

Welcome

Simio Reference Guide

V18



Refer to the Appendix for the License Agreement, and Academic and Runtime Use Policies. There are links on the bottom of each page that allow you to send us feedback about the content. This book accompanies Simio software version 18.279.46998.0

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Contents of Documentation

[Welcome](#)

[Simio Product Family](#)

[Getting Started](#)

[User Interface](#)

[Simio Concepts](#)

- [Object Hierarchy](#)
- [Object Types](#)
- [Processes](#)
- [Projects and Models](#)
- [Queues](#)
- [Tokens and Entities](#)

[Learning Aids](#)

- [SimBits](#)

[Modeling in Simio](#)

- [Navigation](#)
- [Project Window](#)
- [Model Windows](#)
 - [The Facility Window](#)
 - [The Processes Window](#)
 - [Steps](#)
 - [The Definitions Window](#)
 - [Elements, Properties, States, Events, Lists, Tokens, External Console](#)
 - [The Data Window](#)
 - [Tables, Schedules, Changeovers](#)
 - [The Results Window](#)
 - [Interactive Verses Experiment, Pivot Table, Reports, Exporting](#)
- [Standard Object Library](#)
- [Project Library](#)
- [Assigning States](#)
- [Secondary Resources](#)
- [Add-on Process Triggers](#)
- [Reliability](#)
- [Creating New Objects](#)
- [Animation](#)
- [Expression Editor, Functions and Distributions](#)
- [Running the Model](#)
- [Experiments](#)
 - [Controls or Responses and Constraints](#)
 - [Analysis - Experiment Properties](#)
 - [Experiment Response Chart \(SMORE Plot\)](#)
 - [OptQuest Add-In](#)
 - [Scenario Subset Selection](#)
 - [Select Best Scenario Using KN Add-In](#)
 - [An Example Model with an Experiment](#)
- [Model Summary Report](#)
- [Custom Simio Extensions \(i.e. Custom Steps and Elements\)](#)

-
- [Dynamic Selection Rules](#)
 - [Protection](#)

[Appendix](#)

- [System Requirements](#)
 - [Installation Instructions](#)
 - [Check For Updates](#)
 - [Run Time Capability](#)
 - [End User License Agreement](#)
-

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Simio Product Family

Simio is a family of products that includes the Design, Professional and RPS Editions. All three products provide the same powerful 3D object-based modeling environment.

Simio Design Edition

Simio Design is Simio's standard simulation software product and includes the Standard Library for getting started with Simio. It adds a unique and powerful capability that allows modification of object logic, using add-on process-oriented logic. This version of the software also provides the ability to create and distribute custom modeling libraries. Simio Design is the ideal product for professional modelers that want to have full control over complex process logic or want to develop new modeling libraries focused on specific applications areas.

Simio Professional Edition

Simio Professional Edition provides the same functionality as Simio Design, however, it allows building and distribution of models using the freely available Simio Evaluation version as a runtime platform. Models built with the Simio Professional Edition will run and generate results with the Simio Evaluation version. Simio Professional Edition is ideal for consultants that want to deliver a running model to their customer. It also provides the capability to distribute scenarios and replications across computers in a work group. Simio Professional Edition includes Output Tables/States, and Gantt charts.

Simio RPS Edition

Simio RPS Edition adds a powerful set of patented features to build and execute new models for Risk-based Planning and Scheduling (RPS). This edition provides full scheduling capabilities: custom tailor reports, graphs, and tables for use by schedulers. Reduce your risk and costs by analyzing your schedules in ways never before possible!

Simio Professional Runtime Edition

Simio Professional Runtime Edition provides the ability to view and run the model and Experiments as well as view the Results of the model. With a Simio Professional Runtime Edition license, model editing functionality is limited, but this license allows for complete model viewing based on the Edition of software the model was saved in.

Simio RPS Runtime Edition

Simio RPS Runtime Edition provides the ability to view and run the model and Experiments as well as view the Results and Plans of the model. With a Simio RPS Runtime Edition license, model editing functionality is limited, but this license allows for complete model viewing based on the Edition of software the model was saved in.

Simio Trial Edition

Simio Trial Edition is a no cost version of Simio that expires thirty (30) days after installation. Simio Trial Edition has no limits on what can be modeled, and it is perfect for exploring and learning Simio's simulation capabilities.

Simio Academic Edition

Fully functional versions of Simio are available to students and faculty.

OptQuest for Simio

OptQuest for Simio is available as an add-on to the standard Simio products. OptQuest tightly integrates with Simio experiments and uses state-of-the-art algorithms, including Tabu Search, Neural Networks, Scatter Search, and Linear/Integer Programming to generate and evaluate scenarios in search of optimal configurations.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Getting Started

Understanding the Simio Modeling Approach

- Before you begin to build your first model with the instructions below, you might want to first familiarize yourself with the basic [Simio Concepts](#) since modeling in Simio is based on an object orientated approach, which might be different than other modeling software programs you have used in the past.

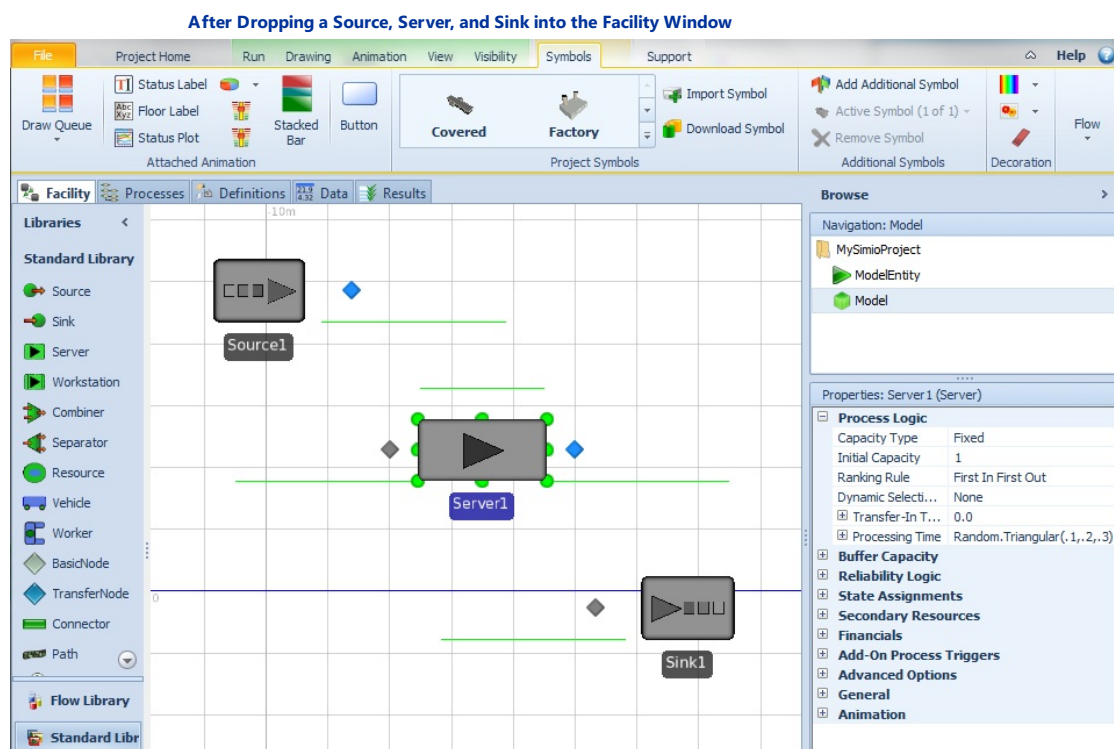
Build a Simple Model

- This section will explain the steps in building a small example model. For this walk through, you will be creating a simple Source-Server-Sink model.

To begin, you need to create a new model within the project. This can be done by clicking the New Model icon in the Project Home ribbon. Once the new model is created, you'll be looking at the Facility window of this new model.

First, click on the [Source](#) object definition in the [Standard Library](#) and drag it into the upper left part of the [Facility Window](#). Once you have dropped it, you should see a Source object instance called Source1. Follow the same process to drag a [Server](#) object to the middle of the Facility Window and a [Sink](#) object to the lower right of the Facility Window. To select any of these objects in order to move them or change their properties, you can click on the name (e.g. Sink1). For this example, we will leave the properties for all the objects at their defaults.

After you have all three objects in the Facility Window, your screen should look something like this:

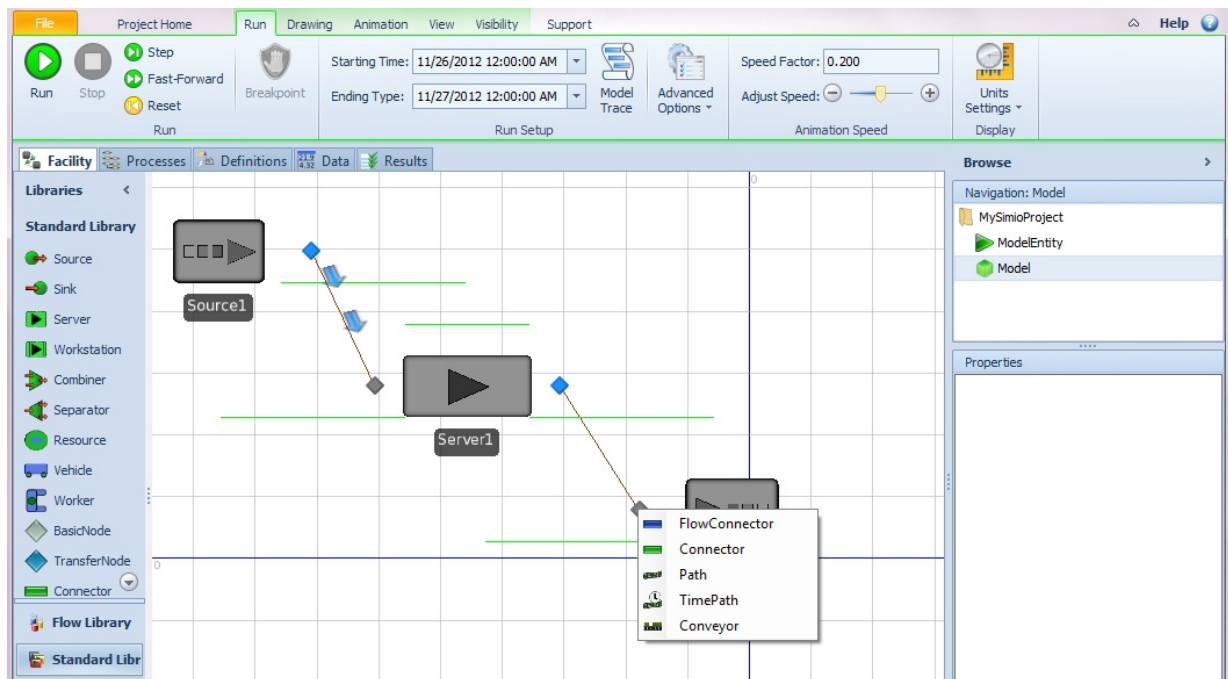


Your next step is to connect the objects. The diamond shapes on the objects are called [Nodes](#). They represent places that Entities can enter and leave. Although we won't be changing anything now, to review or change Node properties, you may click on the diamond shape and you will see its properties appear in the Properties Window in the bottom right side of the interface. Node properties are used to specify Entity destination and Transporter selection logic. Selecting a node in the Facility Window is a bit different from other objects because there are three modes of selection. The first two are common to most objects, the third is unique:

- Click – Selects Node and displays its properties
- Click and drag – Moves the Node to a different screen location. Note that it is still “attached” to its associated object if any (for example, a Server), and if that associated object is moved, the Node position will also change.
- Ctrl+Shift+Click and move – Initiates the creation of a Link between Nodes. The Link type may be preselected in the library panel (as described in third paragraph below) or it will be prompted for when the Link is terminated by clicking on a Node (as described immediately below).

