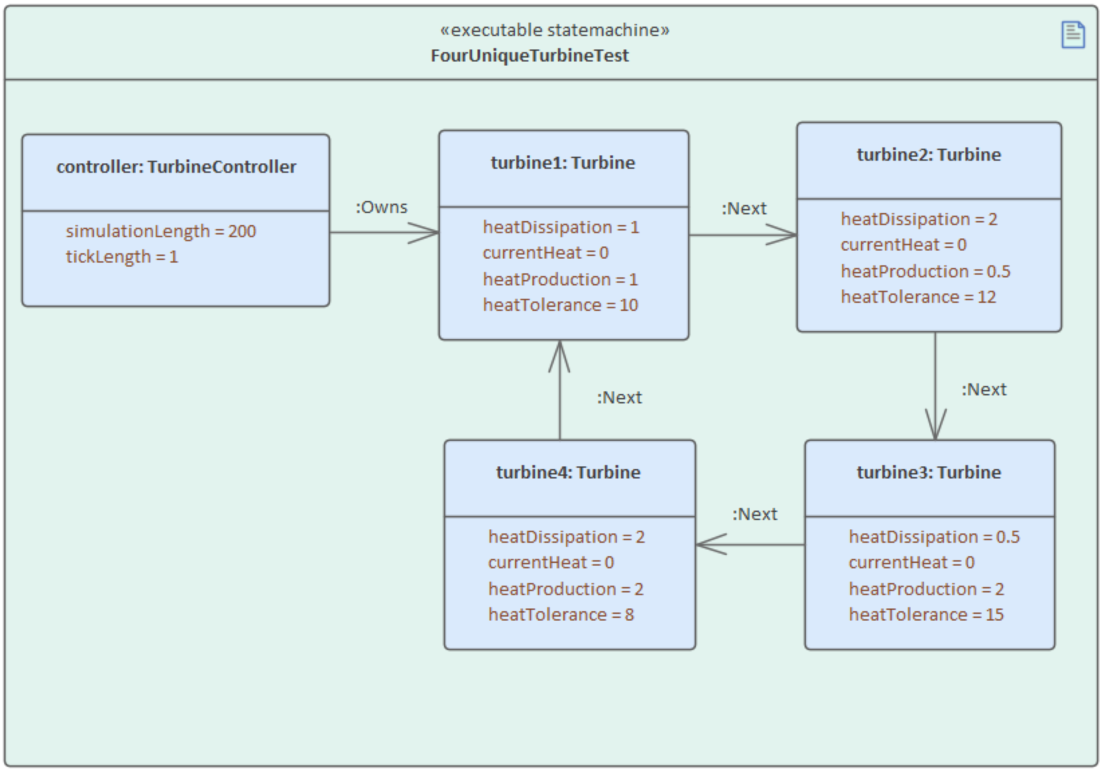


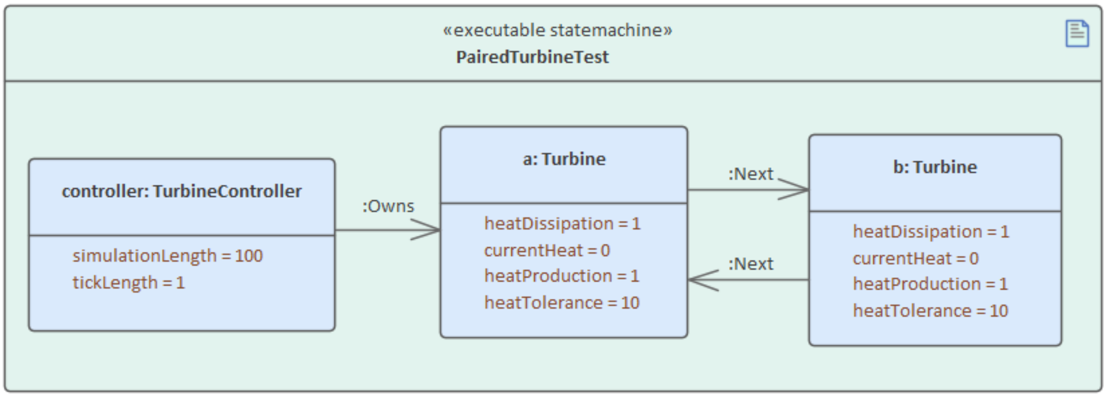
The top level controller.



Example Artifacts

From the example Class and StateMachine diagrams, we can create Executable StateMachines as shown here.

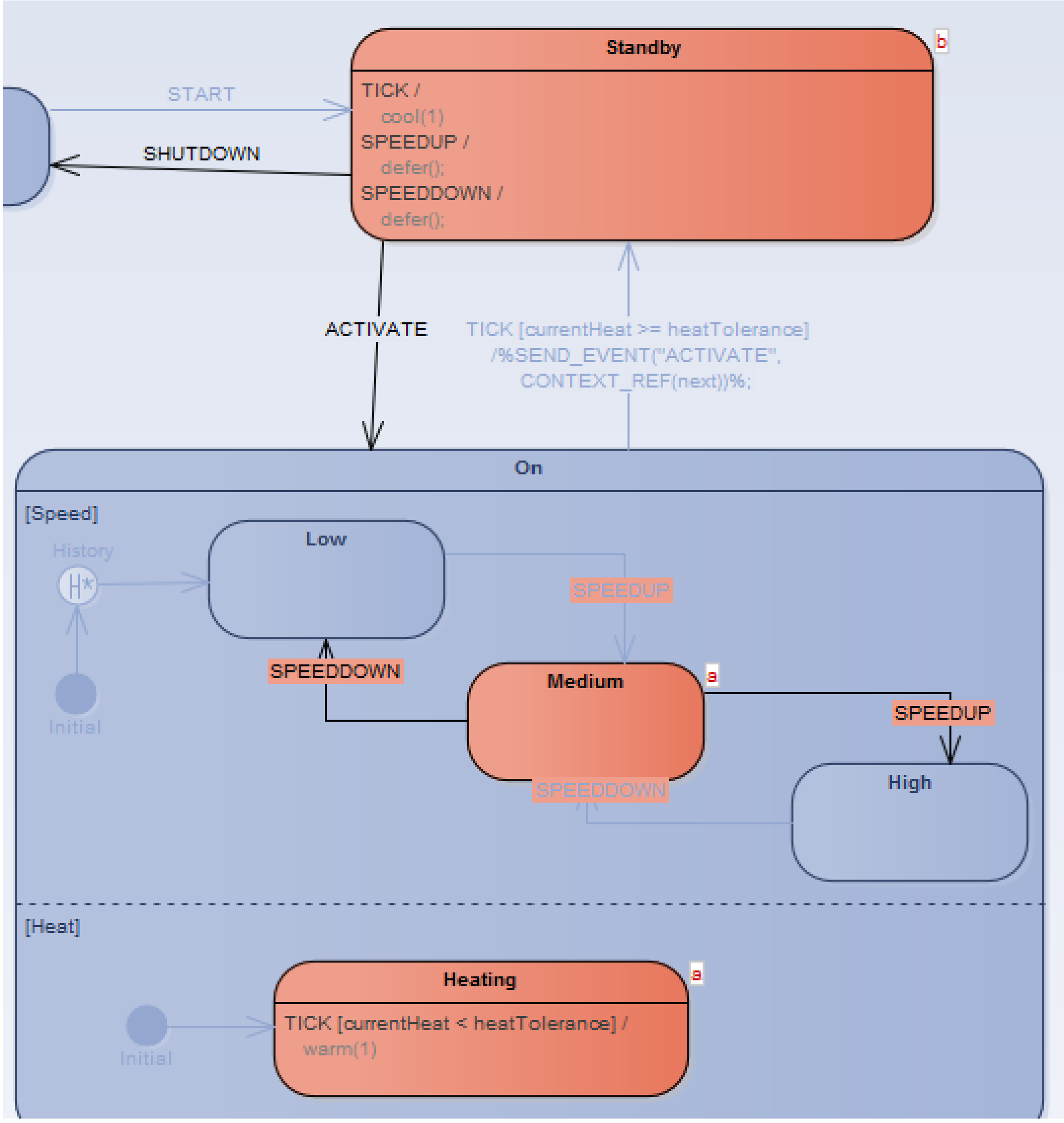




Note how property values have been set for each property, and the links between elements identify the relationships that exist in the Class model.

Simulation Results

When running a simulation, Enterprise Architect will highlight the currently active States in any StateMachines. Where multiple instances of a StateMachine exist, it will also show the names of each instance in that State.


Standby
b
TICK /
cool(1)
SPEEDUP /
defer();
SPEEDDOWN /
defer();
START
SHUTDOWN
ACTIVATE
TICK [currentHeat >= heatTolerance]
/%SEND_EVENT("ACTIVATE",
CONTEXT_REF(next))%;
On
[Speed]
History
Initial
Low
SPEEDUP
SPEEDDOWN
Medium
a
SPEEDUP
SPEEDDOWN
High
[Heat]
Initial
Heating
a
TICK [currentHeat < heatTolerance] /
warm(1)

Example: Simulation Commands

This example demonstrates how we can use the Simulation window to observe Trace messages or send commands to control a StateMachine. Through the example, you can examine:

- An attribute of a context - the member variable defined in the Class, which is the context of the StateMachine; these attributes carry values in the scope of the context for all State behaviors and transition effects, to access and modify
- Each attribute of a Signal - the member variable defined in the Signal, which is referenced by an Event and can serve as an Event Parameter; each Signal Event occurrence might have different instances of a Signal
- The use of the 'Eval' command to query the runtime value of a context's attribute
- The use of the 'Dump' command to dump the current state's active count; it can also dump the current event deferred in the pool

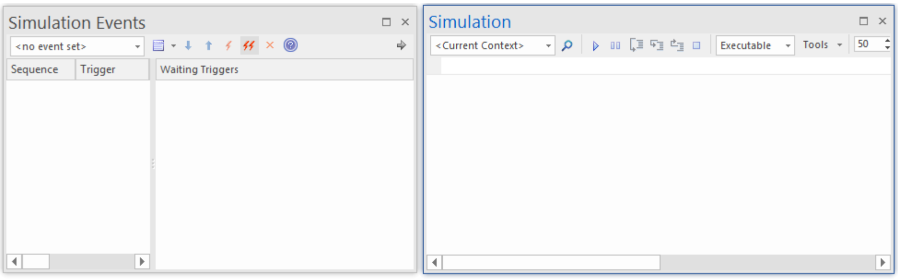
This example is taken from the EAExample model:

Example Model.Model Simulation.Executable StateMachine.Simulation Commands

Access

<table>
<tr>
<td>Ribbon</td>
<td>

- Simulate > Dynamic Simulation > Simulator > Open Simulation Window)
- Simulate > Dynamic Simulation > Events (for the Simulation Events window)

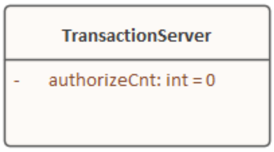


These two windows are frequently used together in the simulation of Executable StateMachines.
</td>
</tr>
</table>

Create Context and StateMachine

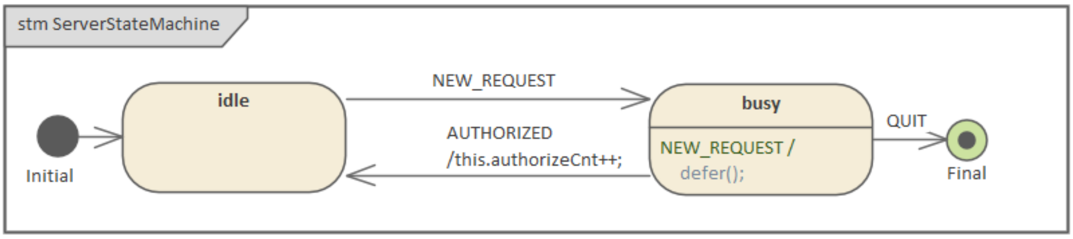
In this section we will create a Class called TransactionServer, which defines a StateMachine as its behavior. We then create an Executable StateMachine Artifact as the simulation environment.

Create the Context of the StateMachine



1. Create a Class element called *TransactionServer.*
2. In this Class, create an attribute called *authorizeCnt* with initial value 0.
3. In the Browser window, right-click on *TransactionServer* and select the 'Add | StateMachine' option.

Create the StateMachine



1. Create an Initial pseudostate called *Initial*.
2. Transition to a State called *idle*.
3. Transition to a State called *busy*, with the trigger NEW_REQUEST.
4. Transition:
   - To a Final pseudostate called *Final*, with the trigger QUIT
   - Back to *idle*, with the trigger AUTHORIZED, with the Effect 'this.authorizeCnt++;'

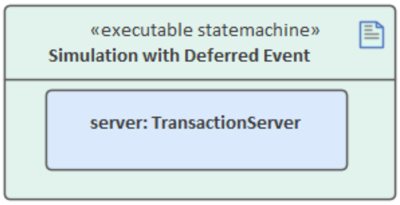
Create a Deferred Event for the State *busy*

1. Draw a self-transition for *busy.*
2. Change the 'kind' of the transition to 'internal'.
3. Specify the Trigger to be the event you want to defer.
4. In the 'Effect' field, type 'defer();'.

Create a Signal and Attributes

1. Create a Signal element called *RequestSignal.*
2. Create an attribute called *requestType* with type 'int'.
3. Configure the Event NEW_REQUEST to reference *RequestSignal.*

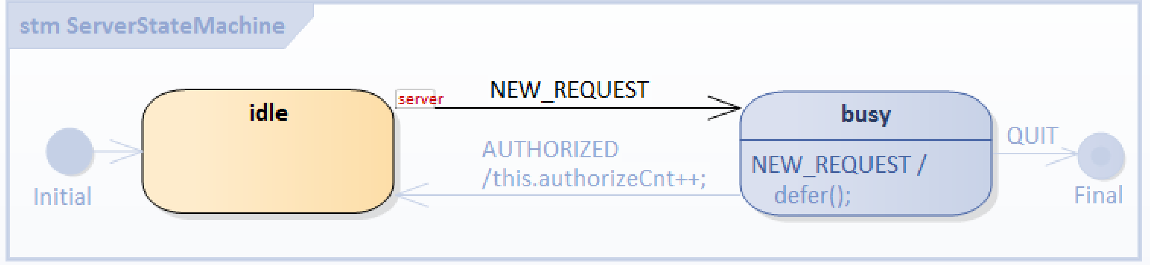
Create the Executable StateMachine Artifact


«executable statemachine»
Simulation with Deferred Event
server: TransactionServer

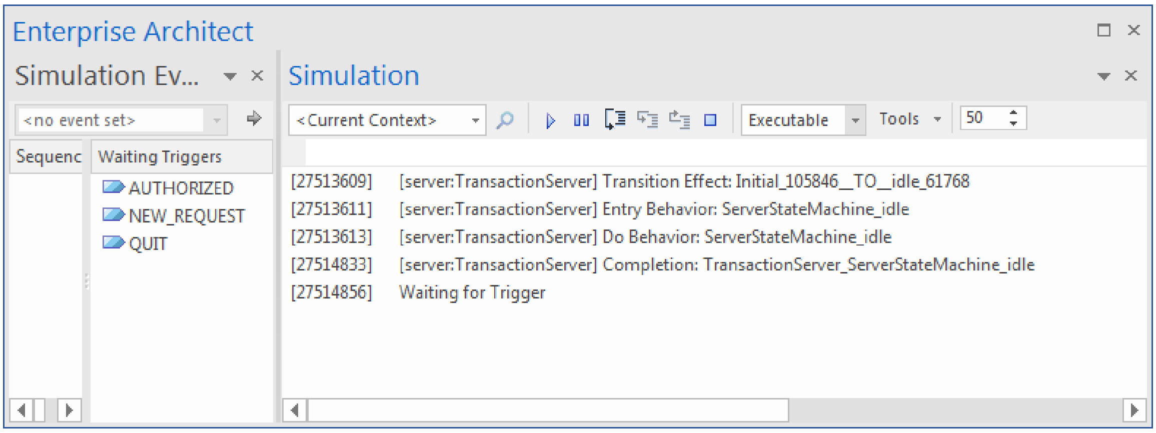
1. From the 'Simulation' page of the Diagram Toolbox, drag an 'Executable StateMachine' icon onto the diagram and call the element *Simulation with Deferred Event.*
2. Ctrl+Drag the *TransactionServer* element from the Browser window and drop it onto the Artifact as a property, with the name *server.*
3. Set the language of the Artifact to JavaScript, which does not require a compiler (for the example; in production you could also use C, C++, C#, or Java, which also support Executable StateMachines).
4. Click on the Artifact and select the 'Simulate > Executable States > Statemachine > Generate, Build and Run' ribbon option.

Simulation Window and Commands

When the simulation starts, *idle* is the current state.



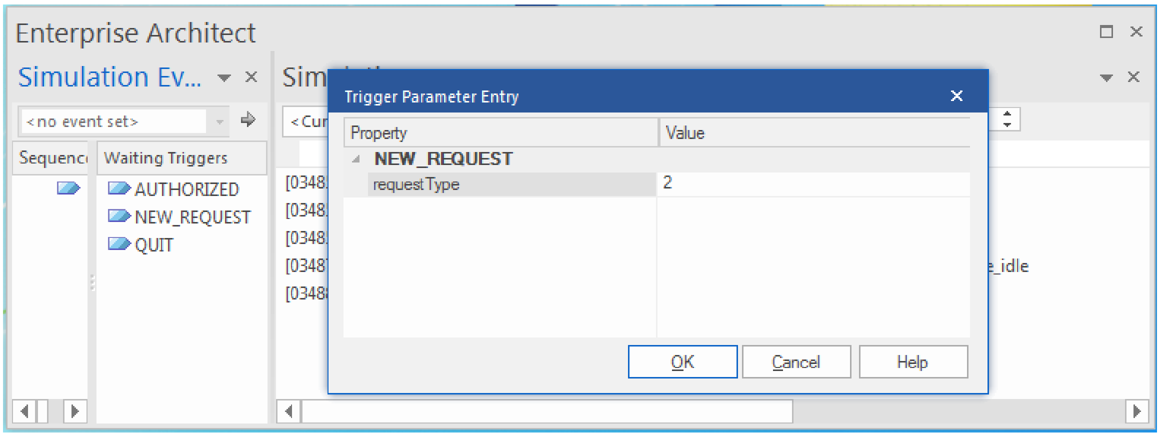
The Simulation window shows that the Transition Effect, Entry and Do behavior is finished for state *idle*, and the StateMachine is waiting for a trigger.



```
[27513609]   [server:TransactionServer] Transition Effect: Initial_105846__TO__idle_61768
[27513611]   [server:TransactionServer] Entry Behavior: ServerStateMachine_idle
[27513613]   [server:TransactionServer] Do Behavior: ServerStateMachine_idle
[27514833]   [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_idle
[27514856]   Waiting for Trigger
```

Event Data via Values for Signal Attributes

For the Trigger Signal Event NEW_REQUEST, the 'Trigger Parameter Entry' dialog displays to prompt for values for the listed attributes defined in the Signal *RequestSignal*, referenced by NEW_REQUEST.



Property	Value
NEW_REQUEST	
requestType	2

Type the value '2' and click on the OK button. The Signal attribute values are then passed to invoked methods such as the State's behaviors and the Transition's effects.

These messages are output to the Simulation window:

[03612562] Waiting for Trigger

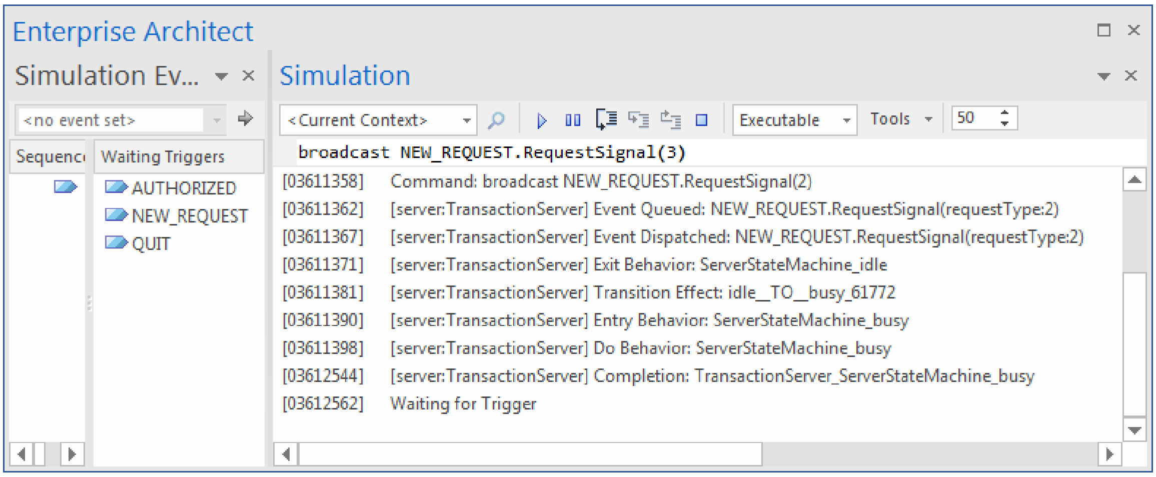
[03611358] Command: **broadcast NEW_REQUEST.RequestSignal(2)**

[03611362] [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:2)

[03611367] [server:TransactionServer] Event Dispatched: NEW_REQUEST.RequestSignal(requestType:2)

```
[03611371]    [server:TransactionServer] Exit Behavior: ServerStateMachine_idle
[03611381]    [server:TransactionServer] Transition Effect: idle__TO__busy_61772
[03611390]    [server:TransactionServer] Entry Behavior: ServerStateMachine_busy
[03611398]    [server:TransactionServer] Do Behavior: ServerStateMachine_busy
[03612544]    [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_busy
[03612562]    Waiting for Trigger
```

We can broadcast events by double-clicking on the item listed in the Simulation Events window. Alternatively, we can type a command string in the text field of the Simulation window (underneath the toolbar).



```
broadcast NEW_REQUEST.RequestSignal(3)
[03611358]    Command: broadcast NEW_REQUEST.RequestSignal(2)
[03611362]    [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:2)
[03611367]    [server:TransactionServer] Event Dispatched: NEW_REQUEST.RequestSignal(requestType:2)
[03611371]    [server:TransactionServer] Exit Behavior: ServerStateMachine_idle
[03611381]    [server:TransactionServer] Transition Effect: idle__TO__busy_61772
[03611390]    [server:TransactionServer] Entry Behavior: ServerStateMachine_busy
[03611398]    [server:TransactionServer] Do Behavior: ServerStateMachine_busy
[03612544]    [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_busy
[03612562]    Waiting for Trigger
```

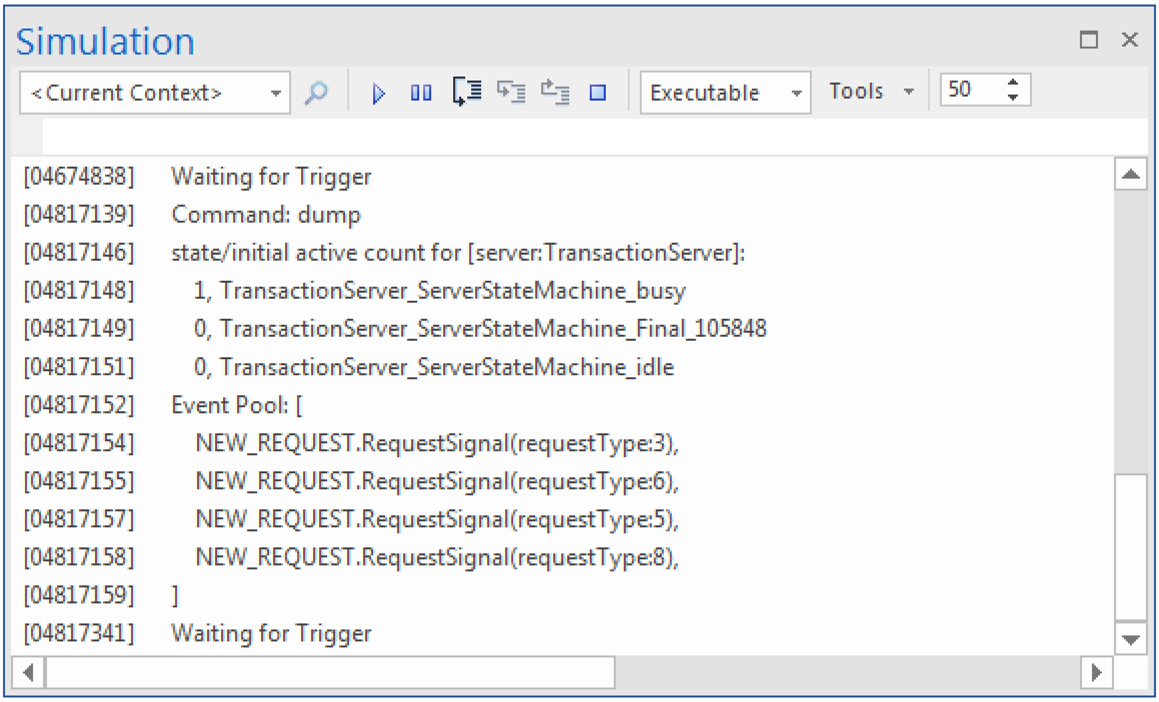
```
[03612562]    Waiting for Trigger
[04460226]    Command: broadcast NEW_REQUEST.RequestSignal(3)
[04460233]    [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:3)
[04461081]    Waiting for Trigger
```

The Simulation message indicates that the event occurrence is deferred (Event Queued, but not dispatched). We can run further commands using the text field:

```
[04655441]    Waiting for Trigger
[04664057]    Command: broadcast NEW_REQUEST.RequestSignal(6)
[04664066]    [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:6)
[04664803]    Waiting for Trigger
[04669659]    Command: broadcast NEW_REQUEST.RequestSignal(5)
[04669667]    [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:5)
[04670312]    Waiting for Trigger
[04674196]    Command: broadcast NEW_REQUEST.RequestSignal(8)
[04674204]    [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:8)
[04674838]    Waiting for Trigger
```

dump: Query 'active count' for a State and Event Pool

Type *dump* in the text field; these results display:



```
[04674838]   Waiting for Trigger
[04817139]   Command: dump
[04817146]   state/initial active count for [server:TransactionServer]:
[04817148]      1, TransactionServer_ServerStateMachine_busy
[04817149]      0, TransactionServer_ServerStateMachine_Final_105848
[04817151]      0, TransactionServer_ServerStateMachine_idle
[04817152]   Event Pool: [
[04817154]      NEW_REQUEST.RequestSignal(requestType:3),
[04817155]      NEW_REQUEST.RequestSignal(requestType:6),
[04817157]      NEW_REQUEST.RequestSignal(requestType:5),
[04817158]      NEW_REQUEST.RequestSignal(requestType:8),
[04817159]   ]
[04817341]   Waiting for Trigger
```

From the 'active count' section, we can see that *busy* is the active state (active count is 1).

Tips: For a Composite State, the active count is 1 (for itself) *plus* the number of active regions.

From the 'Event Pool' section, we can see that there are four event occurrences in the Event Queue. Each instance of the signal carries different data.

The order of the events in the pool is the order in which they are broadcast.

eval: Query Run Time Value of the Context

Trigger AUTHORIZED,

[04817341] Waiting for Trigger

[05494672] Command: broadcast AUTHORIZED

[05494678] [server:TransactionServer] Event Queued: AUTHORIZED

[05494680] [server:TransactionServer] Event Dispatched: AUTHORIZED

[05494686] [server:TransactionServer] Exit Behavior: ServerStateMachine_busy

[05494686] [server:TransactionServer] **Transition Effect: busy__TO__idle_61769**

[05494687] [server:TransactionServer] Entry Behavior: ServerStateMachine_idle

[05494688] [server:TransactionServer] Do Behavior: ServerStateMachine_idle

[05495835] [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_idle

[05495842] [server:TransactionServer] **Event Dispatched: NEW_REQUEST.RequestSignal(requestType:3)**

[05495844] [server:TransactionServer] Exit Behavior: ServerStateMachine_idle

[05495846] [server:TransactionServer] Transition Effect: idle__TO__busy_61772

[05495847] [server:TransactionServer] Entry Behavior: ServerStateMachine_busy

[05495850] [server:TransactionServer] Do Behavior: ServerStateMachine_busy

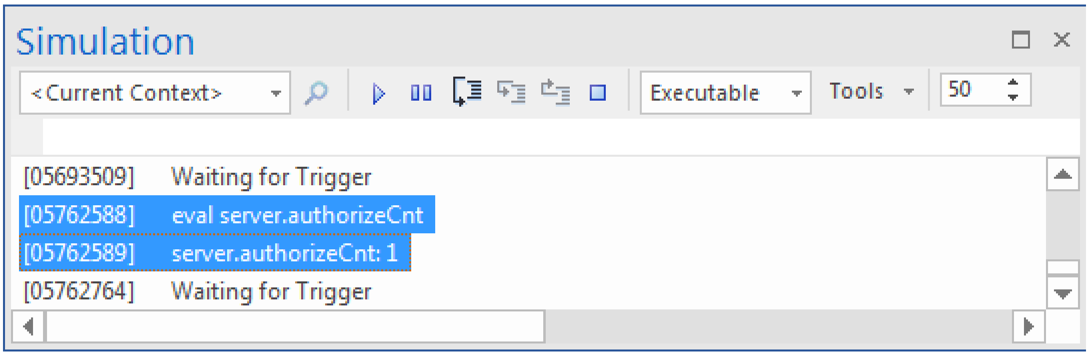
[05496349] [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_busy

[05496367] Waiting for Trigger

- The transition from *busy* to *idle* is made, so we expect the effect to be executed
- One event is recalled from the pool and dispatched when *idle* is completed, causing *busy* to become the active state
- Type *dump* and notice that there are three events left in the pool; the first one is recalled and dispatched

```
[05693348]    Event Pool: [
[05693349]        NEW_REQUEST.RequestSignal(requestType:6),
[05693351]        NEW_REQUEST.RequestSignal(requestType:5),
[05693352]        NEW_REQUEST.RequestSignal(requestType:8),
[05693354]    ]
```

Type *eval server.authorizeCnt* in the text field. This figure indicates that the run time value of 'server.authorizeCnt' is 1.



```
[05693509]    Waiting for Trigger
[05762588]    eval server.authorizeCnt
[05762589]    server.authorizeCnt: 1
[05762764]    Waiting for Trigger
```

Trigger AUTHORIZED again. When the StateMachine is stable at *busy*, there will be two events left in the pool. Run *eval server.suthorizeCnt* again; the value will be 2.

Access Context's Member Variable from State Behavior and Transition Effect

Enterprise Architect's Executable StateMachine supports simulation for C, C++, C#, Java and JavaScript.

For C and C++, the syntax differs from C#, Java and JavaScript in accessing the context's member variables. C and C++ use the pointer '->' while the others simply use '.'; however, you can always use *this.variableName* to access the variables. Enterprise Architect will translate it to this->variableName for C and C++.

So for all languages, simply use this format for the simulation:

```
this.variableName
```

Examples:

In the transition's effect:

```
this.authorizeCnt++;
```

In some state's entry, do or exit behavior:

```
this.foo += this.bar;
```

Note: by default Enterprise Architect is only replacing 'this->' with 'this' for C and C++; For example:

```
this.foo = this.bar + myObject.iCount + myPointer->iCount;
```

will be translated to:

```
this->foo = this->bar + myObject.iCount + myPointer->iCount;
```

A Complete List of Supported Commands

Since the Executable StateMachine Artifact can simulate multiple contexts together, some of the commands can specify an instance name.

run StateMachine:

As each context can have multiple StateMachines, the 'run' command can specify a StateMachine to start with.

- run instance.statemachine
- run all.all
- run instance
- run all
- run

For example:

```
run
run all
run server
run server.myMainStatemachine
```

broadcast & send Event:

- broadcast EventString
- send EventString to instance
- send EventString (equivalent to broadcast EventString)

For example:

```
broadcast Event1
send Event1 to client
```

dump Command:

- dump
- dump instance

For example:

```
dump
dump server
dump client
```

eval Command:

- eval instance.variableName

For example:

```
eval client.requestCnt
eval server.responseCnt
```

exit Command:

- exit

The EventString's Format:

- EventName.SignalName(argument list)

Note: the argument list should match the attributes defined in the signal **by order**.

For example, if the Signal defines two attributes:

- foo
- bar

Then these EventStrings are valid:

- Event1.Signal1(10, 5) --------- foo = 10; bar = 5
- Event1.Signal1(10,) --------- foo = 10; bar is undefined
- Event1.Signal1(,5) --------- bar = 10; foo is undefined
- Event1.Signal1(,) --------- both foo and bar are not defined

If the Signal does not contain any attributes, we can simplify the EventString to:

- EventName

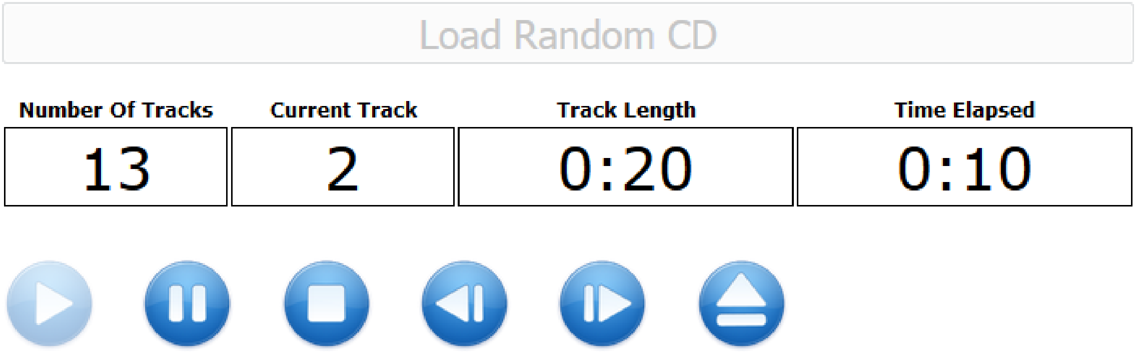
Example: Simulation in HTML with JavaScript

We already know that users can model an Executable StateMachine and simulate it in Enterprise Architect with the generated code. Using the two examples *CD Player* and the *Regular Expression Parser*, we will now demonstrate how you can integrate the generated code with your real projects.

Enterprise Architect provides two different mechanisms for client code to use a StateMachine:

- Active State Based - the client can query the current active state, then 'switch' the logic based on the query result
- Runtime Variable Based - the client does not act on the current active state, but does act on the runtime value of the variables defined in the Class containing the StateMachine

In the *CD Player* example, there are very few states and many buttons on the GUI, so it is quite easy to implement the example based on the Active State Mechanism; we will also query the runtime value for the current track.



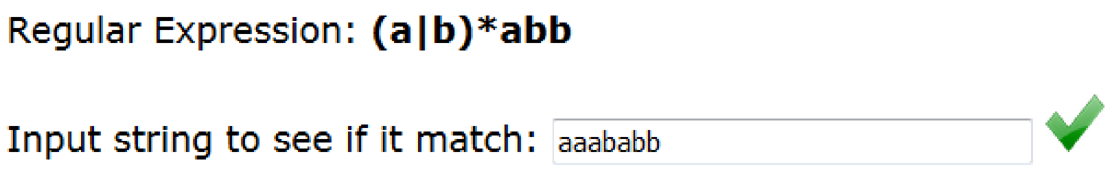
Number Of Tracks: 13

Current Track: 2

Track Length: 0:20

Time Elapsed: 0:10

In the *Regular Expression Parser* example the StateMachine handles everything, and a member variable *bMatch* changes its runtime value when states change. The client does not register how many states are there or which state is currently active.



Regular Expression: **(a|b)*abb**

Input string to see if it match: aaababb ✓

In these topics, we demonstrate how to model, simulate and integrate a CD Player and a Parser for a specified Regular Expression, step by step:

- CD Player
- Regular Expression Parser

CD Player

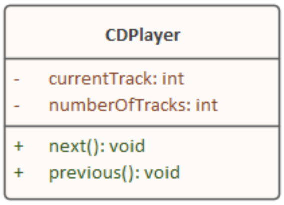
The behavior of a CD Player application might appear intuitive; however, there are many rules related to when the buttons are enabled and disabled, what is displayed in the text fields of the window and what happens when you supply events to the application.

Suppose our example CD Player has these features:

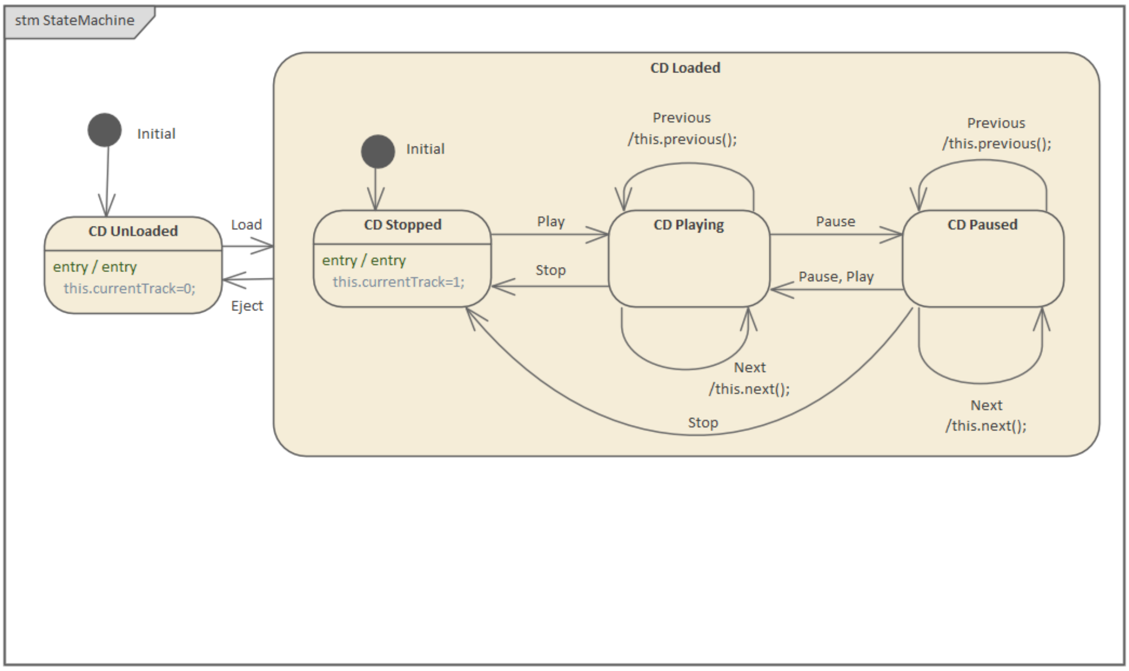
- Buttons - Load Random CD, Play, Pause, Stop, Previous Track, Next Track and Eject
- Displays - Number Of Tracks, Current Track, Track Length and Time Elapsed

StateMachine for CD Player

A Class *CDPlayer* is defined with two attributes: *currentTrack* and *numberOfTracks*.



