



Enterprise Architect

User Guide Series

Dynamic Simulations

How to simulate model behavior? In Sparx Systems Enterprise Architect, you spawn a dynamic process simulating the execution of conceptual behavioral models, analyzing constructs directly in a dynamic environment in which changes can be tested quickly.

Author: Sparx Systems

Date: 2026-05-28

Version: 17.2

CREATED WITH  ENTERPRISE
ARCHITECT

Table of Contents

Dynamic Simulations	4
How it Looks	10
Simulation Windows	12
Set Up Simulation Script	17
Activate Simulation Script	22
Run Model Simulation	24
Simulation Breakpoints	28
Objects and Instances in Simulation	31
Create Objects in a Simulation	34
Destroy Objects in a Simulation	39
Dynamic Simulation with JavaScript	43
Call Behaviors	49
Interaction Operand Condition and Message	
Behavior	52
Guards and Effects	55
Triggers	60
Action Behavior By Type	64
Structured Activity Simulation	67
Activity Return Value Simulation	71
Simulation Events Window	75
Waiting Triggers	82
Re-Signal Triggers	84
Multi-threading - Forks and Joins	86
Trigger Parameters	88

Trigger Sets and Auto-Firing.....	91
Using Trigger Sets to Simulate an Event Sequence.....	96
Multi-threading - Concurrent State Regions.....	98
Using Composite Diagrams.....	100
Win32 User Interface Simulation.....	103
Supported Win32 UI Controls.....	108
Win32 Control Tagged Values.....	128
BPMN Simulation.....	131
Create a BPMN Simulation Model.....	133
Initialize Variables and Conditions.....	136
Comparison of UML Activities and BPMN Processes.....	139

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

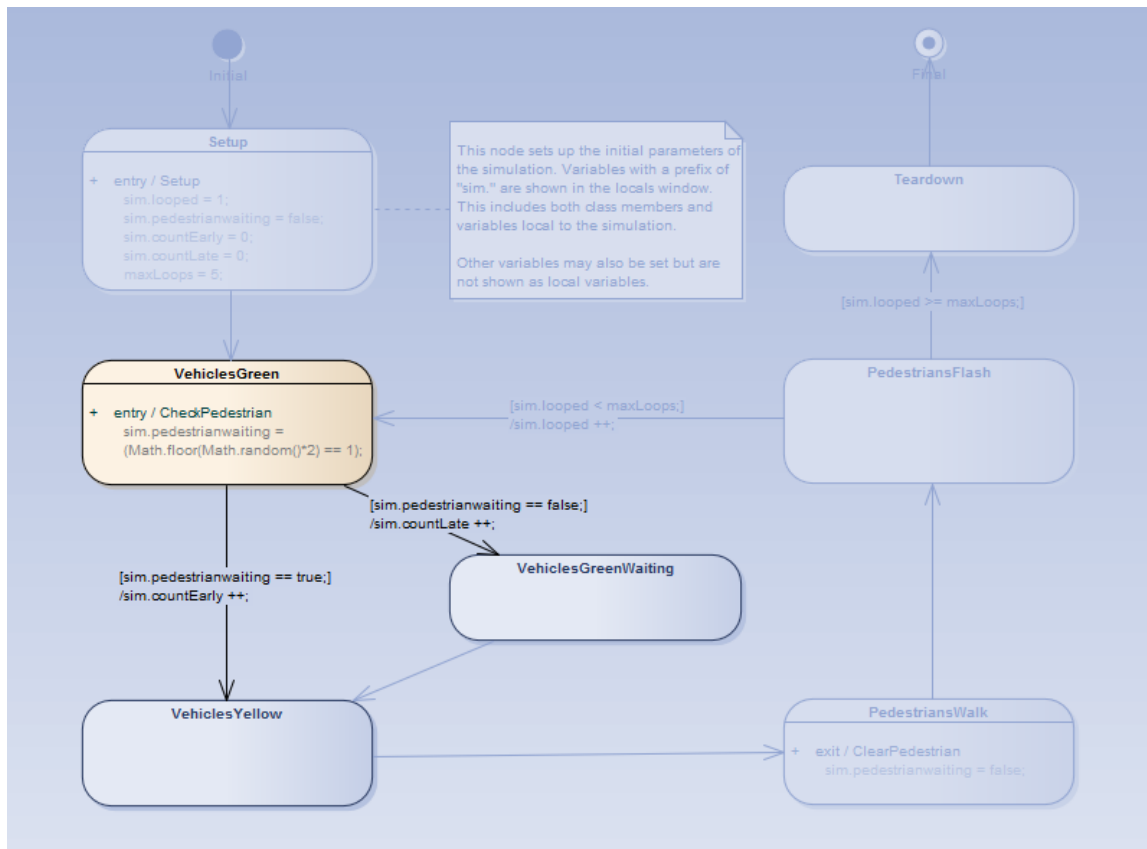
Enterprise Architect has a special way of displaying model information during Simulation. This helps focus attention on the executing or active nodes.

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.



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

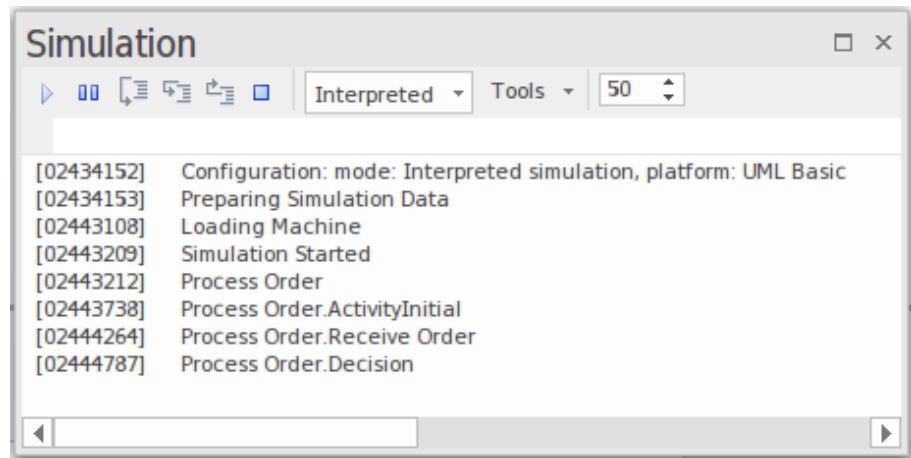
Ribbon	Simulate > Dynamic Simulation > Simulator > Open Simulation Window
--------	--

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.

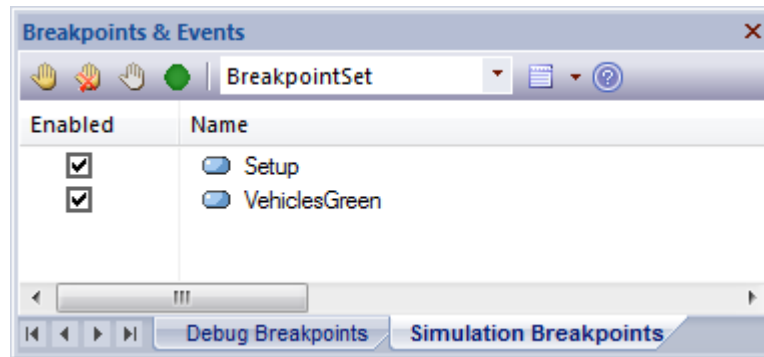
Note the text entry box just underneath the toolbar. This is the Console input area - here you can type simple JavaScript commands such as: `this.count = 4;` to dynamically change a simulation variable named 'count' to 4. In this way you can dynamically influence simulation at run-time.