We will now connect the objects with Paths. Press and hold the Ctrl and Shift buttons on the keyboard while clicking the left mouse button on the Transfer Node of Source1. Move the cursor to the left Node of Server1, and click the left mouse button again. A menu will appear to select the Link type, select Path.

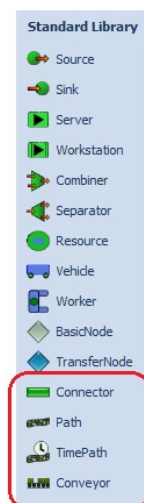
Selecting the Path option after drawing a Link between two objects



Now press the Ctrl+Shift buttons and click the left mouse button on the right Node of Server1. You may place mid-points for the connection by clicking various places in the Facility Window, or you can finish the connection by clicking on the Node of Sink1. If you change your mind about where you are placing the line before finishing the connection and want to remove the connection altogether, then click on the right mouse button. A menu will appear for you to select the Link type, select Path.

There is another method to use if you want to place Links. Click on the Link type of your choice in the Standard Library and then you can left click on the Nodes of your choice. See the circled items in the following screen shot:

Link items in the Standard Library



Double clicking on an object, such as a Path, will put you in a mode to apply that action multiple times sequentially. For example, double clicking on the Path will allow you to connect the Source to the Server and then the Server to the Sink without selecting it again from the library. This is the case with all of the Standard Library objects, as well as many other items in the user interface.

Congratulations, you have built your first Simio model! If you would like to save your [model/project](#), click on the File tab at the top left corner of the main Simio window. Then select the Save As item from the Backstage view. Use the Windows Explorer window to complete the saving of your file. Note: There is an alternative way to add a standard Source, Server and Sink into the Facility Window. It was important to teach you the manual steps as a learning process, but as a time saver for the future, you can simply select the Source/Server/Sink option from the Actions (Add-Ins) icon on the Project Home ribbon. This will add these three objects all connected by Paths.

Now you can click on the [Run](#) button located on the Project Home tab to run your model.

See the [User Interface](#) help topic for additional information on how to navigate through Simio's User Interface.

The Concept of Properties and States

- For a user who is either new to Simio or object oriented programming, spending some time learning about Properties and States will be helpful. Other simulation software programs might use an attribute or a variable, whereas Simio objects use States and Properties to pass information between objects and to get output responses. [Properties](#) and [States](#) are added from the Definitions Window.

Object and Element Name Identifiers

- When naming Objects within the Facility window of a model (such as Source, Server, Sink), as well as Elements within the Definitions window (such as Properties, States, Materials), there are certain naming conventions that should be used. In Simio sprint 156 and prior, all names were required to start with a letter and could contain letters, digits and underscores. In Simio sprint 157+, names may begin with a number. They may still contain letters, numbers and underscores, but may not resolve to an absolute number. For example, the *Name* of a Server can be '123ABC' but may not be '123'. Note that a name such as '3e4' is not allowed as it exponential notation for 3×10^4 .

Simio has a number of available characters that are valid, such that all Unicode uppercase and lowercase letter categories as well as others are supported. (As an example, here is a listing of the Unicode lower case letter category <http://www.fileformat.info/info/unicode/category/Ll/list.htm>).

Queues in Simio

- Queues can be found in many different places throughout Simio. Some queues are animated by default, such as the Station queues associated with the Standard Library objects, but others are not. To learn more about which queues exist in Simio and how to animate a queue, visit the [Queues](#) help page.

Applying Work Schedules

- If certain objects within your system follow a Work Schedule, you'll want to learn how to create schedules and add them to your objects. Schedules are created in the [Data Window](#). To find out more information, see the [Schedules](#) help page and find the [SimBits](#) that contain Schedules to see how Schedules work in Simio.

Modeling with Vehicles

- Simio provides tremendous flexibility and power for modeling with vehicles. Vehicles are "smart" and can make decisions such as whether or not to pickup a rider than has requested a pickup. Begin by reading about the standard [Vehicle](#) object and read through the [Discussion and Example](#) help page. View the [SimBits](#) page to see which models contain vehicles to see the functionality in action.

Importing and Exporting Data

- Importing and Exporting information during the model run is done with the User Defined Steps [Read](#) and [Write](#). The Read and Write Steps require a File to be defined. A [File](#) is defined in the Elements tab of the Process Window. The [WritingToAFile](#) SimBit demonstrates how to use the Write Step.

Results can be exported from Simio from the Results Window. See the [Exporting](#) page for more information about exporting results.

Data can also be imported into (and exported from) a Data Table or a Sequence Table. See the [Tables](#) help page for additional information.

Adding Plots, Gauges, Stacked Bars or Status Labels to View During an Interactive Run

- Plots, Gauges, Stacked Bars or Status Labels can be added to a model so the user can have a dynamic view of how the object's state changes over time. The [Console Window](#) is where the user can place displays that will report the status of an object during the run. These objects can also be placed in the Facility window and the External panel and attached to objects in the Facility window.

Experimentation / Running the Model without Animation

- When a model is run by pressing the Run button from the Facility Window, it is being run in Interactive Mode with full animation and the ability for a user to make certain changes to the model in the middle of the run. However, Simio provides an alternate way to run a model. If a model is run from within an Experiment, there is no animation and therefore the run time is significantly reduced. Creating an [Experiment](#) also allows a user to run multiple scenarios, each with different input parameters and/or output responses, allowing the user to set up true experimentation on their model.

Note: An experiment cannot be run in Evaluation mode (i.e. Simio without a license installed) unless the model is within the Simio Evaluation Edition modeling limits. To see an example of an Experiment, see the [AirportTerminal](#) example model that is included with the installation of Simio. This model contains an experiment that can be run by evaluators.

Debugging Tools

- There are a number of different tools that are helpful for verification and debugging. Simio has a [Trace Window](#) that shows the logic being executed by the objects and tokens in the model. There is also a [Watch Window](#) that displays the current values of States, Properties, Functions and Elements during the interactive run. A user can also put a [Breakpoint](#) on any object in the Facility Window or on any Step in the Processes Window and the model will pause when it hits that object or Step. For more information, see [Debugging the Model](#).

The Standard Library Verses Your Custom Project Library

- Simio provides a [library of standard objects](#) for a user to use for building either simple or complex models. Because of the flexibility of the standard objects and the ability to use [Add On Process triggers](#) on the standard objects, these objects might be sufficient to meet the modeling needs of most people. However, Simio also allows a user to [create their own](#) custom Objects, either from scratch, by subclassing or by copying a Standard Library or other library object to use as a starting point for customization. These custom objects then become part of the [Project Library](#). Clicking on the Project Name in the Navigation window enables the Edit ribbon where subclassing/copying objects can be done. Additionally, right clicking on an object in the Libraries panel provides a subclass option. To subclass an object that already exists with custom data in a model, the right click option on that object within the Facility window allows users to Create Object From This object, which creates a subclass object with default data.

Simio also allows the ability for a user to reuse objects in a project, that were created in another project. This can be done with the Load Library button on the Project Home tab (or right-click on a library to load/unload libraries). Essentially, the user is importing a library of objects from a previous project into this current [project](#) so one or all of the objects from the previous project can be used in the current project.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

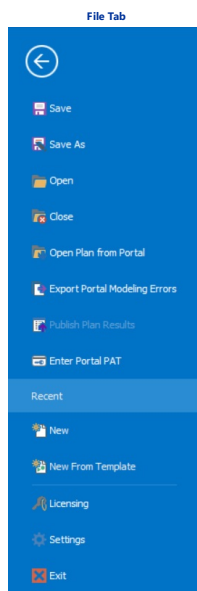
Send comments on this topic to [Support](#)

User Interface

The File Tab

The **File Tab** in the top left corner of the interface, is where the user can open, close or save a new project. The most recent projects can be found here as well. The [New From Template](#) option is available for opening a project template which may consist of existing tables and/or objects. The Licensing option is available for loading, removing or changing a Simio license. See section below on Application Settings for more information on the Settings option.

You can also export to Portal as well as open a model from Portal. The Export Portal Modeling Errors button exports a CSV of any Portal model errors.

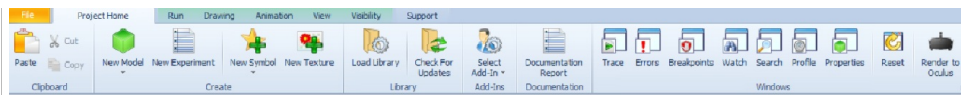


Note: You can load the most recent project at startup by checking the box at the bottom of the File Tab under Recent Projects

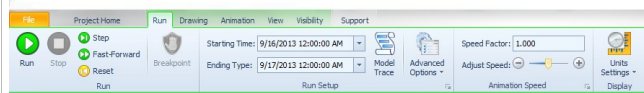
The Ribbon

The Ribbon contains multiple ribbon tabs and within each tab there are groupings of icons, or panels. The ribbon tabs that are visible vary depending on which window the user is viewing in the main part of the application. Below are a few of the ribbon menus.