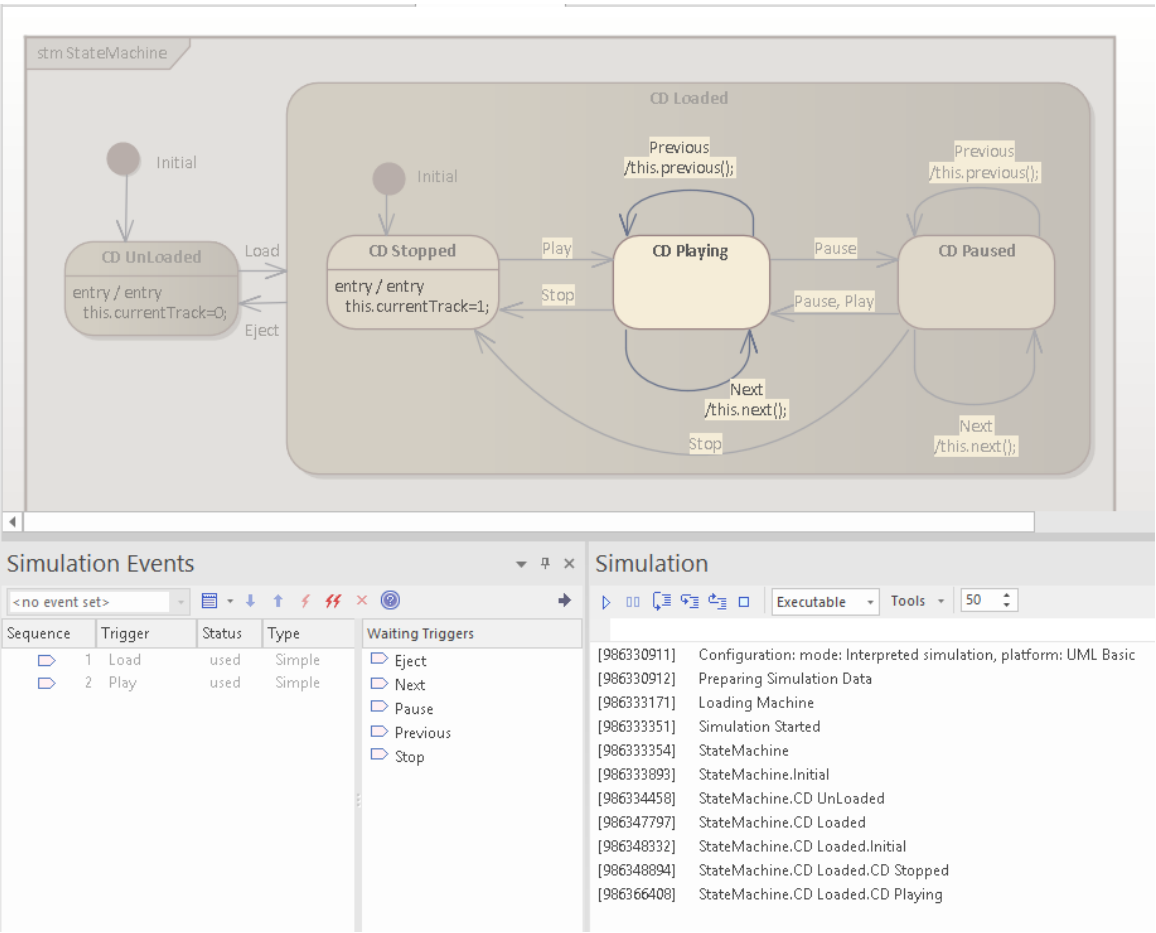
A StateMachine is used to describe the states of the CD Player:



- On the higher level, the StateMachine has two States: *CD UnLoaded* and *CD Loaded*
- *CD Loaded* can be composed of three simple States: *CD Stopped*, *CD Playing*, *CD Paused*
- Transitions are defined with triggers for the events Load, Eject, Play, Pause, Stop, Previous and Next
- State behaviors and transition Effects are defined to change the value of attributes defined in *CDPlayer*; for example, the 'Previous' event will trigger the self transition (if the current state is *CD Playing* or *CD Paused*) and the Effect will be executed, which will decrement the value of *currentTrack* or wrap to the last track

We can create an Executable StateMachine Artifact and create a property typing to *CDPlayer*, then simulate the

StateMachine in Enterprise Architect to make sure the model is correct.



Sequence	Trigger	Status	Type
1	Load	used	Simple
2	Play	used	Simple

Waiting Triggers

- Eject
- Next
- Pause
- Previous
- Stop

```
[986330911]   Configuration: mode: Interpreted simulation, platform: UML Basic
[986330912]   Preparing Simulation Data
[986333171]   Loading Machine
[986333351]   Simulation Started
[986333354]   StateMachine
[986333893]   StateMachine.Initial
[986334458]   StateMachine.CD UnLoaded
[986347797]   StateMachine.CD Loaded
[986348332]   StateMachine.CD Loaded.Initial
[986348894]   StateMachine.CD Loaded.CD Stopped
[986366408]   StateMachine.CD Loaded.CD Playing
```

Inspect the code generated

Enterprise Architect will generate these files in a folder that you have specified:

- Back-end code: CDPlayer.js, ContextManager.js, EventProxy.js
- Client code: ManagerWorker
- Front-end code: statemachineGUI.js, index.html
- Other code: SimulationManager.js

File	Description
/CDPlayer.js	This file defines the Class CDPlayer and its attributes and operations. It also defines the Class's StateMachines with the State behaviors and the transition effects.
/ContextManager.js	This file is the abstract manager of contexts. The file defines the contents that are independent of the actual contexts, which are defined in the generalization of the *ContextManager*, such as *SimulationManager* and *ManagerWorker*. The simulation (Executable StateMachine Artifact) can involve multiple contexts; for example, in a tennis game simulation there will be one *umpire* typed to Class *Umpire,* and two players - *playerA* and *playerB* - typed to Class *Player.* Both Class *Umpire* and Class *Player* will define their own StateMachine(s).
/EventProxy.js	This file defines Events and Signals used in the simulation.

	If we are raising an Event with arguments, we model the Event as a Signal Event, which specifies a Signal Class; we then define attributes for the Signal Class. Each Event occurrence has an instance of the Signal, carrying the runtime values specified for the attributes.
/SimulationManager.js	This file is for simulation in Enterprise Architect.
/html/ManagerWorker.js	This file serves as a middle layer between the front-end and back-end. • The front-end posts a message to request information from the ManagerWorker • Since the ManagerWorker generalizes from ContextManager, it has full access to all the contexts such as querying the current active state and querying the runtime value of a variable • The ManagerWorker will post a message to the front-end with the data it retrieved from the back-end
/html/statemachineGUI.js	This file establishes the communication between the front-end and the ManagerWorker, by defining *stateMachineWorker*. It: • Defines the functions *startStateMachineWebWorker* and *stopStateMachineWebWorker* • Defines the functions *onActiveStateResponse* and *onRuntimeValueResponse* with place-holder code: //to do: write user's logic You could simply replace this comment with your logic, as will be demonstrated later in this topic
/html/index.html	This defines the HTML User Interface, such as the buttons and the input to raise Events or display information. You can define CSS and JavaScript in this file.

Customize index.html and statemachineGUI.js

Make these changes to the generated files:

- Create buttons and displays
- Create a CSS style to format the display and enable/disable the button images
- Create an ElapseTimeWorker.js to refresh the display every second
- Create a TimeElapsed function, set to Next Track when the time elapsed reaches the length of the track
- Create JavaScript as the button 'onclick' event handler
- Once an event is broadcast, request the active State and runtime value for *cdPlayer.currentTrack*
- On initialization, request the active State

In statemachineGUI.js find the function *onActiveStateResponse_cdPlayer*

- In CDPlayer_StateMachine_CDUnLoaded, disable all buttons and enable btnLoad
- In CDPlayer_StateMachine_CDLoaded_CDStopped, disable all buttons and enable btnEject and btnPlay
- In CDPlayer_StateMachine_CDLoaded_CDPlaying, enable all buttons and disable btnLoad and btnPlay
- In CDPlayer_StateMachine_CDLoaded_CDPaused, enable all buttons and disable btnLoad

In statemachineGUI.js find the function *onRuntimeValueResponse*

- In *cdPlayer.currentTrack*, we update the display for current track and track length

The Complete Example

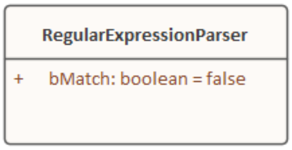
The example can be accessed from the 'Resources' page of the Sparx Systems website, by clicking on this link: CD Player Simulation

Click on the Load Random CD button, and then on the Start Simulation button.

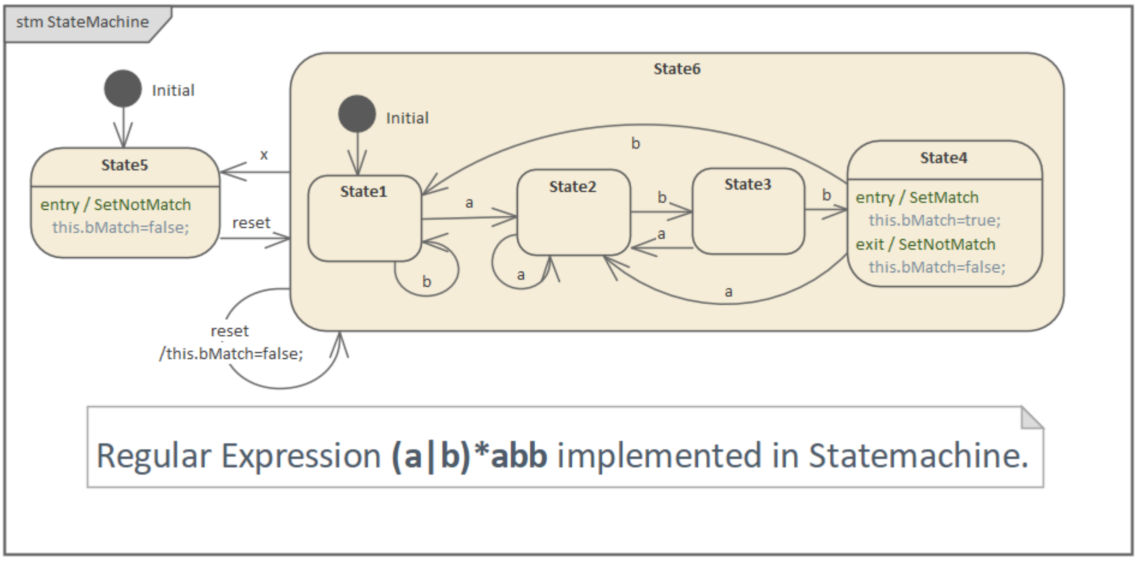
Regular Expression Parser

StateMachine for Regular Expression Parser

The Class *RegularExpressionParser* is defined with one attribute: bMatch.



A StateMachine is used to describe the regular expression (a|b)*abb



- The transition triggers are specified as events *a*, *b*, *x* and *reset*
- On entry to State4, bMatch is set to True; on exit from State4, bMatch is set to False
- On entry to State5, bMatch is set to False
- On self transition of State6, bMatch is set to False

Customize index.html and statemachineGUI.js

Make these changes to the generated files:

- Create an HTML input field and an image to indicate the result
- Create JavaScript as the field's *oninput* event handler
- Create the function 'SetResult′ to toggle the pass/fail image
- Create the function 'getEventStr', which will return 'a' on '*a*' and 'b' on '*b*', but will return 'x' on any other character
- On initialize, broadcast '*reset*'
- On the broadcast event, request the runtime variable 'regxParser.bMatch'

In *statemachineGUI.js*, find the function 'onRuntimeValueResponse'.

- In 'regxParser.bMatch', we will receive 'True' or 'False' and pass it into 'SetResult' to update the image

The Complete Example

The example can be accessed from the 'Resources' page of the Sparx Systems website, by clicking on this link: Regular Expression Parser Simulation

Example: Entering a State

The semantics of entering a State depend on the type of State and the manner in which it is entered.

In all cases, the entry Behavior of the State is executed (if defined) upon entry, but only after any effect Behavior associated with the incoming Transition is completed. Also, if a doActivity Behavior is defined for the State, this Behavior commences execution immediately after the entry Behavior is executed.

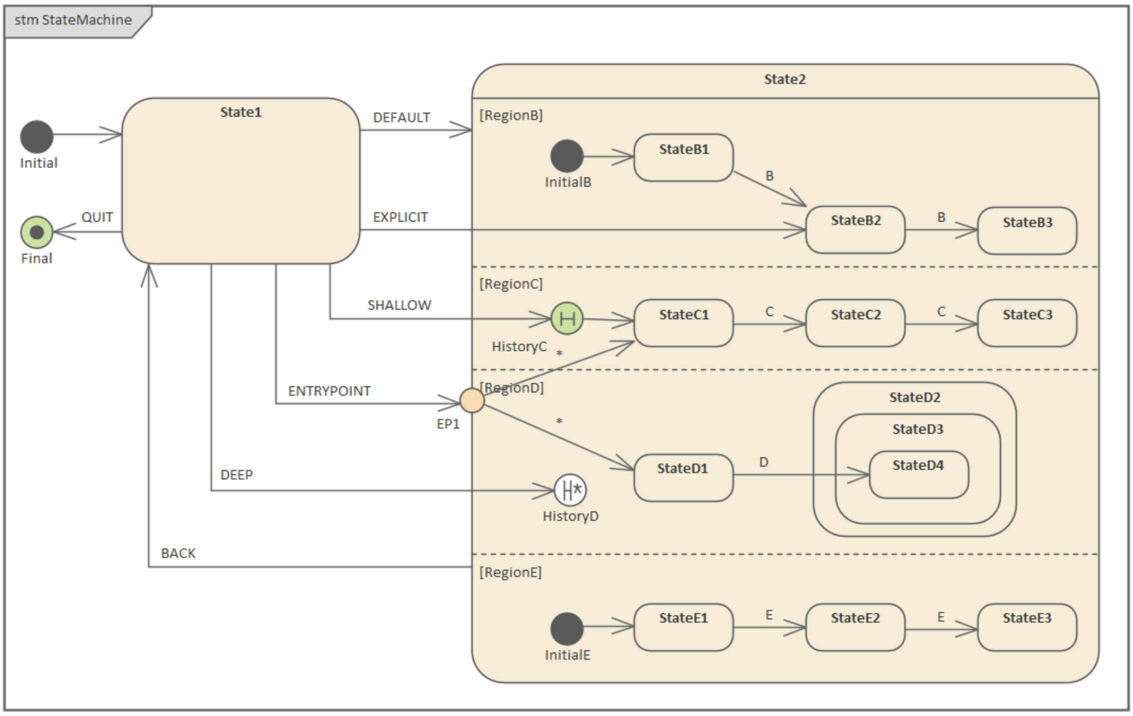
For a Composite State with one or more Regions defined, a number of alternatives exist for each Region:

- *Default entry*: This situation occurs when the owning Composite State is the direct target of a Transition; after executing the entry Behavior and forking a possible doActivity Behavior execution, State entry continues from an initial Pseudostate via its outgoing Transition (known as the default Transition of the State) if it is defined in the Region
  If no initial Pseudostate is defined, this Region will not be active
- *Explicit entry*: If the incoming Transition or its continuations terminate on a directly contained Substate of the owning composite State, then that Substate becomes active and its entry Behavior is executed after the execution of the entry Behavior of the containing composite State
  This rule applies recursively if the Transition terminates on an indirect (deeply nested) Substate
- *Shallow history entry*: If the incoming Transition terminates on a shallowHistory Pseudostate of this Region, the active Substate becomes the Substate that was most recently active (except FinalState) prior to this entry, unless this is the first entry into this State; if it is the first entry into this State or the previous entry had reached a Final, a default shallow history Transition will be taken if it is defined, otherwise the default State entry is applied
- *Deep history entry*: The rule for this case is the same as for shallow history except that the target Pseudostate is of type deepHistory and the rule is applied recursively to all levels in the active State configuration below this one
- *Entry point entry*: If a Transition enters the owning composite State through an entryPoint Pseudostate, then the outgoing Transition originating from the entry point and penetrating into the State in this region is taken; if there are more outgoing Transitions from the entry points, each Transition must target a different Region and all Regions are activated concurrently

For orthogonal States with multiple Regions, if the Transition explicitly enters one or more Regions (in the case of a Fork or entry point), these Regions are entered explicitly and the others by default.

In this example, we demonstrate a model with all these entry behaviors for an orthogonal State.

Modeling a StateMachine



Context of StateMachine

1. Create a Class element named *MyClass*, which serves as the context of the StateMachine.
2. Right-click on *MyClass* in the Browser window and select the 'Add | StateMachine' option.

StateMachine

1. Add to the diagram an *Initial* Node, a State named *State1*, a State named *State2*, and a Final element named *final.*
2. Enlarge *State2* on the diagram, right-click on it and select the 'Advanced | Define Concurrent Substates' option, and define *RegionB, RegionC, RegionD* and *RegionE.*
3. Right-click on *State2* and select the 'New Child Element | Entry Point' option to create the Entry Point *EP1*.
4. In *RegionB*, create the elements *InitialB*, transition to *StateB1*, transition to *StateB2*, transition to *StateB3*; all transitions triggered by Event *B.*
5. In *RegionC*, create the elements shallow *HistoryC* (right-click on History node | Advanced | Deep History | uncheck), transition to *StateC1*, transition to *StateC2*, transition to *StateC3*; all transitions triggered by Event *C.*
6. In *RegionD*, create the elements deep *HistoryD* (right-click on History node | Advanced | Deep History | check), transition to *StateD1*, create *StateD2* as parent of *StateD3*, which is parent of *StateD4*; transition from *StateD1* to *StateD4*; triggered by Event *D.*
7. In *RegionE*, create the elements *InitialE*, transition to *StateE1*, transition to *StateE2*, transition to *StateE3*; all transitions triggered by Event *E.*
8. Draw transitions from the Entry Point *EP1* to *StateC1* and *StateD1.*

Draw transitions for different entry types:

1. Default Entry: *State1* to *State2*; triggered by Event DEFAULT.
2. Explicit Entry: *State1* to *StateB2*; triggered by Event EXPLICIT.
3. Shallow History Entry: *State1* to *HistoryC*; triggered by Event SHALLOW.
4. Deep History Entry: *State1* to *HistoryD*; triggered by Event DEEP.
5. Entry Point Entry: *State1* to *EP1*; triggered by Event ENTRYPOINT.

Other Transitions:

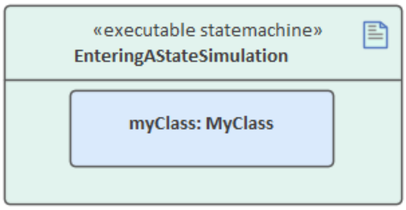
1. Composite State Exit: from *State2* to *State1*; triggered by Event BACK.
2. *State1* to *Final*, triggered by Event QUIT.

Simulation

Artifact

Enterprise Architect supports C, C++, C#, Java and JavaScript. We use JavaScript in this example because we don't need to install a compiler. (For other languages, either Visual Studio or JDK are required.)

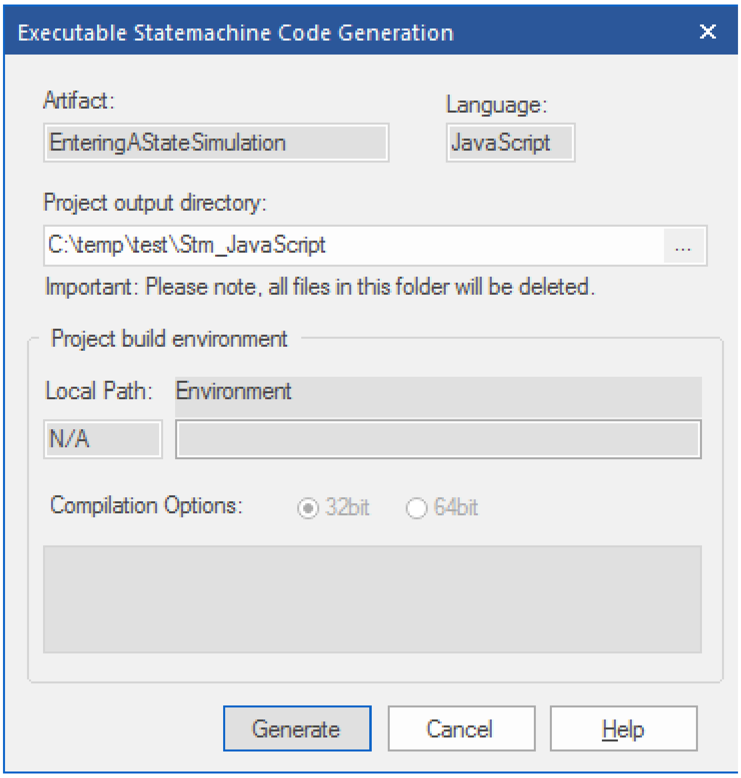
1. On the 'Simulation' page of the Diagram Toolbox, drag the Executable StateMachine icon onto a diagram and create an Artifact named *EnteringAStateSimulation*. Set the language to JavaScript.
2. Ctrl+drag the *MyClass* element from the Browser window onto the *EnteringAStateSimulation* Artifact, select the 'Paste as Property' option and give the Property the name *myClass.*



Code Generation

1. Click on *EnteringAStateSimulation* and select the 'Simulate > Executable States > Statemachine > Generate, Build and Run' ribbon option.
2. Specify a directory for the generated source code.

Note: The contents of this directory will be cleared before generation; make sure you specify a directory that is used only for StateMachine simulation purposes.


Executable Statemachine Code Generation

Artifact: EnteringAStateSimulation

Language: JavaScript

Project output directory: C:\temp\test\Stm_JavaScript

Important: Please note, all files in this folder will be deleted.

Project build environment

Local Path: N/A

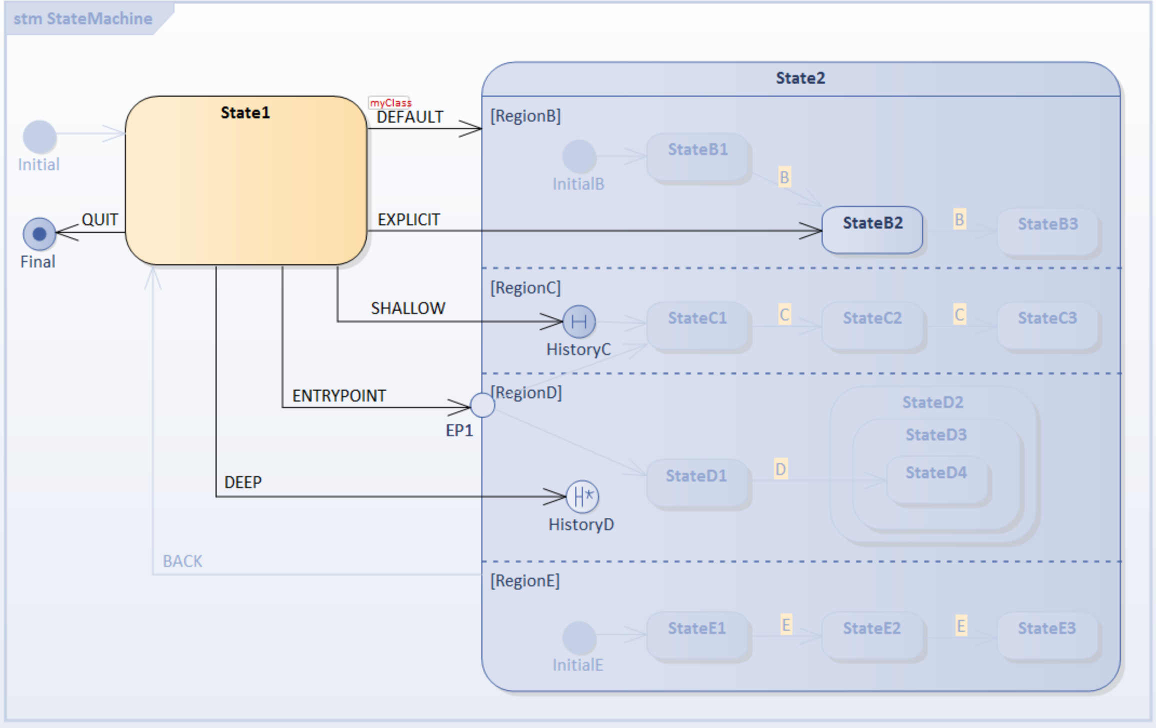
Environment:

Compilation Options: ● 32bit ○ 64bit

Run Simulation

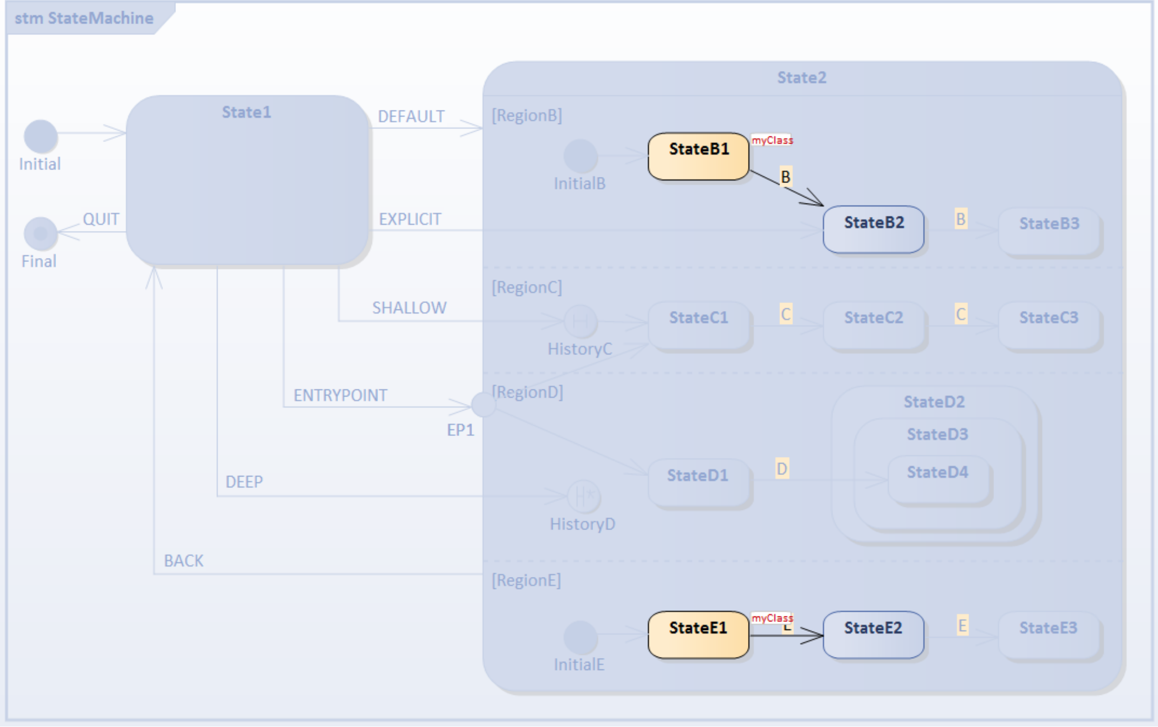
Tips: You can view the execution trace sequence from the Simulation window, which you open by selecting the 'Simulate > Dynamic Simulation > Simulator > Open Simulation Window' ribbon option

When the simulation begins, *State1* is active and the StateMachine is waiting for events.



Open the Simulation Events (Triggers) window using the 'Simulate > Dynamic Simulation > Events' ribbon option.

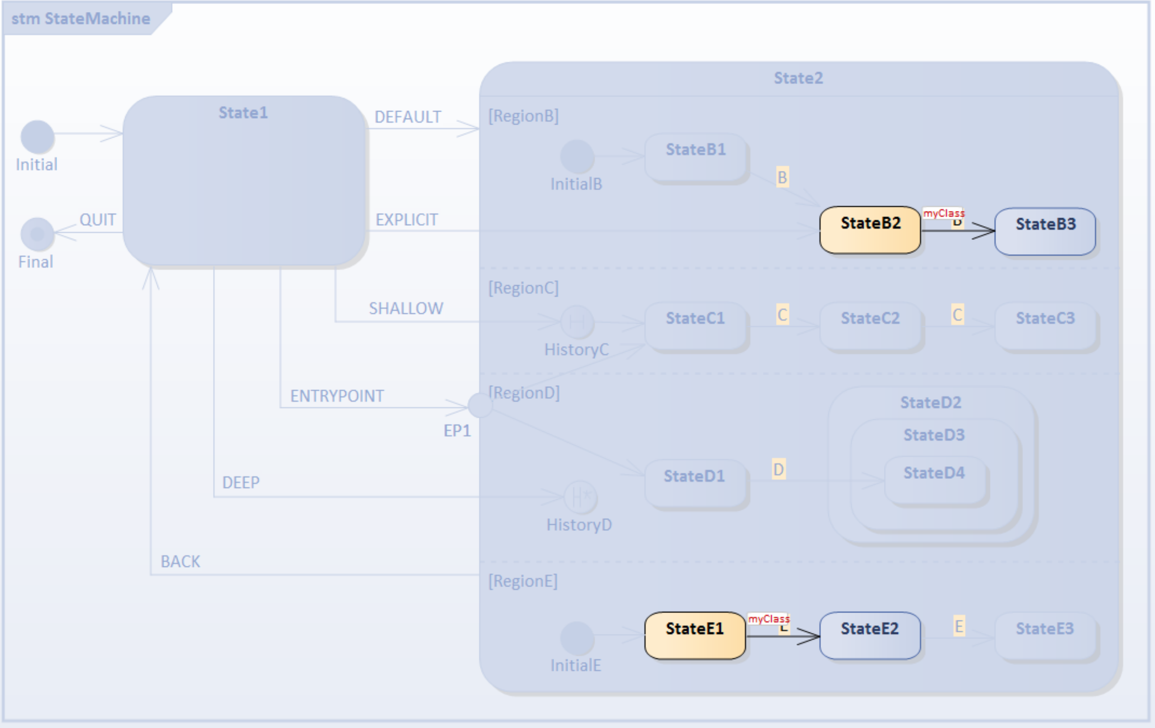
1) Select the Default Entry: Trigger Sequence [DEFAULT].



- *RegionB* is activated because it defines *InitialB*; the transition outgoing from it will be executed, *StateB1* is the active state
- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state
- *RegionC* and *RegionD* are inactive because no Initial Pseudostates were defined

Select the Trigger [BACK] to reset.

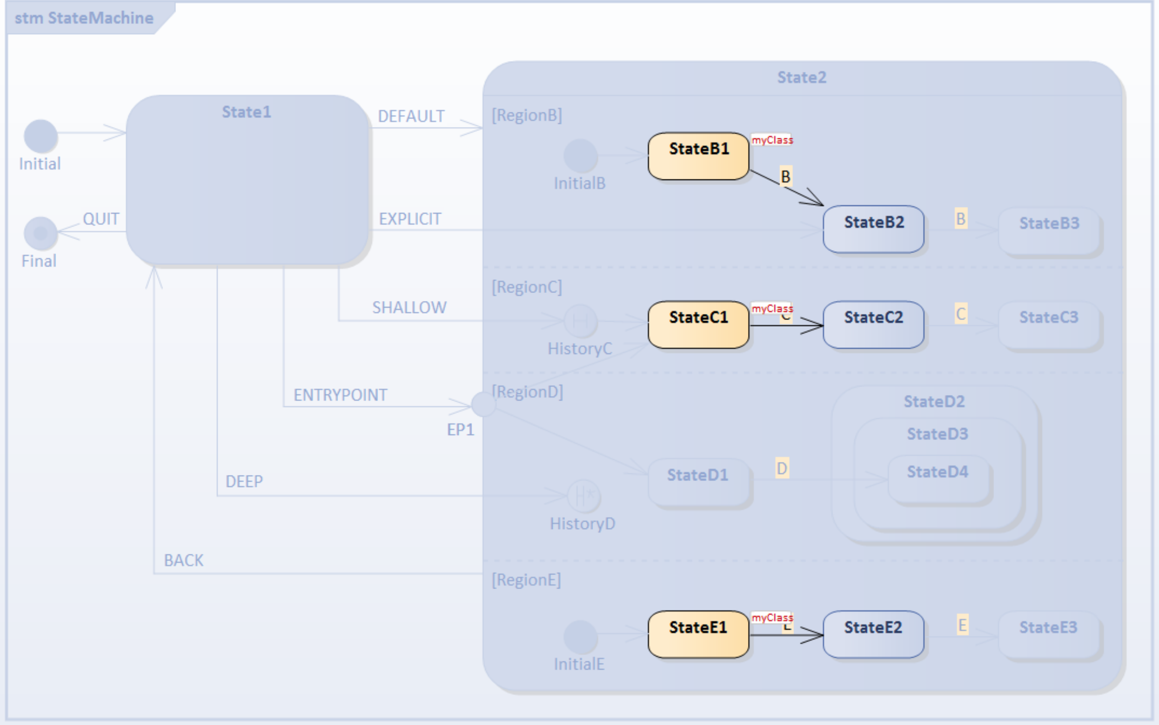
2) Select the Explicit Entry: Trigger Sequence [EXPLICIT].



- *RegionB* is activated because the transition targets the contained vertex *StateB2*
- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state
- *RegionC* and *RegionD* are inactive because no Initial Pseudostates were defined

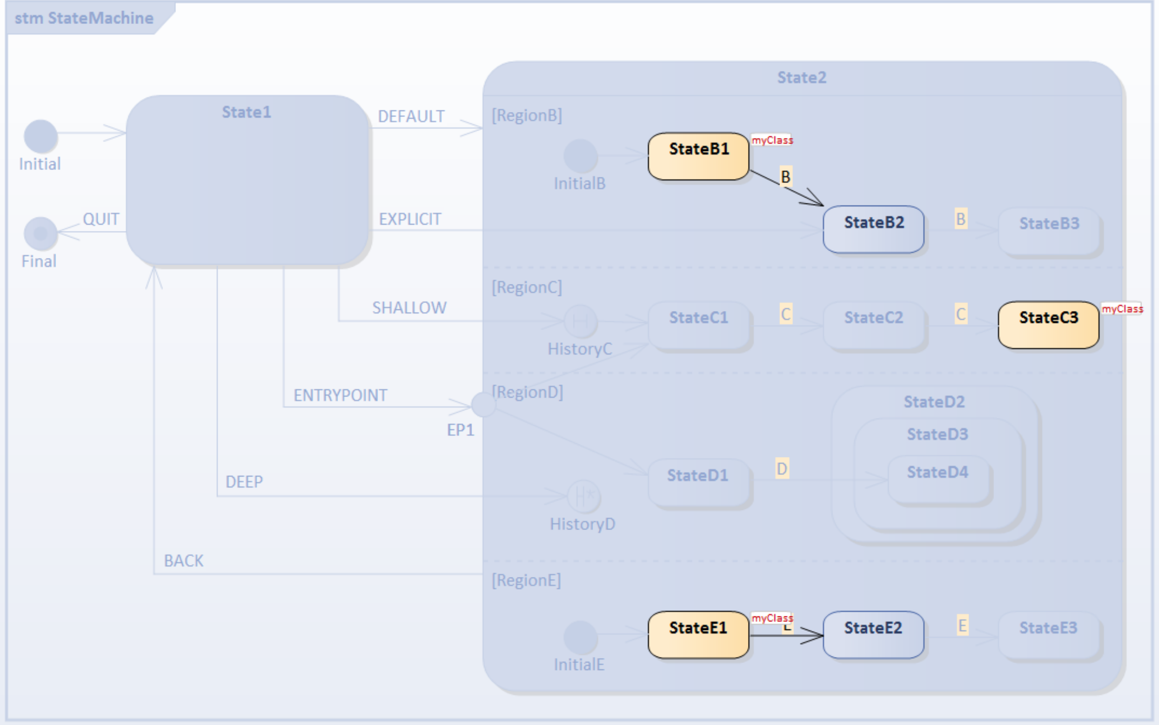
Select the Trigger [BACK] to reset.

3) Select the Default History Transition: Trigger Sequence [SHALLOW].