Breakpoints & Events Window

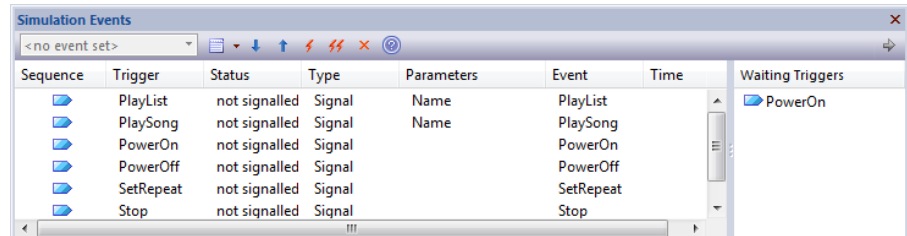
The Simulation process also makes use of the 'Simulation Breakpoints' tab of the Breakpoints & Markers window ('Simulate > Dynamic Simulation > Breakpoints'). Here you set execution breakpoints on specific elements and messages in a Simulation. See the

Simulation Breakpoints topic for more details.



Simulation Events Window

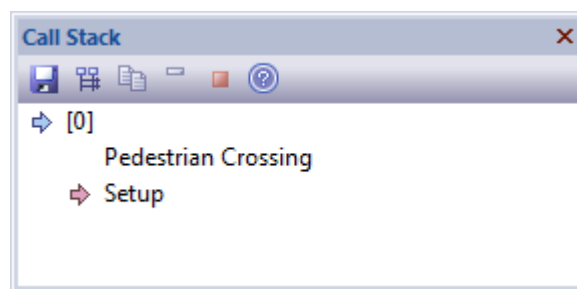
The Simulation Events window ('Simulate > Dynamic Simulation > Events') provides tools to manage and execute triggers. Triggers are used to control the execution of StateMachine transitions.



Call Stack Window

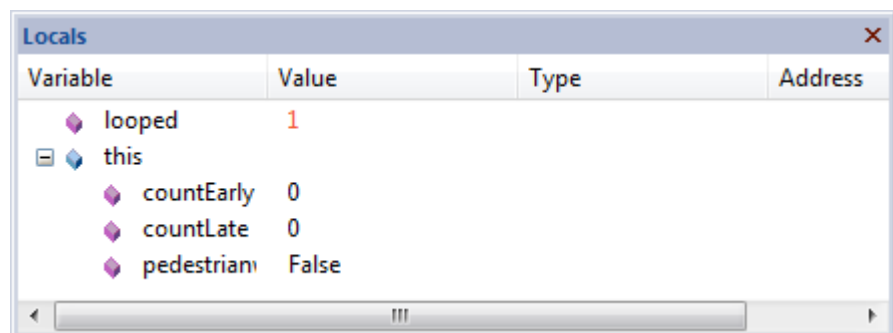
During the Simulation the Call Stack window ('Simulate > Dynamic Simulation > Call Stack') displays information about the threads and current execution context of the Simulation. The Simulator supports multi-threaded Simulations and will include a thread entry for every active and paused thread of execution. 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).



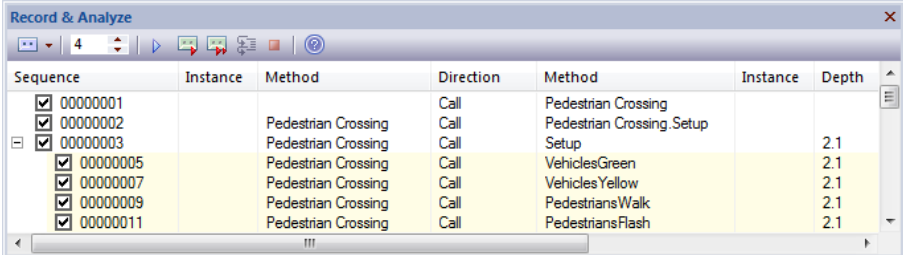
Simulation Local Variable Window

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.



Recording

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.



The screenshot shows the 'Record & Analyze' window with a table of recorded events. The table has columns for Sequence, Instance, Method, Direction, Method, Instance, and Depth. The events are listed in a tree view with checkboxes. The following table represents the data shown in the screenshot:

Sequence	Instance	Method	Direction	Method	Instance	Depth
☒ 00000001						
☒ 00000002		Pedestrian Crossing	Call	Pedestrian Crossing		
☒ 00000003		Pedestrian Crossing	Call	Pedestrian Crossing.Setup		
☒ 00000005		Pedestrian Crossing	Call	Setup		2.1
☒ 00000007		Pedestrian Crossing	Call	VehiclesGreen		2.1
☒ 00000009		Pedestrian Crossing	Call	VehiclesYellow		2.1
☒ 00000011		Pedestrian Crossing	Call	PedestriansWalk		2.1
☒ 00000011		Pedestrian Crossing	Call	PedestriansFlash		2.1

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. <code>sim.count = 0;</code>
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.

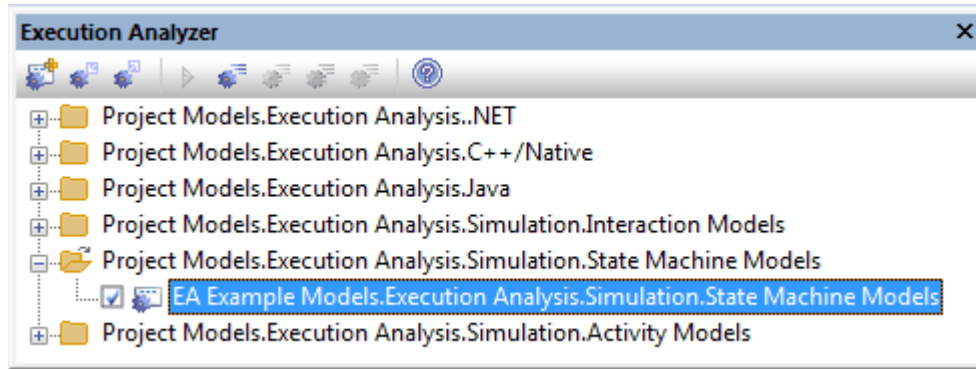
Access

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.