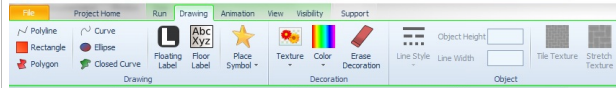
[Some of the Available Ribbon Menus](#)



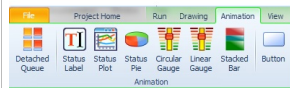
The Project Home ribbon - Always Available



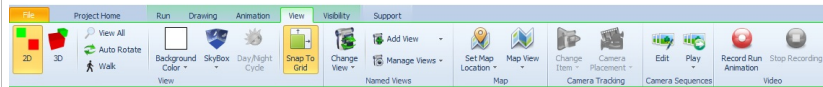
The Run ribbon - Available from the Facility window and the Processes window



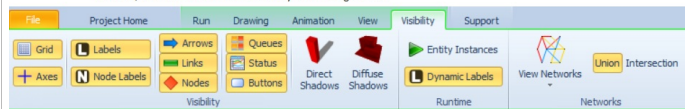
The Drawing ribbon - Available from the Facility window, the External window and the Symbol editing window



The Animation ribbon - Available from the Facility window, the External window and the Console window



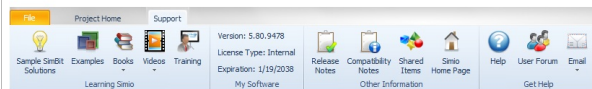
The View ribbon - Available from the Facility window, the External window, the Dashboard window and the Symbol editing window



The Visibility ribbon - Available from the Facility window



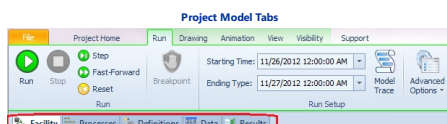
The Appearance ribbon - Available from the Facility window for queue details



The Support ribbon - Always Available

The Project Model Windows

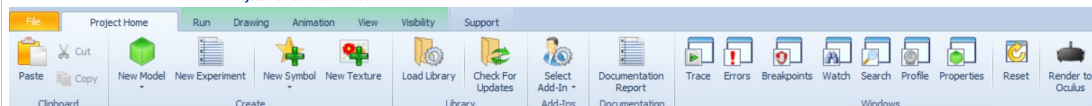
The default window that appears in a new project is the **Facility Window**. Project Window Tabs are used to select between multiple windows that are associated with the active model or experiment. The windows that are available depend on the object class of the selected model.



The Project Home Ribbon

The **Project Home** ribbon is the only ribbon that is displayed when all the different windows are open. This ribbon allows the user to create a new model, a new experiment, add a new symbol, import a new texture, access the report designer and load a library into the current project. This is also where a user can access the cut, copy and paste icons, as well as user add-ins, such as under Actions (all licenses) and Modeling Helpers (visible for RPS licenses only). This ribbon includes icons to turn on and off **Trace** and show or hide the Error window. The Breakpoints icon will display any **breakpoints** that have been set within the model. A breakpoint causes the execution of the model to pause when an entity or a transporter arrives to a specified object, node or link. The Watch icon will display the **Watch Window**, which allows the user to view the values of states, functions and elements that are associated with a certain object instance, during the run. The Search icon will display the **Search Window**, which allows the user to search for a specific text string. The window will then display all instances of that text string, which then can be double-clicked to take the user to a given instance. The Profile icon will open the **Profile Window** which is used for run-time profiling. The Properties icon opens the **Properties Spreadsheet Window** which can be used for viewing and editing properties of selected types of objects. The **Render to Oculus** icon enables or disables rendering the active 3D view to an attached Oculus HMD device.

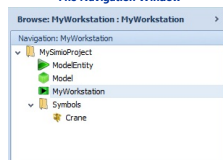
Project Home Ribbon Ribbon



The Navigation Window

The **Navigation window**, which is located in the top right hand side of the interface, is used to switch between the different models that exist within the project. To switch between models, simply click on the desired model. The model which is highlighted is the model that is currently active, or in other words, the model for which the model tabs are associated.

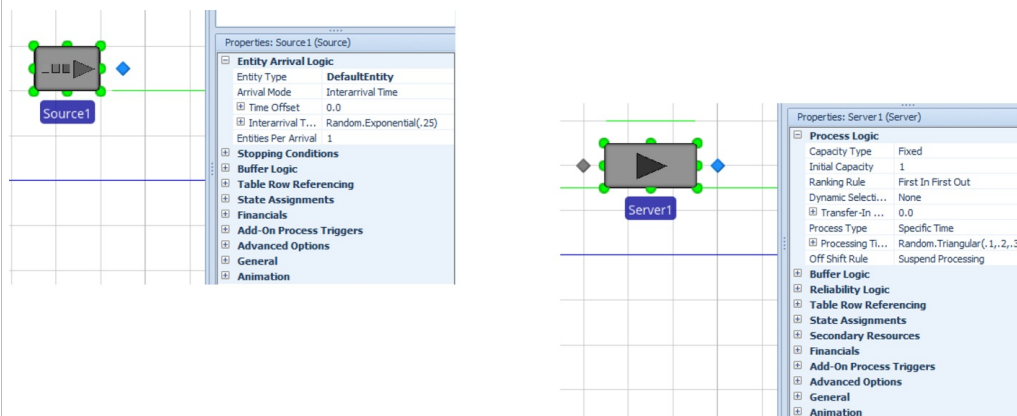
The Navigation Window



When the Project is selected and highlighted, the Project window is displayed. This window lists all the models that exist within this project. By clicking on the panels on the left side of this window, the user can view the Experiments, the Symbols, the Textures and the Path Decorators that exist within this project.

The Properties Window: Viewing and Changing the Properties of an Object

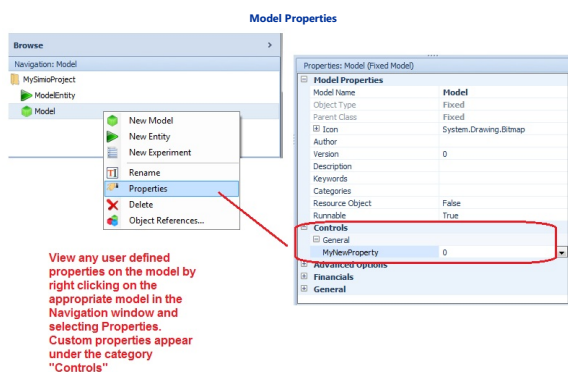
To view the properties of a particular object in the Facility window, click on the object and you will see the object's specific properties in the Properties window. Each object has properties that control its behavior; e.g. the Source has an inter-arrival time, the Path has a property to control entity passing, and the Server has a processing time.



Prior to Simio sprint 169, Simio provided a *Show Commonly Used Properties Only* checkbox at the top of the Properties window. When this mode was enabled, the display was limited to the key set of properties that defined the core behavior of each object. Many standard features such as failures, state assignments, secondary resource allocations, financials, and custom add-on process logic that are provided by the Standard Library objects were not displayed. This mode is no longer available, thus all properties for objects are displayed.

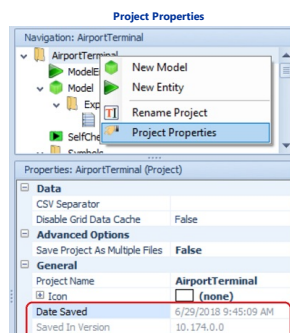
The Properties Window: Viewing and Changing the Properties of a Model

The properties of a model can be accessed by selecting the appropriate model within the Navigation window, right clicking and choosing Model Properties.



The Properties Window: Viewing and Changing the Properties of the Project

The properties of the project can be accessed by selecting the project name within the Navigation window, right clicking and choosing Project Properties. Project properties include important information such as the *Date Saved* and *Saved In Version*.



Right Clicking in The Facility Window

Right clicking on an object in the Facility Window brings up a menu allowing the user to add a Breakpoint to the object or to open the Console window for that object. Depending on the type of object, the user will have different options.

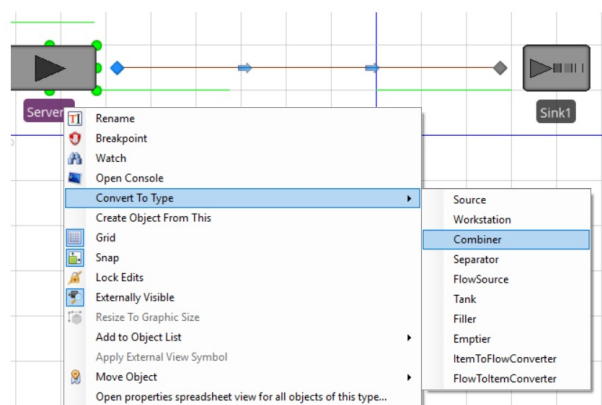
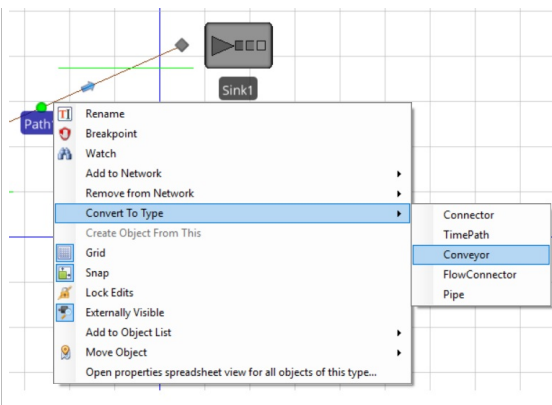
For example, right clicking on a Vehicle or Worker object allows the user to Add To Transporter List. Right clicking on a Node or group of Nodes allows the similar option to Add to Node List. Nodes also have an option for Binding to other nodes (see External Window page for more information on binding and hierarchy).

Right clicking on a link provides options to add or remove this link from a network, change the starting or ending node or convert this link to a different type of link. The link can be added to any networks that currently exist or the user can create a new network from this menu. Allowing the user to change starting/ending node or convert the link to a different type of link with a simple right click allows the user the ability to change the model logic quickly and easily. See Animating Links page for more information.

All objects have the right click option for converting to a different object, such as Server to Combiner or BasicNode to TransferNode. A connected object will give fewer options than an unconnected object as Simio keeps any links between objects. See the Converting Objects page for more information.

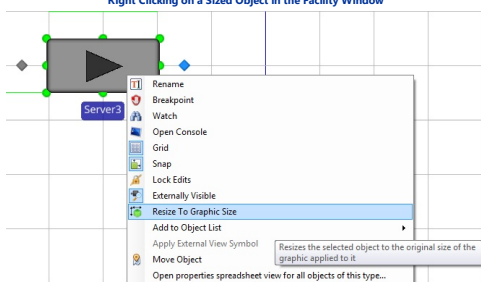
The right click on fixed and agent objects, such as Server, Combiner, Worker, or Vehicle, for example, also provides the option to 'Create Object From This'. This will create a new subclassed object of the object type, load its default values with the values from the properties of this object and replace the object with an instance of the new subclassed object. The subclassed object will be placed within project Navigation window and may be placed again in the model by using the project window. Subclassed objects that are created in the manner will then also have the option to 'Update ** Property Defaults From This' to update any changed property values. See the Creating New Objects page for more information.