- *RegionC* is activated because the transition targets the contained vertex *HistoryC*; since this region is entered for the first time (and the History pseudostate has nothing to 'remember'), the transition outgoing from *HistoryC* to *StateC1* is executed
- *RegionB* is activated because it defines *InitialB*; the transition outgoing from it will be executed, *StateB1* is the active state
- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state
- *RegionD* is inactive because no Initial Pseudostate was defined

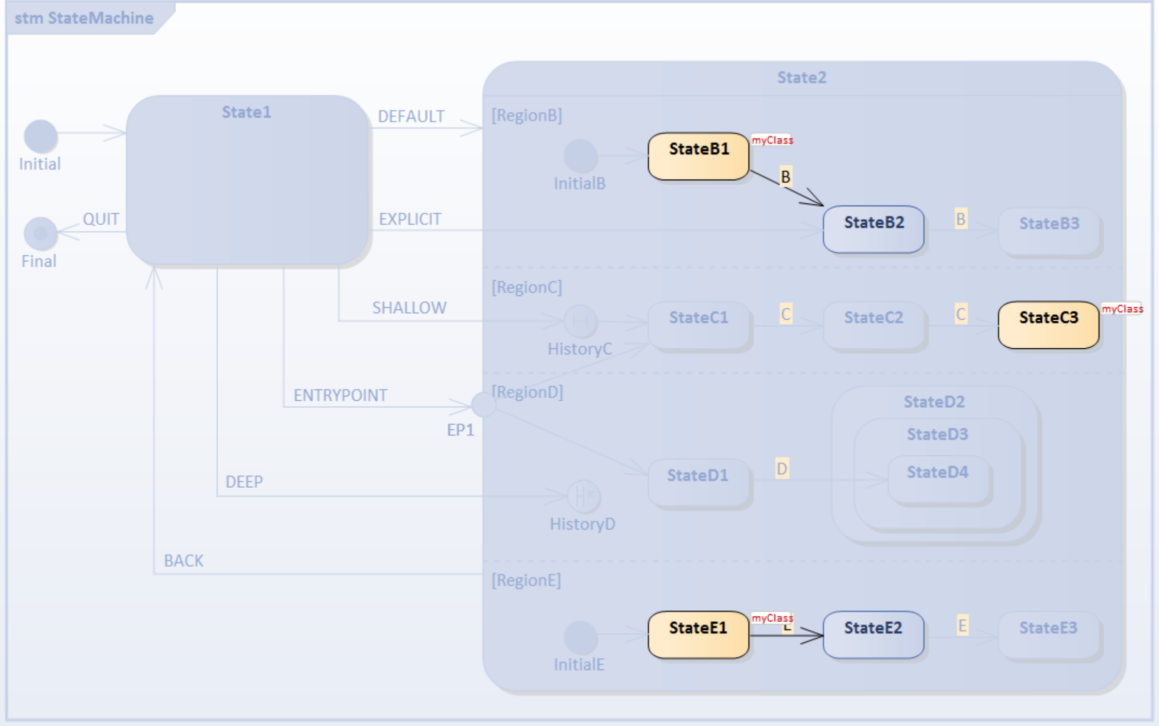
4) Prepare for testing Shallow History Entry: Trigger Sequence [C, C].



- We assume shallow history pseudostate *HistoryC* can remember *StateC3*

Select the Trigger [BACK] to reset.

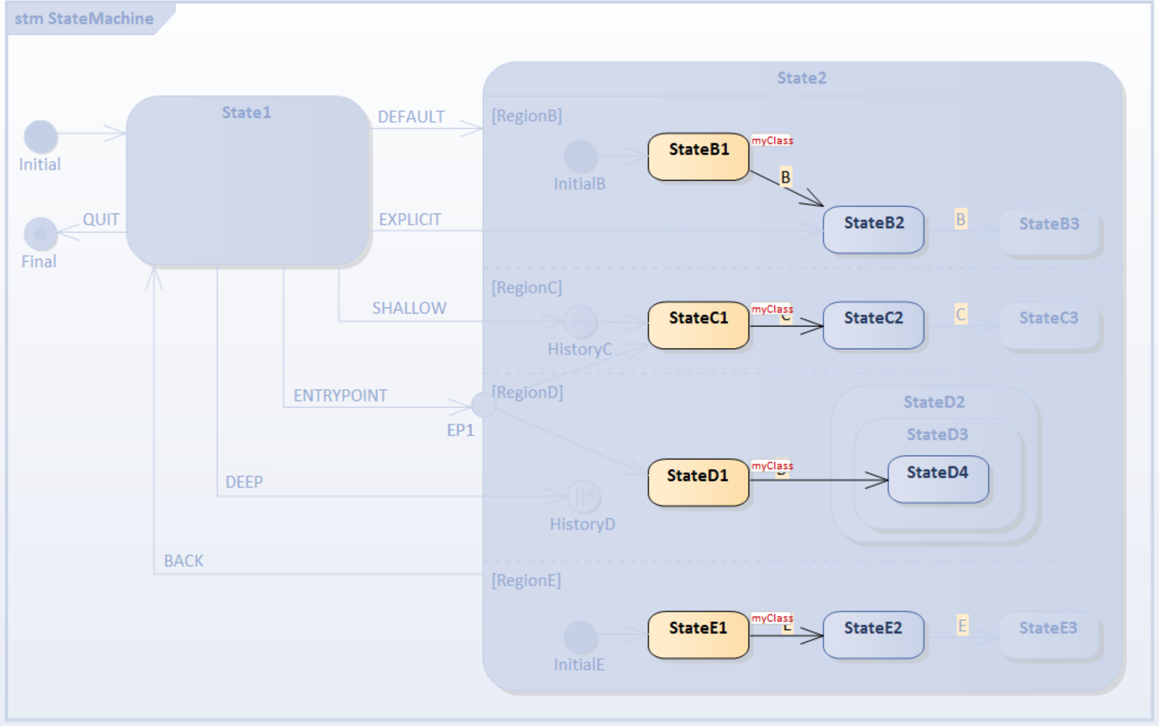
5) Select the Shallow History Entry: Trigger Sequence [SHALLOW].



- For *RegionC*, *StateC3* is activated directly

Select the Trigger [BACK] to reset.

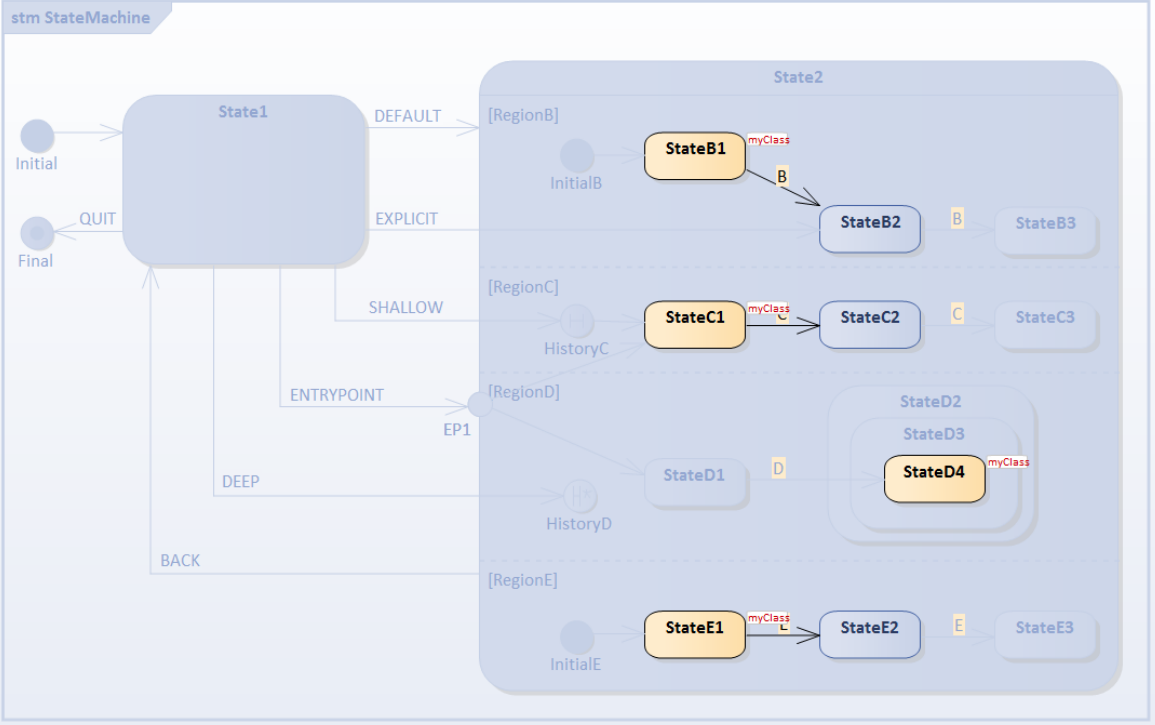
6) Select the Entry Point Entry: Trigger Sequence [ENTRYPOINT].



- *RegionC* is activated because the transition from *EP1* targets the contained *StateC1*
- *RegionD* is activated because the transition from *EP1* targets the contained *StateD1*

- *RegionB* is activated because it defines *InitialB*; the transition outgoing from it will be executed, *StateB1* is the active state
- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state

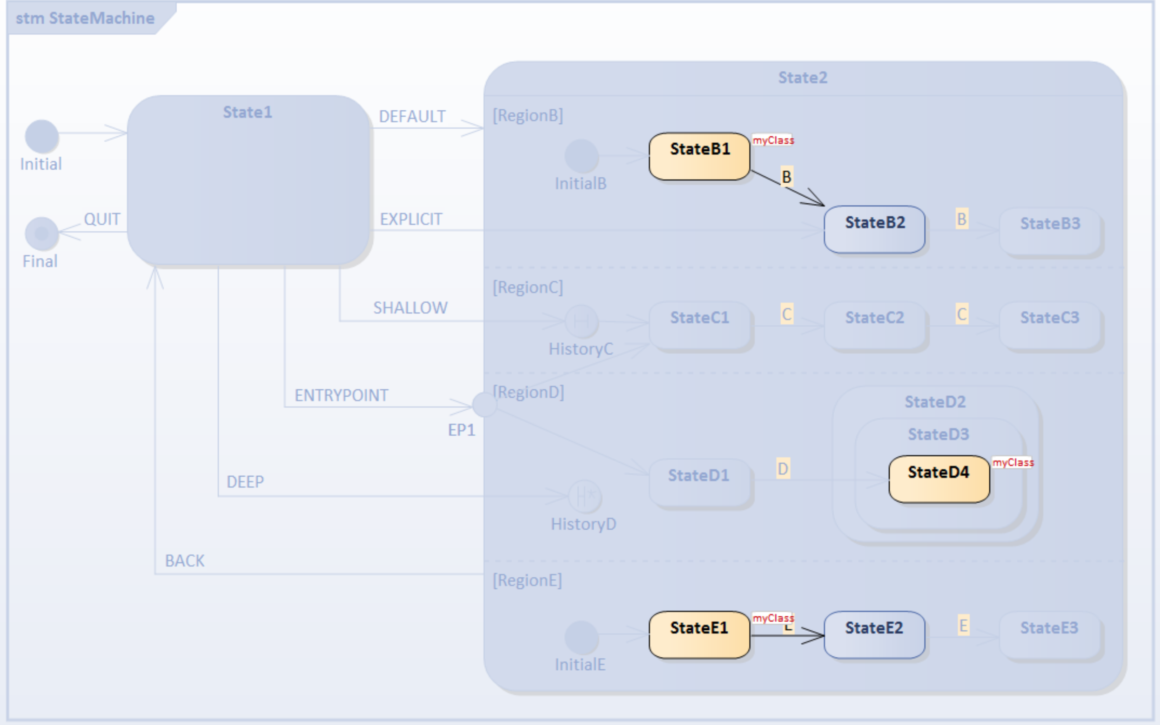
7) Prepare for testing Deep History: Trigger Sequence [D].



- We assume deep history pseudostate *HistoryD* can remember *StateD2*, *StateD3* and *StateD4*

Select the Trigger [BACK] to reset.

8) Select the Deep History Entry: Trigger Sequence [DEEP].



- For *RegionD*, *StateD2*, *StateD3* and *StateD4* are entered; the traces are:

- myClass[MyClass].StateMachine_State1 EXIT
- myClass[MyClass].State1__TO__HistoryD_105793_61752 Effect
- myClass[MyClass].StateMachine_State2 ENTRY
- myClass[MyClass].StateMachine_State2 DO
- myClass[MyClass].InitialE_105787__TO__StateE1_61746 Effect
- myClass[MyClass].StateMachine_State2_StateE1 ENTRY
- myClass[MyClass].StateMachine_State2_StateE1 DO
- myClass[MyClass].InitialB_105785__TO__StateB1_61753 Effect
- myClass[MyClass].StateMachine_State2_StateB1 ENTRY
- myClass[MyClass].StateMachine_State2_StateB1 DO
- myClass[MyClass].StateMachine_State2_StateD2 ENTRY
- myClass[MyClass].StateMachine_State2_StateD2_StateD3 ENTRY
- myClass[MyClass].StateMachine_State2_StateD2_StateD3_StateD4 ENTRY

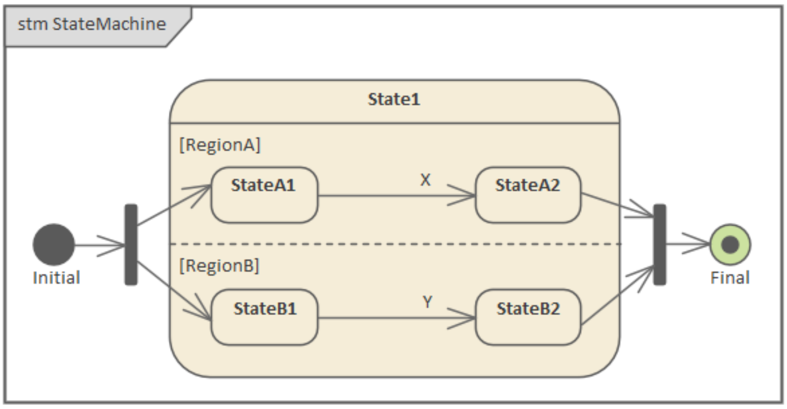
Example: Fork and Join

Fork pseudostates split an incoming Transition into two or more Transitions, terminating in Vertices in orthogonal Regions of a Composite State. The Transitions outgoing from a Fork pseudostate cannot have a guard or a trigger, and the Effect behaviors of the individual outgoing Transitions are, at least conceptually, executed concurrently.

Join pseudostates are a common target Vertex for two or more Transitions originating from Vertices in different orthogonal Regions. Join pseudostates perform a synchronization function, whereby all incoming Transitions have to complete before execution can continue through an outgoing Transition.

In this example, we demonstrate the behavior of a StateMachine with Fork and Join pseudostates.

Modeling StateMachine



Context of StateMachine

- Create a Class element named *MyClass*, which serves as the context of a StateMachine
- Right-click on *MyClass* in the Browser window and select the 'Add | StateMachine' option

StateMachine

- Add an *Initial* Node, a *Fork*, a State named *State1*, a *Join*, and a *Final* to the diagram
- Enlarge *State1*, right-click on it on the diagram and select the 'Advanced | Define Concurrent Substates | Define' option and define *RegionA* and *RegionB*
- In *RegionA*, define *StateA1*, transition to *StateA2*, triggered by event *X*
- In *RegionB*, define *StateB1*, transition to *StateB2*, triggered by event *Y*
- Draw other transitions: *Initial* to *Fork*; *Fork* to *StateA1* and *StateB1*; *StateA2 and StateB2* to *Join*; *Join* to *Final*

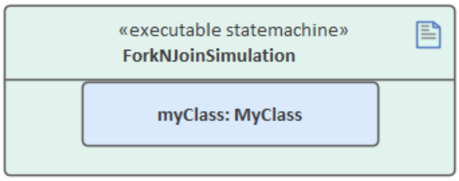
Simulation

Artifact

Enterprise Architect supports C, C++, C#, Java and JavaScript; we will use JavaScript in this example because we don't need to install a compiler (for the other languages, either Visual Studio or JDK are required).

- From the Diagram Toolbox select the 'Simulation' page and drag the Executable StateMachine icon onto the diagram to create an Artifact; name it *ForkNJoinSimulation* and set its 'Language' field to 'JavaScript'
- Ctrl+Drag *MyClass* from the Browser window and drop it on the *ForkNJoinSimulation* Artifact as a Property; give it

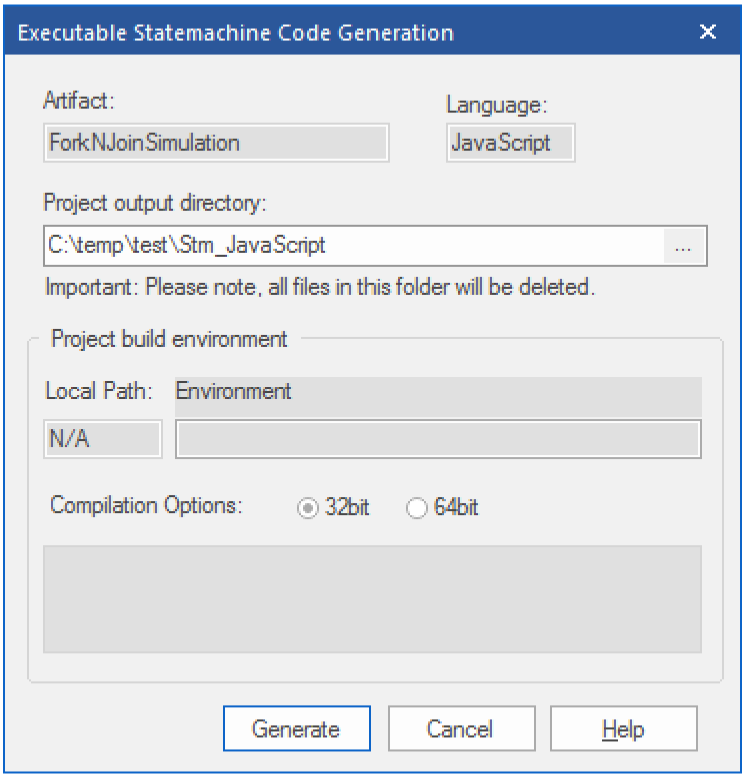
the name *myClass*



Code Generation

- Click on *ForkNJoinSimulation* and select the 'Simulate > Executable States > Statemachine > Generate, Build and Run' ribbon option
- Specify a directory for the generated source code

Note: The contents of this directory will be cleared before generation; make sure you point to a directory that exists only for StateMachine simulation purposes.


Executable Statemachine Code Generation

Artifact: ForkNJoinSimulation

Language: JavaScript

Project output directory: C:\temp\test\Stm_JavaScript

Important: Please note, all files in this folder will be deleted.

Project build environment

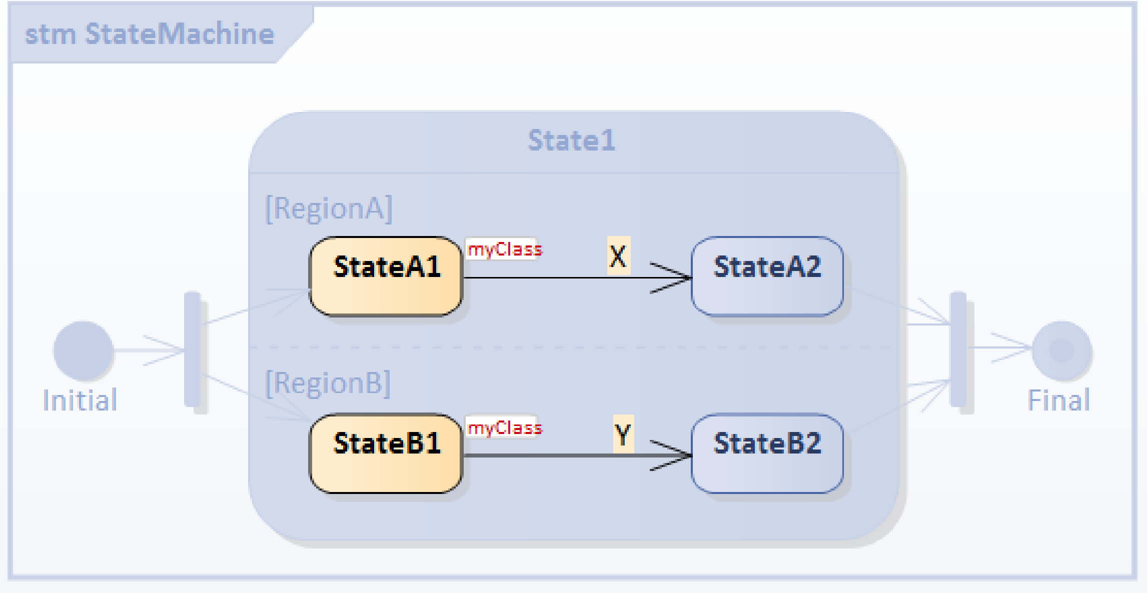
Local Path: N/A

Environment:

Compilation Options: ● 32bit ○ 64bit

Run Simulation

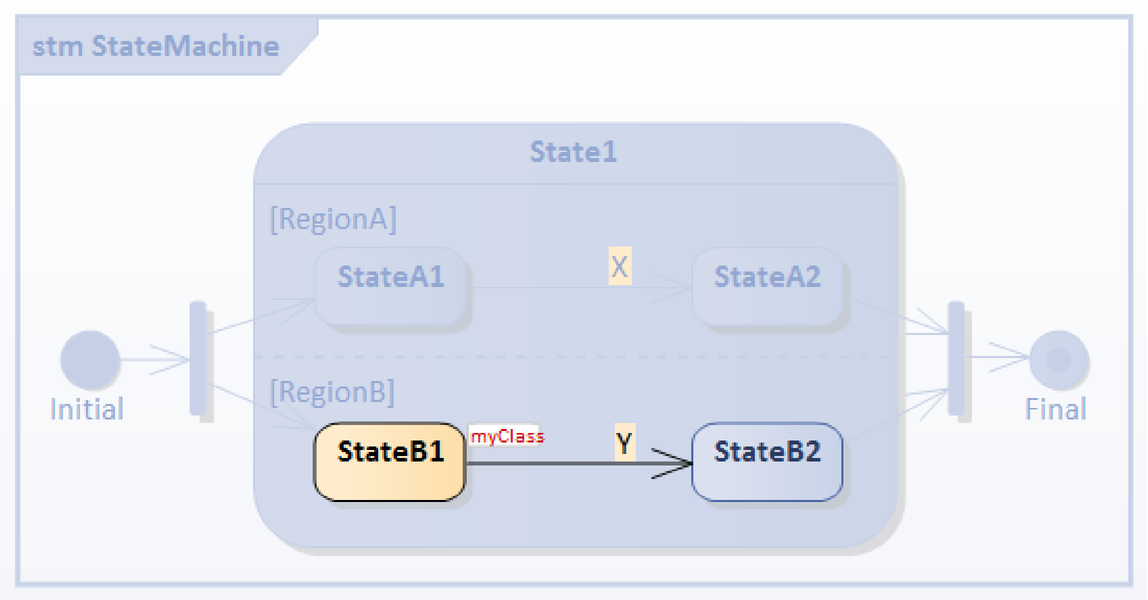
When the simulation is started, *State1*, *StateA1* and *StateB1* are active and the StateMachine is waiting for events.



Select the 'Simulate > Dynamic Simulation > Events' ribbon option to display the Simulation Events window.

On Trigger event *X*, *StateA1* will exit and enter *StateA2*; after the entry and doActivity behavior has run, the completion events of *StateA2* are dispatched and recalled. Then the transition from *StateA2* to the *Join* pseudostate is enabled and traversed.

Note: *Join* must wait for all incoming Transitions to complete before execution can continue through an outgoing Transition. Since the branch from *RegionB* is not complete (because *StateB1* is still active and waiting for triggers) the transition from *Join* to *Final* will not be executed at this moment.



On Trigger event *Y*, *StateB1* will exit and enter *StateB2*; after the entry and doActivity behavior has run, completion events of *StateB2* are dispatched and recalled. Then the transition from *StateB2* to the *Join* pseudostate is enabled and traversed. This satisfies the criteria of all the incoming transitions of *Join* having completed, so the transition from *Join* to *Final* is executed. Simulation has ended.

Tips: You can view the execution trace sequence from the Simulation window ('Simulate > Dynamic Simulation > Simulator > Open Simulation Window' ribbon option).

```
myClass[MyClass].Initial_82285__TO__fork_82286_82286_61745 Effect
```

```
myClass[MyClass].StateMachine_State1 ENTRY
myClass[MyClass].StateMachine_State1 DO
myClass[MyClass].fork_82286_82286__TO__StateA1_57125 Effect
myClass[MyClass].StateMachine_State1_StateA1 ENTRY
myClass[MyClass].StateMachine_State1_StateA1 DO
myClass[MyClass].fork_82286_82286__TO__StateB1_57126 Effect
myClass[MyClass].StateMachine_State1_StateB1 ENTRY
myClass[MyClass].StateMachine_State1_StateB1 DO
Trigger X
myClass[MyClass].StateMachine_State1_StateA1 EXIT
myClass[MyClass].StateA1__TO__StateA2_57135 Effect
myClass[MyClass].StateMachine_State1_StateA2 ENTRY
myClass[MyClass].StateMachine_State1_StateA2 DO
myClass[MyClass].StateMachine_State1_StateA2 EXIT
myClass[MyClass].StateA2__TO__join_82287_82287_57134 Effect
Trigger Y
myClass[MyClass].StateMachine_State1_StateB1 EXIT
myClass[MyClass].StateB1__TO__StateB2_57133 Effect
myClass[MyClass].StateMachine_State1_StateB2 ENTRY
myClass[MyClass].StateMachine_State1_StateB2 DO
myClass[MyClass].StateMachine_State1_StateB2 EXIT
myClass[MyClass].StateB2__TO__join_82287_82287_57132 Effect
myClass[MyClass].StateMachine_State1 EXIT
myClass[MyClass].join_82287_82287__TO__Final_105754_57130 Effect
```

Example: Deferred Event Pattern

Enterprise Architect supports the Deferred Event Pattern.

To create a Deferred Event in a State:

1. Create a self transition for the State.
2. Change the 'kind' of the transition to 'internal'.
3. Specify the Trigger to be the event you want to defer.
4. In the 'Effect' field, type 'defer();'.

To Simulate:

1. Select 'Simulate > Dynamic Simulation > Simulator > Open Simulation Window'. Also select 'Simulate > Dynamic Simulation > Events' to open the Simulation Events window.
2. The Simulator Events window helps you to trigger events; double-click on a trigger in the 'Waiting Triggers' column.
3. The Simulation window shows the execution in text. You can type 'dump' in the Simulator command line to show how many events are deferred in the queue; the output might resemble this:

```
24850060]      Event Pool: [NEW,NEW,NEW,NEW,NEW,]
```

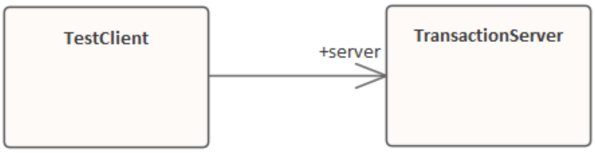
Deferred Event Example

This example shows a model using Deferred Events, and the Simulation Events window showing all available Events.

We firstly set up the contexts (the Class elements containing the StateMachines), simulate them in a simple context and raise the event from outside it; then simulate in a client-server context with the Send event mechanism.

Create Context and StateMachine

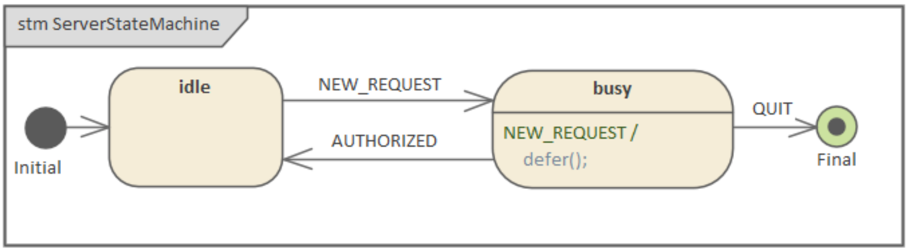
Create the server context



Create a Class diagram and:

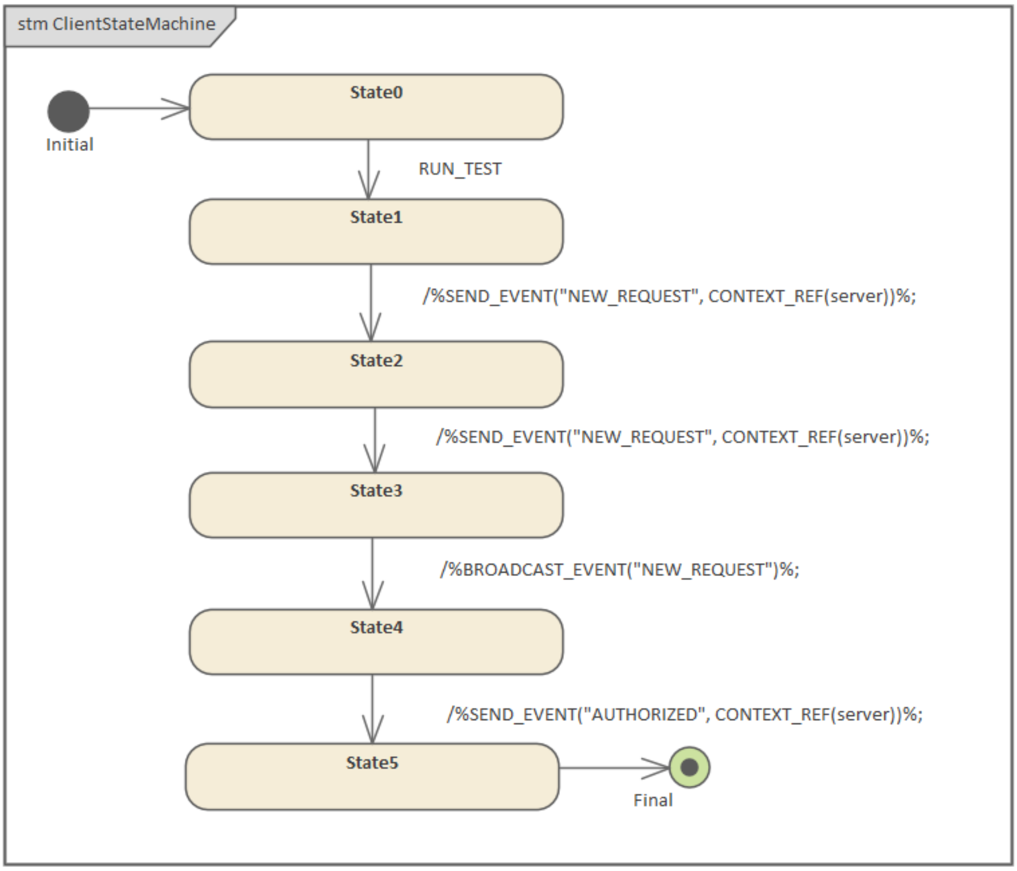
1. A Class element *TransactionServer*, to which you add a StateMachine *ServerStateMachine*.
2. A Class element *TestClient*, to which you add a StateMachine *ClientStateMachine*.
3. An Association from *TestClient* to *TransactionServer*, with the target role named *server.*

Modeling for *ServerStateMachine*



1. Add an Initial Node *Initial* to the StateMachine diagram, and transition to a State *idle*.
2. Transition (with event NEW_REQUEST as Trigger) to a State *busy*.
3. Transition (with event QUIT as Trigger) to a Final State *Final.*
4. Transition (with event AUTHORIZED as Trigger) to *idle*.
5. Transition (with event NEW_REQUEST as Trigger and *defer();* as effect) to *busy*

Modeling for *ClientStateMachine*

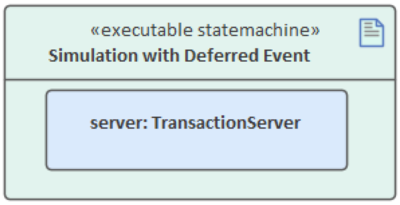

stm ClientStateMachine
Initial
State0
RUN_TEST
State1
/%SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;
State2
/%SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;
State3
/%BROADCAST_EVENT("NEW_REQUEST")%;
State4
/%SEND_EVENT("AUTHORIZED", CONTEXT_REF(server))%;
State5
Final

1. Add an Initial Node *Initial* to the StateMachine diagram, and transition to a State *State0*.
2. Transition (with event RUN_TEST as trigger) to a State *State1*.
3. Transition (with effect: %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;) to a State *State2.*
4. Transition (with effect: %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;) to a State *State3*.
5. Transition (with effect: %BROADCAST_EVENT("NEW_REQUEST")%;) to a State *State4*.
6. Transition (with effect: %SEND_EVENT("AUTHORIZED", CONTEXT_REF(server))%;) to a State *State5*.

7. Transition to a Final State *Final.*

Simulation in a simple context

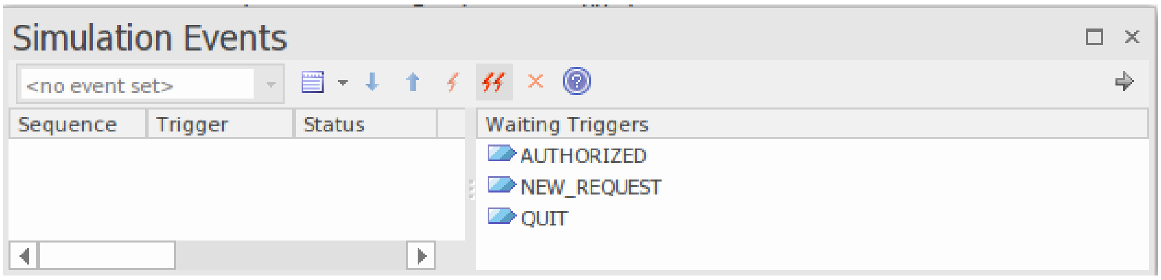
Create the Simulation Artifact


«executable statemachine»
Simulation with Deferred Event
server: TransactionServer

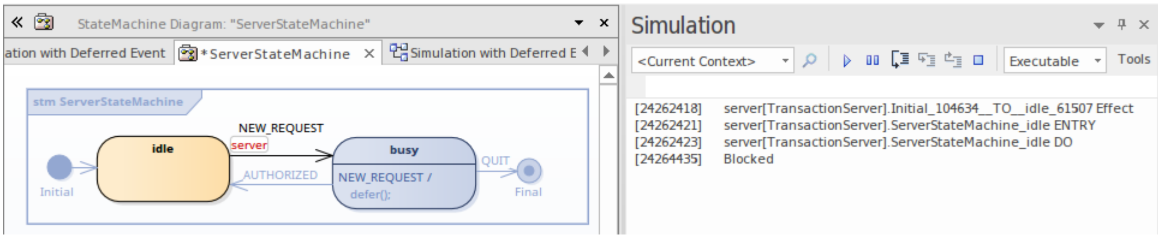
1. Create an Executable StateMachine Artifact with the name *Simulation with Deferred Event* and the 'Language' field set to *JavaScript.*
2. Enlarge it, then Ctrl+drag the *TransactionServer* element onto the Artifact and paste it as a property with the name *server.*

Run the Simulation

1. Select the Artifact, then select the 'Simulate > Executable States > Statemachine > Generate, Build and Run' option, and specify a directory for your code (Note: all the files in the directory will be deleted before simulation starts).
2. Click on the Generate button.
3. Select the 'Simulate > Dynamic Simulation > Events' option to open the Simulation Event window.

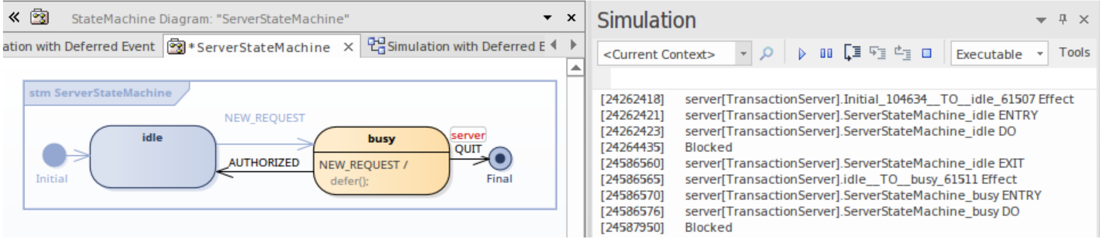


When simulation starts, *idle* will be the active state.



```
[24262418]    server[TransactionServer].Initial_104634__TO__idle_61507 Effect
[24262421]    server[TransactionServer].ServerStateMachine_idle ENTRY
[24262423]    server[TransactionServer].ServerStateMachine_idle DO
[24264435]    Blocked
```

1. Double-click on NEW_REQUEST in the Simulation Event window to execute it as the Trigger; *idle* is exited and *busy* is activated.