- 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

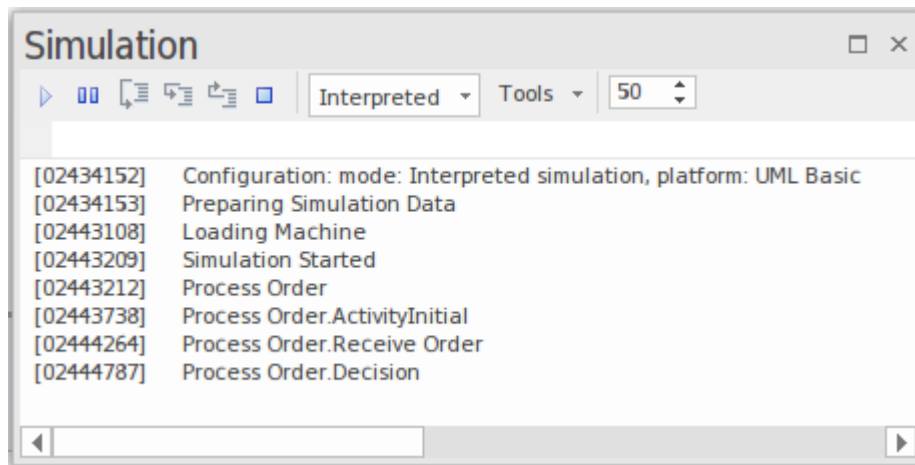
Ribbon	Simulate > Dynamic Simulation > Simulator > Open Simulation Window Simulate > Run Simulation > Start
--------	--

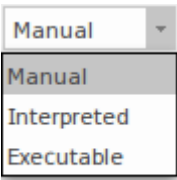
Edition Specific Details

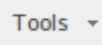
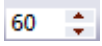
In the Professional Edition, if a branch is encountered in the execution, the Simulator prompts you to choose the appropriate path to take in your execution.

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



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.
	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
	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.
	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.

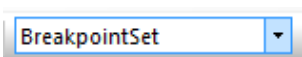
The breakpoints are stored as Breakpoint Sets for a given Enterprise Architect project.

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.
	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	All simulation objects, Simulation

Variables	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

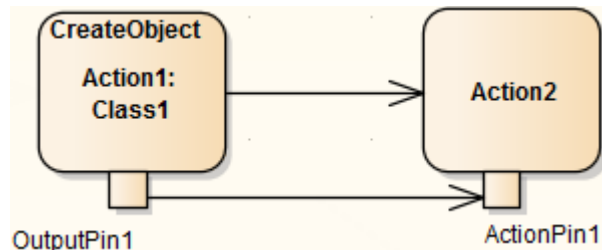
In the Simulation model you can then set up the Object properties themselves (such as setting run-states to re-set the initial value of an attribute) or the behaviors of Actions to act on the Object (such as passing it along a process flow) and observe what happens to the Object in a Simulation.

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.



- 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");
```

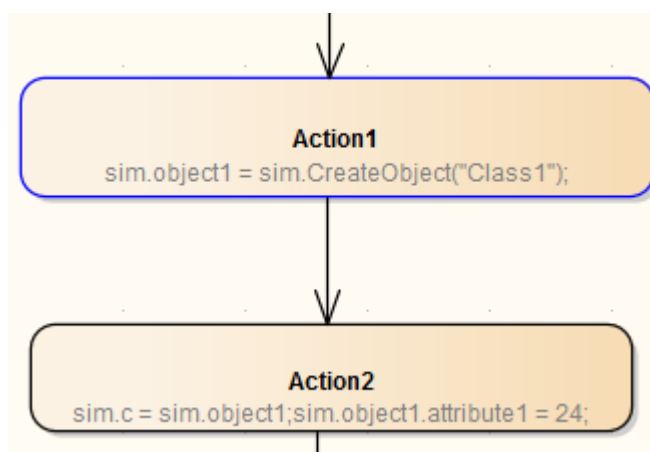
or

```
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.

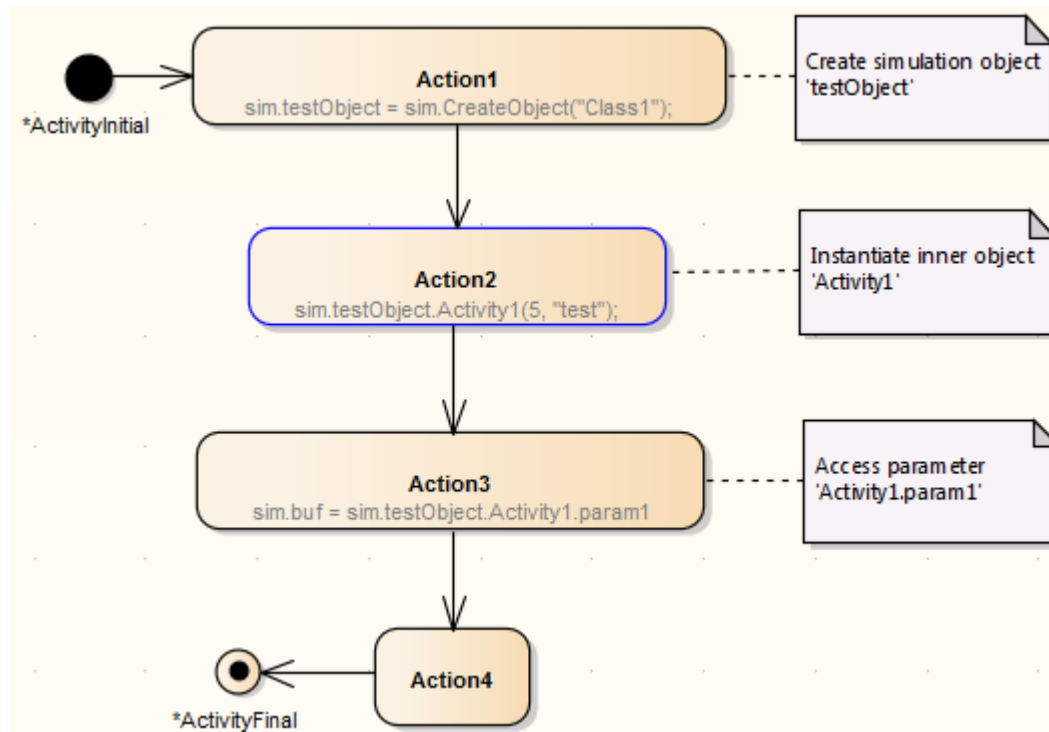
As for the CreateObject Action element, the Object is created during the Simulation and can be passed down to and processed by 'downstream' elements. In this example, the created Object is identified as `sim.object1` and in Action 2 it is accessed and one of its attributes given a different value (also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



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'.



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.

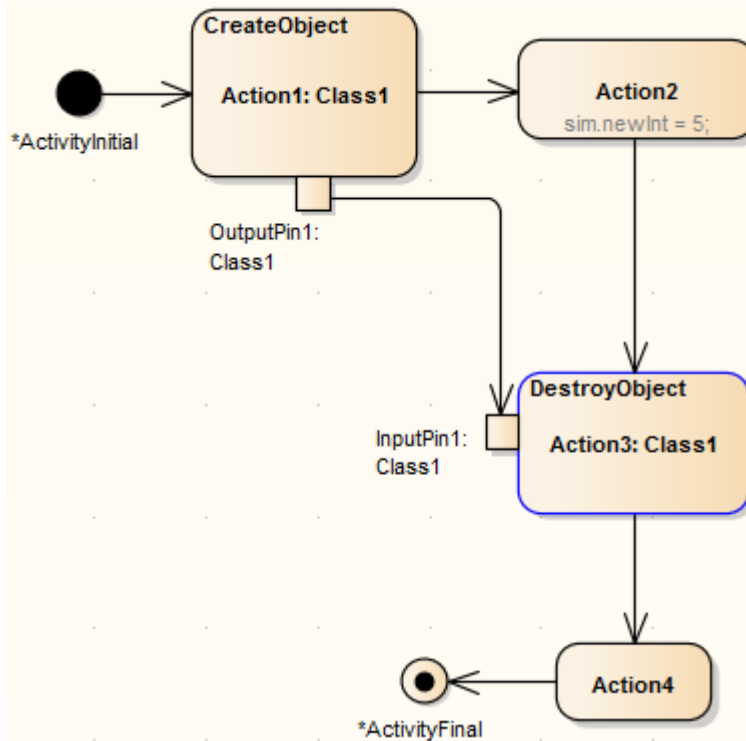
You have two options for destroying the Objects in your Simulation model:

- 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 <i>input</i>.
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.  <pre> graph TD Start((*ActivityInitial)) --> CreateObject[CreateObject Action1: Class1] CreateObject --> Action2[Action2 sim.newInt = 5;] CreateObject -- "OutputPin1: Class1" --> DestroyObject[DestroyObject Action3: Class1] Action2 --> DestroyObject DestroyObject --> Action4[Action4] Action4 --> End(((*ActivityFinal))) </pre>
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

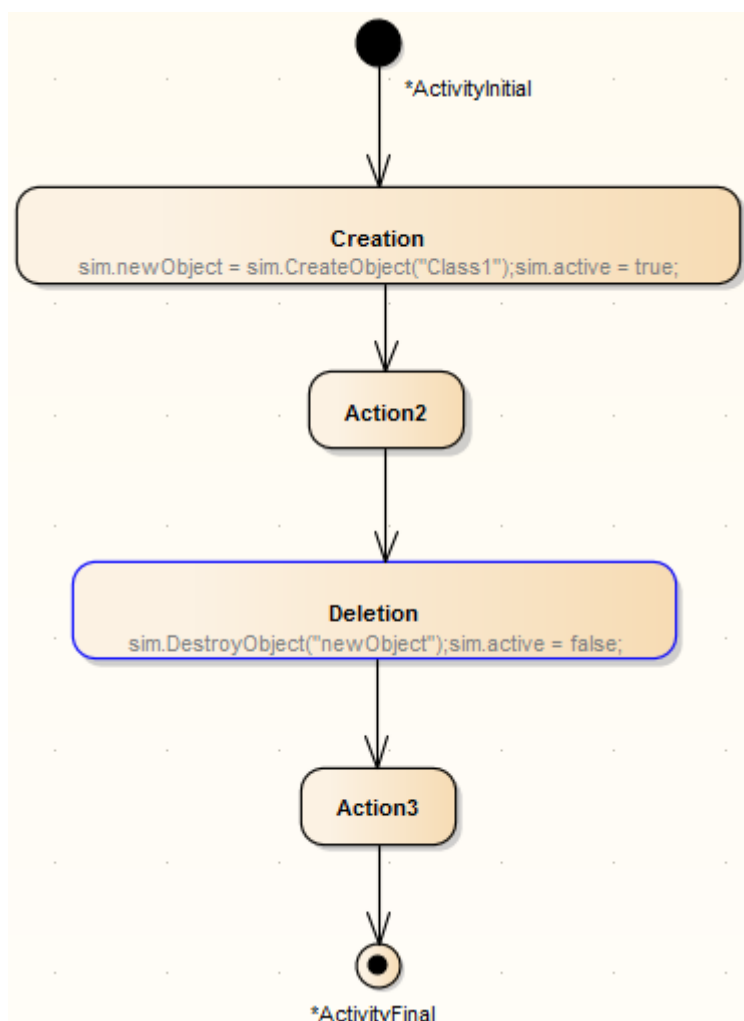
In the 'Properties' dialog of the Action element, in the 'Effect' field on the 'Effect' page, type either:

`sim.DestroyObject ("objectname")`

or

`delete sim.objectFullName`

For example:



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:

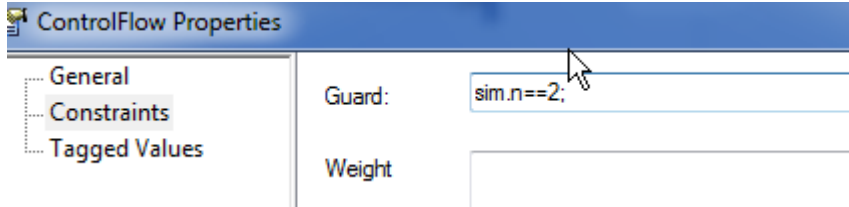
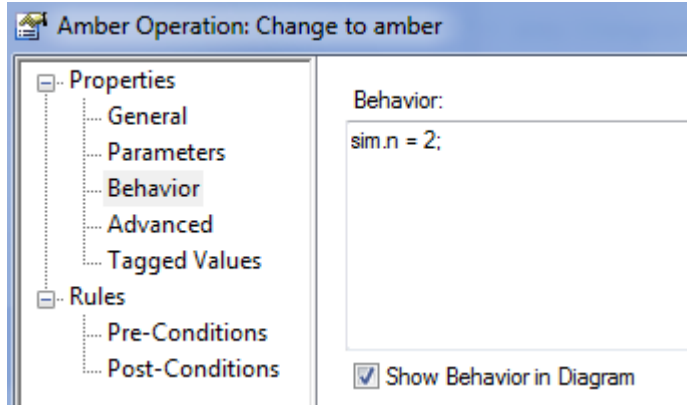
- `sim.logger`
- `sim.Customer.name`
- `this.count`
- `this.Account.amount`

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 <pre>sim.n = 1;</pre>
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. 
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.  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: <pre>this.name="Odd Even"; var logger = new COMObject("MySim.Logger"); logger.Show(); logger.Log("Simulation started");</pre>
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 <i>Octave Solver</i>, <i>MATLAB Solver</i> and <i>Solvers Help</i> 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". <pre>BroadcastSignal("CancelPressed");</pre> 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: <pre>BroadcastSignal("{996EAF52-6843-41f7-8966-BCAA0ABEC41F}");</pre>

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: <pre>if (IS_IN("WaitingForInput")) BroadcastSignal("CancelPressed")</pre> 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.

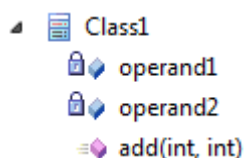
Call Behaviors

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.

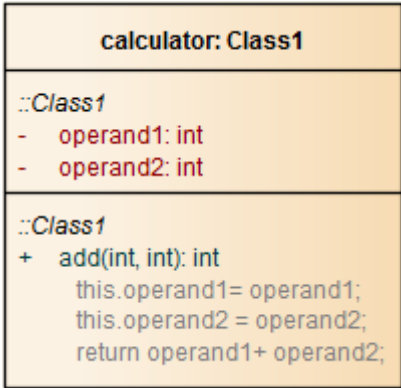
Invoke the Behavior of a Class

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*.



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 (<i>this</i> or <i>sim</i>) to the behavior definition. In the example: <pre>this.operand1=operand1;</pre>

	<pre>this.operand2=operand2; return operand1+operand2</pre>
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.  <pre> classDiagram class "calculator: Class1" { +operand1: int +operand2: int +add(int, int): int } </pre> The diagram shows a class box for 'calculator: Class1'. It has two attributes: 'operand1: int' and 'operand2: int'. It has one operation: 'add(int, int): int'. The operation's body is: '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: <pre>sim.result=sim.calculator.add(7,9)</pre>
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.