Right Clicking on an Object in the Facility Window



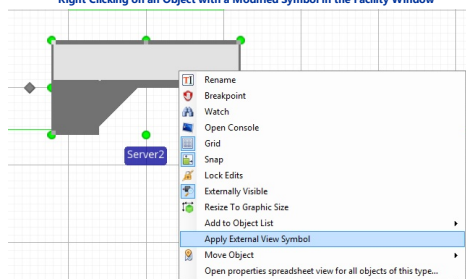
In addition to being able to set breakpoints and watch objects through the Right click menu, the options to make the object 'Externally Visible', as well as to 'Resize to Graphic Size' are also available for selection. When an object is 'Externally Visible' (on as a default), it is visible within the **External** window and will be seen if this object is used in hierarchy. The 'Resize to Graphic Size' is available when the graphic for the object has been changed from its original size. The 'Resize to Graphic Size' can be used to resize the object graphic(s) to the original size(s). This is useful in hierarchical models as well, as if an object in hierarchy has been changed such that the External view of the object is different, any models with that object in it will be sized based on the original object. The 'Resize to Graphic Size' can be then used to restore objects to their original sizes. Finally, you can Delete an object by right-clicking on it and selecting Delete.

Right Clicking on a Sized Object in the Facility Window



If an object's symbol has been changed from its default symbol to another graphic symbol, the option to 'Apply External View Symbol' is made available through the right click menu. This will change the new symbol back to the original symbol as defined through the External View of the object.

Right Clicking on an Object with a Modified Symbol in the Facility Window



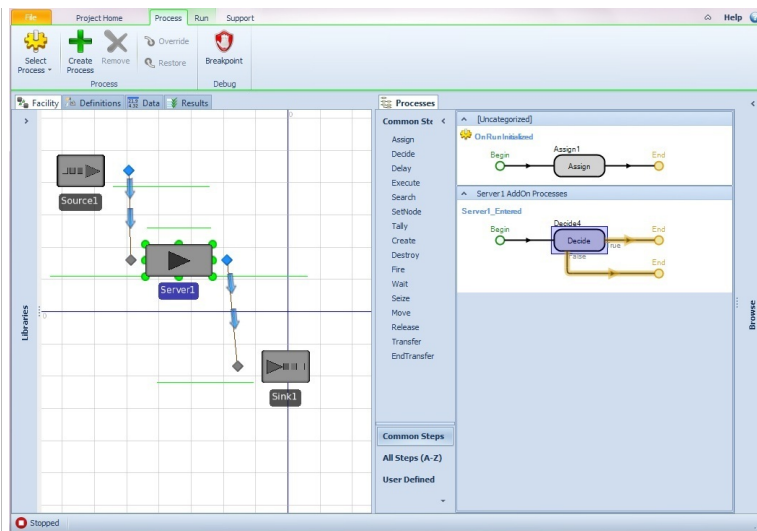
The Grid System

The Grid system that is available in the Facility window, the External Window and the window that is displayed for a New Symbol, has tick marks as well as labels to help you figure out spacing and placement. In Simio's grid system, one unit = 1 meter. Simio's grid displays labels of tens and then as you zoom out, it displays labels by hundreds. Continue zooming out and it will display kilometers. The zoom function is performed by right clicking within the grid and moving the mouse up and down. Or if you have a mouse with a scroll wheel, simply spin the wheel up or down for standard zoom in or out. Using the Alt+Scroll wheel provides 10x finer zoom than the standard scroll wheel zoom.

Arranging the Windows

It is possible to display multiple windows such as the Facility window and the Processes window at the same time. In order to access these windows and organize them as you desire, start by right-clicking on the tab of any window. A drop down menu will appear. In the example below, the Facility window was hidden behind the Process window because the user is working in the Processes window, which is the default behavior. Upon right clicking at the top of the Processes model tab, the user selects **New Vertical Tab Group**.

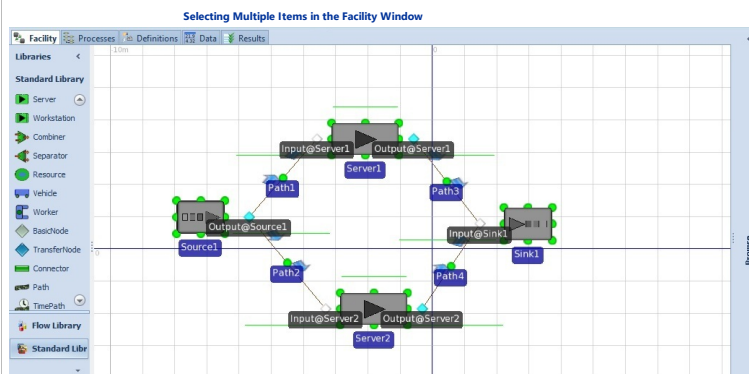
Vertical Grouping of Windows



Multi-Select

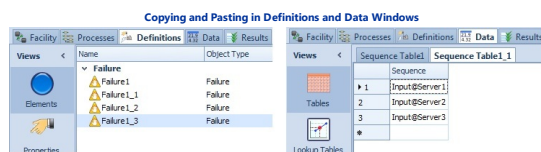
Simio supports Multi-select in the Facility window, as well as the Console and External windows of the Definitions tab. Multi-select allows users to select multiple objects for the purpose of moving, copying and/or deleting them. Multi-select in free space will do a box select, meaning it will select all objects within the box. Ctrl+click on any object adds (or removes) it to the selection set. While multiple objects are selected, they may be moved, copied or deleted.

Multi-select is also very useful for editing similar properties of objects within the above specified windows. Multi-select is also available within the Processes window for selecting Steps for the purpose of property editing. You can select multiple objects and the intersection of their set of properties will appear in the Properties window. If the objects share values for a property, the value will show up, otherwise the value will be blank. This feature is useful for editing multiple items, such as selecting multiple conveyors and changing the desired speed, selecting multiple links and changing them to bidirectional or selecting several transfer nodes and changing the transport logic to ride on a given vehicle. The [Properties Spreadsheet Window](#) is also useful for making edits to multiple objects and is available for the Facility window objects, as well as for the Processes window processes and steps.

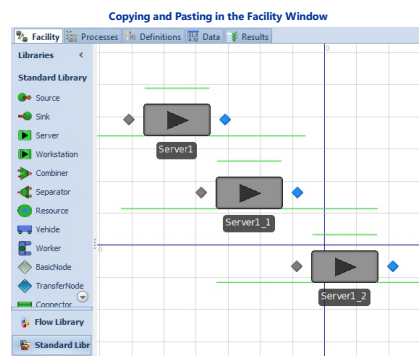


Cut / Copy / Paste

Simio supports Ctrl-X (Cut), Ctrl-C (Copy) and Ctrl-V (Paste) for all of the items within the Definitions and Data windows. This includes the Elements, Properties, States, Events, Lists, Tokens, External Nodes (in the External Window) in the Definitions Window, as well as the Tables, Function Tables, Rate Tables, Schedules and Changeovers in the Data Window. To copy an item, simply highlight the item and use Ctrl-C. To paste the item, use Ctrl-V and an identical copy of the item and any properties will be defined with a unique name, based on the original name and a unique number. The copy can then be renamed, if desired.



Simio also supports Ctrl-X (Cut), Ctrl-C (Copy) and Ctrl-V (Paste) for all of the items on the Drawing and Animation Ribbon tabs, as well items within the Facility and Process Windows. Ctrl-D (Delete) is currently supported only within the Facility window.



Undo / Redo

Simio supports comprehensive undo and redo capability across the product to make it easier to safely explore and to recover from any accidents. The Undo and Redo buttons are located on the quick access toolbar on the upper left. Ctrl-Z

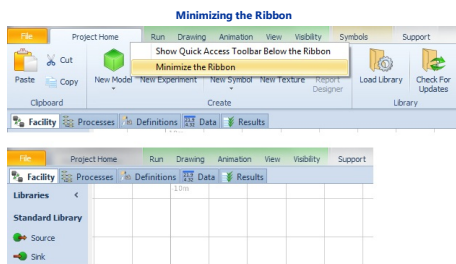
and Ctrl-Y also work.

Undo and Redo Buttons on the Quick Access Toolbar



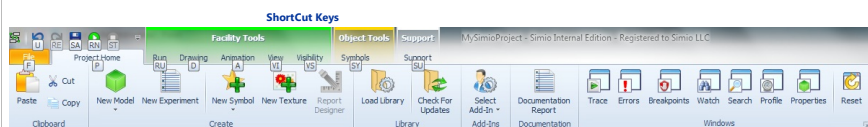
Minimizing the Ribbon

In order to get the maximum amount of working space in the interface, you can minimize the ribbon menu at the top of the interface. This is done by right clicking anywhere in the ribbon menu and selecting 'Minimize the Ribbon'. Similarly, it can be viewed again by right clicking on the ribbon tabs and selecting 'Minimize the Ribbon' again.



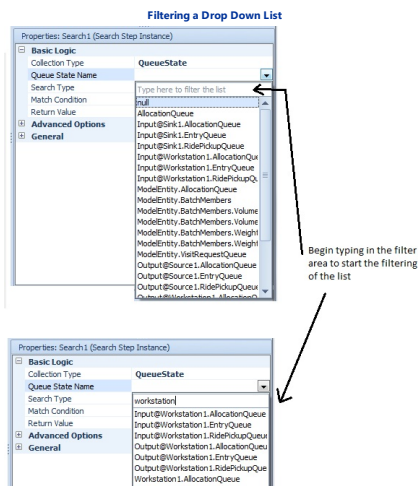
Clicking the ALT button

When the user clicks the ALT button, the hints for the shortcut keystrokes appear in the User Interface. While these shortcuts are displayed, the user simply needs to click on any of the keys that are shown and this will navigate them through the interface. For example, upon clicking the ALT button, the letter **F** appears over the icon for the File Tab. While the **F** is displayed, if the user clicks on the **F** key, the File tab's Backstage view will appear. The shortcut keys will disappear from the interface if the user clicks the ALT button again.



Filtering Drop Down Lists

Some of the drop down lists you'll find throughout the product will allow you to filter the list to narrow your choices. The example below shows the filter area where you can type in text to start the filtering of the list. This is extremely helpful in large models when the full list might be large and overwhelming.



Understanding the Content of the Windows

If you would like more information on where to find something in the interface, visit the [Windows](#) page to learn what can be found in each Simio Window, such as the Facility Window, the Definitions Window, the Processes Window, etc.

Application Settings

Within the File menu is a Settings option which will open the Application Settings window as shown below. The window is divided into separate categories.

The Graphics grouping includes the *Graphics Type*, which indicates the graphics driver to use for 3D display. Changes in this setting will take effect the next time Simio is started. The *DirectX Rendering Device* indicates the type of graphics driver to use for the 3d display. The *Compressed Texture Hardware Support* indicates if hardware support for compressed textures should be used. The 'Auto' setting will disable it for known bad graphic drivers. Users can manually enable or disable the hardware support for compressed textures. Changes to this setting will take effect the next time Simio is started. The *Hardware Instancing Draw Support* setting indicates if hardware support for instanced drawing should be used. Instanced drawing improves performance by making less draw calls for repeated geometry on the screen. The 'Auto' setting will disable it for unsupported hardware. The *Anti-aliasing Support* setting indicates if hardware support for anti-aliased drawing should be used. Anti-aliased drawing improves the visual fidelity of what is drawn at the expense of performance and memory. The 'Auto' setting will disable it for potentially weaker or unsupported hardware or graphics types (i.e., this is enabled for DirectX11 by default and disabled for everything else). Changes to this setting will take effect the next time Simio is started. The *Hardware Skeletal Animation* option indicates if hardware support for skeletal pose animations (like walking people) to be used. Hardware skeletal animation can offload the processing needed for skeletal pose animations (like walking people) to the GPU. The 'Auto' setting will disable it for potentially weaker or unsupported hardware or graphics types (i.e., it is enabled for DirectX11, disabled for others). Changes to this setting will take effect the next time Simio is started.