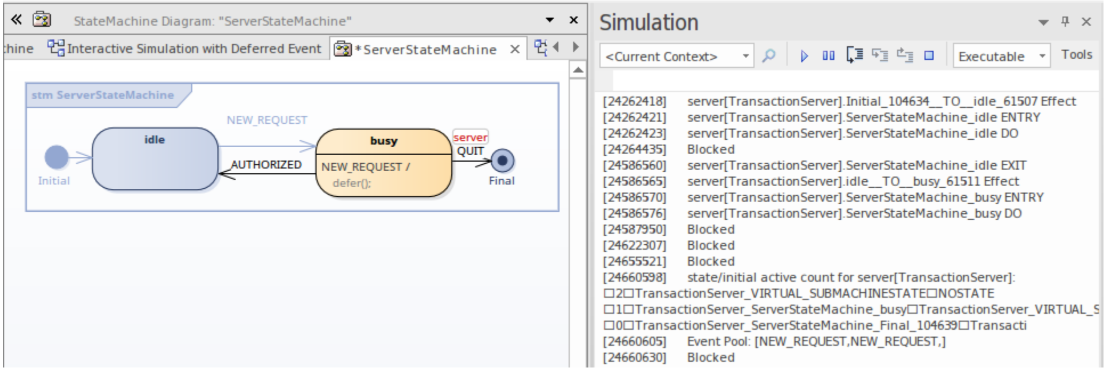


```
[24262418]    server[TransactionServer].Initial_104634__TO__idle_61507 Effect
[24262421]    server[TransactionServer].ServerStateMachine_idle ENTRY
[24262423]    server[TransactionServer].ServerStateMachine_idle DO
[24264435]    Blocked
[24586560]    server[TransactionServer].ServerStateMachine_idle EXIT
[24586565]    server[TransactionServer].idle__TO__busy_61511 Effect
[24586570]    server[TransactionServer].ServerStateMachine_busy ENTRY
[24586576]    server[TransactionServer].ServerStateMachine_busy DO
[24587950]    Blocked
```

2. Double-click on NEW_REQUEST in the Simulation Event window to execute it again as the Trigger; *busy* remains

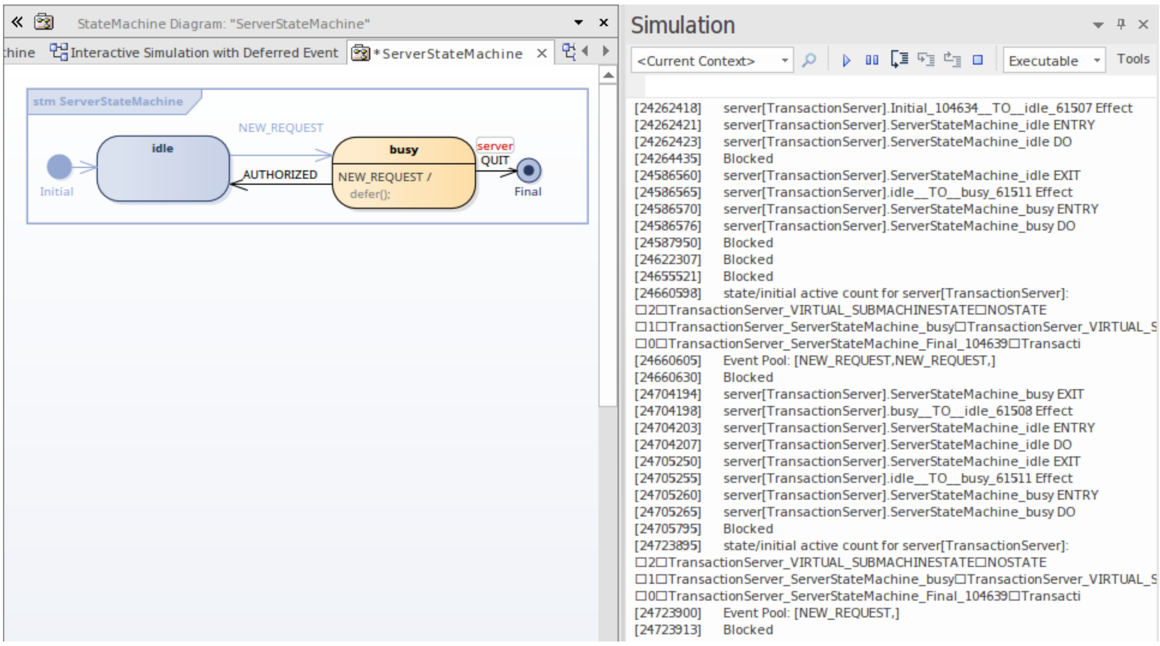
activated, and an instance of NEW_REQUEST is appended in the Event Pool.

3. Double-click on NEW_REQUEST in the Simulation Event window to execute it a third time as the Trigger; *busy* remains activated, and an instance of NEW_REQUEST is appended in the Event Pool.
4. Type *dump* in the Simulation window command line; notice that the event pool has two instances of NEW_REQUEST.



```
[24262418]	server[TransactionServer].Initial_104634__TO__idle_61507 Effect
[24262421]	server[TransactionServer].ServerStateMachine_idle ENTRY
[24262423]	server[TransactionServer].ServerStateMachine_idle DO
[24264435]	Blocked
[24586560]	server[TransactionServer].ServerStateMachine_idle EXIT
[24586565]	server[TransactionServer].idle__TO__busy_61511 Effect
[24586570]	server[TransactionServer].ServerStateMachine_busy ENTRY
[24586576]	server[TransactionServer].ServerStateMachine_busy DO
[24587950]	Blocked
[24622307]	Blocked
[24655521]	Blocked
[24660598]	state/initial active count for server[TransactionServer]:
□2□TransactionServer_VIRTUAL_SUBMACHINESTATE□NOSTATE
□1□TransactionServer_ServerStateMachine_busy□TransactionServer_VIRTUAL_S
□0□TransactionServer_ServerStateMachine_Final_104639□Transacti
[24660605]	Event Pool: [NEW_REQUEST,NEW_REQUEST,]
[24660630]	Blocked
```

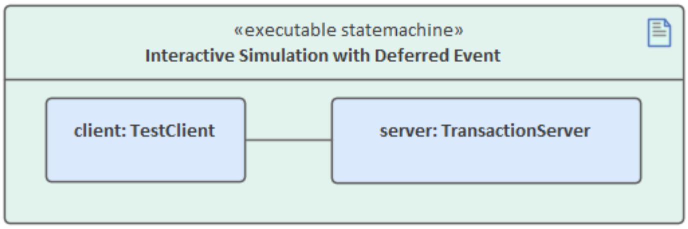
5. Double-click on AUTHORIZED in the Simulation Event window to execute it as the Trigger; these actions take place:
   - *busy* is exited and *idle* becomes active
   - a NEW_REQUEST event is retrieved from the pool, *idle* is exited and *busy* becomes active
6. Type *dump* in the Simulation window command line; there is now only one instance of NEW_REQUEST in the Event Pool.



```
[24262418]	server[TransactionServer].Initial_104634__TO__idle_61507 Effect
[24262421]	server[TransactionServer].ServerStateMachine_idle ENTRY
[24262423]	server[TransactionServer].ServerStateMachine_idle DO
[24264435]	Blocked
[24586560]	server[TransactionServer].ServerStateMachine_idle EXIT
[24586565]	server[TransactionServer].idle__TO__busy_61511 Effect
[24586570]	server[TransactionServer].ServerStateMachine_busy ENTRY
[24586576]	server[TransactionServer].ServerStateMachine_busy DO
[24587950]	Blocked
[24622307]	Blocked
[24655521]	Blocked
[24660598]	state/initial active count for server[TransactionServer]:
□2□TransactionServer_VIRTUAL_SUBMACHINESTATE□NOSTATE
□1□TransactionServer_ServerStateMachine_busy□TransactionServer_VIRTUAL_S
□0□TransactionServer_ServerStateMachine_Final_104639□Transacti
[24660605]	Event Pool: [NEW_REQUEST,NEW_REQUEST,]
[24660630]	Blocked
[24704194]	server[TransactionServer].ServerStateMachine_busy EXIT
[24704198]	server[TransactionServer].busy__TO__idle_61508 Effect
[24704203]	server[TransactionServer].ServerStateMachine_idle ENTRY
[24704207]	server[TransactionServer].ServerStateMachine_idle DO
[24705250]	server[TransactionServer].ServerStateMachine_idle EXIT
[24705255]	server[TransactionServer].idle__TO__busy_61511 Effect
[24705260]	server[TransactionServer].ServerStateMachine_busy ENTRY
[24705265]	server[TransactionServer].ServerStateMachine_busy DO
[24705795]	Blocked
[24723895]	state/initial active count for server[TransactionServer]:
□2□TransactionServer_VIRTUAL_SUBMACHINESTATE□NOSTATE
□1□TransactionServer_ServerStateMachine_busy□TransactionServer_VIRTUAL_S
□0□TransactionServer_ServerStateMachine_Final_104639□Transacti
[24723900]	Event Pool: [NEW_REQUEST,]
[24723913]	Blocked
```

Interactive simulation via Send/Broadcast Event

Create the Simulation Artifact

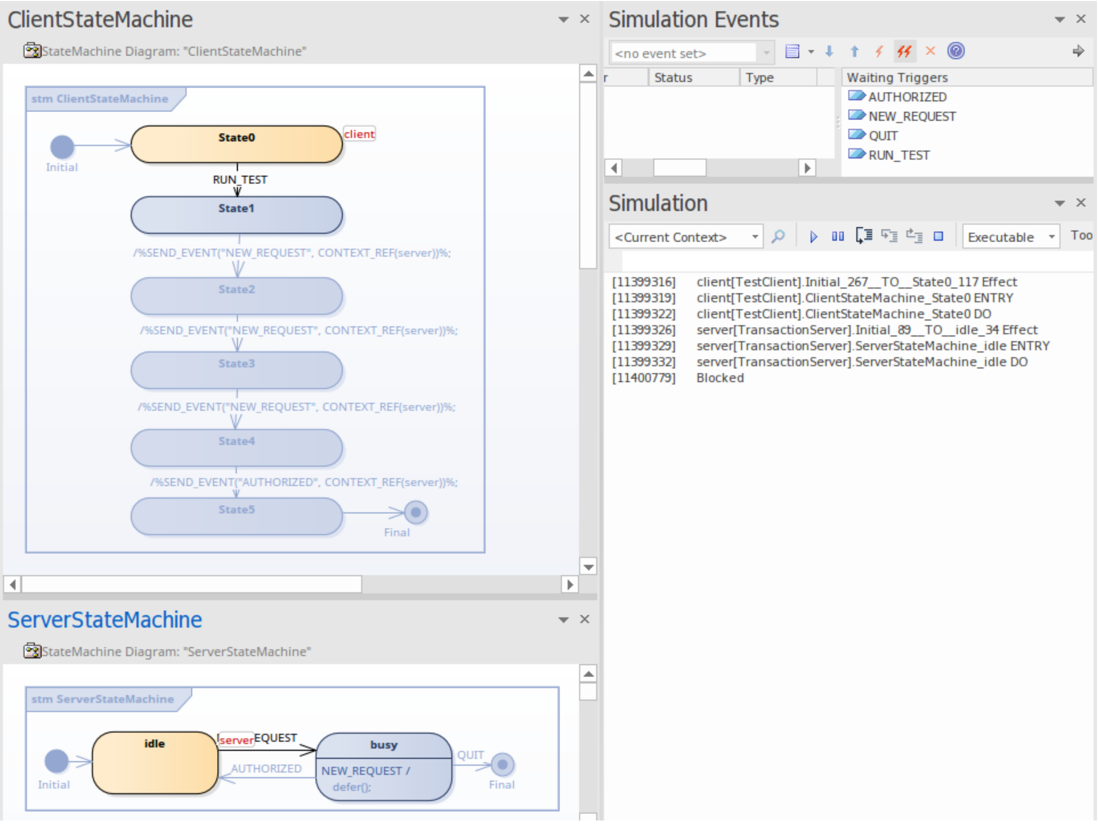

«executable statemachine»
Interactive Simulation with Deferred Event
client: TestClient
server: TransactionServer

1. Create an Executable StateMachine Artifact with the name *Interactive Simulation with Deferred Event* and the 'Language' field set to *JavaScript;* enlarge the element.
2. Ctrl+Drag the *TransactionServer* element onto the Artifact, and paste it as a property with the name *server.*
3. Ctrl+Drag the *TestClient* element onto the Artifact, and paste it as a property with the name *client*.
4. Create a connector from *client* to *server*.
5. Click on the connector and press Ctrl+L to select the association from the *TestClient* element to the *TransactionServer* element.

Run Interactive Simulation

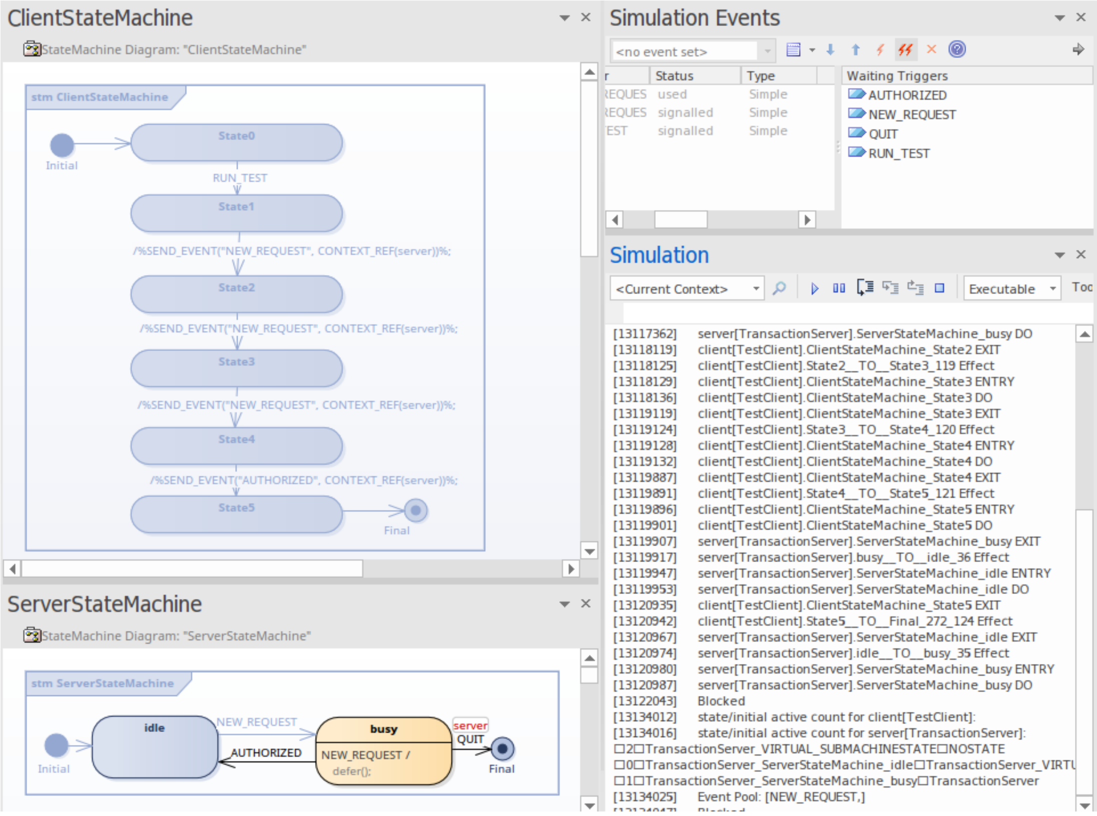
1. Launch the simulation in the same way as for the simple context.

   Once the simulation has started, the *client* remains at *State0* and the *server* remains at *idle*.



```
[11399316]    client[TestClient].Initial_267__TO__State0_117 Effect
[11399319]    client[TestClient].ClientStateMachine_State0 ENTRY
[11399322]    client[TestClient].ClientStateMachine_State0 DO
[11399326]    server[TransactionServer].Initial_89__TO__idle_34 Effect
[11399329]    server[TransactionServer].ServerStateMachine_idle ENTRY
[11399332]    server[TransactionServer].ServerStateMachine_idle DO
[11400779]    Blocked
```

2. Double-click on RUN_TEST in the Simulation Event window to trigger it. The event NEW_REQUEST will be triggered three times (by SEND_EVENT and BROADCAST_EVENT) and AUTHORIZED will be triggered once by SEND_EVENT.


ClientStateMachine
StateMachine Diagram: "ClientStateMachine"
stm ClientStateMachine
Initial
State0
RUN_TEST
State1
/%SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;
State2
/%SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;
State3
/%SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;
State4
/%SEND_EVENT("AUTHORIZED", CONTEXT_REF(server))%;
State5
Final
ServerStateMachine
StateMachine Diagram: "ServerStateMachine"
stm ServerStateMachine
Initial
idle
NEW_REQUEST
AUTHORIZED
busy
NEW_REQUEST / defer();
server
QUIT
Final
Simulation Events
<no event set>
Simulation
<Current Context>
Executable
Too

r	Status	Type
REQUES	used	Simple
REQUES	signalled	Simple
EST	signalled	Simple

Waiting Triggers

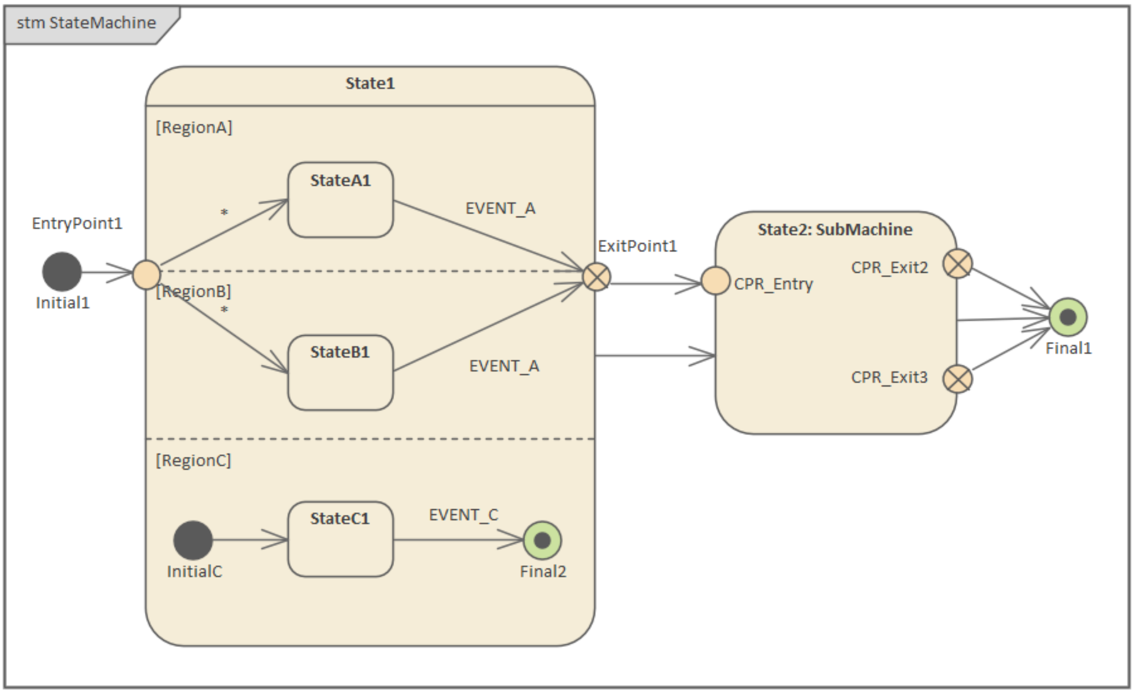
- AUTHORIZED
- NEW_REQUEST
- QUIT
- RUN_TEST

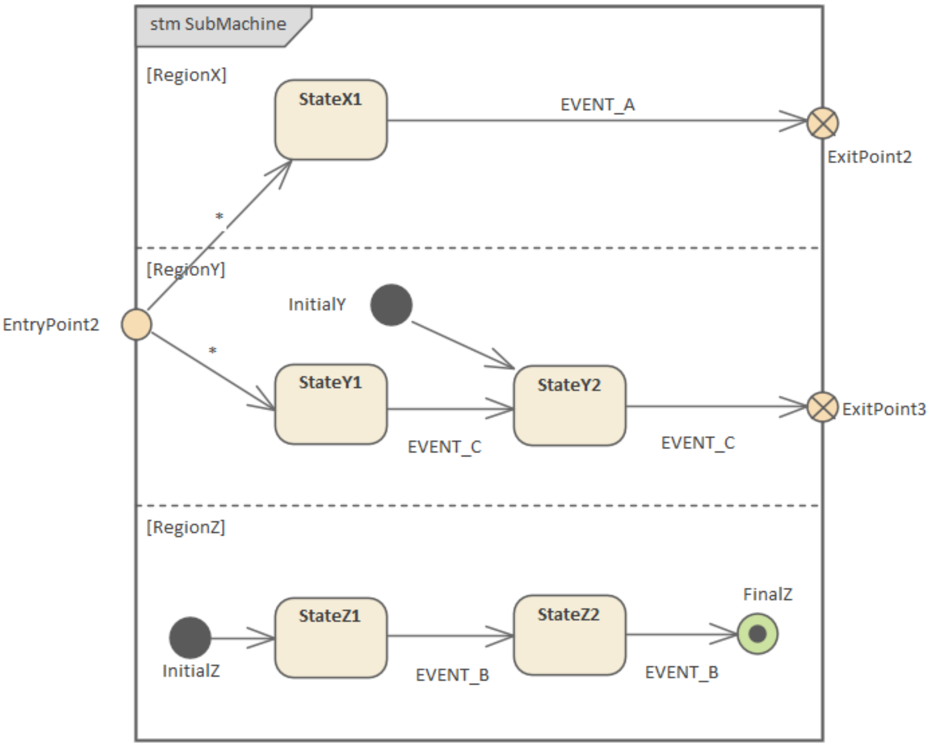
```
[13117362]    server[TransactionServer].ServerStateMachine_busy DO
[13118119]    client[TestClient].ClientStateMachine_State2 EXIT
[13118125]    client[TestClient].State2__TO__State3_119 Effect
[13118129]    client[TestClient].ClientStateMachine_State3 ENTRY
[13118136]    client[TestClient].ClientStateMachine_State3 DO
[13119119]    client[TestClient].ClientStateMachine_State3 EXIT
[13119124]    client[TestClient].State3__TO__State4_120 Effect
[13119128]    client[TestClient].ClientStateMachine_State4 ENTRY
[13119132]    client[TestClient].ClientStateMachine_State4 DO
[13119887]    client[TestClient].ClientStateMachine_State4 EXIT
[13119891]    client[TestClient].State4__TO__State5_121 Effect
[13119896]    client[TestClient].ClientStateMachine_State5 ENTRY
[13119901]    client[TestClient].ClientStateMachine_State5 DO
[13119907]    server[TransactionServer].ServerStateMachine_busy EXIT
[13119917]    server[TransactionServer].busy__TO__idle_36 Effect
[13119947]    server[TransactionServer].ServerStateMachine_idle ENTRY
[13119953]    server[TransactionServer].ServerStateMachine_idle DO
[13120935]    client[TestClient].ClientStateMachine_State5 EXIT
[13120942]    client[TestClient].State5__TO__Final_272_124 Effect
[13120967]    server[TransactionServer].ServerStateMachine_idle EXIT
[13120974]    server[TransactionServer].idle__TO__busy_35 Effect
[13120980]    server[TransactionServer].ServerStateMachine_busy ENTRY
[13120987]    server[TransactionServer].ServerStateMachine_busy DO
[13122043]    Blocked
[13134012]    state/initial active count for client[TestClient]:
[13134016]    state/initial active count for server[TransactionServer]:
□2□TransactionServer_VIRTUAL_SUBMACHINESTATE□NOSTATE
□0□TransactionServer_ServerStateMachine_idle□TransactionServer_VIRTU
□1□TransactionServer_ServerStateMachine_busy□TransactionServer
[13134025]    Event Pool: [NEW_REQUEST,]
[13134047]    Blocked
```

Type *dump* in the Simulation window command line, There is one instance of NEW_REQUEST left in the Event Pool. The result matches our manual triggering test.

Example: Entry and Exit Points (Connection Point References)

Enterprise Architect provides support for Entry and Exit points, and for Connection Point References. In this example, we define two StateMachines for *MyClass - StateMachine* and *SubMachine.*





- *State1* is a Composite State (also called an Orthogonal State because it has multiple Regions) with three Regions:

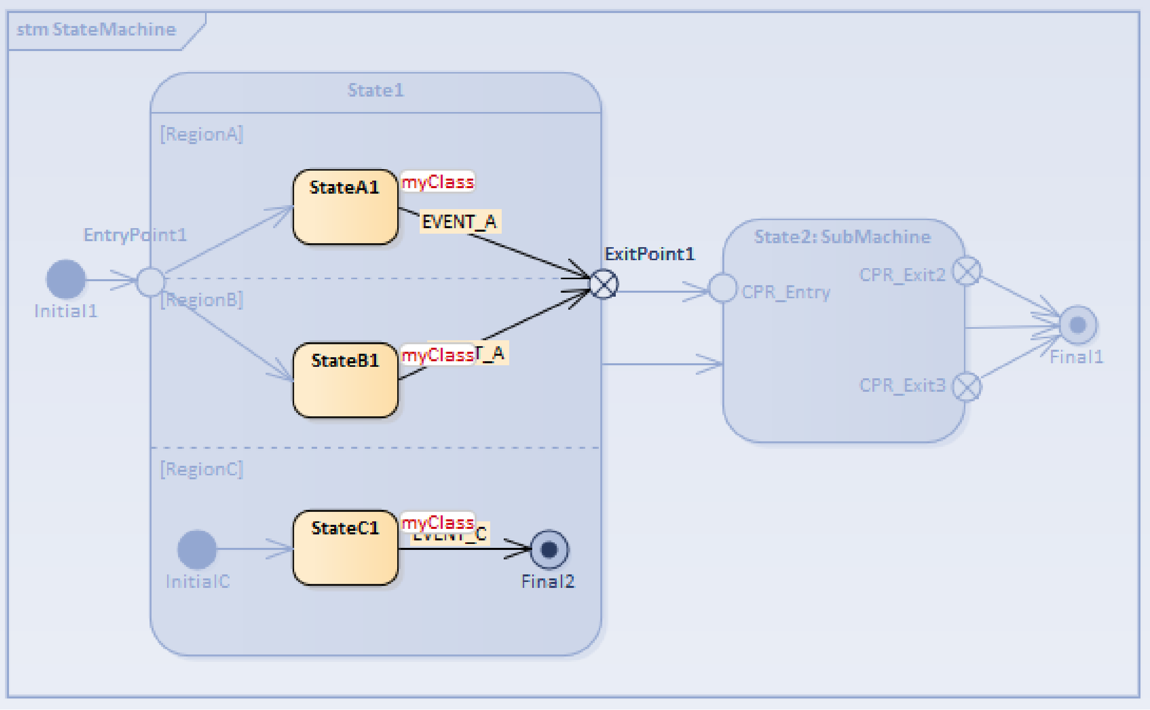
RegionA, *RegionB* and *RegionC*

- *State2* is a SubMachine State calling *SubMachine*, which has three Regions: *RegionX*, *RegionY*, and *RegionZ*
- *EntryPoint1* is defined on *State1* to activate two of the three Regions; *EntryPoint2* is defined on *SubMachine* to activate two of the three Regions
- *ExitPoint1* is defined on *State1*; two exit points *ExitPoint2* and *ExitPoint3* are defined on *SubMachine*
- Connection Point References are defined on *State2* and bind to the Entry/Exit Points of the typing SubMachine
- Initial nodes are defined to demonstrate default activation of the Regions

Entering a State: Entry Point Entry

EntryPoint1 on *State1*

When a Transition targeted on *EntryPoint1* is enabled, *State1* is activated followed by the contained Regions.

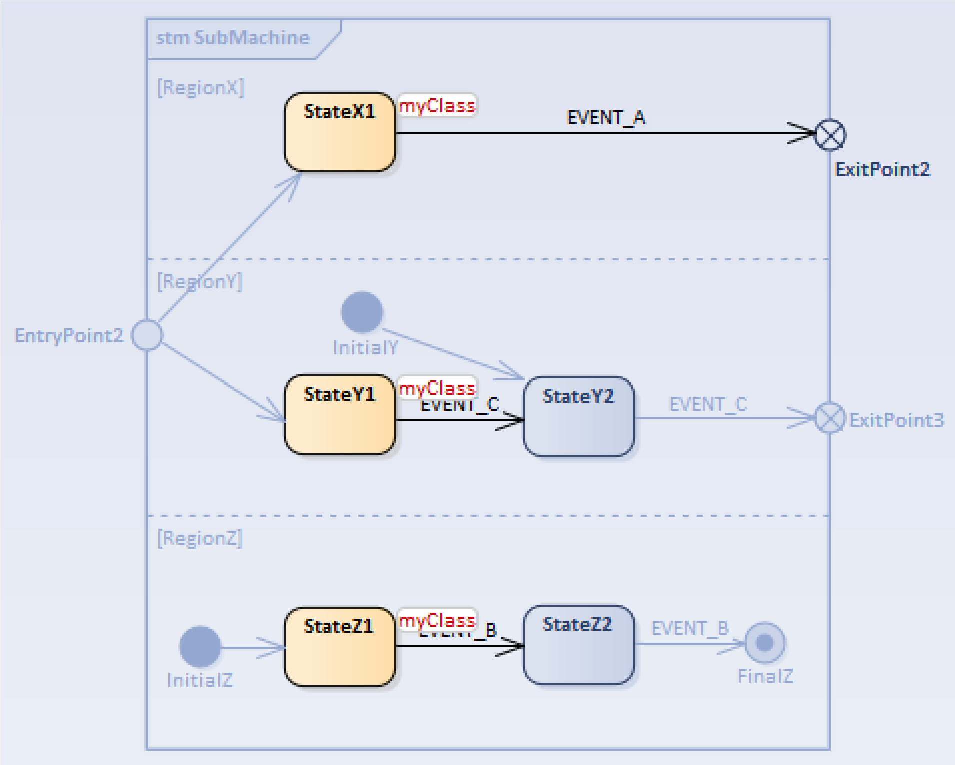


- Explicit activation occurs for *RegionA* and *RegionB*, because each of them is entered by a Transition terminating on one of the Region's contained Vertices
- Default activation occurs for *RegionC*, because it defines an Initial pseudostate *InitialC* and the Transition originating from the *InitialC* to *StateC1* starts execution

EntryPoint2 on *SubMachine*

The Trigger Sequence to be simulated is: [EVENT_C, EVENT_A].

When a Transition targeted on Connection Point Reference *CPR_Entry* on *State2* is enabled, *State2* is activated, followed by the SubMachine's activation through the binding entry points.



- Explicit activation occurs for *RegionX* and *RegionY*, because each of them is entered by a Transition terminating on one of the Region's contained Vertices - *StateX1* in *RegionX*, *StateY1* in *RegionY*
- Default activation occurs for *RegionZ*, because it defines an Initial pseudostate *InitialZ* and the Transition originating from *InitialZ* to *StateZ1* starts execution

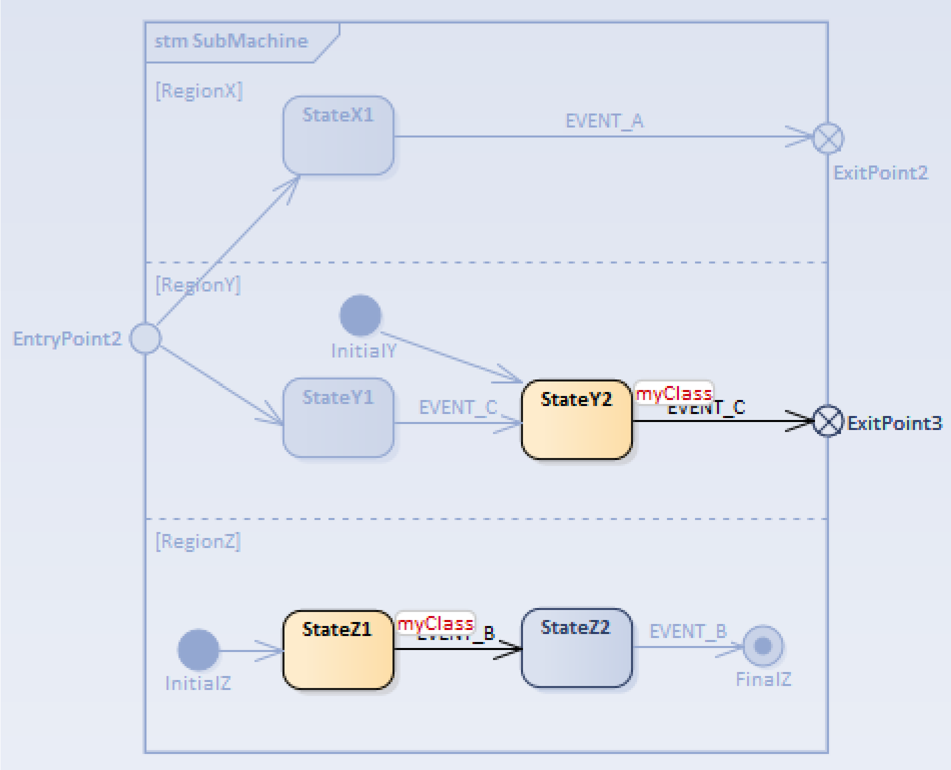
Entering a State: Default Entry

This situation arises when the Composite State is the direct target of a Transition.

Default Entry of *State2*

The Trigger Sequence to be simulated is: [EVENT_A, EVENTC].

When a Transition targeted directly on *State2* is enabled, *State2* is activated, followed by default activation for all the Regions of the SubMachine.



- *RegionX's* State is inactive because it does not define an Initial node
- *RegionY* is activated through *InitialY* and the Transition to *StateY2* is executed
- *RegionZ* is activated through *InitialZ* and the Transition to *StateZ1* is executed

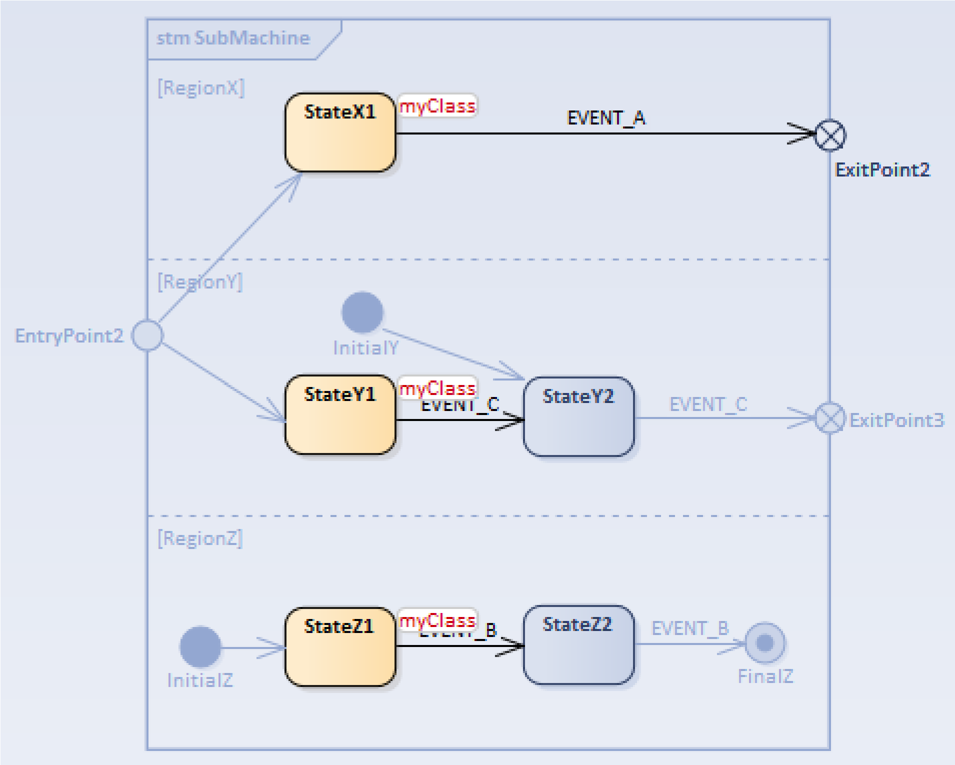
State Exit

State1 Exit

- Trigger Sequence [EVENT_C, EVENT_A]: *RegionC* is inactivated first, then *RegionA* and *RegionB*; after the exit behavior of *State1* is executed, the Transition outgoing from *ExitPoint1* is enabled
- Trigger Sequence [EVENT_A, EVENT_C]: *RegionA* and *RegionB* are inactivated first, then *RegionC*; after the exit behavior of *State1* is executed, the Transition outgoing directly from *State1* is enabled

State2 Exit

Trigger Sequence [EVENT_C, EVENT_A], so the current state resembles this:



- Trigger Sequence [EVENT_A, EVENT_C, EVENT_C, EVENT_B, EVENT_B]: *RegionX* is inactivated first, then *RegionY*, and *RegionZ* is the last; after the exit behavior of *State2* is executed, the Transition outgoing directly from *State2* is enabled
- Trigger Sequence [EVENT_A, EVENT_B, EVENT_B, EVENT_C, EVENT_C]: *RegionX* is inactivated first, then *RegionZ*, and *RegionY* is the last; after the exit behavior of *State2* is executed, the Transition outgoing from *CPR_Exit3* is enabled (*ExitPoint3* on *SubMachine* is bound to *CPR_Exit3* of *State2*)
- Trigger Sequence [EVENT_C, EVENT_C, EVENT_B, EVENT_B, EVENT_A]: *RegionY* is inactivated first, then *RegionZ*, and *RegionX* is the last; after the exit behavior of *State2* is executed, the Transition outgoing from *CPR_Exit2* is enabled (*ExitPoint2* on *SubMachine* is bound to *CPR_Exit2* of *State2*)

Example: History Pseudostate

State History is a convenient concept associated with Regions of Composite States, whereby a Region keeps track of the configuration a State was in when it was last exited. This allows easy return to that State configuration, if necessary, when the Region next becomes active (for example, after returning from handling an interrupt), or if there is a local Transition that returns to its history.