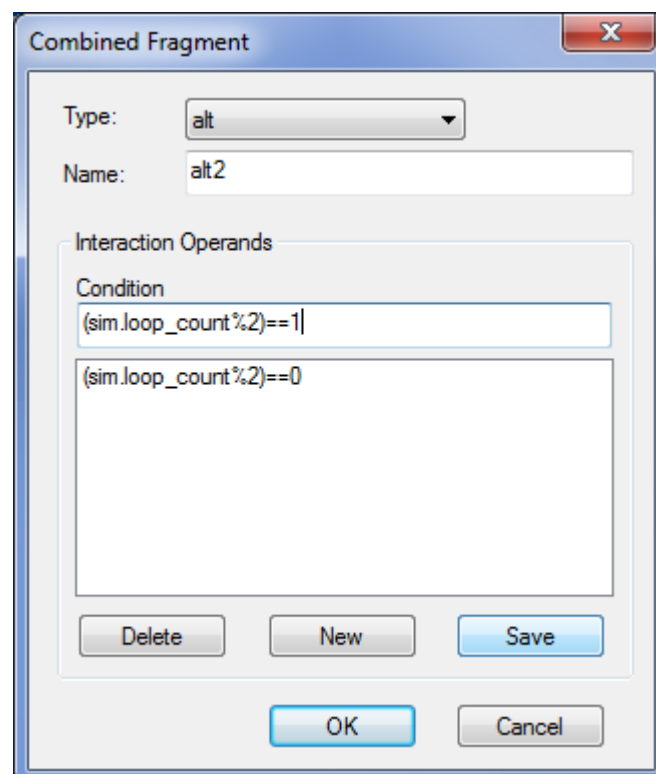
Interaction Operand Conditions

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.



Evaluation Semantics

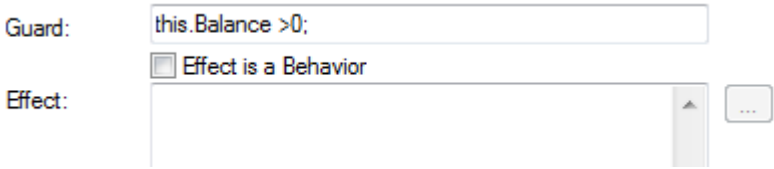
During execution the Simulator evaluates any Operand Condition within the CombinedFragment; the CombinedFragment type and the outcome of the evaluation can determine the path that the Simulation continues on.

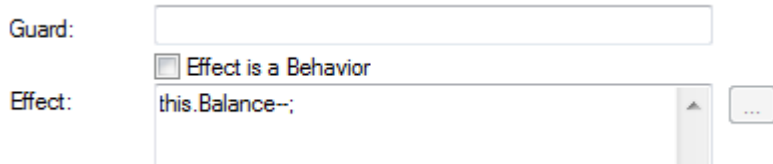
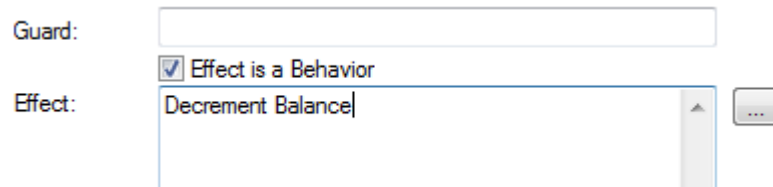
Guards and Effects

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 '<i>this</i>' to indicate that the variable is a member of the current Class context. 
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: this.Balance--; ☒ Effect is a Behavior
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: Decrement Balance ☒ Effect is a Behavior

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: • <code>sim</code> (for example, <code>sim.pedestrianwaiting</code>) - typically used for global Simulation variables, or • <code>this</code> (for example, <code>this.CustomerNumber</code>) - 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, <code>this.customer.name</code> 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

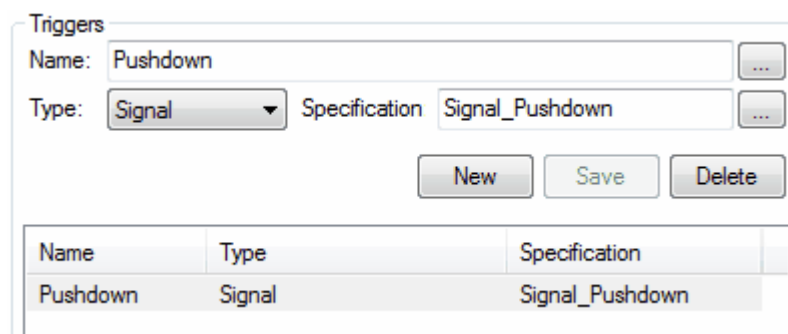
- Transitions must be defined that will fire when the simulation receives the Signal or Event
- The current Simulation State(s) or its parent(s) must have an outgoing transition that accepts that trigger
- The transition activated must be unguarded or have a guard that will evaluate to True

Managing Triggers

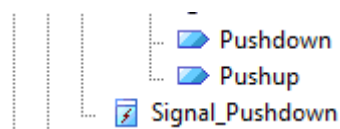
Action	Detail
Creating Triggers	Triggers are either created as an instance of a Signal element or as an anonymous event. Triggers are connected to

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



A trigger will appear in the 'Project' tab of the Browser window, as illustrated here:



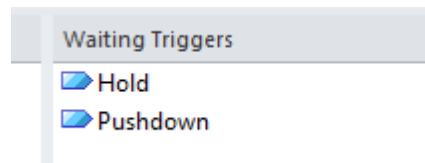
Using Triggers

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.



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

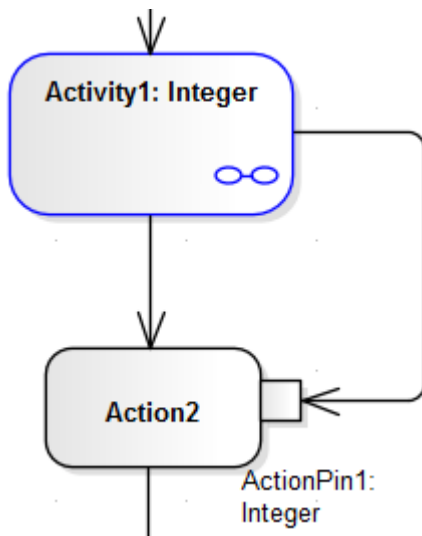
An Activity is likely to produce a return value as the output of the process it represents. You can simulate how that return value is passed on to the next stage in the process, under three scenarios:

- 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.)

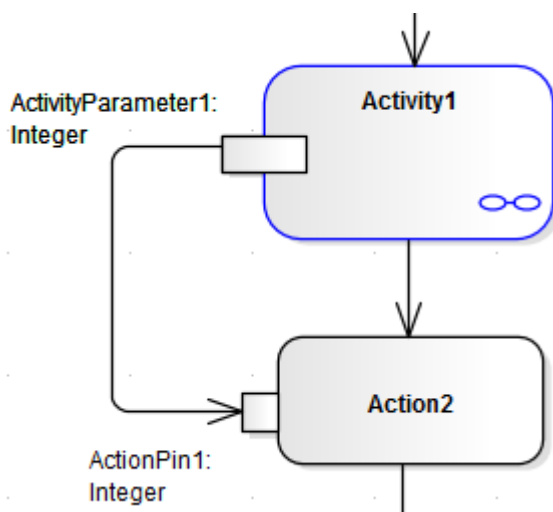
The Activity has a return value, which is transferred from the Activity element to an Action Pin on the next Action in the process via an Object Flow connector.