This window also includes a Libraries grouping which allows the user to specify whether or not to *Load Standard Library* and/or *Load Flow Library* and/or *Load Extra Library* upon opening Simio. By default, these libraries are both loaded. The *Additional Libraries To Load* option may also be indicated and may include a semi-colon separated list of fully qualified path names to additional libraries to load when opening a model.

The Simio Portal grouping includes the *Simio Portal URL* property, which determines the URL for the Simio Portal installation. For example, "https://mysimioserver". Portal functionality is only available for Simio RPS Edition.

The Experimentation grouping provides a place for a user to specify a list of addresses where Simio will look to find replication runners to be used with distributed runs in an experiment. This is useful if the replication runners you have will not work with the automatic WS-Discovery protocol or if you are connected via VPN. To enter an address, open the repeat group window for the Replication Runner Addresses line and type in the computer name or address and port where the *Simio Replication Runners* exist.

The GUI grouping includes the option to *Restore Docking Window Locations* in the software. If set to true, Simio will attempt to restore the top level docking windows to the same location they were when Simio last shut down. It also includes the ability to *Display Deprecated Steps In Processes Windows Panels*. When this setting is changed, it will require closing and re-opening Simio if the Processes window had already been opened. The option to *Display Deprecated Properties in Properties Window* is also available. This grouping includes the ability to *Display Deprecated Object Or Element Types*, where a change may require restarting Simio. Finally, this GUI grouping includes the *Font Scale (percentage)* indicating how much to scale the fonts used in various parts of the UI. A value of 100 is the default size, while using a larger size may help when using the UI on a projector in front of large groups. *Cache Dashboard Datasources* if set to true, uses

intermediate data file extracts when loading datasource data into dashboards in order to lower overall dashboard load times.

The *Show Version in Title Bar* if set to 'True', will show the current software version in the title bar text.

The Workflow grouping has the option to *Start In Scheduling Mode*. If this option is set to true and Simio is opened with a license that supports it (Simio RPS, for example), Simio will open in 'scheduling mode'. If you change this option, you have to restart Simio for it to take effect. Users can also use the *-start-in-scheduling-mode* command line option.

The Runtime grouping includes the option to *Use Scheduled Breakpoints*, which, if set to 'True', the breakpoints will be scheduled onto the event calendar, instead of suspending the run directly. Enabling this may make runs with breakpoints act differently than runs without breakpoints.

The Add-ins grouping has an option that was added to Sprint 154 for compatability reasons, which will *Allow Add-ins To Create Properties With Invalid Names* (i.e., such as those with spaces). This option is set to false by default.

The Project Recovery grouping was added to Sprint 187 and includes two options. *Minutes Between Recovery Save* indicates the number of minutes between automatically saving the active project for recovery. A value of '0' indicates that no recovery saves should take place. *Save For Auto-Recovery Before Run*, if set to 'True', will save the active project for recovery before the start of an interactive or experiment run. Project Recovery is used to restore potentially lost work after an unexpected Simio shutdown, such as an unexpected computer shutdown. A notification in the bottom right corner of Simio will be shown when autosaving. If Project Recovery is used, then when opening Simio after an unexpected Simio shutdown, the user will be asked if the recovery file should be used. Clicking OK opens the recovery save, saves it in a new file named [ProjectName]_Recovered, and deletes the recovery save. Clicking Cancel keeps the recovery save. Users can explicitly delete them by deleting the contents of the %localappdata%\Simio\Recovery directory, where %localappdata% is configured on most systems to be C:\Users\[user name]\AppData\Local. As of Sprint 200, if the user clicks Cancel on opening the recovery save file, a dialog will then prompt for deleting the project recovery save data.

Application Settings	
+ Graphics	
Render Type	Direct3D11
DirectX Rendering Device	
Compressed Texture Hardware Support	Auto
Hardware Instancing Draw Support	Auto
Anti-aliasing Support	Auto
Hardware Textured Animations	Auto
+ Libraries	
Load Standard Library	True
Load Plug Library	True
Additional Libraries To Load	
+ Simio Portal	
Simio Portal URL	https://test.portal.simioportal.com/app/highload/permissions
+ Experimentation	
Production Server Addresses	
+ GUI	
Restore Existing Window Locations	False
Display Depreciated Steps In Processors Window Panels	False
Display Depreciated Properties In Processor Windows	False
Display Depreciated Object Or Element Types	False
Start Guide (experimental)	000
+ Workflow	
Start In Scheduling Mode	False
+ Add-ins	
Use Scheduled Breakpoints	False
Allow Add-ins To Create Properties With Invalid Names	False
+ Project Recovery	
Minutes Between Recovery Save	5
Save For Auto-Recovery Before Run	True

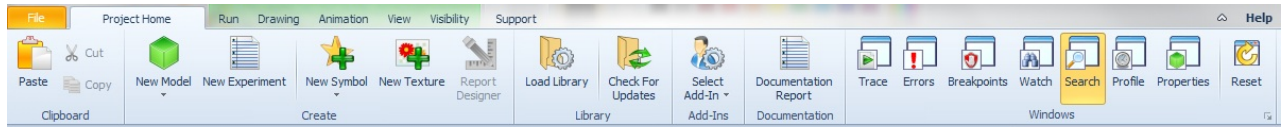
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

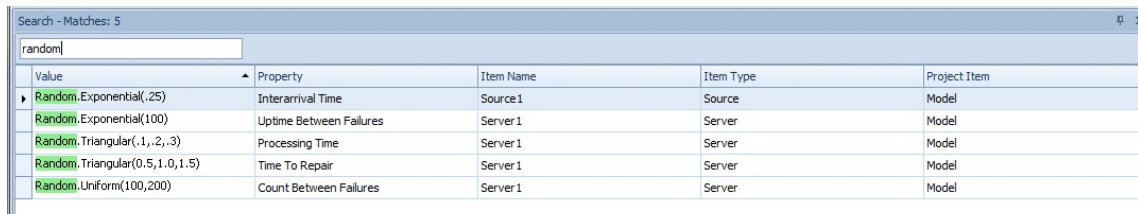
Search Window

Searching the Model

The Search icon within the Project Home window opens up the Search window to allow the user to search for any given text string.



Search Icon within the Project Home ribbon



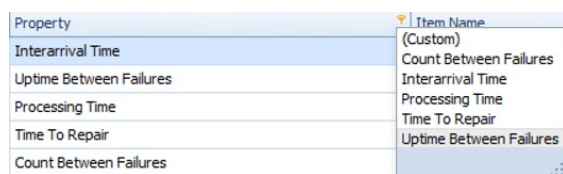
Search Window allowing Search of any text string

When you type something into the Search window's text box, the window will display everything that it finds that contains that text. There are four columns in the search results:

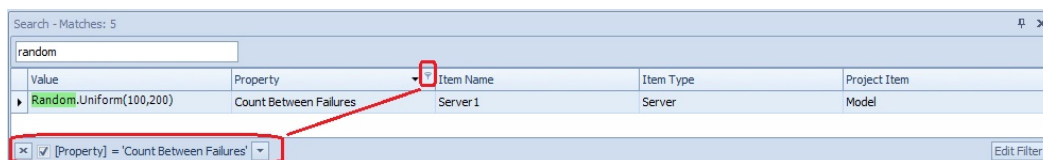
- "Value" is the actual piece of text containing the search string, with the search string highlighted.
- "Item" is the name for this piece of text; i.e. "Name" if it is the name of some object, or "Time To Repair" for a property.
- "Object Name" is the object containing the piece of text.
- "Project Item Name" is the top-level model containing the object.

Double-clicking on a row in the results will take the user to the window containing the item where the text is found, select the item, and then (if possible) select the property where the text is found. If the text found is a property value and that value is currently switched out, the user is simply taken to the item that contains that value. Currently, the Search window will search "Simio property" values (that is, property values on Steps, Elements, Processes, Object Instances), and Object names (where "Object" refers to any of the things a user names, such as aforementioned, but also the object in the Definitions, and Data windows). Search is not yet supported in Schedule item text, Changeover matrices, or Lookup / Rate table values.

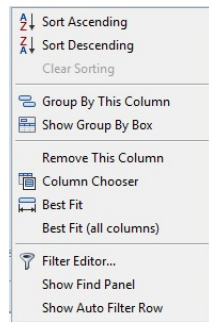
The columns within the Search window can be manipulated to sort and rearrange, as well as to filter them. To use this, click the filter glyph in a column header, and select an option from it.



When filtering, only those rows matching the filter criteria are displayed, and a filter bar appears the bottom.



The user may also right click on a column to sort, group or filter items in the column.