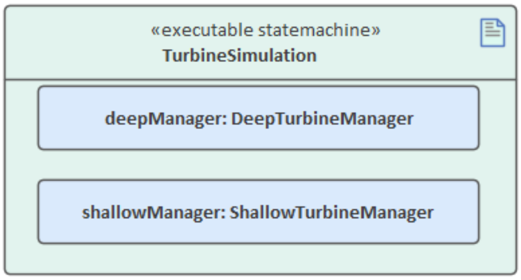
Enterprise Architect supports two types of History Pseudostate:

- Deep History - representing the full State configuration of the most recent visit to the containing Region; the effect is the same as if the Transition terminating on the deepHistory Pseudostate had, instead, terminated on the innermost State of the preserved State configuration, including execution of all entry Behaviors encountered along the way
- Shallow History - representing a return to only the top-most substate of the most recent State configuration, which is entered using the default entry rule

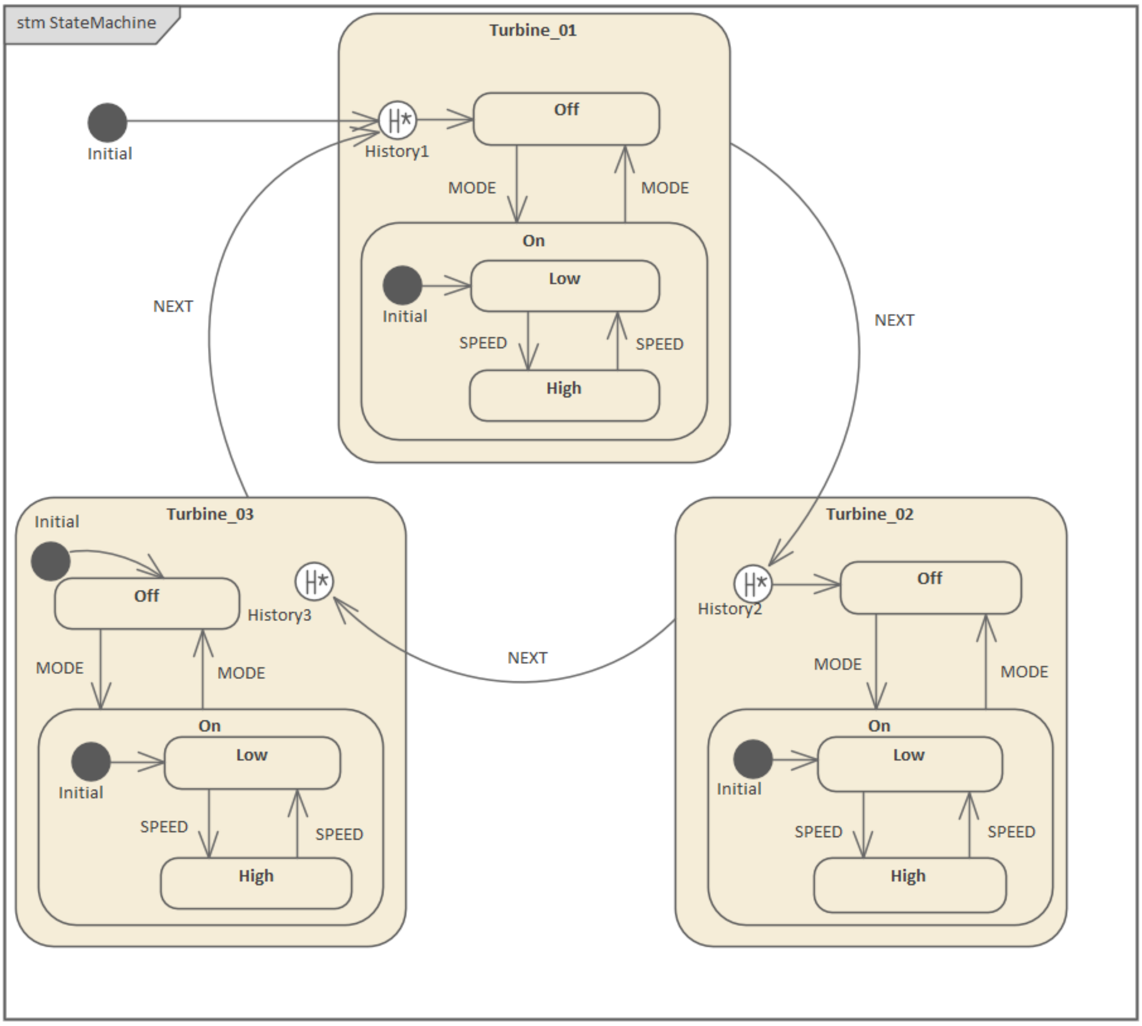
In this example, the Classes *DeepTurbineManager* and *ShallowTurbineManager* are exactly the same except that the contained StateMachine for the first has a deepHistory Pseudostate and for the second has a shallowHistory Pseudostate.

Both StateMachines have three Composite States: *Turbine_01*, *Turbine_02* and*Turbine_03*, each of which has *Off* and *On* States and a History Pseudostate in its Region.

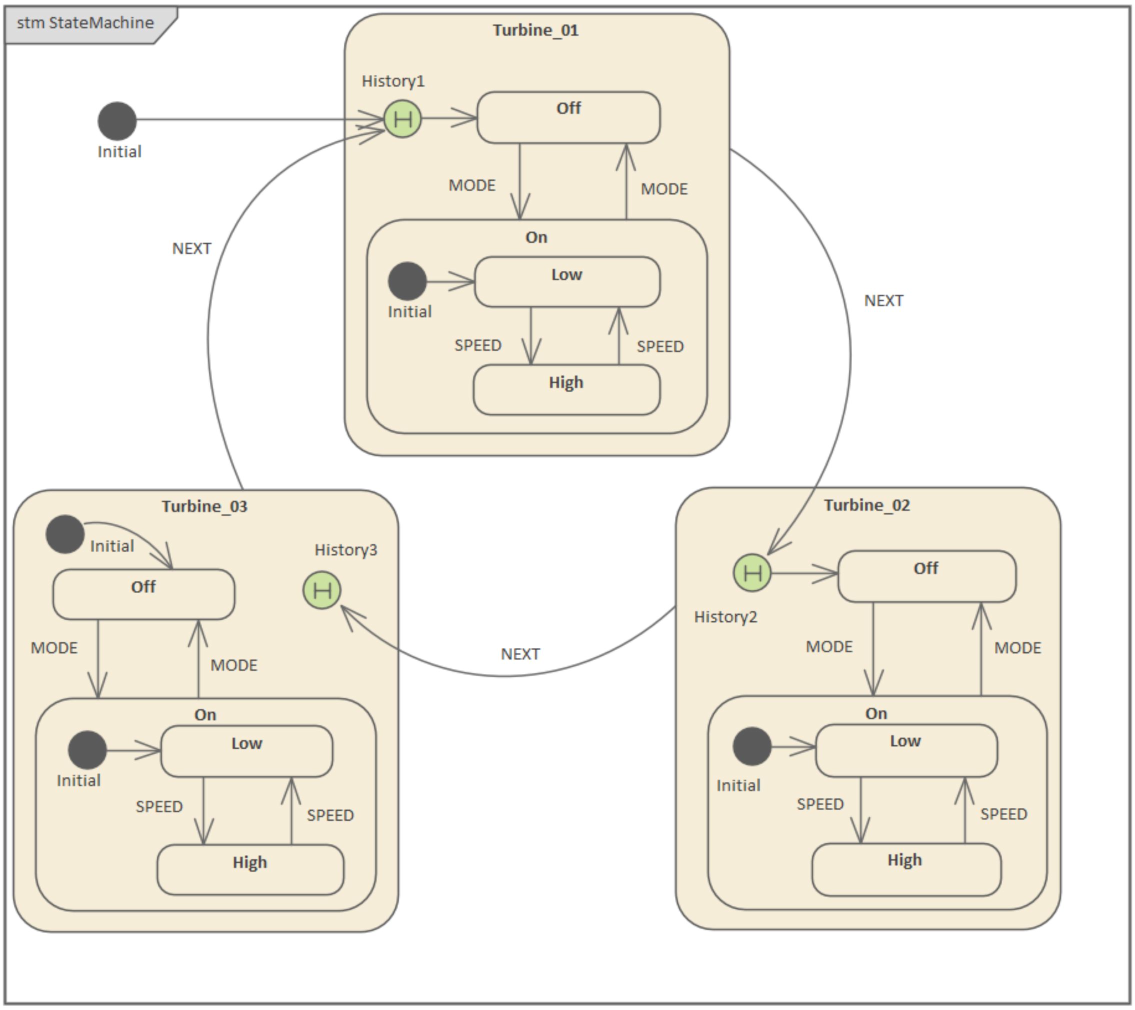
In order to better observe the difference between Deep History and Shallow History, we execute the two StateMachines in one simulation.


«executable statemachine»
TurbineSimulation
deepManager: DeepTurbineManager
shallowManager: ShallowTurbineManager

The StateMachine in *DeepTurbineManager* is illustrated in this diagram:



The StateMachine in *ShallowTurbineManager* is illustrated in this diagram:

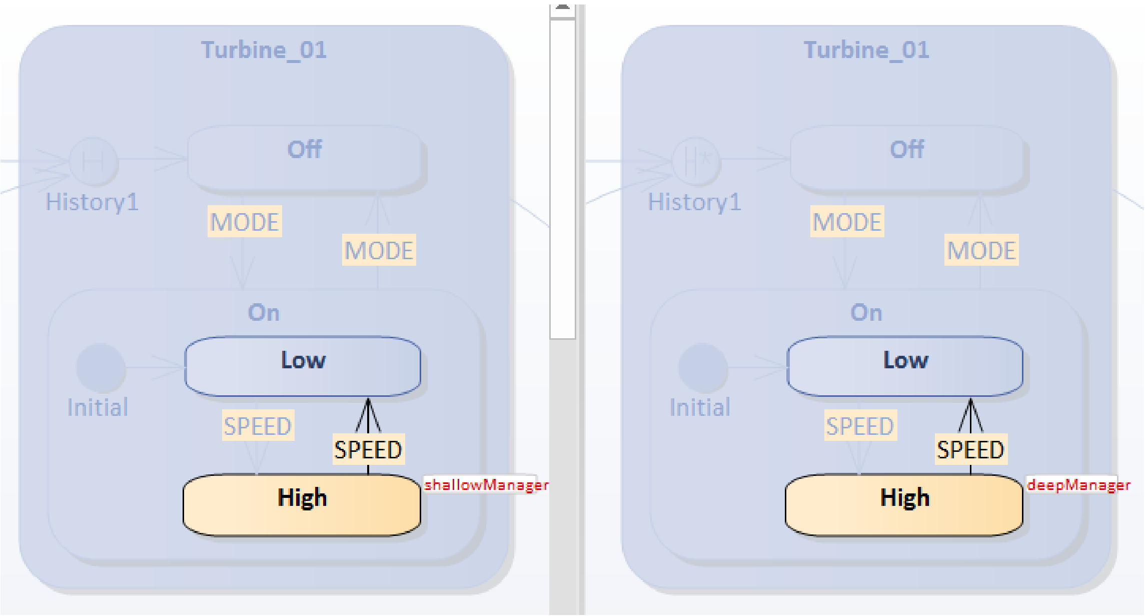


Tip: If you right-click on the History node on the diagram and select the 'Advanced | Deep History' option, you can toggle the type of History Pseudostate between shallow and deep.

First Time Activation of States

After simulation starts, *Turbine_01* and its substate *Off* are activated.

Trigger Sequence: [MODE, SPEED]

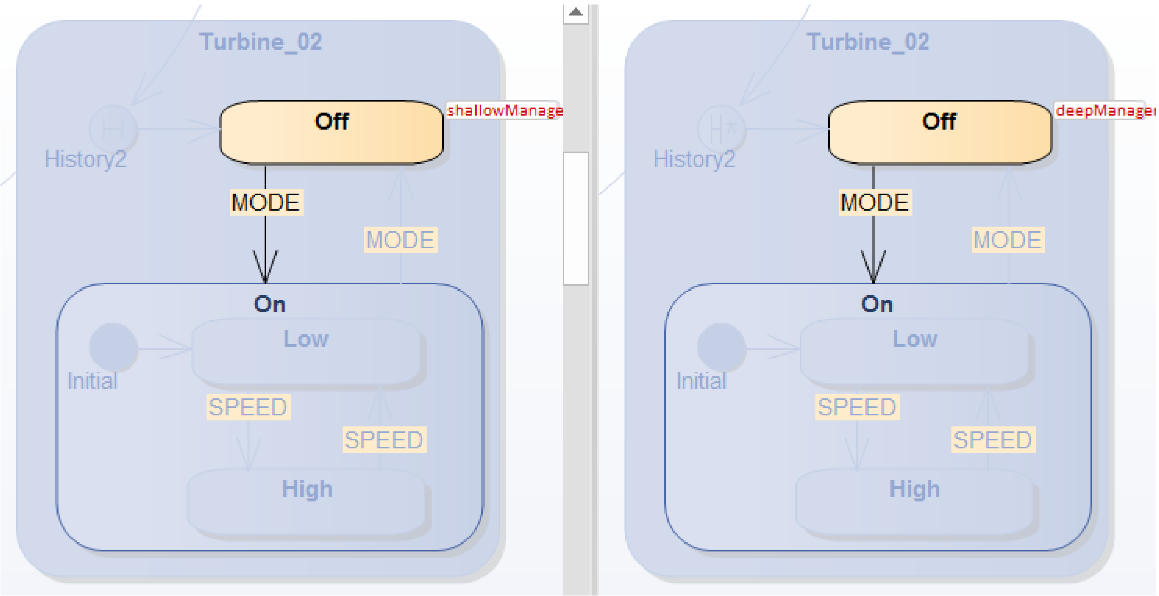


Then the active State configuration includes:

- Turbine_01
- Turbine_01.On
- Turbine_01.On.High

This applies to both *deepManager* and *shallowManager.*

Trigger Sequence: [NEXT]



This trace sequence can be observed from the Simulation window (Simulate > Dynamic Simulation > Simulator > Open Simulation Window):

```
01    shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_High EXIT
02    shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On EXIT
```

03 shallowManager[ShallowTurbineManager].StateMachine_Turbine_01 EXIT
04 shallowManager[ShallowTurbineManager].Turbine_01__TO__History2_105720_61730 Effect
05 shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 ENTRY
06 shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 DO
07 shallowManager[ShallowTurbineManager].History2_105720__TO__Off_61731 Effect
08 shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off ENTRY
09 shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off DO

Note: Since *deepManager* has exactly the same trace as *shallowManager*, the trace for *deepManager* is filtered out from this sequence.

We can learn that:

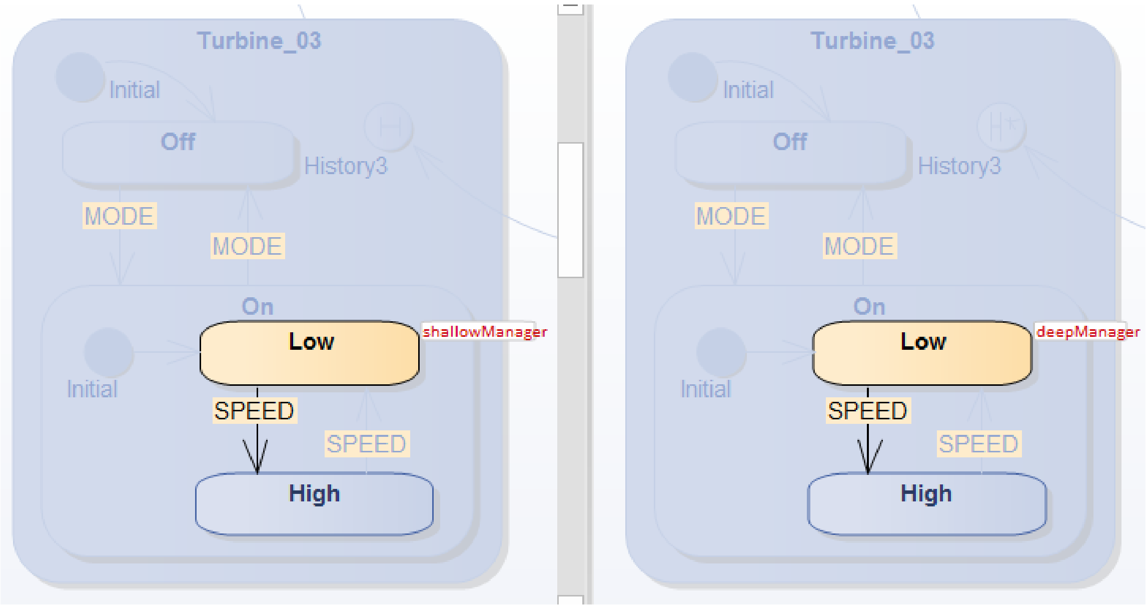
- Exiting a Composite State commences with the innermost State in the active State configuration (see lines 01 - 03 in the trace sequence)
- The Default History Transition is only taken if execution leads to the History node (see line 04) and the State has never been active before (see line 07)

Then the active State configuration includes:

- Turbine_02
- Turbine_02.Off

This applies to both *deepManager* and *shallowManager.*

Trigger Sequence: [NEXT, MODE]



This trace sequence can be observed from the Simulation window:

Trigger [NEXT]
01 shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off EXIT
02 shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 EXIT

```
03   shallowManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725 Effect
04   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRY
05   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 DO
06   shallowManager[ShallowTurbineManager].Initial_105706__TO__Off_61718 Effect
07   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_Off ENTRY
08   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_Off DO
      Trigger [MODE]
   Message omitted...
```

Note: Since *deepManager* has exactly the same trace as *shallowManager,* the trace for *deepManager* is filtered out from this sequence.

We can learn that:

- Since there is no default History Transition defined for *History3*, the standard default entry of the State is performed; an Initial node is found in the Region contained by *Turbine_03,* so the Transition originating from *Initial* is enabled (see line 06)

Then the active state configuration includes:

- Turbine_03
- Turbine_03.On
- Turbine_03.On.Low

This applies to both *deepManager* and *shallowManager.*

History Entry of States

As a reference, we show the Deep History snapshot of each Turbine after its first activation:

Turbine_01

- Turbine_01.On
- Turbine_01.On.High

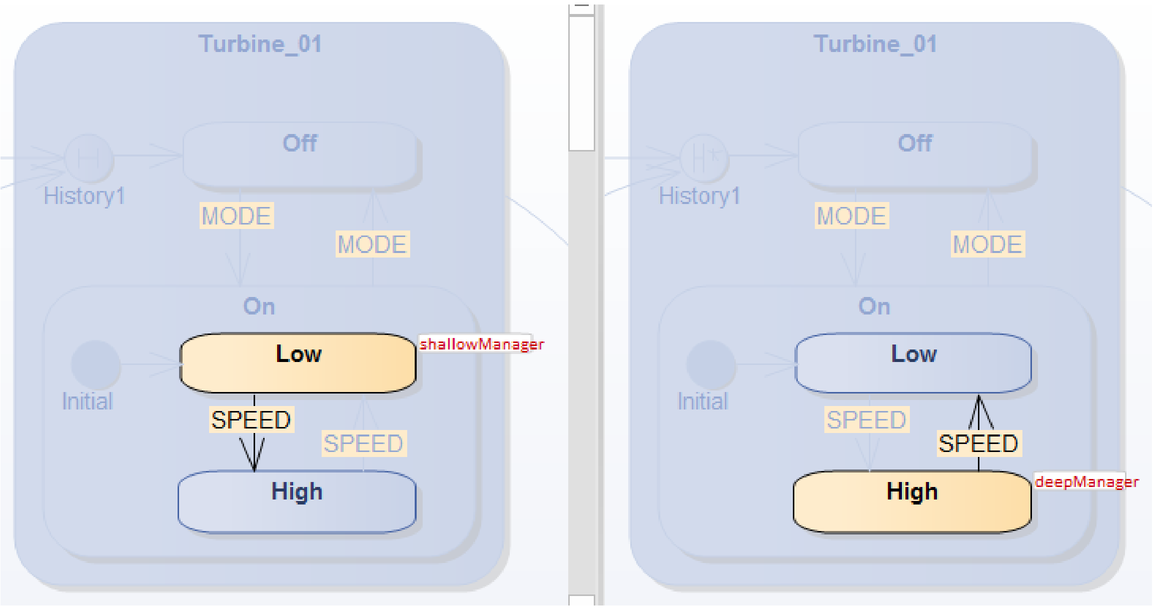
Turbine_02

- Turbine_02.Off

Turbine_03

- Turbine_03.On
- Turbine_03.On.Low

When we further Trigger NEXT, Turbine_01 will be activated again.



This trace sequence can be observed from the Simulation window:

For shallowManager:

```
        Trigger [NEXT]
    01  shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low EXIT
    02  shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On EXIT
    03  shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 EXIT
    04  shallowManager[ShallowTurbineManager].Turbine_03__TO__History1_105711_61732 Effect
    05  shallowManager[ShallowTurbineManager].StateMachine_Turbine_01 ENTRY
    06  shallowManager[ShallowTurbineManager].StateMachine_Turbine_01 DO
    07  shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On ENTRY
    08  shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On DO
    09  shallowManager[ShallowTurbineManager].Initial_105721__TO__Low_61729 Effect
    10  shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low ENTRY
    11  shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low DO
```

We can learn that:

- The shallowHistory node restores Turbine_01 as far as Turbine_01.On
- Then the Region contained by Composite State Turbine_01.On will be activated by the *Initial* node, which activated at *Low*

For deepManager:

```
        Trigger [NEXT]
    01  deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low EXIT
    02  deepManager[DeepTurbineManager].StateMachine_Turbine_03_On EXIT
    03  deepManager[DeepTurbineManager].StateMachine_Turbine_03 EXIT
    04  deepManager[DeepTurbineManager].Turbine_03__TO__History1_105679_61708 Effect
    05  deepManager[DeepTurbineManager].StateMachine_Turbine_01 ENTRY
```

```
06    deepManager[DeepTurbineManager].StateMachine_Turbine_01 DO
07    deepManager[DeepTurbineManager].StateMachine_Turbine_01_On ENTRY
08    deepManager[DeepTurbineManager].StateMachine_Turbine_01_On_High ENTRY
```

We can learn that:

- The *deepHistory* node restores Turbine_01 as far as Turbine_01.On.High

Trigger [NEXT] to exit Turbine_01 and activate Turbine_02

Both *shallowManager* and *deepManager* activate Turbine_02.Off, which is the History snapshot when they exited.

Trigger [NEXT] to exit Turbine_02 and activate Turbine_03

Both *shallowManager* and *deepManager* activate Turbine_03.On.Low. However, the sequences of *shallowManager* and *deepManager* are different.

For *shallowManager*, the *shallowHistory* can only restore as far as Turbine_03.On. Since an *Initial* node is defined in Turbine_03.On, the Transition originating from *Initial* will be enabled and Turbine_03.On.Low is reached.

```
01    shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off EXIT
02    shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 EXIT
03    shallowManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725 Effect
04    shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRY
05    shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 DO
06    shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On ENTRY
07    shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On DO
08    shallowManager[ShallowTurbineManager].Initial_105727__TO__Low_61728 Effect
09    shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low ENTRY
10    shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low DO
```

For *deepManager*, the *deephistory* can restore as far as Turbine_03.On.Low directly.

```
01    deepManager[DeepTurbineManager].StateMachine_Turbine_02_Off EXIT
02    deepManager[DeepTurbineManager].StateMachine_Turbine_02 EXIT
03    deepManager[DeepTurbineManager].Turbine_02__TO__History3_105680_61701 Effect
04    deepManager[DeepTurbineManager].StateMachine_Turbine_03 ENTRY
05    deepManager[DeepTurbineManager].StateMachine_Turbine_03 DO
06    deepManager[DeepTurbineManager].StateMachine_Turbine_03_On ENTRY
07    deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low ENTRY
```

Event Macro: EVENT_PARAMETER

EVENT_PARAMETER is a function macro used to access attribute of a signal instance, in State's behavior, Transition's guard and effect. This macro will be expanded to executable code according to the simulation language.

Access Default Macro Definition

Ribbon | Develop | Source Code | Options | Edit Code Templates | Language | Stm Event Parameter

Usage Format

%EVENT_PARAMETER(SignalType, SignalAttributeName)%

For example: A signal 'MySignal' has two attributes, 'foo:int' and 'bar:int'; these use cases are valid:

- Transition's effect: %EVENT_PARAMETER(MySignal, foo)%
- State's behavior: %EVENT_PARAMETER(MySignal, bar)%
- Transition's guard: %EVENT_PARAMETER(MySignal, bar)% > 10
- State's behavior: %EVENT_PARAMETER(MySignal, bar)%++
- Trace the value to Simulation Window: %TRACE(EVENT_PARAMETER(MySignal, foo))%

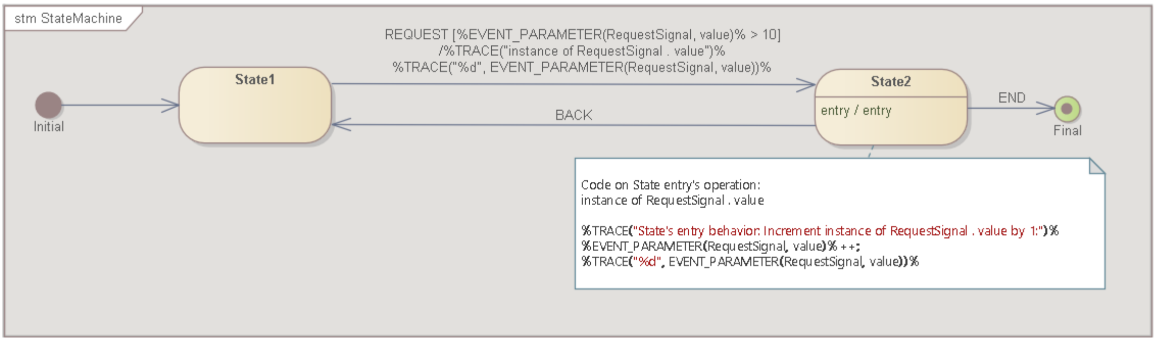
Macro Expansion Example

For Signal "MySignal" with attribute "value", Examples of macro expansion for Transition with signal trigger:%EVENT_PARAMETER(MySignal, value)%

C	((MySignal*)signal)->value
C++	static_cast<MySignal*>(signal)->value
C#	((MySignal)signal).value
Java	((EventProxy.MySignal)signal).value
JavaScript	signal.value

Example

This example demonstrated how to use EVENT_MACRO in Transition's effect, guard and State's behavior.


stm StateMachine
REQUEST [%EVENT_PARAMETER(RequestSignal, value)% > 10]
/%TRACE("instance of RequestSignal . value")%
%TRACE("%d", EVENT_PARAMETER(RequestSignal, value))%
State1
State2
entry / entry
BACK
END
Initial
Final

```
Code on State entry's operation:
instance of RequestSignal . value

%TRACE("State's entry behavior: Increment instance of RequestSignal . value by 1:")%
%EVENT_PARAMETER(RequestSignal, value)% ++;
%TRACE("%d", EVENT_PARAMETER(RequestSignal, value))%
```

While running the simulation,

(1) trigger REQUEST and specify number 1 in for attribute value.

Since the condition for the guard is false, the active state will remain as State1.

(2) trigger REQUEST and specify number 11 for the attribute value.

Since the condition for the guard is true, the active state will change from State1 to State2;

The Transition's effect is executed. Here we traced the runtime value for the signal's attribute to the simulation window..

State2's behavior is executed. Here we incremented the runtime value for the signal's attribute and traced it to the simulation window.

```
[32608107]    [Part1:TransactionServer] Transition Effect: Initial_4019__TO__State1_4420
[32608118]    [Part1:TransactionServer] Entry Behavior: StateMachine_State1
[32608124]    [Part1:TransactionServer] Do Behavior: StateMachine_State1
[32608877]    [Part1:TransactionServer] Completion: TransactionServer_StateMachine_State1
[32608907]    Waiting for Trigger
[32613165]    Command: broadcast REQUEST.RequestSignal(1)
[32613214]    [Part1:TransactionServer] Event Queued: REQUEST.RequestSignal(value:1)
[32613242]    [Part1:TransactionServer] Event Dispatched: REQUEST.RequestSignal(value:1)
[32613279]    Waiting for Trigger
[32619541]    Command: broadcast REQUEST.RequestSignal(11)
[32619546]    [Part1:TransactionServer] Event Queued: REQUEST.RequestSignal(value:11)
[32619551]    [Part1:TransactionServer] Event Dispatched: REQUEST.RequestSignal(value:11)
[32619557]    [Part1:TransactionServer] Exit Behavior: StateMachine_State1
[32619562]    [Part1:TransactionServer] Transition Effect: State1__TO__State2_4421
[32619567]    instance of RequestSignal . value
[32619571]    11
[32619576]    [Part1:TransactionServer] Entry Behavior: StateMachine_State2
[32619584]    State's entry behavior: Increment instance of RequestSignal . value by 1:
[32619590]    12
[32619594]    [Part1:TransactionServer] Do Behavior: StateMachine_State2
[32620168]    [Part1:TransactionServer] Completion: TransactionServer_StateMachine_State2
[32620211]    Waiting for Trigger
[32622266]    Command: broadcast END
[32622272]    [Part1:TransactionServer] Event Queued: END
[32622310]    [Part1:TransactionServer] Event Dispatched: END
```

```
[32622349]      [Part1:TransactionServer] Exit Behavior: StateMachine_State2
[32622359]      [Part1:TransactionServer] Transition Effect: State2__TO__Final_4023_4423
[32622896]      [Part1:TransactionServer] Completion: TransactionServer_VIRTUAL_SUBMACHINESTATE
```

Limitation and Workaround

Since the macro expansion involve type casting, it is user's responsibility to ensure the type casting is valid.

And here are workarounds for some common cases that the type casting will encounter issues in the model.

- **One Transition has multiple triggers of different signal types**

For example, a Transition has triggerA(specified SignalA) and triggerB(specified SignalB); A macro %EVENT_PARAMETER(SignalA, attributeOfA)% will not work when this Transition is triggered by triggerB.

We suggest to create two Transitions, one with triggerA(specified SignalA), and the other with triggerB(SignalB).

- **One State is the target of multiple Transitions of different signal triggers**

For example, both TransitionA and TransitionB are targeting to MyState, TransitionA is triggered by SignalA, and TransitionB is triggered by SignalB.

If EVENT_PARAMETER is used in state's behavior code, one macro will not be able to work for both cases.

We suggest to move the logic dealing with signal's attribute from State's behavior to Transition's effect.

- **Customize template and generated code**

User can also change the default template to add Run Time Type Identification (RTTI) before type casting. User can also modify the generated code after generation, which can also satisfy the simulation purpose after compilation.

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

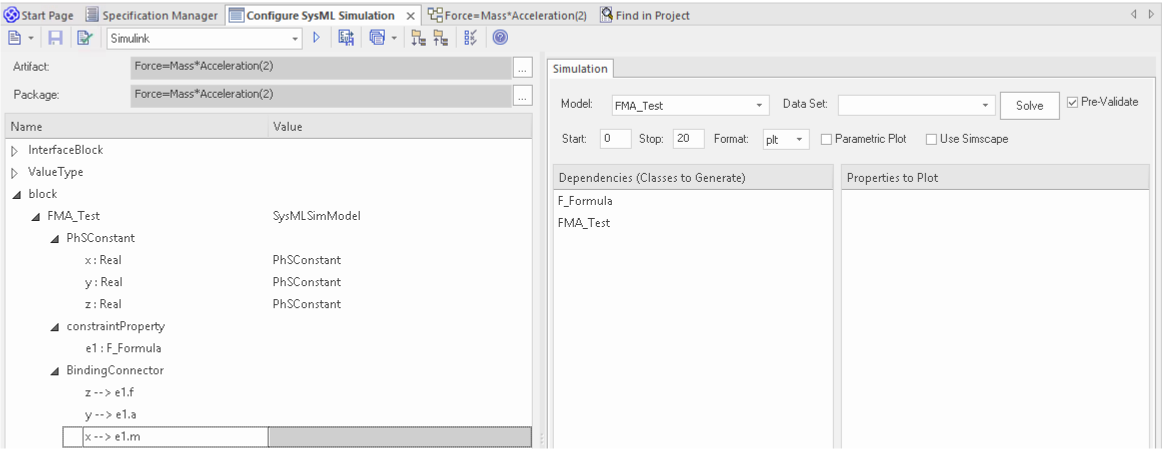
These sections describe the process of defining a Parametric model, annotating the model with additional information to drive a simulation, and running a simulation to generate a graph of the results.

Section	Description
Introduction to SysML Parametric Models	SysML Parametric models support the engineering analysis of critical system parameters, including the evaluation of key metrics such as performance, reliability and other physical characteristics. These models combine Requirements models with System Design models, by capturing executable constraints based on complex mathematical relationships. Parametric diagrams are specialized Internal Block diagrams that help you, the modeler, to combine behavior and structure models with engineering analysis models such as performance, reliability, and mass property models. For further information on the concepts of SysML Parametric models, refer to the official OMG SysML website and its linked sources.
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.



Simulink

Artifact: Force=Mass*Acceleration(2)

Package: Force=Mass*Acceleration(2)

Name	Value
InterfaceBlock	
ValueType	
block	
FMA_Test	SysMLSimModel
PhSConstant	
x : Real	PhSConstant
y : Real	PhSConstant
z : Real	PhSConstant
constraintProperty	
e1 : F_Formula	
BindingConnector	
z --> e1.f	
y --> e1.a	
x --> e1.m	

Model: FMA_Test Data Set: ☑ Pre-Validate

Start: 0 Stop: 20 Format: plt □ Parametric Plot □ Use Simscape

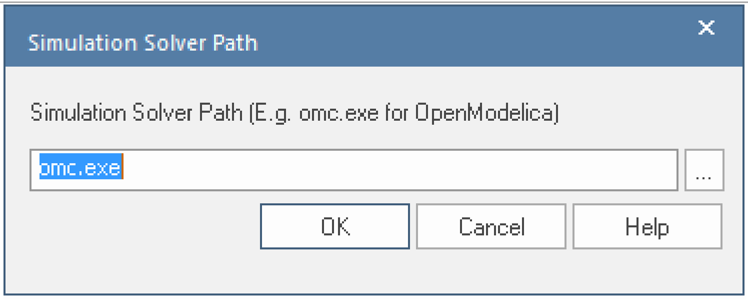
Dependencies (Classes to Generate)	Properties to Plot
F_Formula	
FMA_Test	

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

	 Simulation Solver Path Simulation Solver Path (E.g. omc.exe for OpenModelica) omc.exe
	Click on this button to save the configuration to the current Artifact.
	Click on this icon to specifically validate the model against the SysML configuration now. The results of the validation display in the 'SysML Simulation' tab of the System Output window. You can also select an option to automatically pre-validate the model before each simulation is executed. See the 'Pre-validate' option in the *Simulation Tab* table.
	Click on this icon to expand every item in the hierarchy in the 'Name' column of the window.
	Click on this icon to collapse all the expanded items in the model hierarchy in the 'Name' column of the window.
	Click on this icon to display a list of object types that can be suppressed in the simulation. Click on the checkbox against each object to suppress, or click on the All button to select all items for suppression. You can also use the Filter Bar at the top of the 'Option' column to only display items having the specified letter or text string in the name.
	Click on the drop-down arrow and select the application under which the simulation is being run - such as OpenModelica or Simulink.
	Click on this button to generate, compile and run the current configuration, and display the results.
	After simulation, the result file is generated in either plt, mat or csv format. That is, with the filename: • ModelName_res.mat (the default for OpenModelica) • ModelName_res.plt or • ModelName_res.csv Click on this button to specify a directory into which Enterprise Architect will copy the result file.
	Click on this button to select from these options: • Run Last Code - Execute the most recently generated code • Generate Code — Generate the code without compiling or running it • Open Simulation Directory — Open the directory into which OpenModelica or Simulink code will be generated • Edit Templates — Customize the code generated for OpenModelica or Simulink, using the Code Template Editor

Simulation Artifact and Model Selection

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

Package Objects

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

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

Element Type	Behavior
ValueType	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 stereotypes have built-in properties for *isContinuous*, *isConserved*, and *changeCycle*. The Properties will be listed under either PhSConstant or PhSVariable and the Value cannot be changed. It's also possible to define the settings within the Configure SysML Simulation window. In this case they will be listed under 'Properties'. Properties within a Block can be configured to be either SimConstants or

	SimVariables. For a SimVariable, you configure these attributes: - isContinuous — determines whether the property value varies continuously ('true', the default) or discretely ('false') - isConserved — determines whether values of the property are conserved ('true') or not ('false', the default); when modeling for physical interaction, the interactions include exchanges of conserved physical substances such as electrical current, force or fluid flow - changeCycle — specifies the time interval at which a discrete property value changes; the default value is '0' - changeCycle can be set to a value other than 0 only when isContinuous = 'false' - The value of changeCycle must be positive or equal to 0
Port	No configuration required.
SimFunction	Functions are created as operations in Blocks or ConstraintBlocks, stereotyped as 'SimFunction'. No configuration is required in the Configure SysML Simulation window.
Generalization	No configuration required.
Binding Connector	Binds a property to a parameter of a constraint property. No configuration required; however, if the properties are different, the system provides an option to synchronize them.
Connector	Connects two Ports. No configuration required in the Configure SysML Simulation view. However, you might have to configure the properties of the Port's type by determining whether the attribute isConserved should be set as 'False' (for potential properties, so that equality coupling is established) or 'True' (for flow/conserved properties, so that sum-to-zero coupling is established).
Constraint Block	No configuration required.

Simulation Tab

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

Field	Action
Model	Click on the drop-down arrow and select the top-level node (a SysMLSimModel element) for the simulation. The list is populated with the names of the Blocks defined as top-level, model nodes.
Data Set	Click on the drop-down arrow and select the dataset for the selected model.
Pre-Validate	Select this checkbox to automatically validate the model before each simulation of the model is executed.
Start	Type in the initial wait time before which the simulation is started, in seconds (default value is 0).