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:

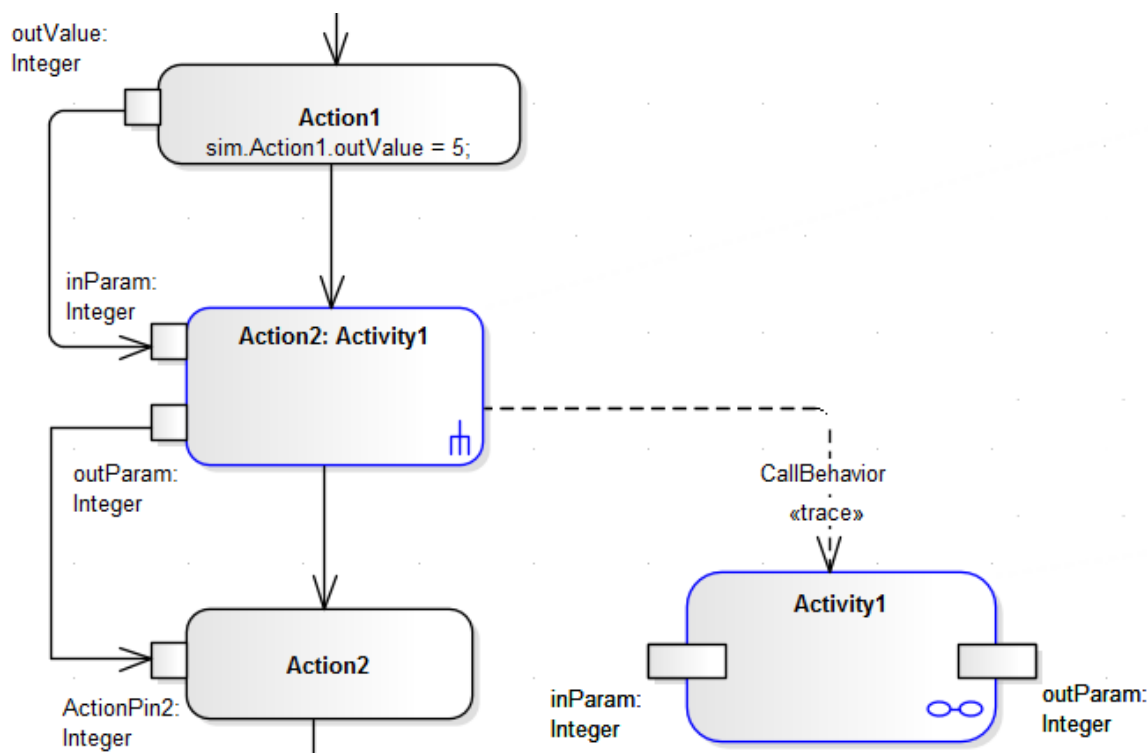


In the Locals window, you can either observe the Parameter's default value pass through to the ActionPin, or you can use JavaScript in the Activity's child Actions to simulate an update of the value within the Activity. For example:

```
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

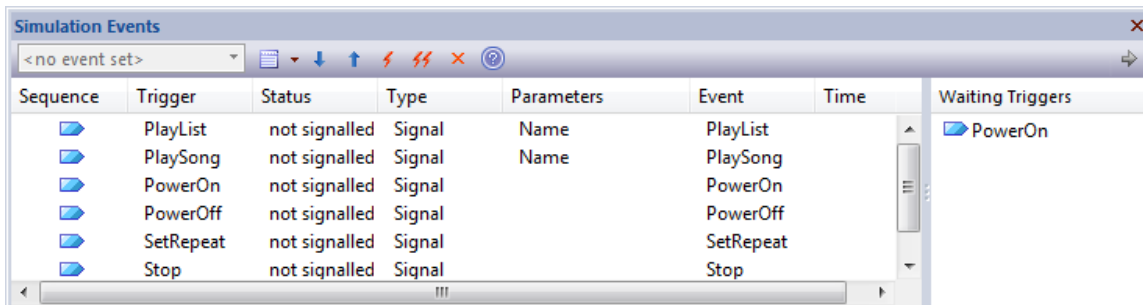
The Simulation Events window is where you manage triggers and sets of events in a simulation. Its main functions are to:

- 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.



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
	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

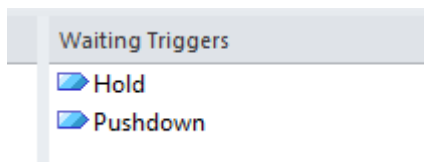
Ribbon	Simulate > Dynamic Simulation > Events The right hand side pane lists available Triggers.
--------	--

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.



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.

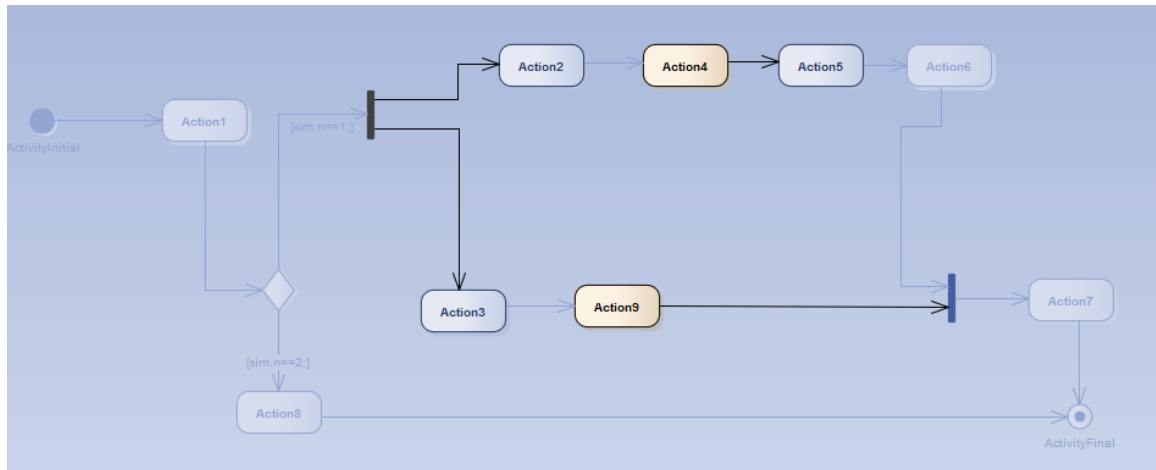
This image demonstrates re-signalling in action. When a signal is re-signalled, a new copy is made and placed at the end of the signalled triggers list, where it is automatically fired again.

Sequence	Trigger	Status	Type	Parameters
 1	Pushdown	used	Signal	
 2	Pushup	used	Signal Selected	
 3	Pushdown	used	Removed Selected	
 4	Pushup	used	Re-Signal Selected	
				Set All to Unsignalled
				Clear Trigger List

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

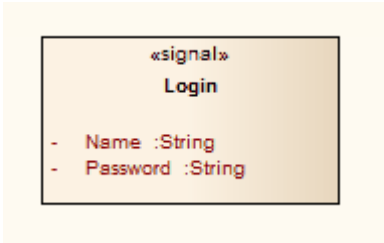
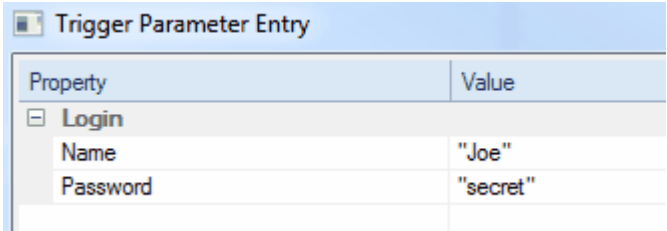


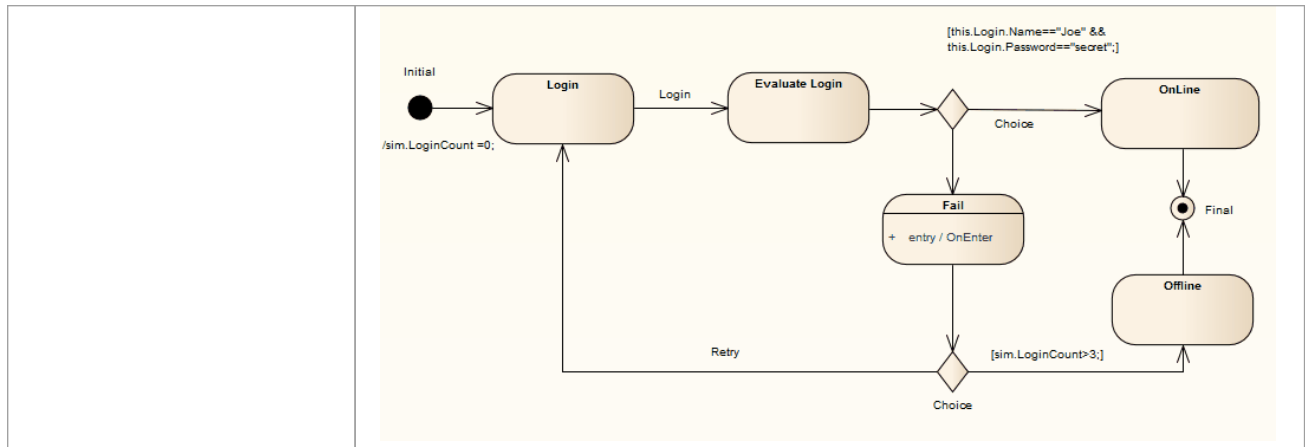
Trigger Parameters

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.  <pre> «signal» Login - Name :String - Password :String </pre>								
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.  <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Login</td> <td></td> </tr> <tr> <td> Name</td> <td>"Joe"</td> </tr> <tr> <td> Password</td> <td>"secret"</td> </tr> </tbody> </table>	Property	Value	Login		Name	"Joe"	Password	"secret"