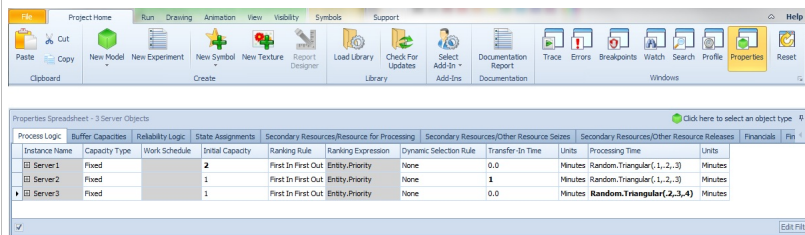
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Properties Spreadsheet Window

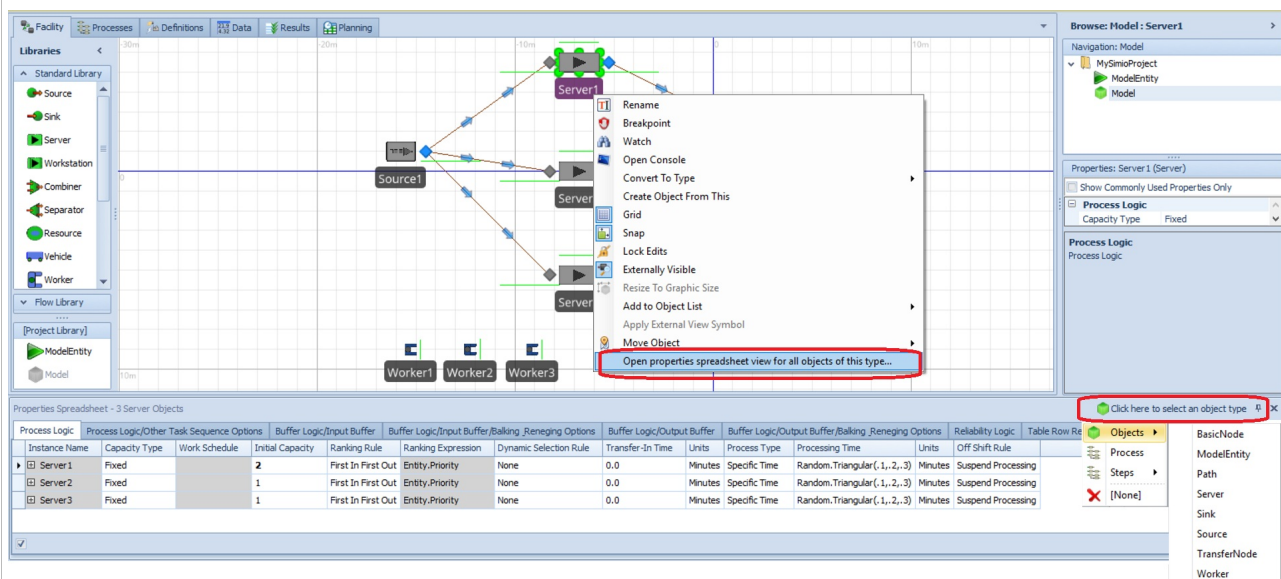
Viewing the Properties Spreadsheet Window

The Properties icon within the Project Home window opens up the Properties Spreadsheet window to allow the user to view and change properties of a specified object type, process or step in a spreadsheet interface. This window is available for use within the Facility and Processes windows.

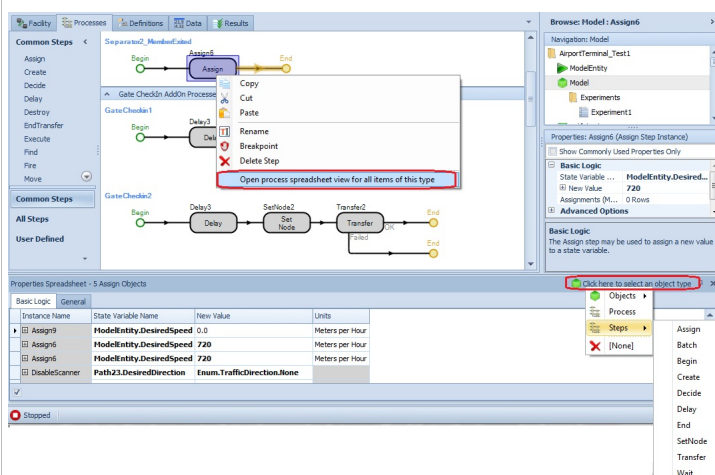


When the Properties Spreadsheet window is first opened, the 'Click here to select an object type' button in the top right may be used to select from a list of objects currently in the Facility window, Process names within the Processes window or Steps within the Processes window.

Within the Facility window, for example, if the model has a Source, two Servers and a Sink, connected by multiple Paths, the list of objects will include Source, Server, Sink, BasicNode, TransferNode and Path. Click on the object type to view a list of those particular objects in a spreadsheet format. Alternatively, if you have a particular object highlighted within the Facility window (i.e., such as a Server), if you right click on that highlighted object you will see the option to 'Open properties spreadsheet view for all objects of this type...' which will present those objects in the window.

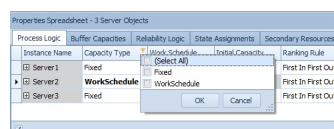


Similarly, within the Processes window, if the model has multiple steps, the list of possible steps may be accessed using the 'Click here to select an object type' and selecting the particular step type desired. Alternatively, right-clicking on a particular type of step, for example an Assign step, will open the properties spreadsheet for all Assign steps in the model for editing. Ctrl-click of multiple steps in the Processes window also allows editing of similar properties of those steps through the Properties window on the right.

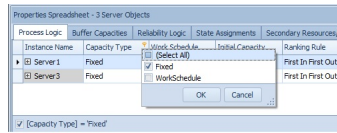


Within the Properties Spreadsheet window, you can edit single property entries for a given object or copy/paste values into multiple columns. To select multiple entry rows, you may click a row, then use Ctrl-click to highlight additional rows – editing of multiple rows simultaneously can continue to be done within the Properties window on the right.

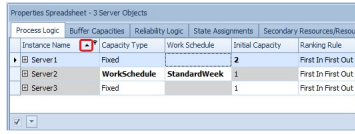
The columns within the Properties Spreadsheet window can be manipulated to sort as well as to filter them. To filter, click the filter glyph in a column header and select an option from it.



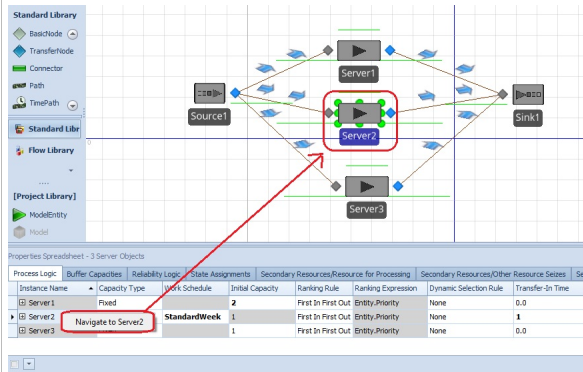
When filtering, only those rows matching the filter criteria are displayed, and a filter bar appears at the bottom.



The user may also right click on a column to see the sort arrow for displaying items in the column alphabetically.



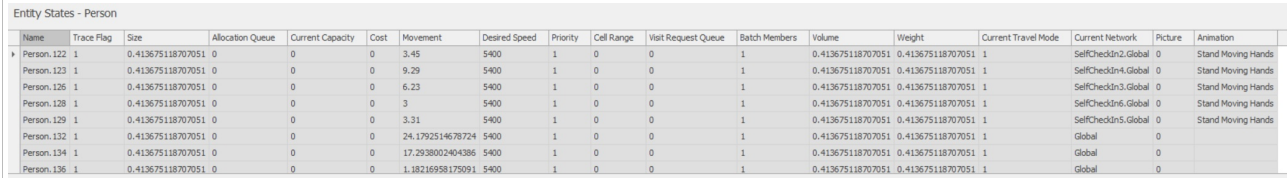
Right-clicking on a particular object's row (i.e., such as Server2 in a list of Servers) will highlight and allow you to 'navigate' to that particular object within the Facility or Processes window.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

The Entities icon within the Project Home window opens up the Entity States window to allow the user to double click on any entity listed in the window and navigate to where the entity is in animation.



Note: Entities that are in Queues and not visible in the model will not be found by the Entity States Window.

Copyright 2006-2025, Simio LLC. All Rights Reserved

20 / 1555

Converting Objects

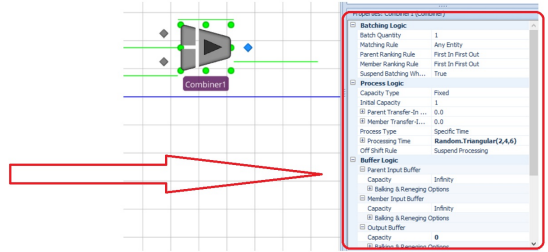
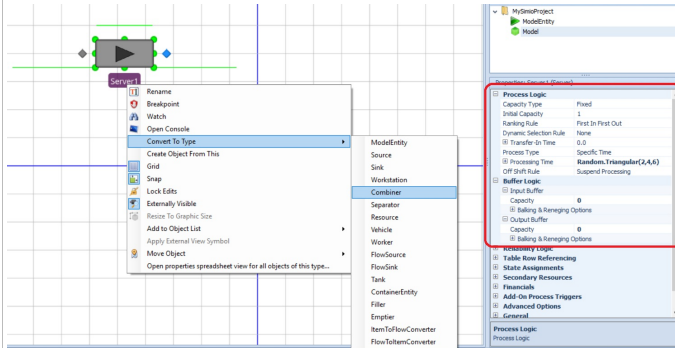
Convert To Type

Within the Facility window, the right-click option on an object allows the user to select 'Convert To Type' to convert the current object to a different object of a similar class. For example, a Path may be converted to any other type of 'Link' class, such as Connector, TimePath, Conveyor, FlowConnector or Pipe.

There are several rules that Simio uses to determine how to convert the data within an object from one object to another. Let's take the conversion from a Server to a Combiner for example. When 'Convert To Type' is used, Simio will look for a common base class between the From (Server) and To (Combiner) object (in this case Fixed).

For each property of that base class:

1. If the from object has a different value than its default value, copy the value over to the to object.
2. If the to object has a value equal to its default (it should, it's a new object), and the to object does NOT have type information (i.e. does NOT contain something like "Server Whatever"), and the from object DOES have type information, then do NOT copy the value over. Simio assumes this means that the from object has type information for which the to object makes no sense.
3. If the to object has any type information, do NOT copy value over. Simio assumes it is there for a specific reason and so that information is kept.
4. Otherwise, if the from value and to values are different, and they are both switched and visible, copy the value over.



Note that the Processing Time property was converted from Server to Combiner. Additionally, the Output Buffer Capacity of '0' was converted.

It's important to note that the Input Buffer Capacity of '0' DID NOT convert, as it is NOT the same property as the Parent Input Buffer Capacity of the Combiner.

Oculus Support

Render to Oculus button found on the Project Home ribbon will enable or disable rendering the active 3D view to an attached Oculus HMD device. Simio supports the Oculus Consumer Edition but currently does not support Oculus Touch controls.

In order to use the software with an Oculus HMD device, Simio must first be run in DirectX mode. This can be set by going to File > Settings and changing the Graphics Type (under Graphics heading). You may also have to explicitly specify the device the Oculus is attached to in File > Settings > DirectX Rendering Device. When running under DirectX, a list of possible device names can be found in the Simio3DStatus.log file in the "My Documents" directory, on the line starting "Available rendering devices are". Simio must be restarted for any graphics changes to take place. Users may have to go to the Oculus Home app and enable "Unknown Sources". On certain multi-GPU setups, like Optimus, users may need to start Simio by running it under a specific GPU. Under Optimus, this can be done by right-clicking on Simio and selecting "Run with graphics processor" and selecting the one in which Oculus is using. On more advanced GPU setups (for example, laptops with both integrated and discrete GPUs, plus an external GPU), you may need to disable one or more of those GPUs in windows Device Manager to get multi-gpu runtimes like Optimus correctly assigning Simio to the GPU the Oculus is attached to.