Stop	Type in the number of seconds for which the simulation will execute.
Format	Click on the drop-down arrow and select either 'plt', 'csv' or 'mat' as the format of the result file, which could potentially be used by other tools.
Parametric Plot	• Select this checkbox to plot Legend A on the y-axis against Legend B on the x-axis. • Deselect the checkbox to plot Legend(s) on the y-axis against time on the x-axis Note: With the checkbox selected, you must select two properties to plot.
Use Simscape	(if the selected math tool is Simulink) Select the checkbox if you also want to process the simulation in Simscape.
Dependencies	Lists the types that must be generated to simulate this model.
Properties to Plot	Provides a list of variable properties that are involved with the simulation. Select the checkbox against each property to plot.

Creating a Parametric Model

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

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

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

You would also take into consideration:

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

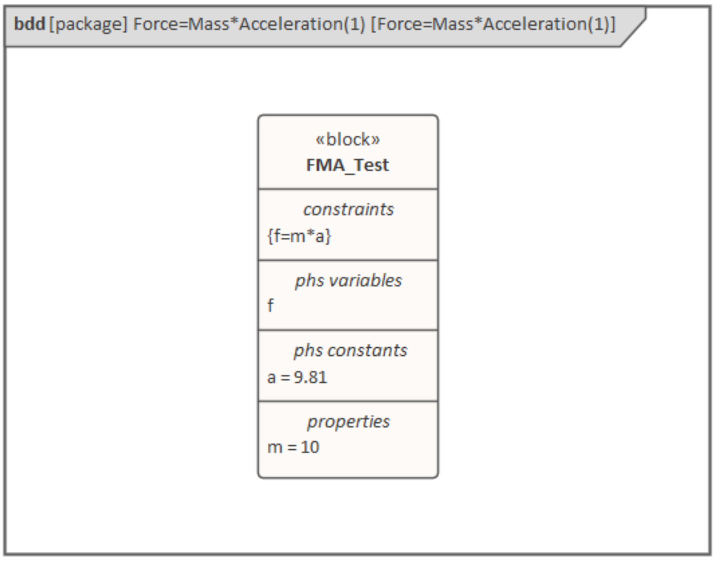
Access

Ribbon	Simulate > System Behavior > Modelica/Simulink > SysMLSim Configuration Manager

Defining inline Constraint Equations on a Block

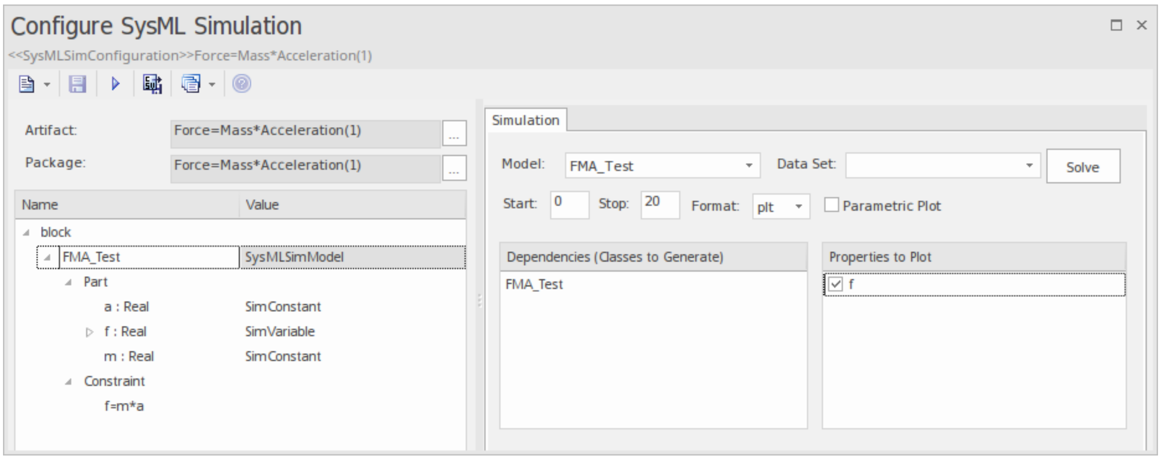
Defining constraints directly in a Block is straightforward and is the easiest way to define constraint equations.

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



Tip: You can define multiple constraints in one Block.

1. Create a SysMLSim Configuration Artifact 'Force=Mass*Acceleration(1)' and point it to the Package 'FMA_Test'.
2. For 'FMA_Test', in the 'Value' column set 'SysMLSimModel'.
3. For Parts 'a', 'm' and 'f', in the 'Value' column: set 'a' and 'm' to 'PhSConstant' and (optionally) set 'f' to 'PhSVariable'.
4. On the 'Simulation' tab, in the 'Properties to Plot' panel, select the checkbox against 'f'.
5. Click on the Solve button to run the simulation.



Configure SysML Simulation

<<SysMLSimConfiguration>>Force=Mass*Acceleration(1)

Artifact: Force=Mass*Acceleration(1)

Package: Force=Mass*Acceleration(1)

Name	Value
block	
FMA_Test	SysMLSimModel
Part	
a : Real	SimConstant
f : Real	SimVariable
m : Real	SimConstant
Constraint	
f=m*a	

Model: FMA_Test

Data Set:

Start: 0 Stop: 20 Format: plt □ Parametric Plot

Dependencies (Classes to Generate): FMA_Test

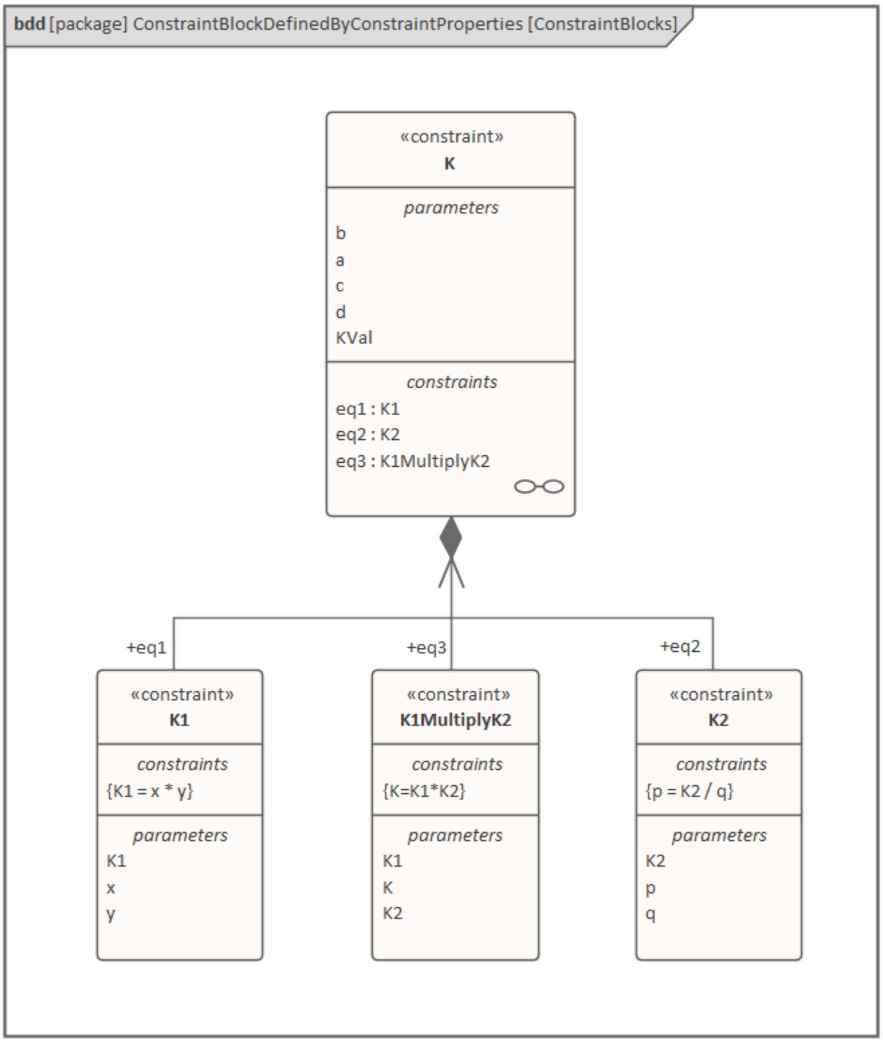
Properties to Plot: ☑ f

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

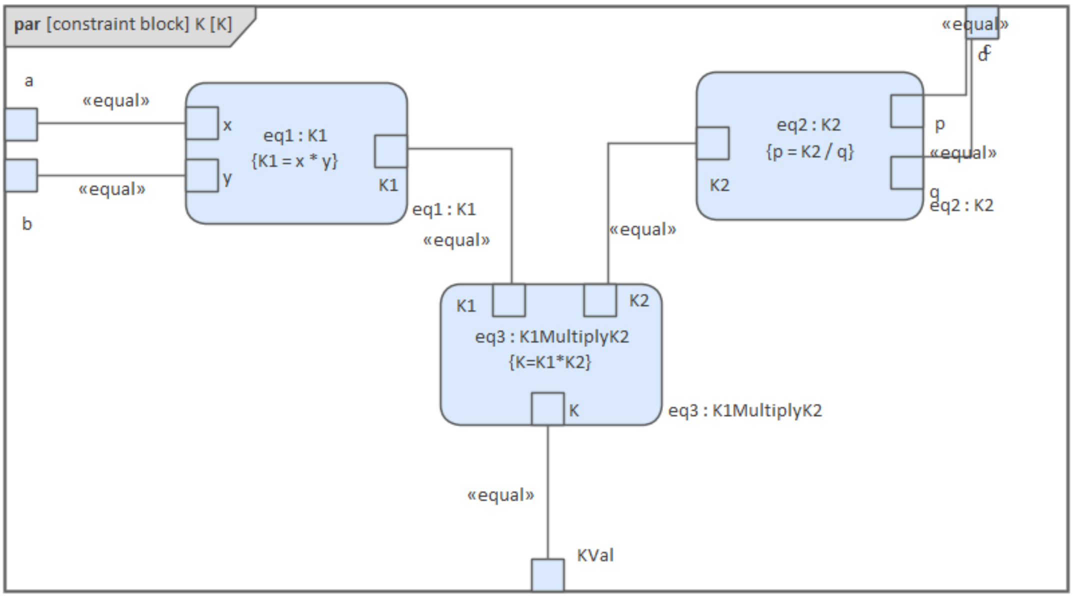
Connected Constraint Properties

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

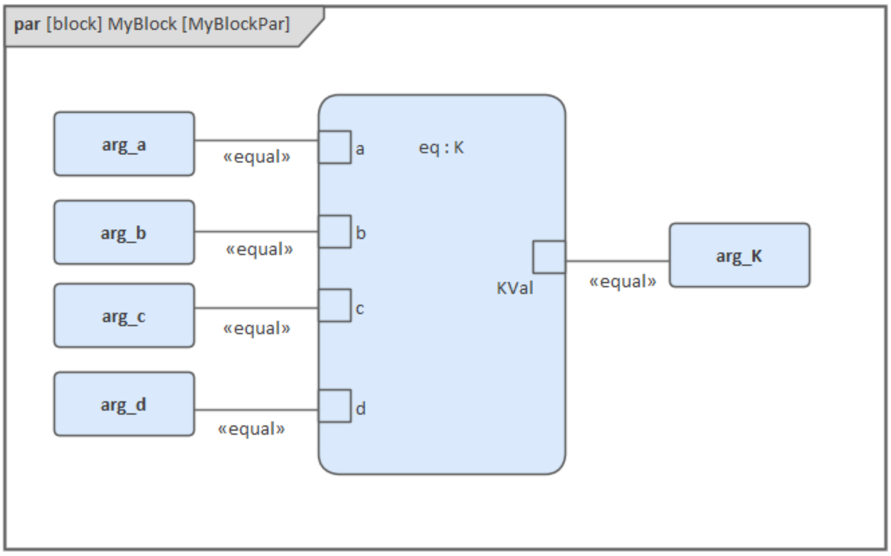
In this figure, ConstraintBlock 'K' defines parameters 'a', 'b', 'c', 'd' and 'KVal', and three constraint properties 'eq1', 'eq2' and 'eq3', typed to 'K1', 'K2' and 'K1MultiplyK2' respectively.



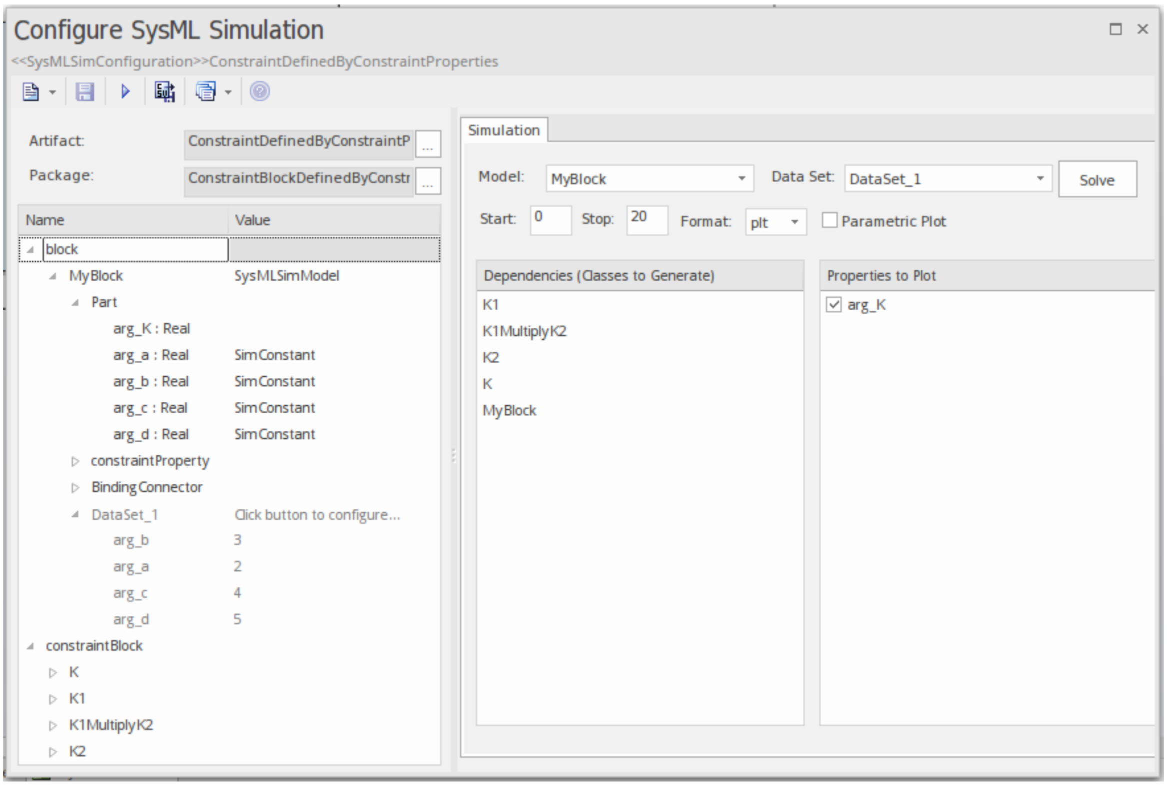
Create a Parametric diagram in ConstraintBlock 'K' and connect the parameters to the constraint properties with Binding connectors, as shown:



- 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


Configure SysML Simulation

<<SysMLSimConfiguration>>ConstraintDefinedByConstraintProperties

Artifact: ConstraintDefinedByConstraintP

Package: ConstraintBlockDefinedByConstr

Name	Value
block	
MyBlock	SysMLSimModel
Part	
arg_K : Real	
arg_a : Real	SimConstant
arg_b : Real	SimConstant
arg_c : Real	SimConstant
arg_d : Real	SimConstant
constraintProperty	
BindingConnector	
DataSet_1	Click button to configure...
arg_b	3
arg_a	2
arg_c	4
arg_d	5
constraintBlock	
K	
K1	
K1MultiplyK2	
K2	

Simulation

Model: MyBlock Data Set: DataSet_1 Solve

Start: 0 Stop: 20 Format: plt □ Parametric Plot

Dependencies (Classes to Generate)
K1
K1MultiplyK2
K2
K
MyBlock

Properties to Plot
☑ arg_K

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.

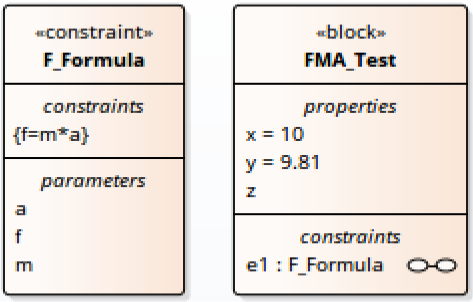
We can easily solve K2 to be p * q in this example, but in some complex examples it is extremely hard to solve a variable from an equation; however, the Enterprise Architect SysMLSim can still get it right.

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

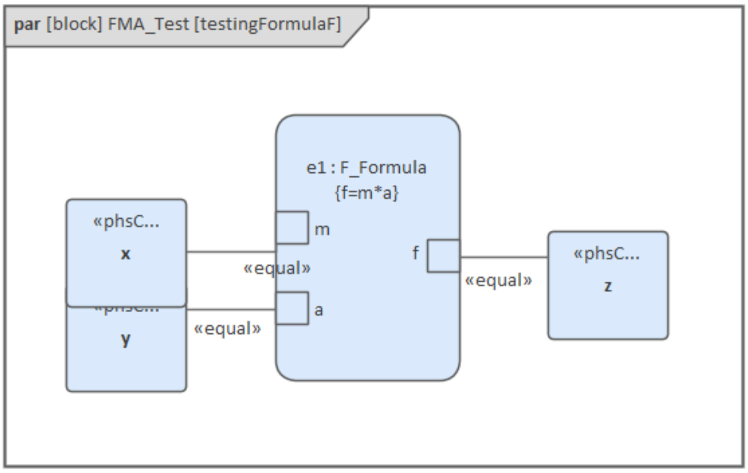
If one equation is commonly used in many Blocks, a ConstraintBlock can be created for use as a constraint property in each Block. These are the changes we make, based on the previous example:

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

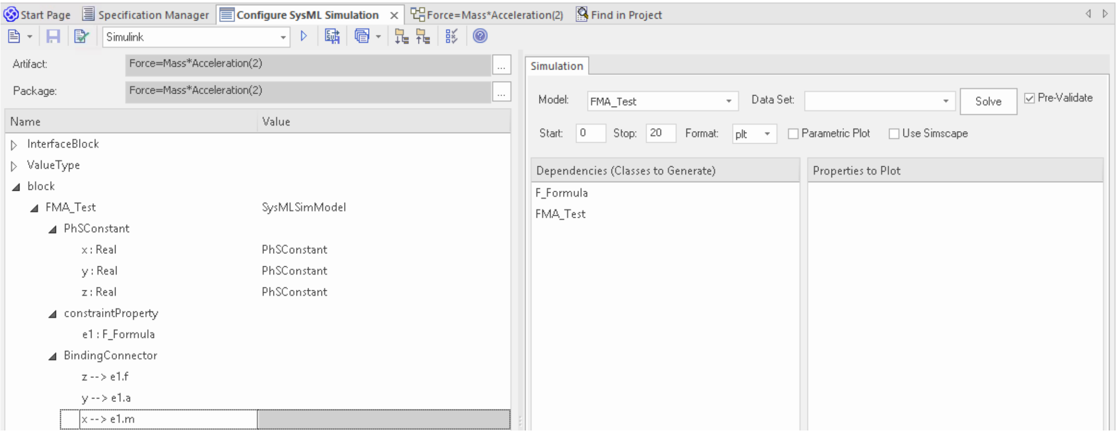


Tip: Primitive type 'Real' will be applied if property types are empty

- 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



Artifact: Force=Mass*Acceleration(2)

Package: Force=Mass*Acceleration(2)

Name	Value
InterfaceBlock	
ValueType	
block	
FMA_Test	SysMLSimModel
PhSConstant	
x : Real	PhSConstant
y : Real	PhSConstant
z : Real	PhSConstant
constraintProperty	
e1 : F_Formula	
BindingConnector	
z --> e1.f	
y --> e1.a	
x --> e1.m	

Model: FMA_Test Data Set: ☑ Pre-Validate

Start: 0 Stop: 20 Format: plt □ Parametric Plot □ Use Simscape

Dependencies (Classes to Generate)

F_Formula
FMA_Test

Properties to Plot

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;
```

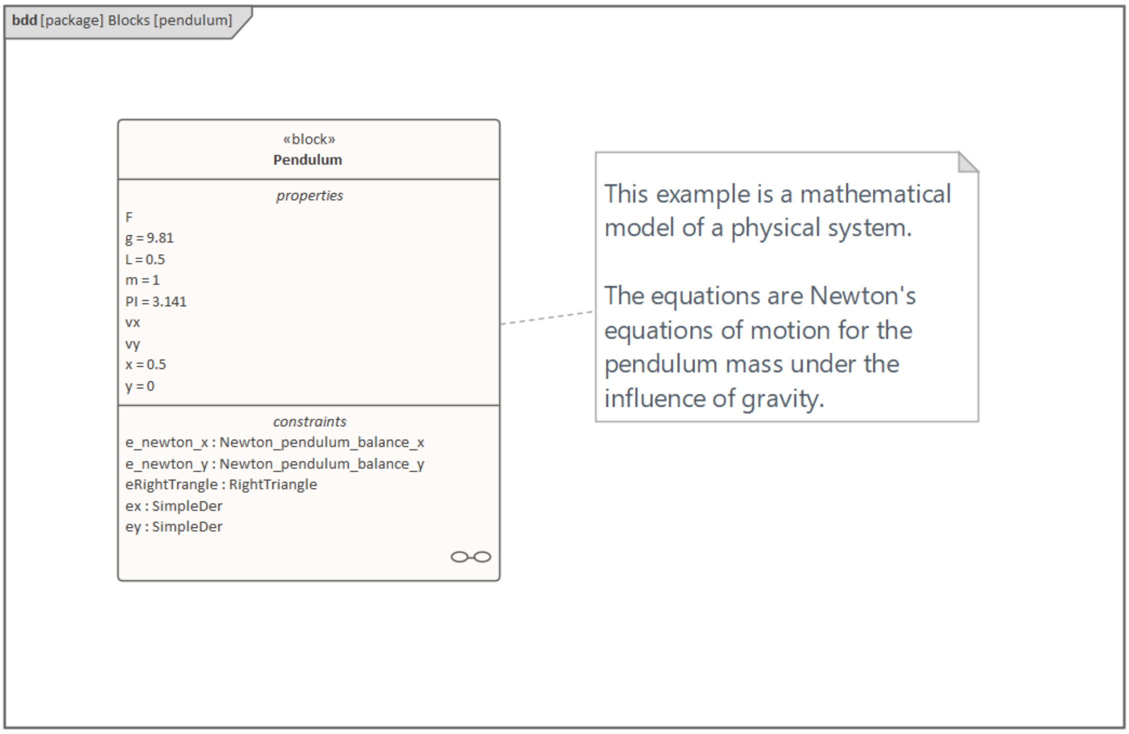
Each connect equation is actually expanded to two equations (there are two properties defined in ChargePort), one for equality coupling, the other for sum-to-zero coupling:

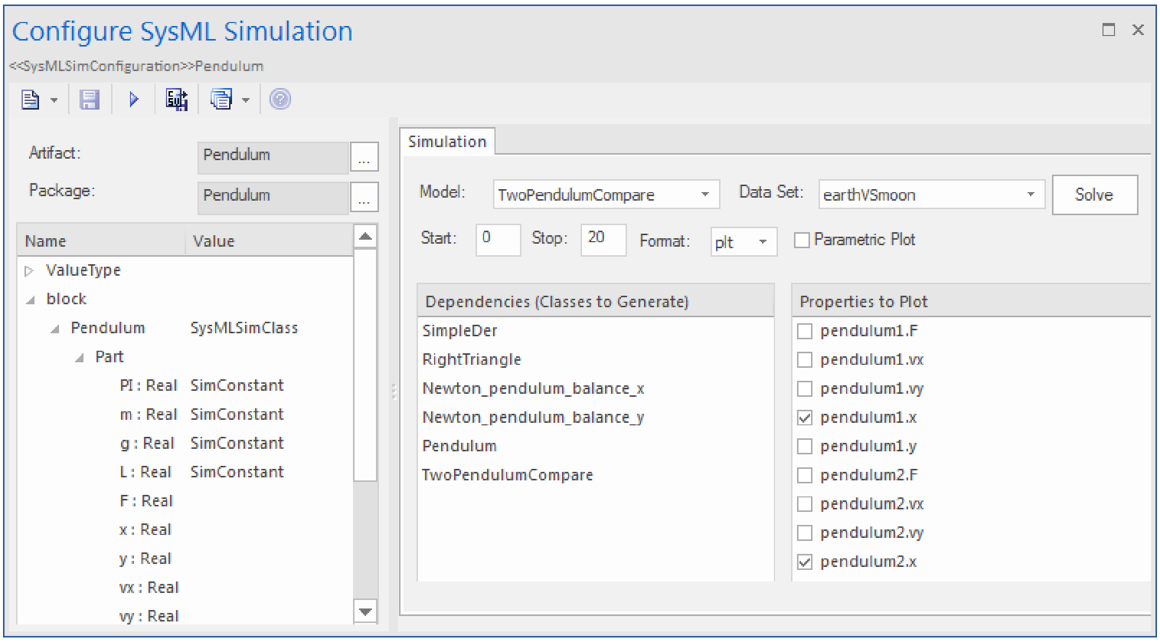
```
source.p.v = resistor.n.v;
source.p.i + resistor.n.i = 0;
```

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'.


bdd [package] Blocks [pendulum]
«block»
Pendulum
properties
F
g = 9.81
L = 0.5
m = 1
PI = 3.141
vx
vy
x = 0.5
y = 0
constraints
e_newton_x : Newton_pendulum_balance_x
e_newton_y : Newton_pendulum_balance_y
eRightTrangle : RightTriangle
ex : SimpleDer
ey : SimpleDer
This example is a mathematical model of a physical system.
The equations are Newton's equations of motion for the pendulum mass under the influence of gravity.



Configure SysML Simulation

<<SysMLSimConfiguration>>Pendulum

Artifact: Pendulum

Package: Pendulum

Name	Value
▷ ValueType	
◢ block	
◢ Pendulum	SysMLSimClass
◢ Part	
PI : Real	SimConstant
m : Real	SimConstant
g : Real	SimConstant
L : Real	SimConstant
F : Real	
x : Real	
y : Real	
vx : Real	
vy : Real	

Simulation

Model: TwoPendulumCompare Data Set: earthVSmoon Solve

Start: 0 Stop: 20 Format: plt □ Parametric Plot

Dependencies (Classes to Generate)

SimpleDer
RightTriangle
Newton_pendulum_balance_x
Newton_pendulum_balance_y
Pendulum
TwoPendulumCompare

Properties to Plot

□ pendulum1.F
□ pendulum1.vx
□ pendulum1.vy
☑ pendulum1.x
□ pendulum1.y
□ pendulum2.F
□ pendulum2.vx
□ pendulum2.vy
☑ pendulum2.x

The generated Modelica code resembles this:

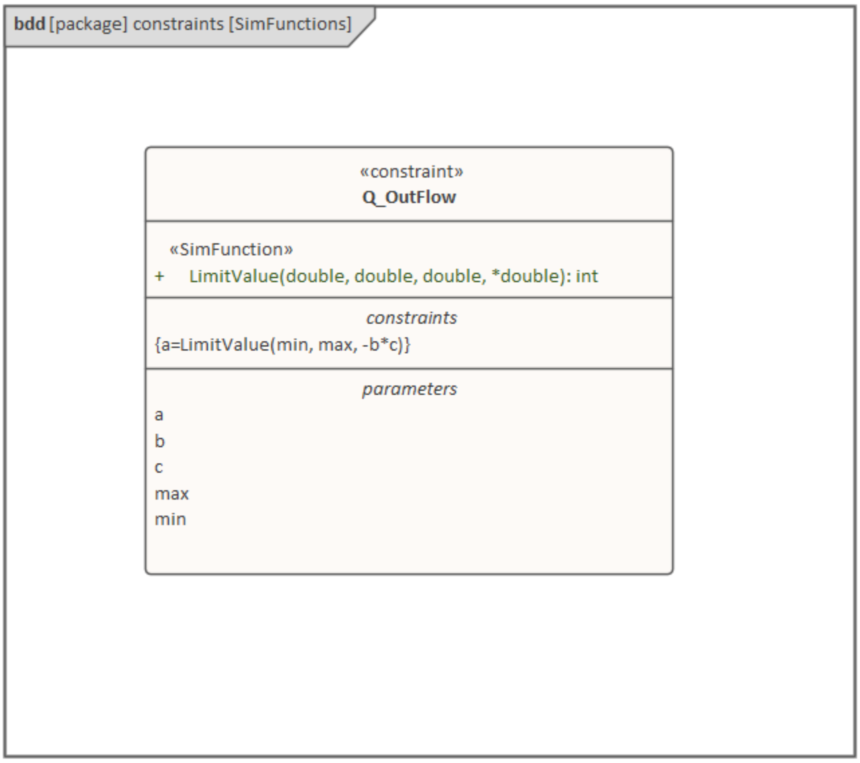
```
class Pendulum
    parameter Real PI = 3.141;
    parameter Real m = 1;
    parameter Real g = 9.81;
    parameter Real L = 0.5;
    Real F;
    Real x (start=0.5);
    Real y (start=0);
    Real vx;
    Real vy;
    ......
equation
......
end Pendulum;
```

- 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.


bdd [package] constraints [SimFunctions]
«constraint»
Q_OutFlow
«SimFunction»
+ LimitValue(double, double, double, *double): int
constraints
{a=LimitValue(min, max, -b*c)}
parameters
a
b
c
max
min

- 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;
```

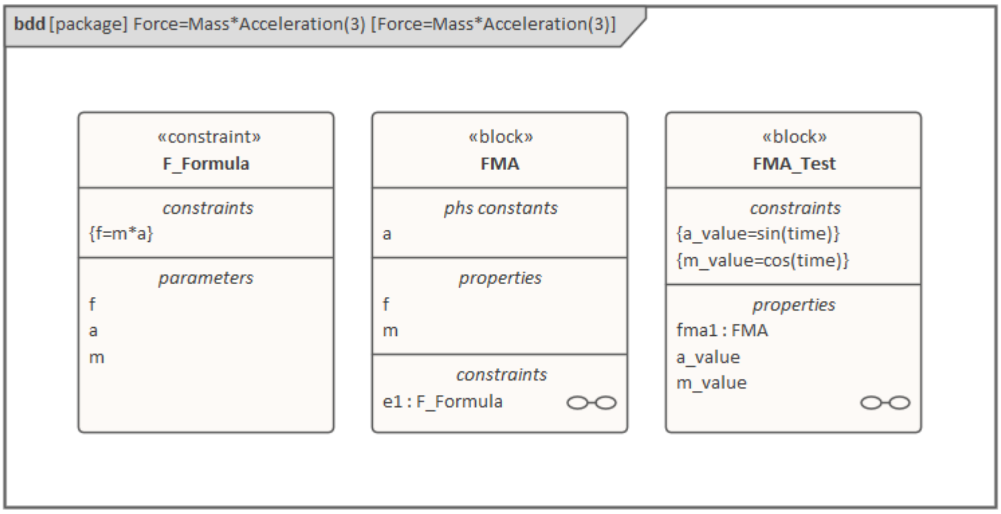
When generating code, Enterprise Architect will collect all the operations stereotyped as 'SimFunction' defined in ConstraintBlocks and Blocks, then generate code resembling this:

```
function LimitValue
   input Real pMin;
   input Real pMax;
   input Real p;
   output Real pLim;
```

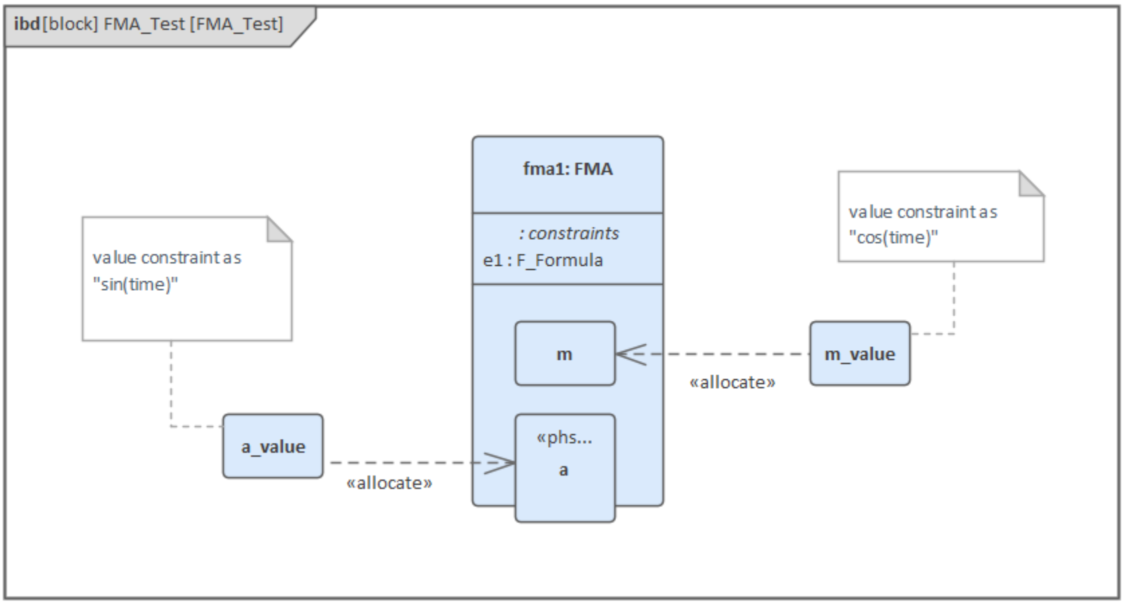
```
        algorithm
            pLim :=
                if p > pMax then
                        pMax
               else if p < pMin then
                        pMin
               else
                       p;
end LimitValue;
```

Value Allocation

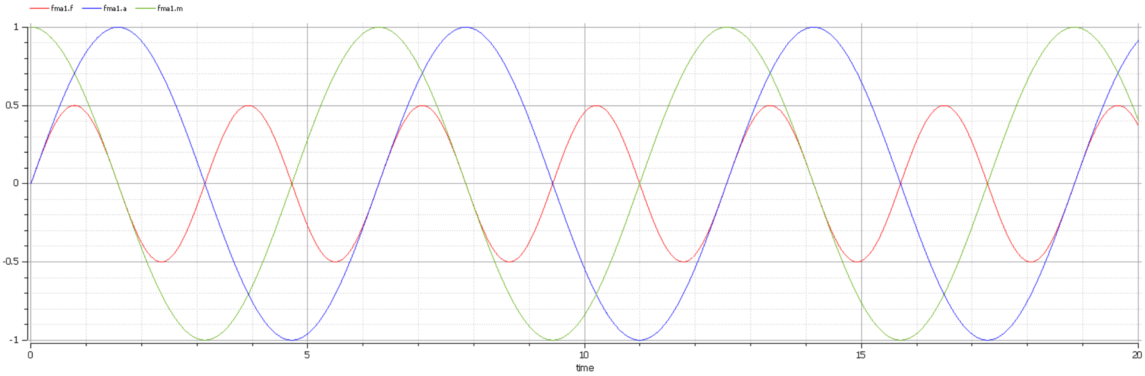
This figure shows a simple model called 'Force=Mass*Acceleration'.



- 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'


ibd [block] FMA_Test [FMA_Test]
fma1: FMA
: constraints
e1 : F_Formula
m
«phs...
a
value constraint as "sin(time)"
value constraint as "cos(time)"
a_value
m_value
«allocate»
«allocate»

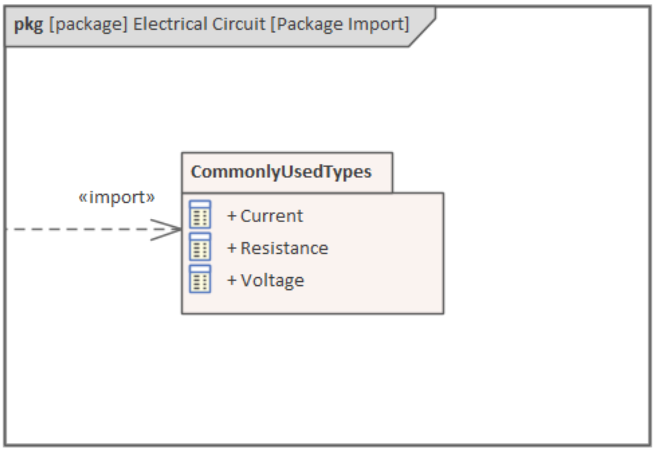
- Select the 'Properties to Plot' checkboxes against 'fma1.a', 'fma1.m' and 'fma1.f'
- Click on the Solve button to simulate the model



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.


pkg [package] Electrical Circuit [Package Import]
«import»
CommonlyUsedTypes
+ Current
+ Resistance
+ Voltage

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

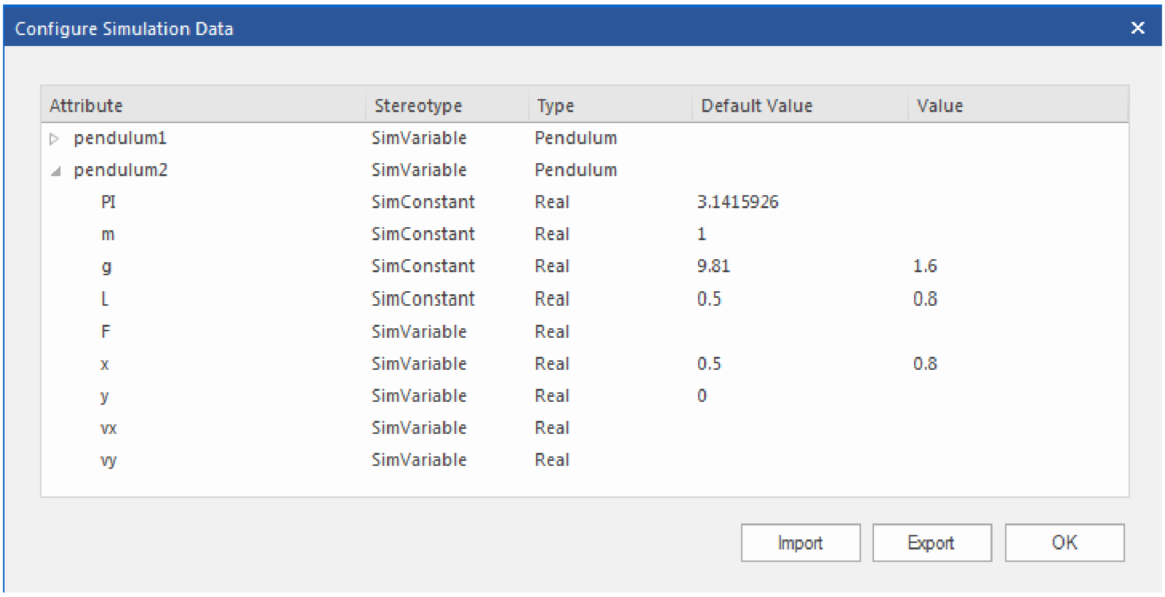
Ribbon	Simulate > System Behavior > Modelica/Simulink > SysMLSim Configuration Manager > in "block" group > Name column > Context menu on block element > Create Simulation DataSet

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.