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.								



Trigger Sets and Auto-Firing

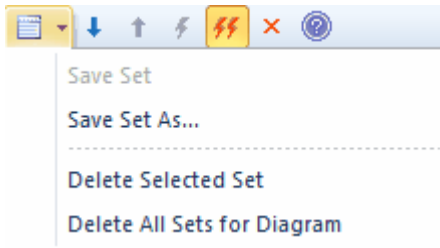
Trigger Sets are an effective means of automating and streamlining the execution, testing and validation of simulation models. By re-using sets of triggers (with or without parameters) it is possible to quickly and efficiently walk through many simulation scenarios, either manually or automatically using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

Access

Ribbon	Simulate > Dynamic Simulation > Events
--------	--

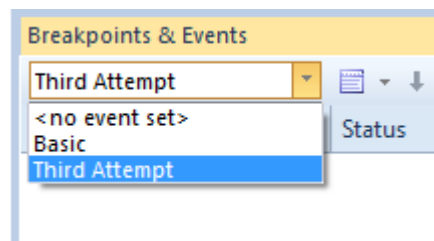
About Trigger Sets

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.  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.



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

Sequence	Trigger	Status	Type	Parameters
1	Login	not signalled	Signal	Joe,guess1
2	Retry	not signalled	Simple	
3	Login	not signalled	Signal	Joe,secret
4	Retry	not signalled	Simple	
5	Login	not signalled	Signal	"Joe","secret"

Auto-Firing

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.
Auto-Firing Rules	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
--	---

Using Trigger Sets to Simulate an Event Sequence

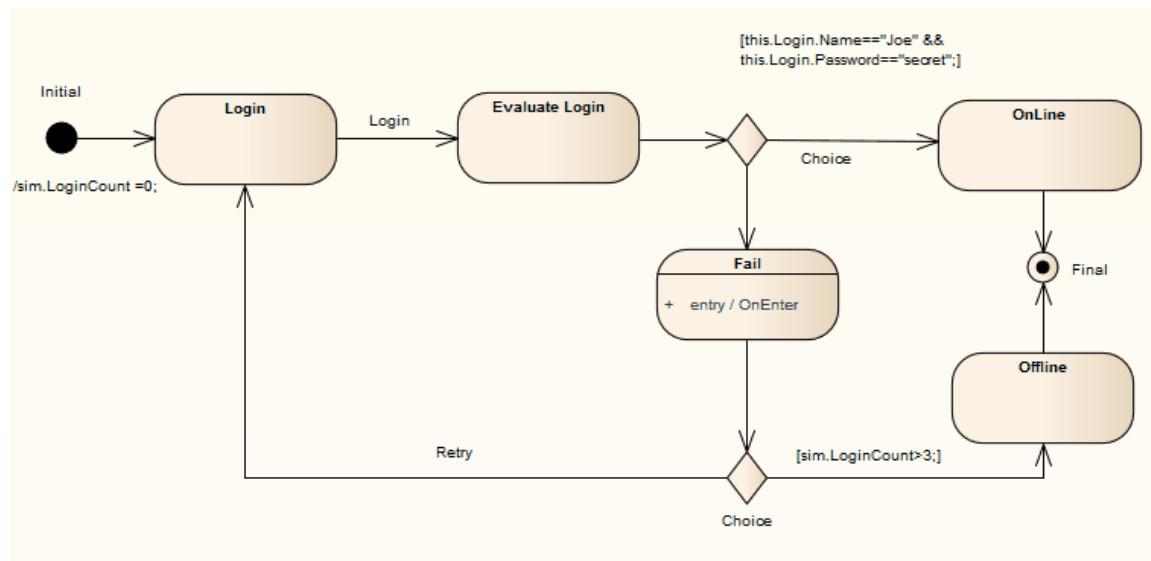
As a simple example of how useful trigger sets are, consider this example trigger set and associated diagram.

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

Breakpoints & Events				
Third Attempt				
Sequence	Trigger	Status	Type	Parameters
1	Login	not signalled	Signal	Joe,guess1
2	Retry	not signalled	Simple	
3	Login	not signalled	Signal	Joe,secret
4	Retry	not signalled	Simple	
5	Login	not signalled	Signal	"Joe","secret"

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



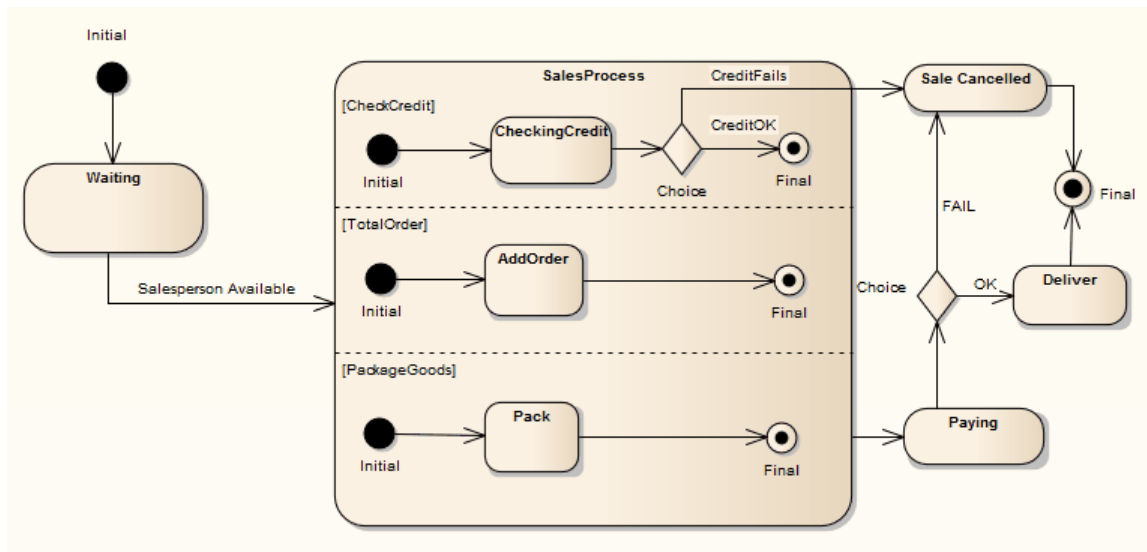
Multi-threading - Concurrent State Regions

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