Since the Oculus view in Simio is simply a view, no simulation modeling can be done in this mode. To move around in the Oculus view, the desktop 3D view should be focus (users may need to click in it), and then the following keyboard controls may be used:

- W – Forward
- S – Back
- A – Strafe left
- D – Strafe right
- Q – Turn left
- E – Turn right
- R – Rise (in elevation)
- F – Fall (in elevation)
- V – Reset elevation

Users with an XInput controller plugged in (Xbox 360, Logitech F310, etc.) may also use the following movement controls:

- Left Thumbstick – Move forward/back, strafe left/right
- Right Thumbstick – Turn left/right
- Left Trigger – Rise (in elevation)
- Right Trigger – Fall (in elevation)
- Right Shoulder – Reset elevation
- A - Go to location in center of vision
- B - Return to origin

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Simio Concepts

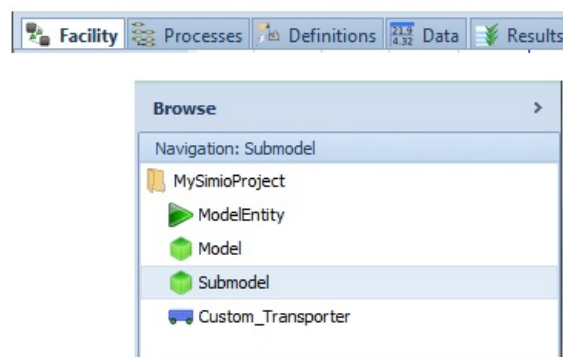
The Simio Object Paradigm

Simio is a *S*mulation *M*odeling framework based on *I*ntelligent *O*bjects. This may be a bit different than other simulation packages that you may be familiar with, even those that market themselves as object oriented. Simio is designed from the ground up to support the object modeling paradigm; however it also supports the seamless use of multiple modeling paradigms including a process orientation and event orientation. It also fully supports both discrete and continuous systems, along with large scale applications based on agent-based modeling. These modeling paradigms can be freely mixed within a single model.

The intelligent objects are built by modelers and then may be reused in multiple modeling projects. Objects can be stored in libraries and easily shared. A beginning modeler may prefer to use pre-built objects from libraries; however the system is designed to make it easy for even beginning modelers to build their own intelligent objects for use in building hierarchical models. An object might be a machine, robot, airplane, customer, doctor, tank, bus, ship, or any other thing that you might encounter in your system. A model is built by combining objects that represent the physical components of the system. A Simio model looks like the real system. The model logic and animation is built as a single step. An object may be [animated](#) to reflect the changing state of the object. For example, a forklift truck raises and lowers its lift, a robot opens and closes its gripper, and a battle tank turns its turret. The animated model provides a moving picture of the system in operation. Objects are built using the concepts of object orientation. However unlike other object oriented simulation systems, the process of building an object is very simple and completely graphical. There is no need to write programming code to create new objects. The activity of building an object in Simio is identical to the activity of building a model – in fact there is no difference between an object and a model. This concept is central to the design of Simio. Whenever you build a model it is by definition an object that can be instantiated into another model. For example, if you combine two machines and a robot into a model of a work cell, the work cell model is itself an object that can then be instantiated any number of times into other models. The work cell is an object just like the machines and robot are objects. In Simio there is no way to separate the idea of building a model from the concept of building an object. Every model that is built in Simio is automatically a building block that can be used in building higher level models.

Each object in Simio has its own [Processes](#), [Elements](#), [Properties](#), [States](#), and [Events](#). It also has an External view, which determines how the object will appear when placed into another model in the Facility Window of another model. Therefore, each object found in Simio's [Standard Library](#) has its own Processes, Elements, Properties, States, Events and External View. And since the main model is also an object in itself (a fixed object type), it also has it's own Processes, Elements, Properties, States, and Events.

When a user is working with multiple models within a project, it is important to remain aware of which model is the current active model. This can be found by looking in the Navigation window, found in the top right side of the interface, and finding the highlighted model. This is the active model. This means that the windows which are displayed in the main part of the interface are the windows associated with that active model. To view the windows for a different model, simply click on the appropriate model in the Navigation window and you will see a different set of tabs(windows) in the main part of the interface. The following image shows the model tabs for the active model called SubModel.



Even though each object within the standard library has its own Processes, Elements, Properties, States, and Events, the user cannot see those components of the object. Simio hides these details to prevent the user from making any changes to the logic of the standard objects. The standard objects were built to provide standard functionality that will meet most basic modeling needs, so hiding the details of these objects brings simplicity. If the user would like to see the Processes,

Elements, Properties, States, and Events of an object, they would need to subclass the object from the standard library into their own Project Library. Once the object is in the Project Library, the model windows of that object can now be viewed by making it the active model in the project, which is done by selecting the model in the Navigation window. This is where the user could see the Processes of any standard object (from the Processes Window) or the States and Properties of each object (from the Definitions Window). Now that the object is part of the [Project Library](#), it can be modified (i.e. new properties, new states, new tables, etc). If the user would like to modify a Process, they first need to select the Process from within the Processes Window and click on the Override button in the Ribbon. By overriding a Process and making changes, the user is changing the logic that Simio put into this object to make it work as it does in our Standard Library.

- View the [Object Hierarchy](#)
- Read about the different [Object Types](#) that exist in Simio and examples of each
- Read about [Processes](#), what they are and how they can be used in your model.
- Read about [Projects and Models](#) and the difference between the two.
- Read about [Queues](#) and see where they exist in the product.
- Read about [Tokens and Entities](#) and how they differ.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Object Hierarchy

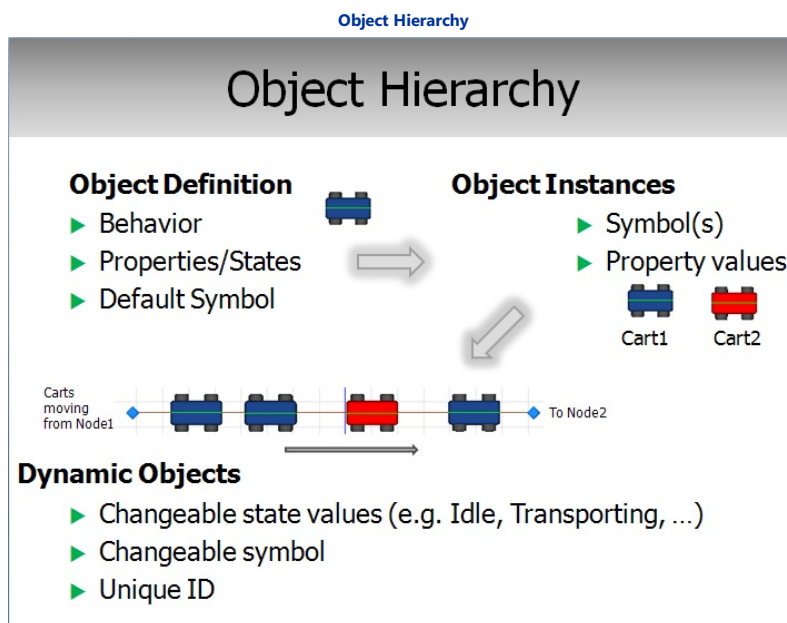
What is an Object?

Simio employs an object approach to modeling, whereby models are built by combining objects that represent the physical components of the systems. An object is a self-contained modeling construct that defines that construct's characteristics, data, behavior, user interface, and animation. Objects are the most common constructs used to build models. You've already used objects each time you built a model using the Standard Library --- the general-purpose set of objects that comes standard with Simio.

An object has its own custom behavior that responds to events in the system as defined by its internal model. For example, a production-line model is built by placing objects that represent machines, conveyors, forklift trucks, and aisles, while a hospital might be modeled using objects that represent staff, patient rooms, beds, treatment devices, and operating rooms. In addition to building your models using the Standard Object Library objects, you can also build your own libraries of objects that are customized for specific application areas. And you can modify and extend the Standard Library object behavior using process logic. Refer to [Simio Concepts](#) for additional information on the specifics of an object.

Object Hierarchy

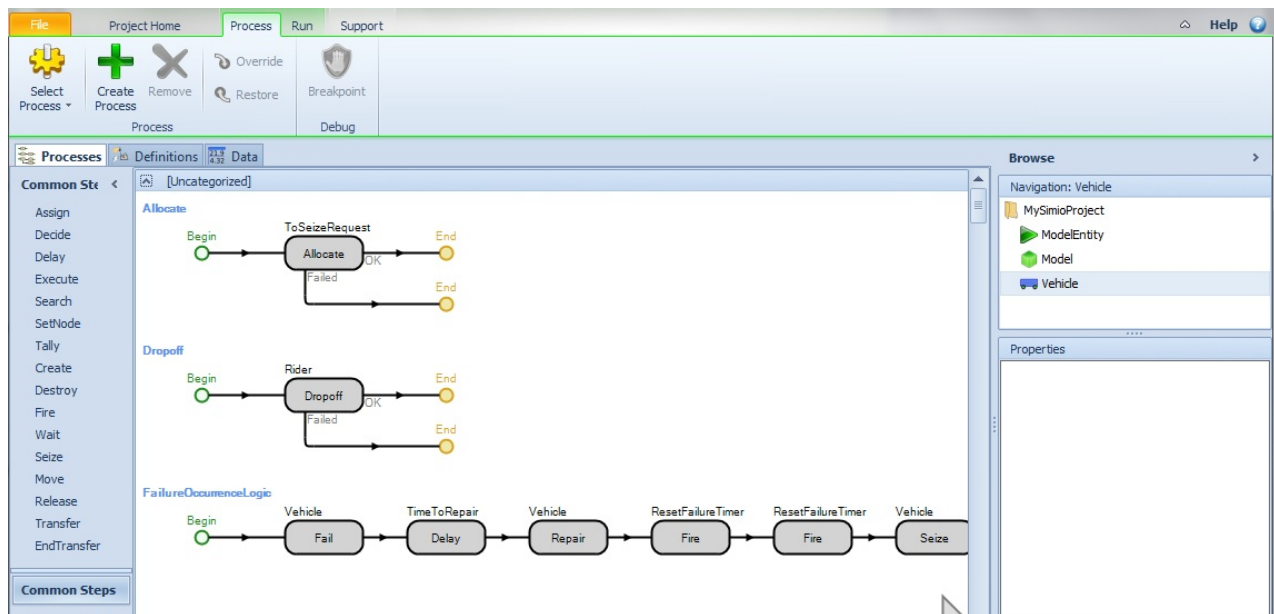
One of the important and unique internal design features of Simio is the use of a three tier object structure that separates an object into an **object definition**, **object instance**, and **object realization**.



Object Definition

An **object definition** is a Library Object that is supplied by Simio or created by the Modeler. An object definition is just that - a definition of how the object is to act by making use of specific [Processes](#), [Events](#), [States](#), and [Properties](#) that work together to define how the object interacts with Entities and other Objects. An object definition specifies the behavior of that object and is shared by all instances of the object in all models in which it is used. Object definitions reside in a library, either the Standard Library, a custom library, or a project library. New object definitions can be made by selecting New Model and selecting the class of model to be built. Alternatively, a new object definition can be generated by 'sub-classing' an object from an existing library object.