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

Column	Description
Attribute	The 'Attribute' column provides a tree view of all the properties in the Block being edited.
Stereotype	The 'Stereotype' column identifies, for each property, if it has been configured to be a constant for the duration of the simulation or variable, so that the value is expected to change over time.
Type	The 'Type' column describes the type used for simulation of this property. It can be either a primitive type (such as 'Real') or a reference to a Block contained in the model. Properties referencing Blocks will show the child properties specified by the referenced Block below them.
Default Value	The 'Default Value' column shows the value that will be used in the simulation if no override is provided. This can come from the 'Initial Value' field in the SysML model or from the default dataset of the parent type.
Value	The 'Value' column allows you to override the default value for each primitive value.
Export / Import	Click on these buttons to modify the values in the current dataset using an external application such as a spreadsheet, and then re-import them to the list.

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

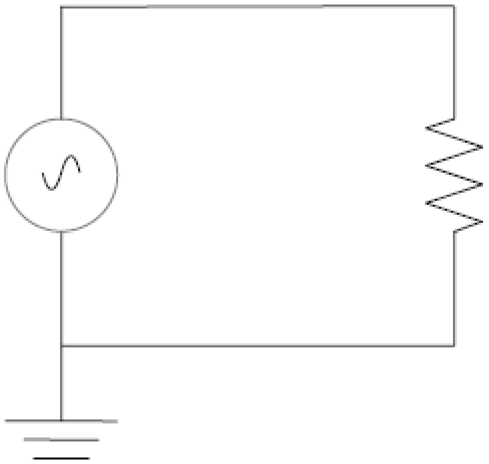
Model	Description
Electrical Circuit Simulation Example	The first example is of the simulation of loading an electrical circuit. The example starts with an electrical circuit diagram and converts it to a parametric model. The model is then simulated and the voltage at the source and target terminals 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 Regulator	The final example shows the water levels of two water tanks where the water is being distributed between them. We first simulate a well-balanced system, then we simulate a system where the water will overflow from the second tank.

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

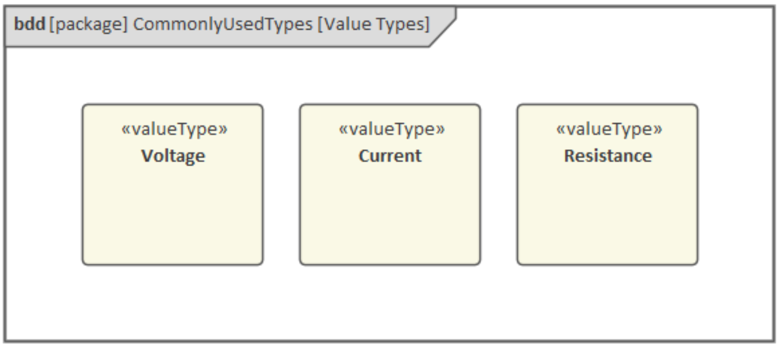
The electrical circuit we are going to model, shown here, uses a standard electrical circuit notation.

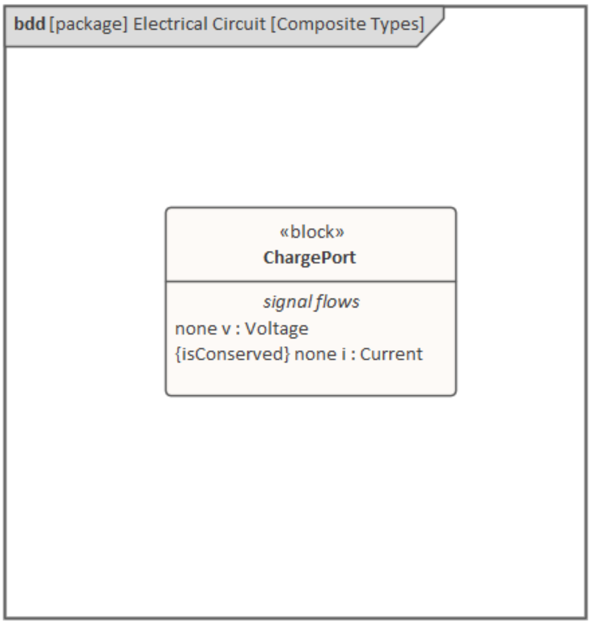


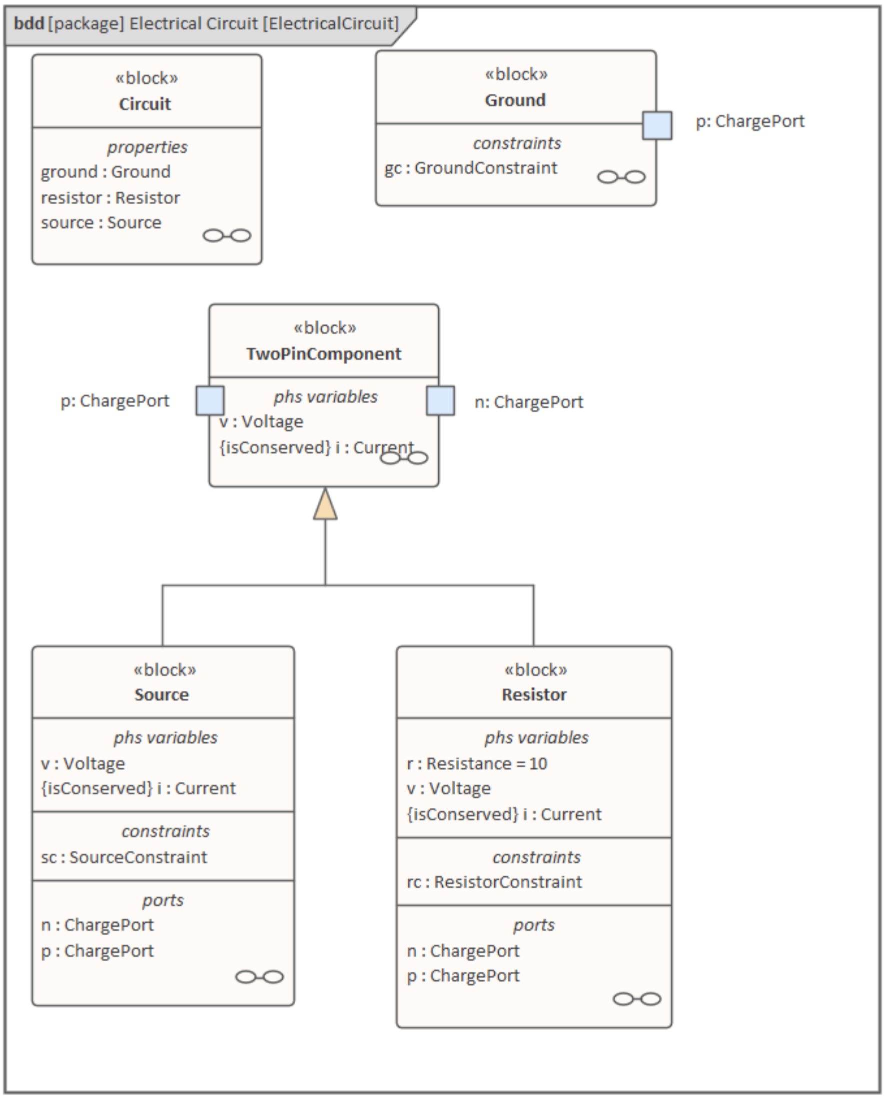
The circuit includes an AC power source, an earth and a resistor, connected to each other by electrical wire.

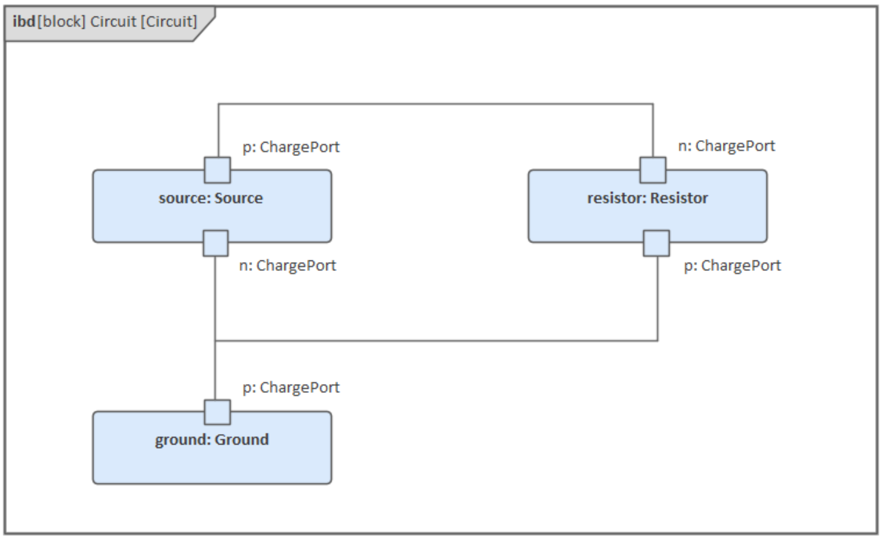
Create SysML Model

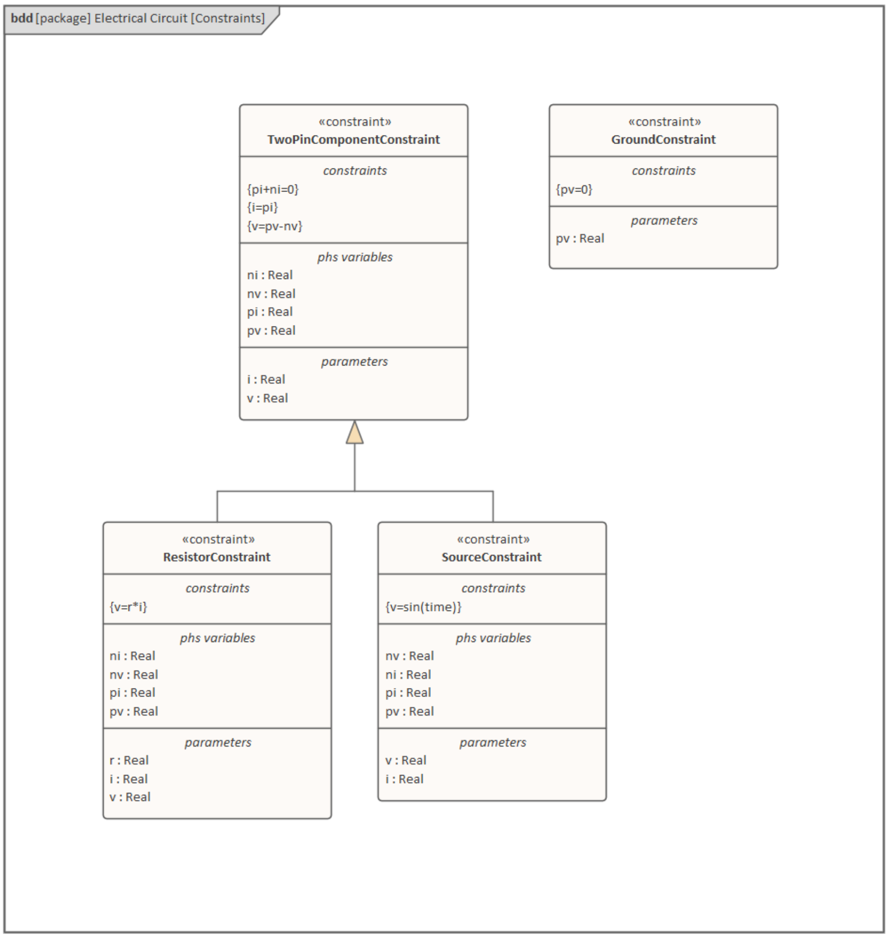
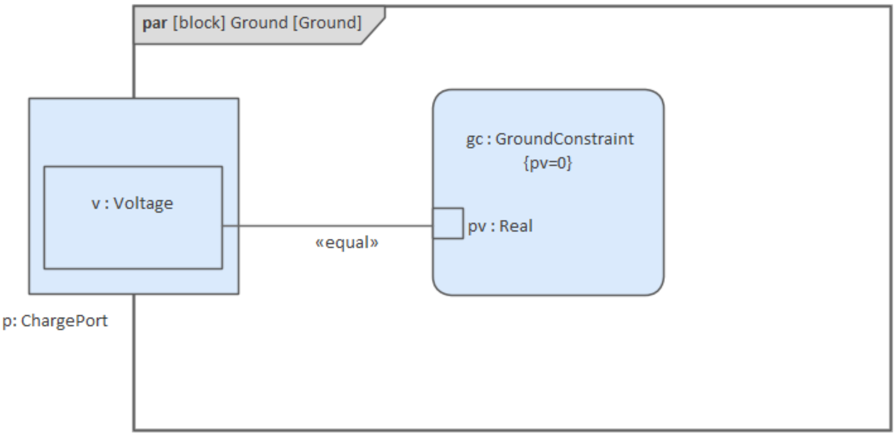
This table shows how we can build up a complete SysML model to represent the circuit, starting at the lowest level types and building up the model one step at a time.

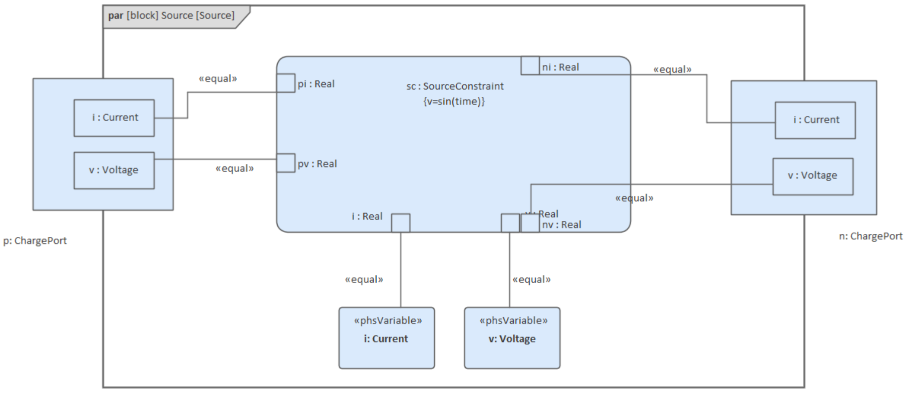
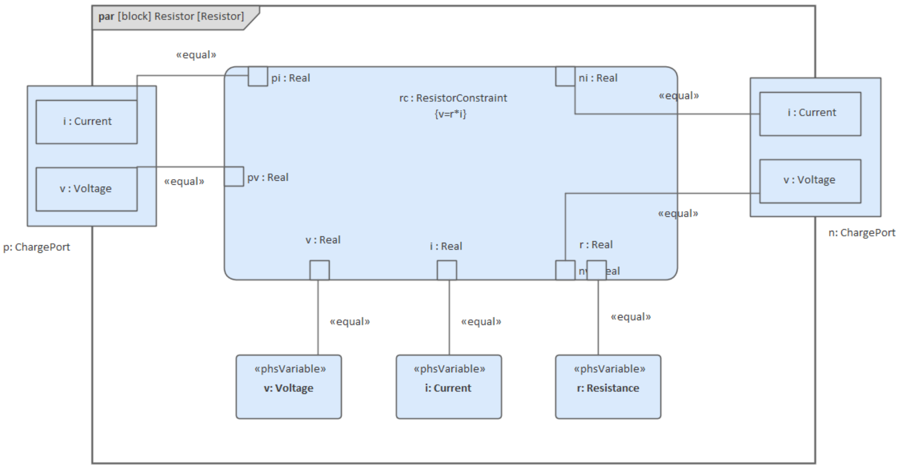
Component	Action
Types	Define Value Types for the Voltage, Current and Resistance. Unit and quantity kind are not important for the purposes of simulation, but would be set if defining a complete SysML model. These types will be generalized from the primitive type 'Real'. In other models, you can choose to map a Value Type to a corresponding simulation type separate from the model.  bdd [package] CommonlyUsedTypes [Value Types] «valueType» Voltage «valueType» Current «valueType» Resistance 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.

	 bdd [package] Electrical Circuit [Composite Types] «block» ChargePort signal flows none v : Voltage {isConserved} none i : Current
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.

	 bdd [package] Electrical Circuit [ElectricalCircuit] «block» Circuit properties ground : Ground resistor : Resistor source : Source «block» Ground constraints gc : GroundConstraint p: ChargePort «block» TwoPinComponent p: ChargePort phs variables v : Voltage {isConserved} i : Current n: ChargePort «block» Source phs variables v : Voltage {isConserved} i : Current constraints sc : SourceConstraint ports n : ChargePort p : ChargePort «block» Resistor phs variables r : Resistance = 10 v : Voltage {isConserved} i : Current constraints rc : ResistorConstraint ports n : ChargePort p : ChargePort
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.

	 ibd [block] Circuit [Circuit] p: ChargePort n: ChargePort source: Source resistor: Resistor n: ChargePort p: ChargePort p: ChargePort ground: Ground 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.

	 bdd [package] Electrical Circuit [Constraints] «constraint» TwoPinComponentConstraint constraints {pi+ni=0} {i=pi} {v=pv-nv} phs variables ni : Real nv : Real pi : Real pv : Real parameters i : Real v : Real «constraint» GroundConstraint constraints {pv=0} parameters pv : Real «constraint» ResistorConstraint constraints {v=r*i} phs variables ni : Real nv : Real pi : Real pv : Real parameters r : Real i : Real v : Real «constraint» SourceConstraint constraints {v=sin(time)} phs variables nv : Real ni : Real pi : Real pv : Real parameters v : Real i : Real
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.  par [block] Ground [Ground] v : Voltage p: ChargePort «equal» gc : GroundConstraint {pv=0} pv : Real 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  par [block] Source [Source] «equal» i : Current v : Voltage «equal» pi : Real pv : Real sc : SourceConstraint {v=sin(time)} ni : Real «equal» i : Current v : Voltage «equal» i : Real Real nv : Real p: ChargePort n: ChargePort «equal» «equal» «phsVariable» i: Current «phsVariable» v: Voltage 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  par [block] Resistor [Resistor] «equal» i : Current v : Voltage «equal» pi : Real pv : Real rc : ResistorConstraint {v=r*i} ni : Real «equal» i : Current v : Voltage «equal» v : Real i : Real r : Real eal p: ChargePort n: ChargePort «equal» «equal» «equal» «phsVariable» v: Voltage «phsVariable» i: Current «phsVariable» r: Resistance

Configure Simulation Behavior

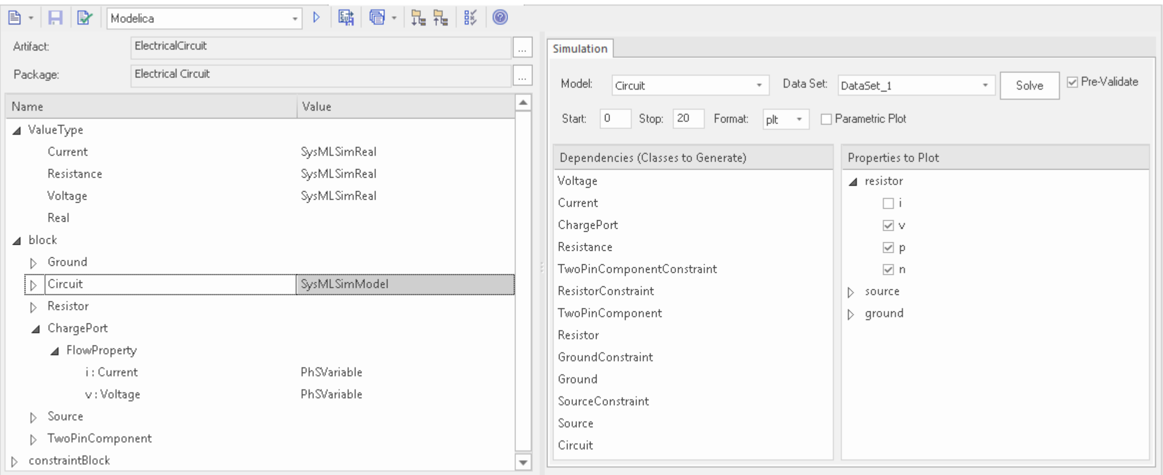
This table shows the detailed steps of the configuration of SysMLSim.

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.



Modelica

Artifact: ElectricalCircuit

Package: Electrical Circuit

Name	Value
ValueType	
Current	SysMLSimReal
Resistance	SysMLSimReal
Voltage	SysMLSimReal
Real	
block	
Ground	
Circuit	SysMLSimModel
Resistor	
ChargePort	
FlowProperty	
i : Current	PhSVariable
v : Voltage	PhSVariable
Source	
TwoPinComponent	
constraintBlock	

Simulation

Model: Circuit Data Set: DataSet_1 Solve ☑ Pre-Validate

Start: 0 Stop: 20 Format: plt □ Parametric Plot

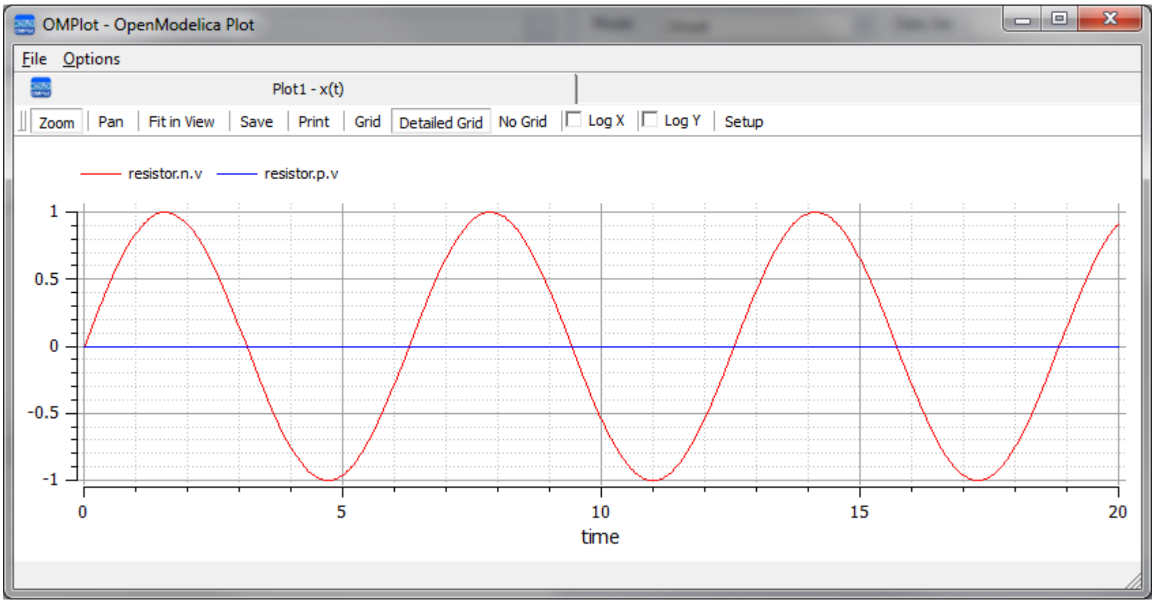
Dependencies (Classes to Generate)

Voltage
Current
ChargePort
Resistance
TwoPinComponentConstraint
ResistorConstraint
TwoPinComponent
Resistor
GroundConstraint
Ground
SourceConstraint
Source
Circuit

Properties to Plot

- resistor
  - □ i
  - ☑ v
  - ☑ p
  - ☑ n
- source
- ground

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

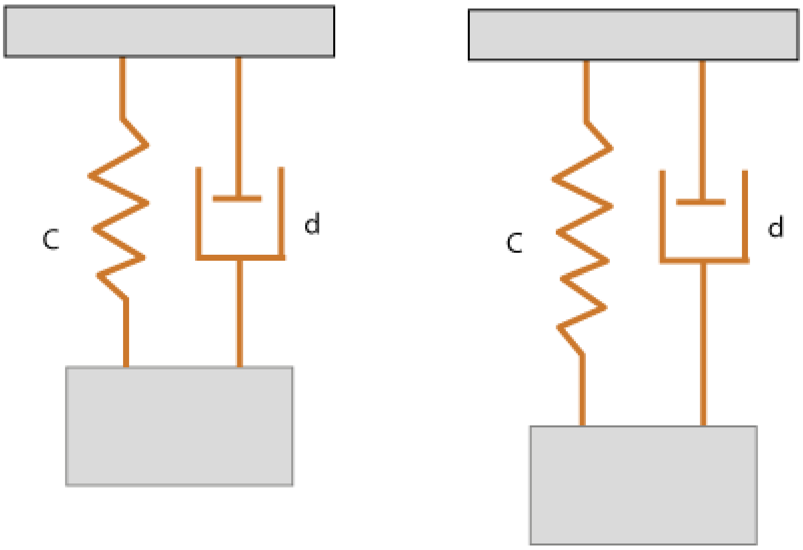

OMPlot - OpenModelica Plot
File Options
Plot1 - x(t)
Zoom
Pan
Fit in View
Save
Print
Grid
Detailed Grid
No Grid
□ Log X
□ Log Y
Setup
resistor.n.v
resistor.p.v
1
0.5
0
-0.5
-1
0
5
10
15
20
time

Mass-Spring-Damper Oscillator Simulation Example

In this section, we will walk through the creation of a SysML parametric model for a simple Oscillator composed of 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

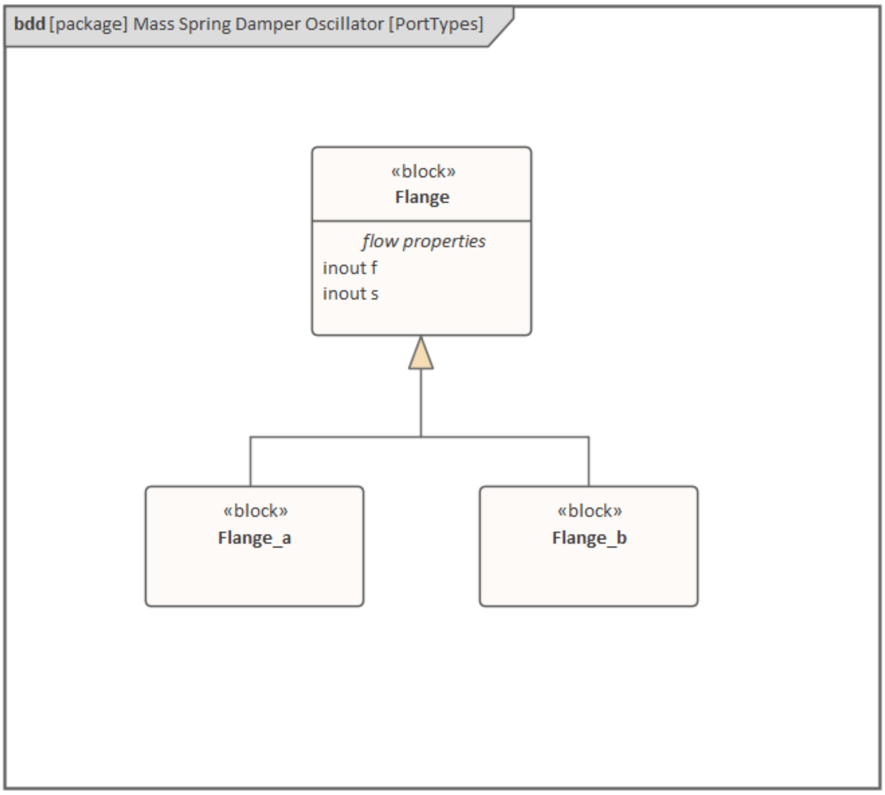
A mass is hanging on a spring and damper. The first state shown here represents the initial point at time=0, just when the mass is released. The second state represents the final point when the body is at rest and the spring forces are in equilibrium with gravity.

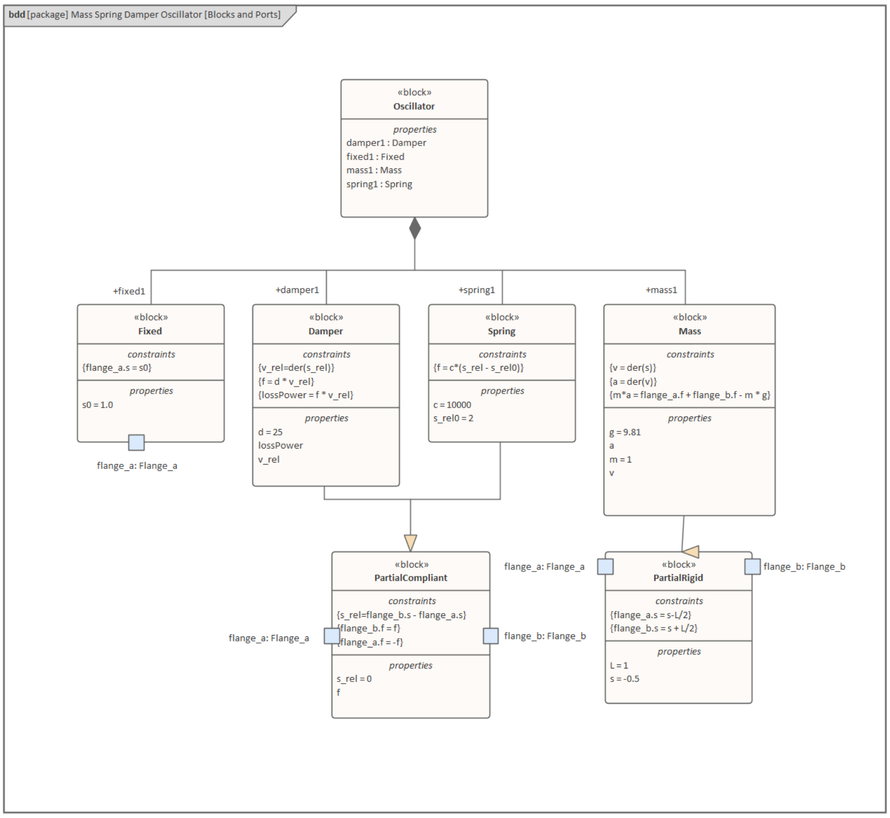


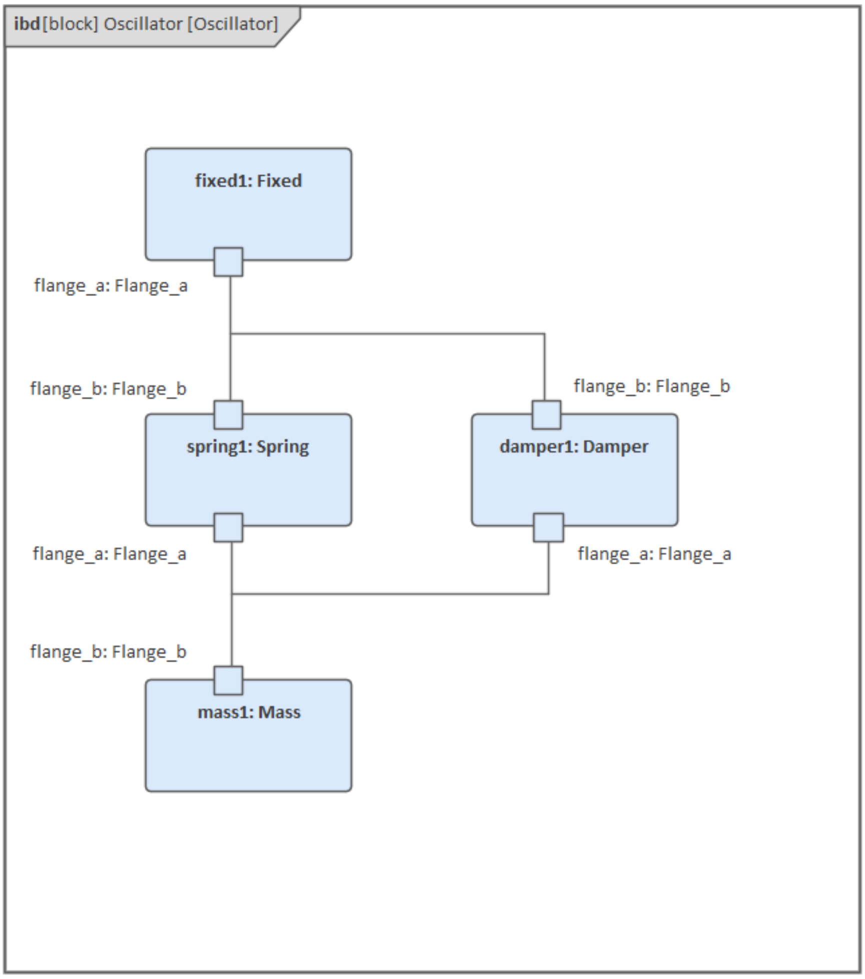
Create SysML Model

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.

	 bdd [package] Mass Spring Damper Oscillator [PortTypes] «block» Flange flow properties inout f inout s «block» Flange_a «block» Flange_b
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

	 bdd [package] Mass Spring Damper Oscillator [Blocks and Ports] «block» Oscillator properties damper1 : Damper fixed1 : Fixed mass1 : Mass spring1 : Spring +fixed1 +damper1 +spring1 +mass1 «block» Fixed constraints {flange_a.s = s0} properties s0 = 1.0 flange_a: Flange_a «block» Damper constraints {v_rel=der(s_rel)} {f = d * v_rel} {lossPower = f * v_rel} properties d = 25 lossPower v_rel «block» Spring constraints {f = c*(s_rel - s_rel0)} properties c = 10000 s_rel0 = 2 «block» Mass constraints {v = der(s)} {a = der(v)} {m*a = flange_a.f + flange_b.f - m * g} properties g = 9.81 a m = 1 v «block» PartialCompliant constraints {s_rel=flange_b.s - flange_a.s} {flange_b.f = f} {flange_a.f = -f} properties s_rel = 0 f flange_a: Flange_a flange_b: Flange_b «block» PartialRigid constraints {flange_a.s = s-L/2} {flange_b.s = s + L/2} properties L = 1 s = -0.5 flange_a: Flange_a flange_b: Flange_b
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'

	 ibd [block] Oscillator [Oscillator] fixed1: Fixed flange_a: Flange_a flange_b: Flange_b spring1: Spring flange_b: Flange_b damper1: Damper flange_a: Flange_a flange_a: Flange_a flange_b: Flange_b mass1: Mass
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.