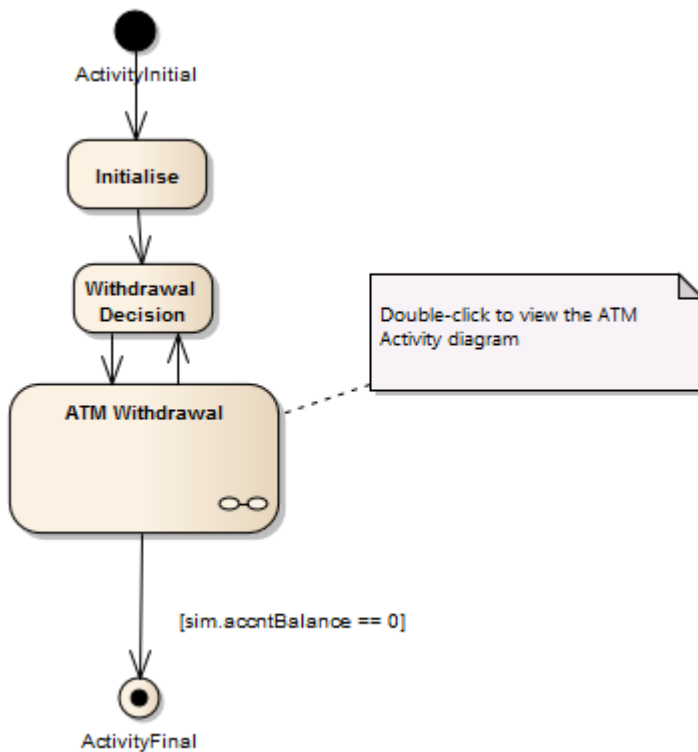


Using Composite Diagrams

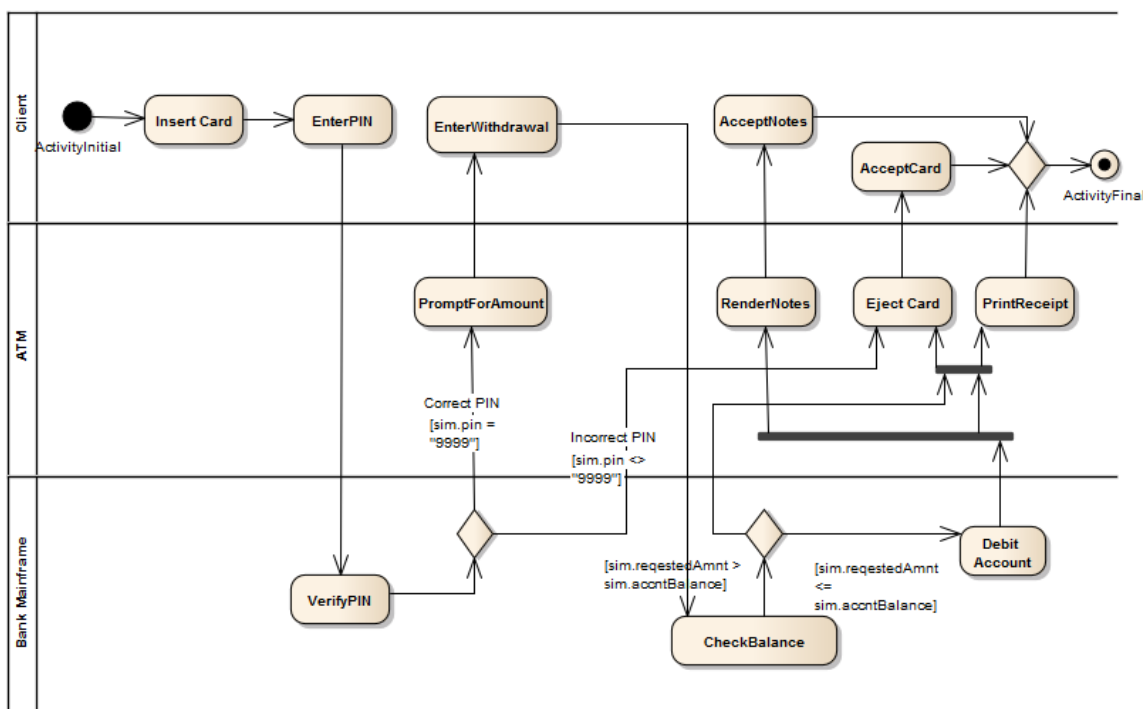
If you want to simulate processing that includes a branch represented on a different diagram (for example, to reduce complexity on the main diagram, or to hide areas of processing that are only actioned under an exception), you can use a Composite element to represent and access the branch on its child 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.



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: <pre>UIBroadcastSignal("Login",{Name: dialog.Login.Username.Text, Password: dialog.Login.Password.Text});</pre>

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 /	Customizable design properties

Rich Edit Control	• 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
	Create a Business Process or BPEL diagram from

1	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	Ctrl+2 > 'Tags' tab of the Properties

Shortcuts	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	The basic behavior unit in an Activity is the Activity element. A number of different Task Types are available. These

is the Action element. UML provides many different forms of Actions, although the simulation makes use of a small subset of 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	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.

node, except for an explicit Fork Node. To restrict flow on a Control Flow, add a Guard.	
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	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

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.	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 only some outputs from a node for UML Activities. If you needed this you add later Control Flows with the appropriate Guards.	A Gateway node set to Inclusive is used to explicitly model the situation where all outgoing flows with a true condition are executed concurrently.

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.