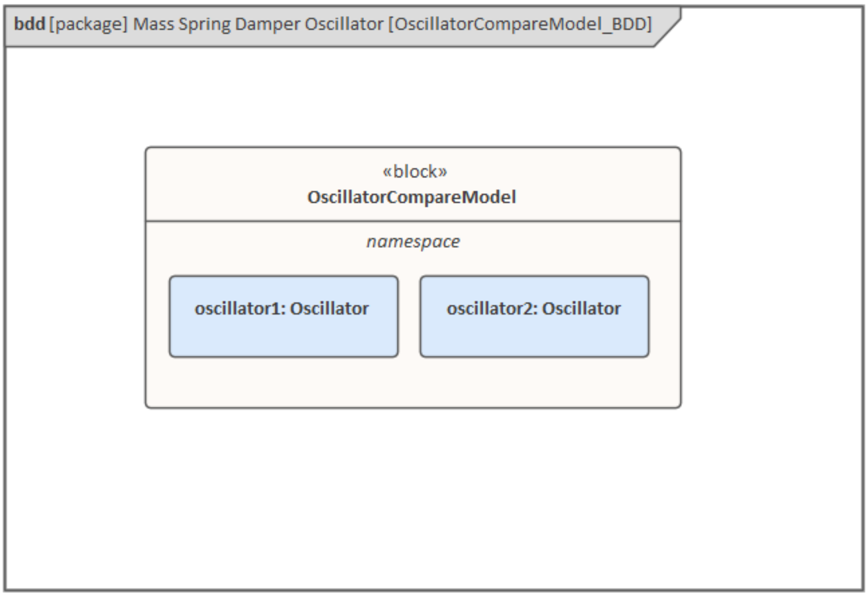
Two Oscillator Compare Plan

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*

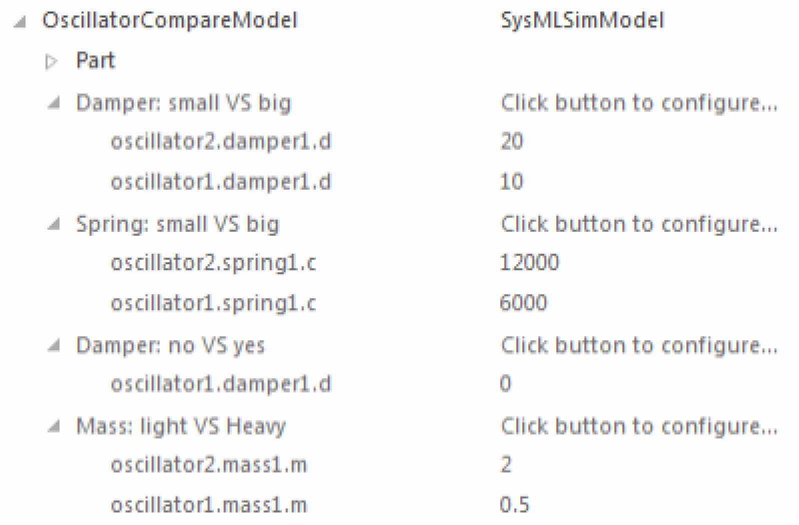


Setup DataSet and Run Simulation

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

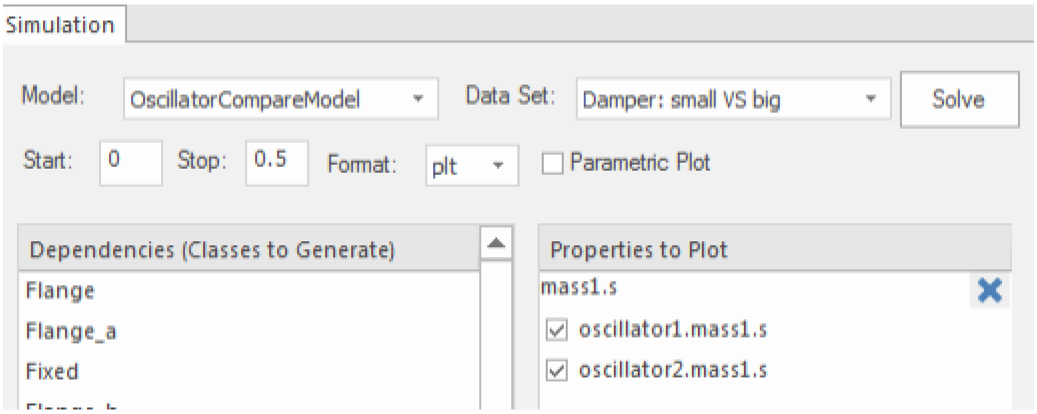
The configured page resembles this:



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

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

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



Simulation

Model: OscillatorCompareModel

Data Set: Damper: small VS big

Solve

Start: 0 Stop: 0.5 Format: plt

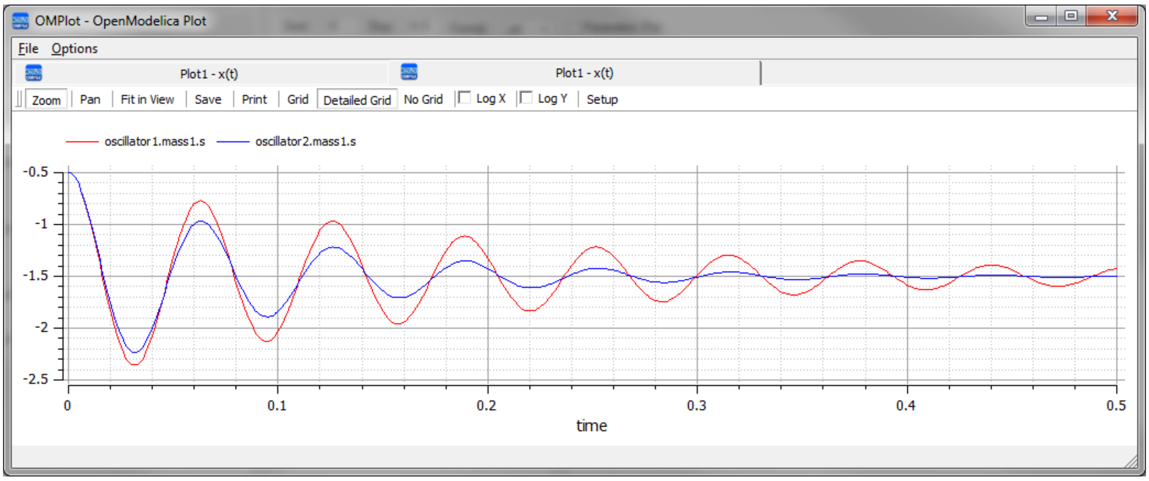
□ Parametric Plot

Dependencies (Classes to Generate)
Flange
Flange_a
Fixed

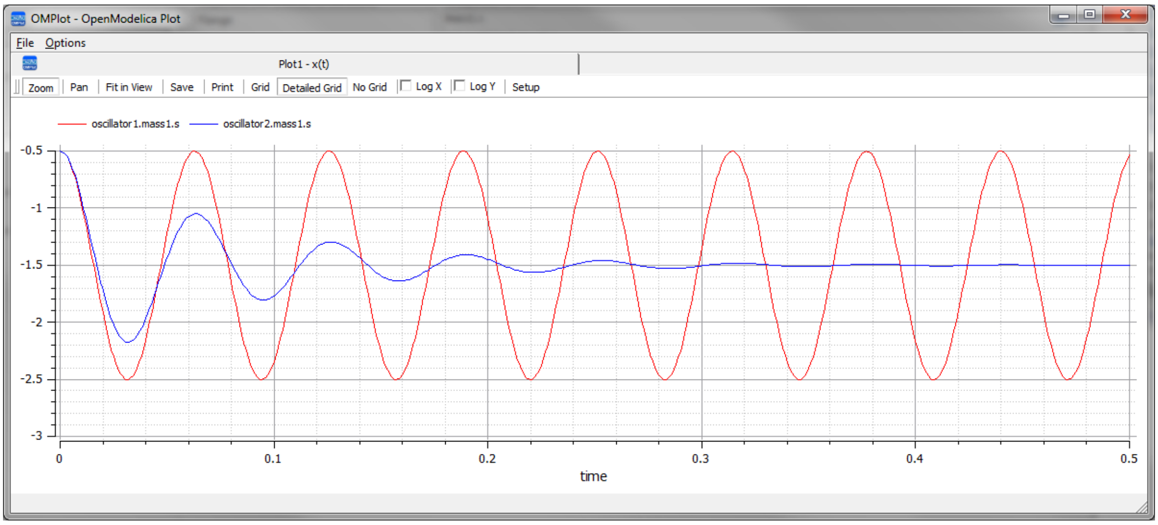
Properties to Plot
mass1.s
☑ oscillator1.mass1.s
☑ oscillator2.mass1.s

These are the simulation results:

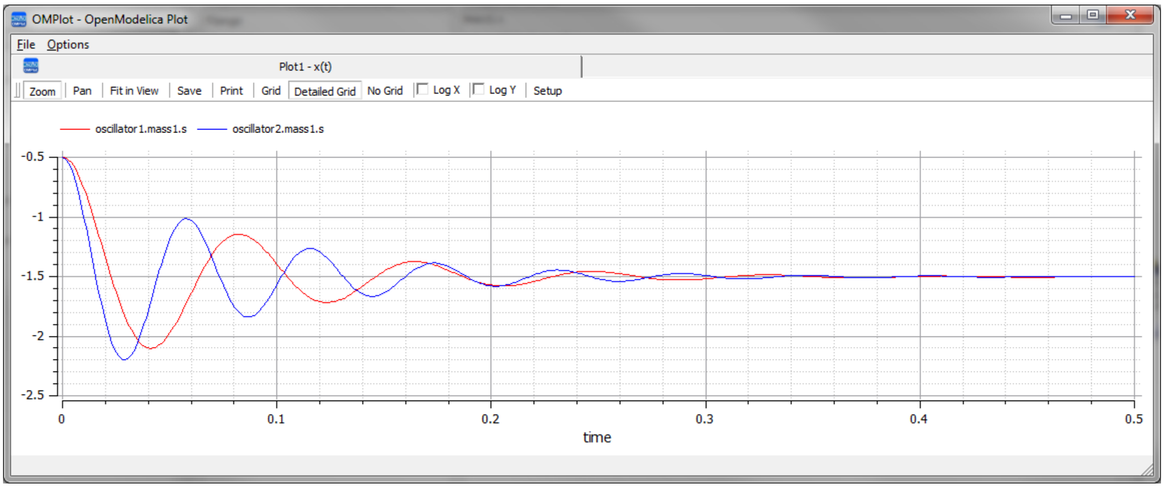
- Damper, small vs big: the smaller damper allows the body to oscillate more



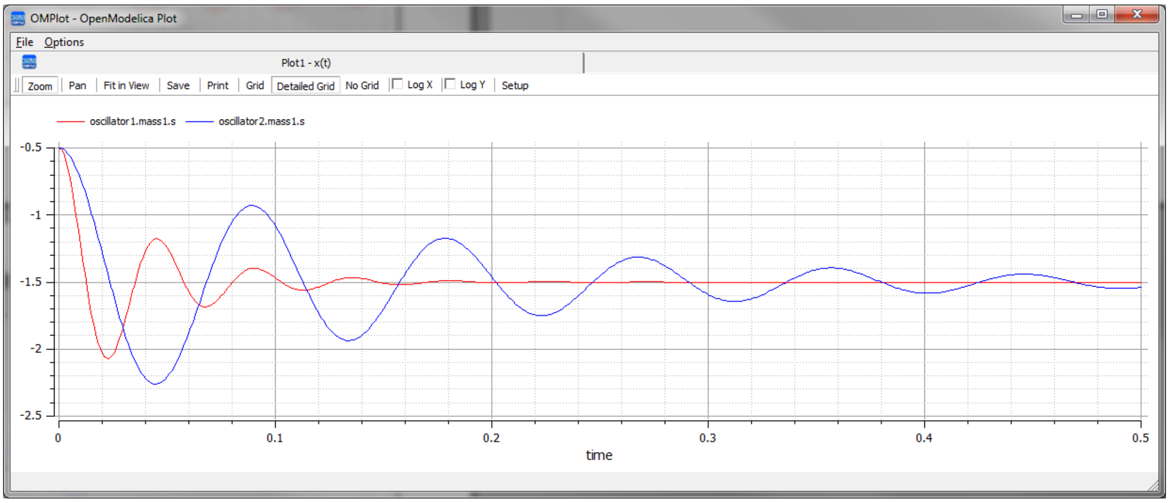
- Damper, no vs yes: the oscillator never stops without a damper


OMPlot - OpenModelica Plot
File Options
Plot1 - x(t)
Zoom Pan Fit in View Save Print Grid Detailed Grid No Grid □ Log X □ Log Y Setup
oscillator1.mass1.s
oscillator2.mass1.s
-0.5
-1
-1.5
-2
-2.5
-3
0
0.1
0.2
0.3
0.4
0.5
time

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


OMPlot - OpenModelica Plot
File Options
Plot1 - x(t)
Zoom Pan Fit in View Save Print Grid Detailed Grid No Grid □ Log X □ Log Y Setup
oscillator1.mass1.s
oscillator2.mass1.s
-0.5
-1
-1.5
-2
-2.5
0
0.1
0.2
0.3
0.4
0.5
time

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


OMPlot - OpenModelica Plot
File Options
Plot1 - x(t)
Zoom Pan Fit in View Save Print Grid Detailed Grid No Grid □ Log X □ Log Y Setup
oscillator1.mass1.s
oscillator2.mass1.s
-0.5
-1
-1.5
-2
-2.5
0
0.1
0.2
0.3
0.4
0.5
time

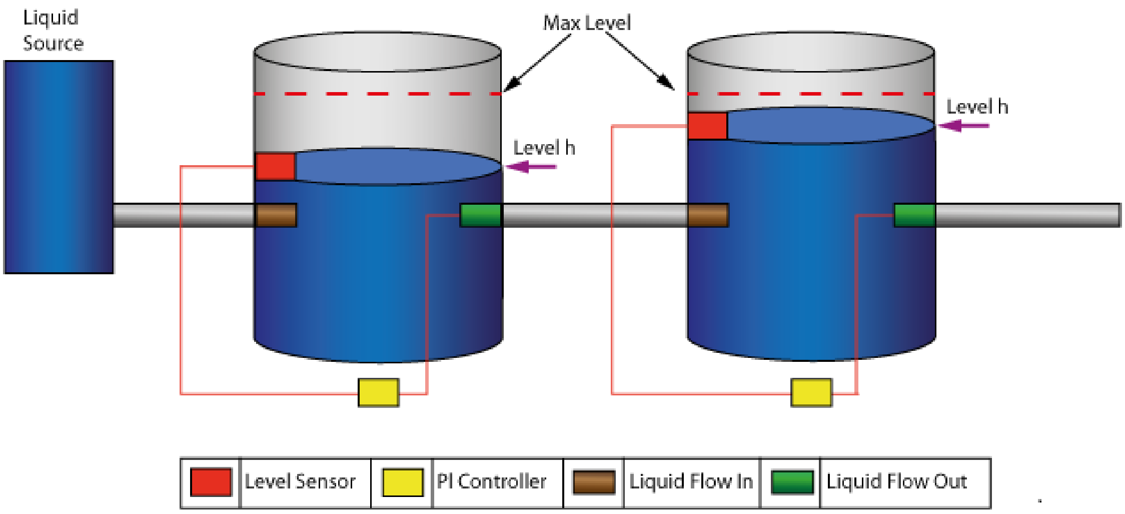
Water Tank Pressure Regulator

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.

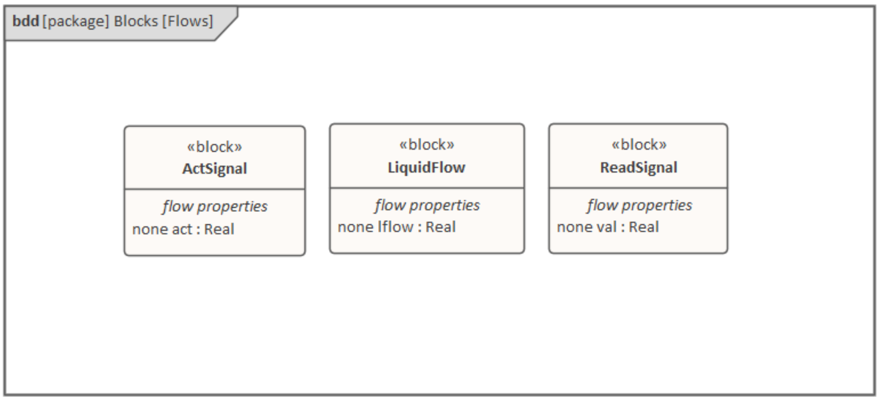
System being modeled

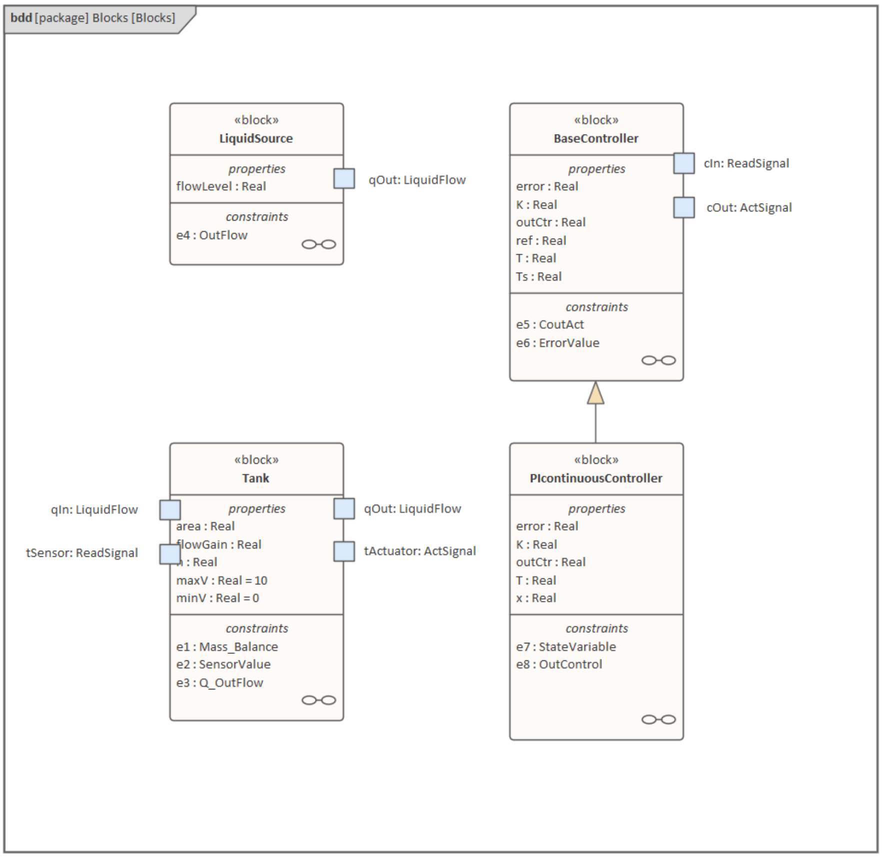
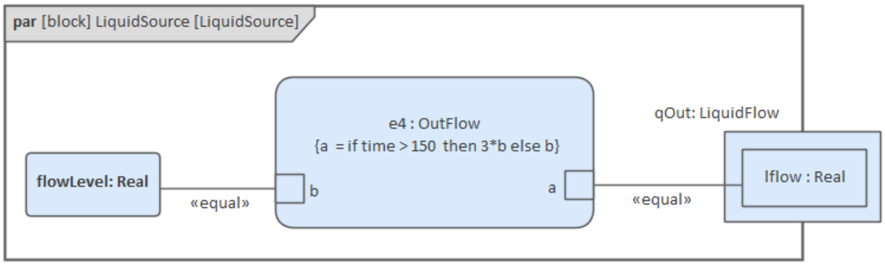
This diagram depicts two tanks connected together, and a water source that fills the first tank. Each tank has a proportional–integral (PI) continuous controller connected to it, which regulates the level of water contained in the tanks at a reference level. While the source fills the first tank with 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.

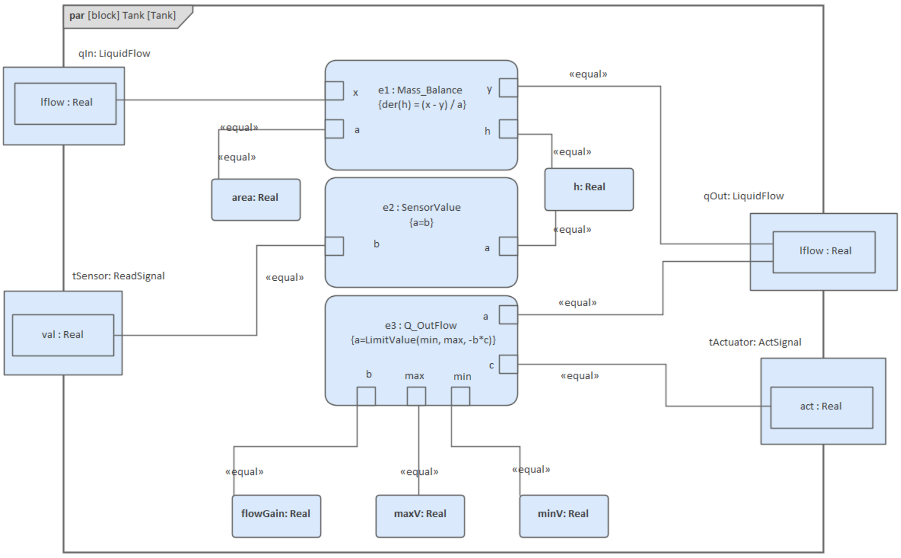


Create SysML Model

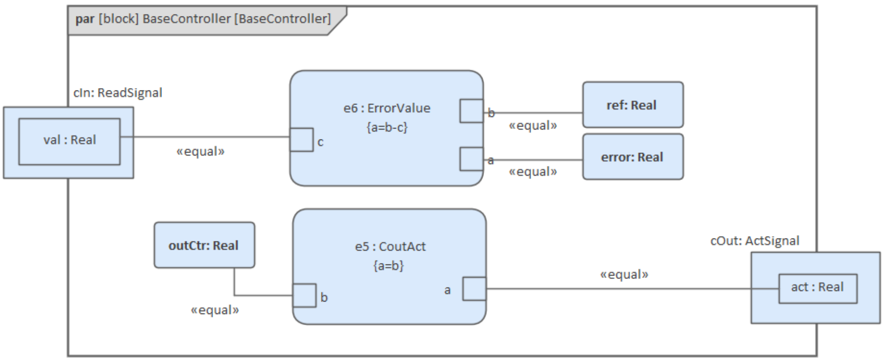
Component	Discussion
Port Types	The tank has four Ports that are typed to these three Blocks: • ReadSignal: Reading the fluid level; this has a property 'val' with unit 'm' • ActSignal: The signal to the actuator for setting valve position • LiquidFlow: The liquid flow at inlets or outlets; this has a property 'lflow' with unit 'm^3/s"

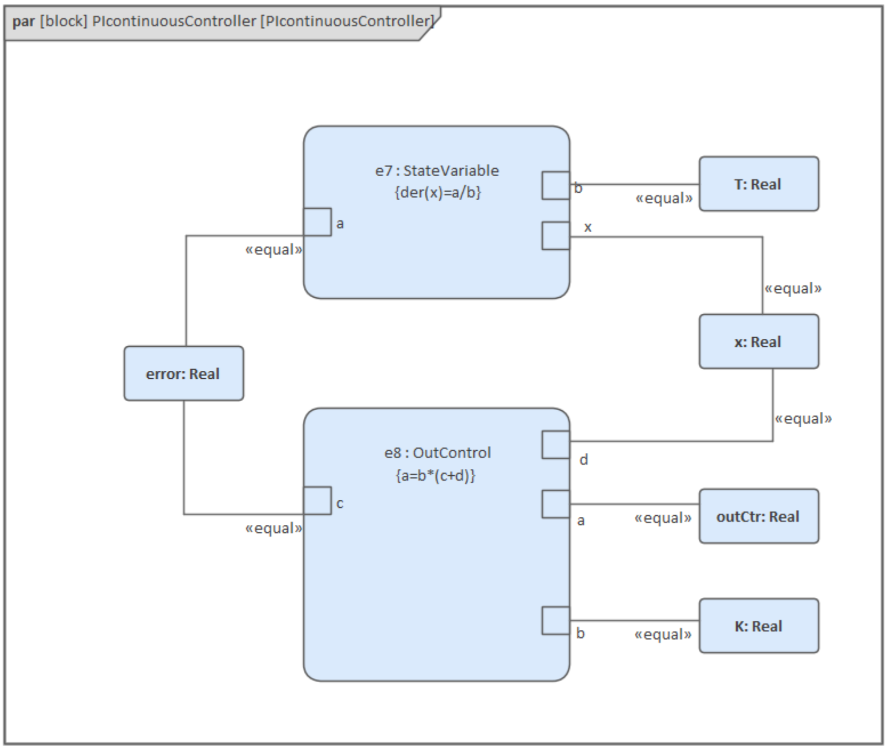
	 bdd [package] Blocks [Flows] «block» ActSignal flow properties none act : Real «block» LiquidFlow flow properties none lflow : Real «block» ReadSignal flow properties none val : Real
Block Definition Diagram	LiquidSource: The water entering the tank must come from somewhere, therefore we have a liquid source component in the tank system, with the property *flowLevel* having a unit of 'm^3/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='m^3'): 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 = 'm^3/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

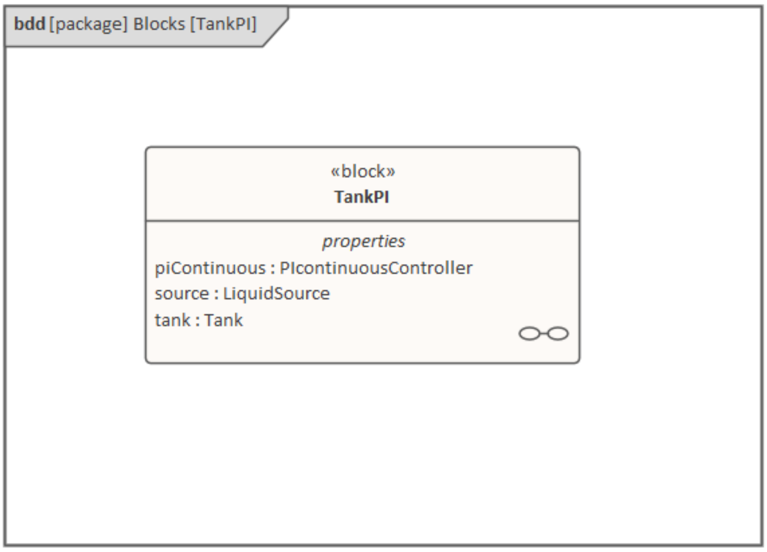
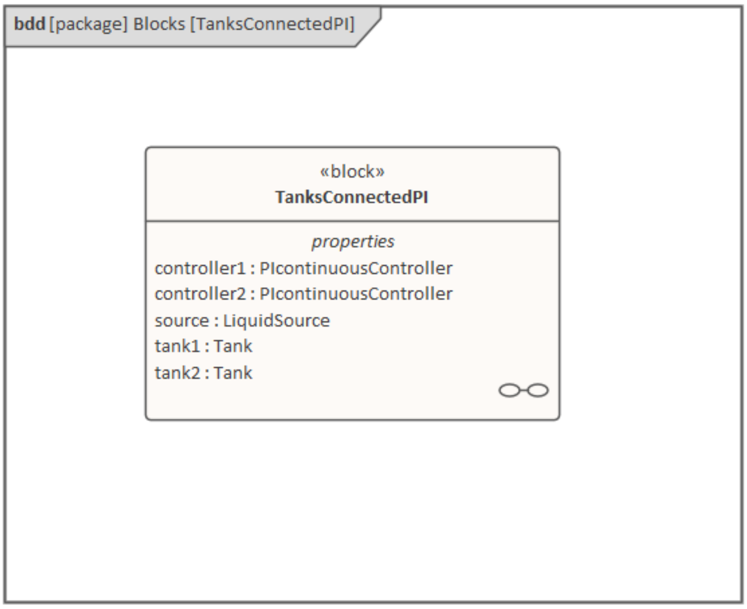
	 bdd [package] Blocks [Blocks] «block» LiquidSource properties flowLevel : Real constraints e4 : OutFlow qOut: LiquidFlow «block» BaseController properties error : Real K : Real outCtr : Real ref : Real T : Real Ts : Real constraints e5 : CoutAct e6 : ErrorValue cIn: ReadSignal cOut: ActSignal «block» Tank properties area : Real flowGain : Real : Real maxV : Real = 10 minV : Real = 0 constraints e1 : Mass_Balance e2 : SensorValue e3 : Q_OutFlow qIn: LiquidFlow tSensor: ReadSignal qOut: LiquidFlow tActuator: ActSignal «block» PIcontinuousController properties error : Real K : Real outCtr : Real T : Real x : Real constraints e7 : StateVariable e8 : OutControl
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.  par [block] LiquidSource [LiquidSource] flowLevel: Real «equal» e4 : OutFlow {a = if time > 150 then 3*b else b} b a «equal» qOut: LiquidFlow lflow : Real 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.  bdd [package] Blocks [TankPI] «block» TankPI properties piContinuous : PIcontinuousController source : LiquidSource tank : Tank This is the Internal Block diagram for a system with two connected tanks.  bdd [package] Blocks [TanksConnectedPI] «block» TanksConnectedPI properties controller1 : PIcontinuousController controller2 : PIcontinuousController source : LiquidSource tank1 : Tank tank2 : Tank

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.

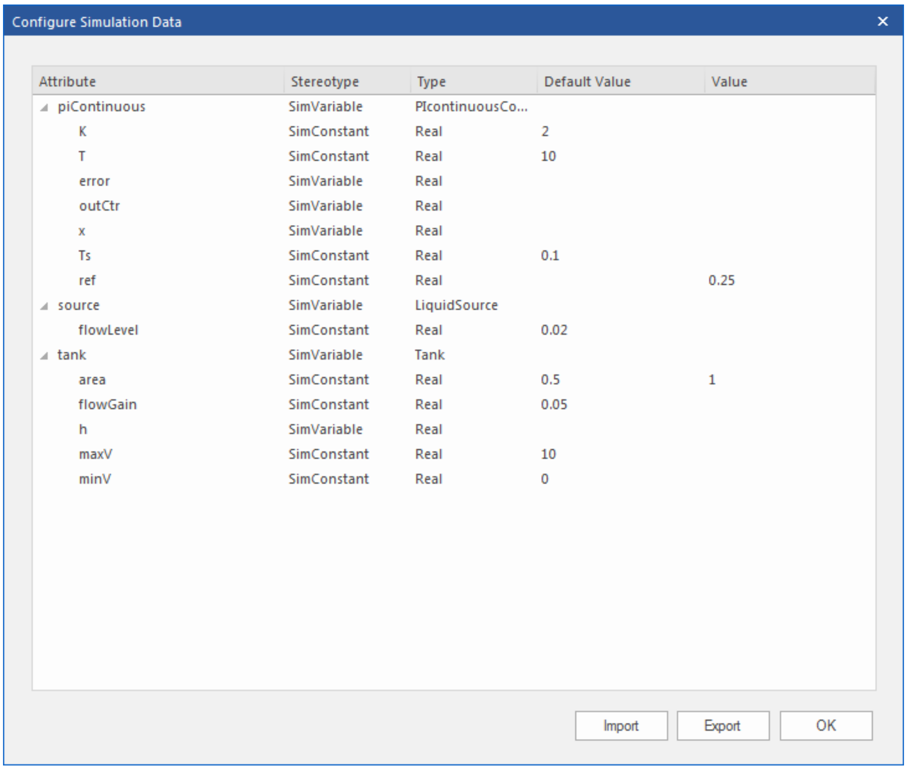
Block	Properties
LiquidSource	Configure as 'SysMLSimClass'. Properties configuration: • flowLevel: set as 'SimConstant'
Tank	Configure as 'SysMLSimClass'. Properties configuration: • area: set as 'SimConstant' • flowGain: set as 'SimConstant' • maxV: set as 'SimConstant' • minV: set as 'SimConstant'
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

<table>
<tr><td></td><td>maxV: 10
minV: 0</td></tr>
<tr><td>BaseController</td><td>T: 10
K: 2
Ts: 0.1</td></tr>
<tr><td>PIcontinuousController</td><td>No configuration needed.
By default, the specific Block will use the configured values from super Block's default dataSet.</td></tr>
<tr><td>TankPI</td><td>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.



Configure Simulation Data

<table>
<tr><th>Attribute</th><th>Stereotype</th><th>Type</th><th>Default Value</th><th>Value</th></tr>
<tr><td>piContinuous</td><td>SimVariable</td><td>PIcontinuousCo...</td><td></td><td></td></tr>
<tr><td>K</td><td>SimConstant</td><td>Real</td><td>2</td><td></td></tr>
<tr><td>T</td><td>SimConstant</td><td>Real</td><td>10</td><td></td></tr>
<tr><td>error</td><td>SimVariable</td><td>Real</td><td></td><td></td></tr>
<tr><td>outCtr</td><td>SimVariable</td><td>Real</td><td></td><td></td></tr>
<tr><td>x</td><td>SimVariable</td><td>Real</td><td></td><td></td></tr>
<tr><td>Ts</td><td>SimConstant</td><td>Real</td><td>0.1</td><td></td></tr>
<tr><td>ref</td><td>SimConstant</td><td>Real</td><td></td><td>0.25</td></tr>
<tr><td>source</td><td>SimVariable</td><td>LiquidSource</td><td></td><td></td></tr>
<tr><td>flowLevel</td><td>SimConstant</td><td>Real</td><td>0.02</td><td></td></tr>
<tr><td>tank</td><td>SimVariable</td><td>Tank</td><td></td><td></td></tr>
<tr><td>area</td><td>SimConstant</td><td>Real</td><td>0.5</td><td>1</td></tr>
<tr><td>flowGain</td><td>SimConstant</td><td>Real</td><td>0.05</td><td></td></tr>
<tr><td>h</td><td>SimVariable</td><td>Real</td><td></td><td></td></tr>
<tr><td>maxV</td><td>SimConstant</td><td>Real</td><td>10</td><td></td></tr>
<tr><td>minV</td><td>SimConstant</td><td>Real</td><td>0</td><td></td></tr>
</table>

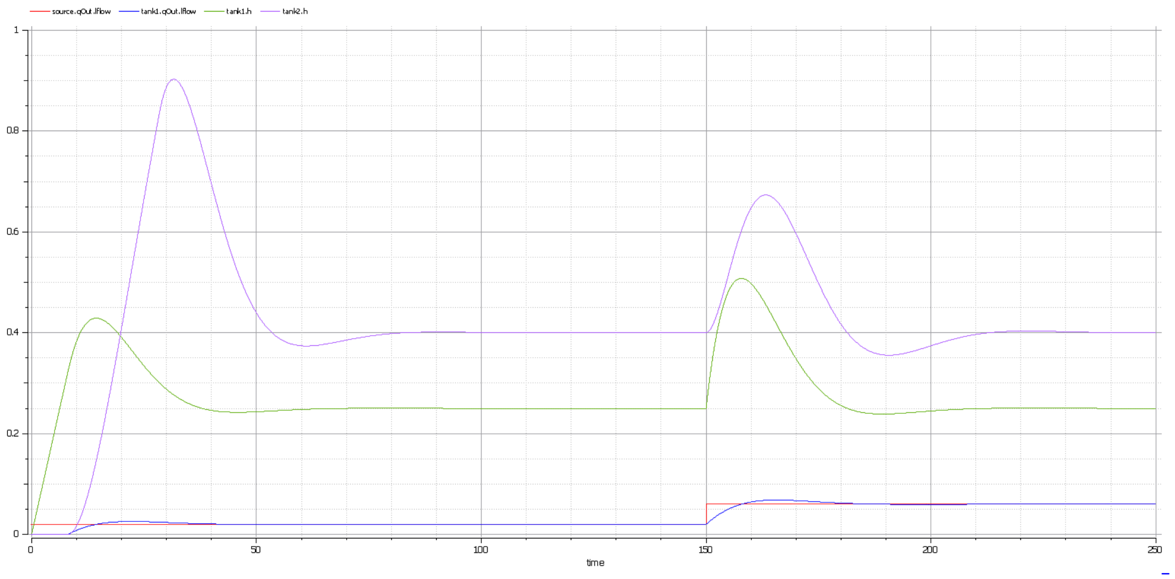
Click on the OK button and return to the Configuration Manager. Then these values are configured:

- tank.area: 1 this overrides the default value 0.5 defined in the Tank Block's data set
- piContinuous.ref: 0.25
</td></tr>
<tr><td>TanksConnectedPI</td><td>- controller1.ref: 0.25
- controller2.ref: 0.4</td></tr>
</table>

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:

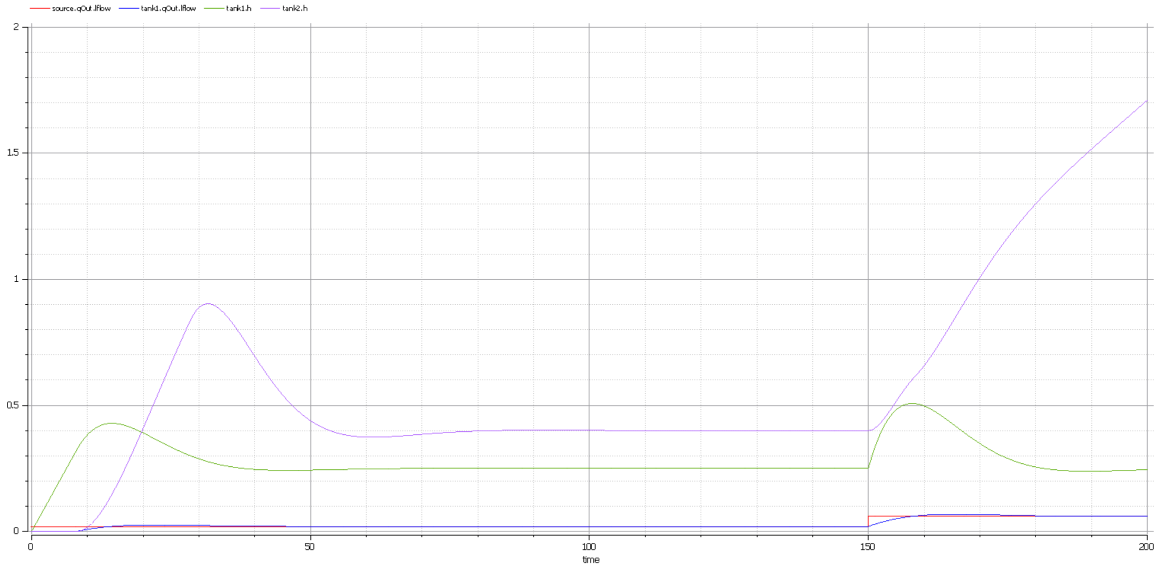
- The liquid flow increases sharply at time=150, to 0.06 m^3/s, a factor of three of the previous flow (0.02 m^3/s)
- Tank1 regulated at height 0.25 and tank2 regulated at height 0.4 as expected (we set the parameter value through the data set)
- Both tank1 and tank2 regulated twice during the simulation;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