Vehicle Definition (Processes)



Vehicle Definition (Properties)

Name	Object Type	Display Name
Properties (Inherited)		
WorkDayExceptions.Properties (Inherited)		
WorkPeriodExceptions.Properties (Inherited)		
Properties		
ParkWhileBusy	Boolean Property	Park While Busy
TaskSelectionStrategy	Enum Property	Task Selection Strategy
LoadTime	Expression Property	Load Time
UnloadTime	Expression Property	Unload Time
ParkToLoadUnload	Boolean Property	Park to Load/Unload
MinimumDwellTimeType	List Property	Minimum Dwell Time Type
MinimumDwellTime	Expression Property	Minimum Dwell Time
MinimumDwellTimeExpirationEventName	Event Property	Event Name
MinimumDwellTimeCondition	Expression Property	Dwell Only If
InitialNode	Node Property	Initial Node (Home)
RoutingType	List Property	Routing Type
RouteSequence	Sequence Property	Route Sequence
IdleAction	List Property	Idle Action
OffShiftAction	List Property	Off Shift Action
FailureType	List Property	Failure Type
FailureEventName	Event Property	Event Name
CountBetweenFailures	Expression Property	Count Between Failures
UptimeBetweenFailures	Expression Property	Uptime Between Failures
TimeToRepair	Expression Property	Time To Repair
RunInitializedAddOnProcess	Process Element Property	Run Initialized
RunEndingAddOnProcess	Process Element Property	Run Ending
AllocatedAddOnProcess	Process Element Property	Allocated

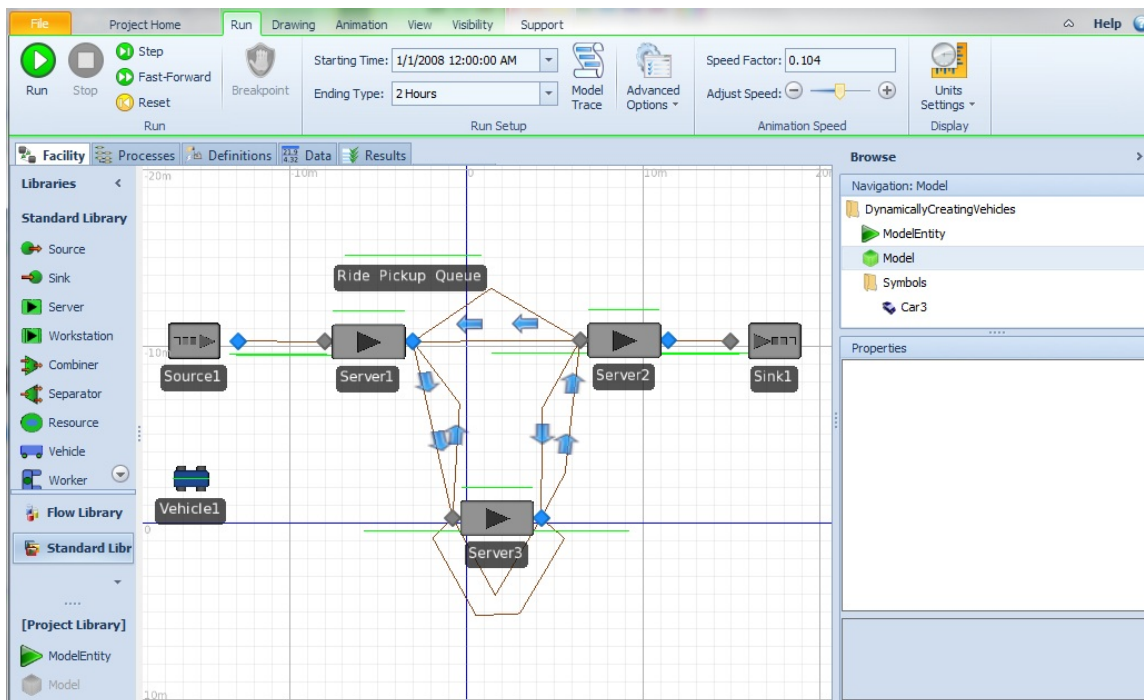
Object Instance

An **Object Instance** is simply an occurrence of an Object Definition within a parent object, such as a model or another object definition. This is what you get when you place a definition into the Facility window.

In the SimBit example below, the 'model' (which is an object definition), contains a number of object instances within the Facility window. Each Server (Server1, Server2, and Server3) is an Instance of the Object Definition named 'Server'. Similarly, Source1, Sink1, all Paths, Vehicle1, and all Input/output Nodes are all Instances of Source, Server, Path, Vehicle, BasicNode, and TransferNode Object Definitions.

Each instance of an Object Definition is differentiated by customizing property values and/or creating Add-On processes within the Instance. In this example, each Server has a different Processing Time property value. Also, some TransferNodes have Ride on Transporter set to True while others are set to False. This instance data is in turn shared by all object realizations.

SimBit 'DynamicallyCreatingVehicles'

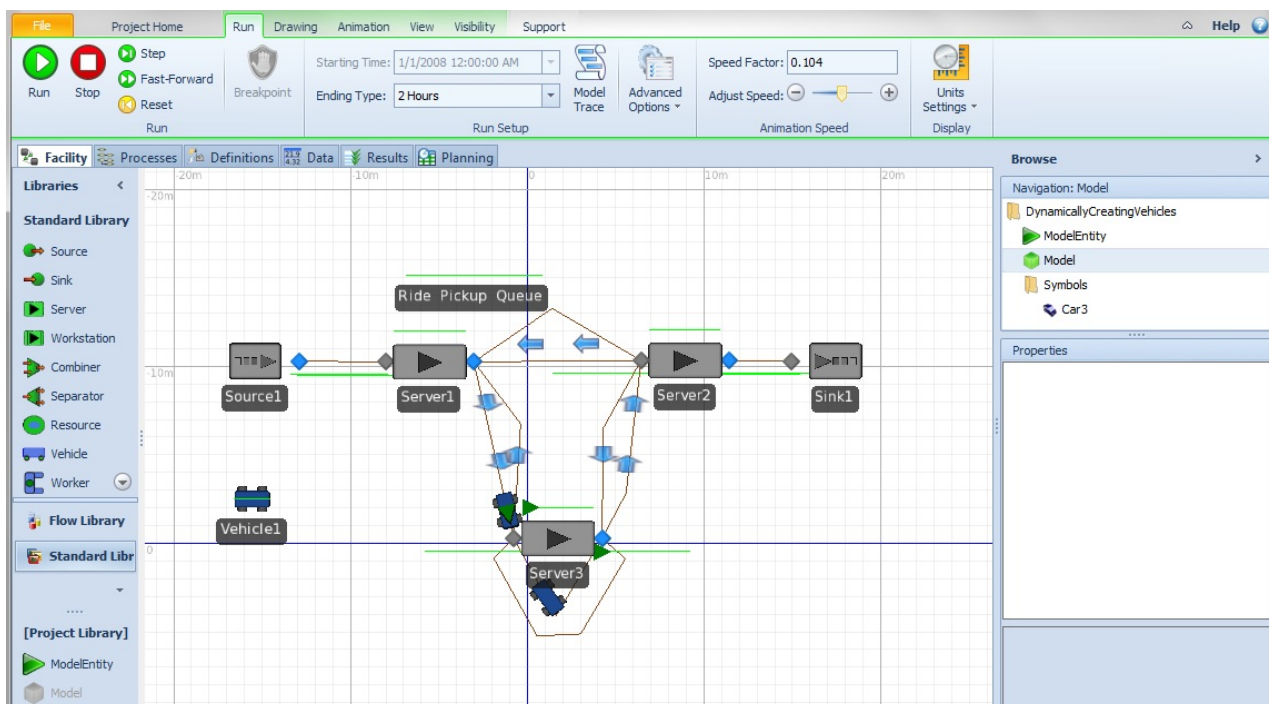


Object Realization or Object Runspace

Once a simulation run is started, Object Instances are represented with object realizations. Object realizations hold the state values for that object instance. Each realization of an object instance is unique because of its unique combination of state and function values.

All objects have a fixed runspace. A fixed runspace is the physical X,Y,Z location where an object instance is created once a run is started and is located where the object instance was placed in the Facility window. Dynamic Objects are created at this location and then are transferred to another location. Below, Vehicle1's fixed runspace is located at the bottom left hand corner of the screen, the stationary vehicle labeled Vehicle1.

In the case of dynamic objects, there is also a dynamic runspace created to represent the number in system for a particular object instance. Each dynamic runspace has independent values for states and functions. You can see the three dynamic runspace instances of Vehicle1 moving throughout the system. Each of these three vehicles have different X,Y,Z locations, they are doing different tasks – one of them is going to pick up an entity while another is carrying an entity to its destination – and therefore have different values for their Destination, RideStation.Capacity.Remaining function, etc.



In summary, the Vehicle Object Definition is the "Model" supplied with the Simio Standard Library. It is the collection of all the Properties, States, Event, Processes, Elements, etc. that make a Vehicle act the way the standard Vehicle acts in your model.

The Vehicle Object Instance is Vehicle1 that is shown in the bottom left hand corner of the Facility window. This is where the properties such as Desired Speed, Initial Ride Capacity, Task Selection Strategy, etc. are specified. These property values further shape this Vehicle's behavior to represent the behavior of the vehicle the user is trying to represent.

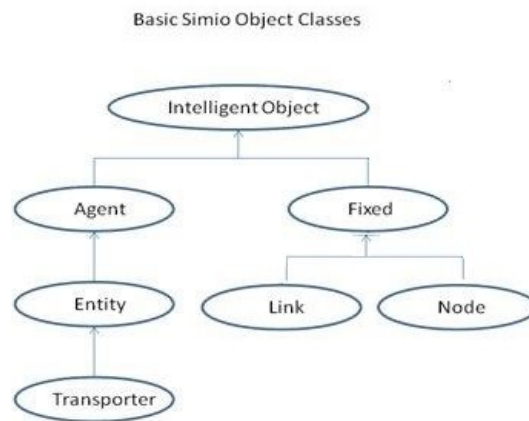
When you run the model, it will create a static runspace (or realization) to represent the states common to all Vehicle1s (such as Population.NumberInSystem). It will also create a dynamic runspace for each of the three vehicles that are in the system. Each one of these vehicles has different values for Location, Heading, Pitch, NumberRiders, TimeCreated, etc. that differentiates that Vehicle from the others. These can be then referenced by Vehicle1[1], Vehicle1[2], and Vehicle1[3].

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Object Types

There are six basic classes of intelligent objects in Simio:



Listed below are the **Object Types** in Simio:

Type	Derived From	Description
Intelligent Object	None	A base object with the optional ability to be seized, released, and follow an availability schedule. View all the Functions available for Intelligent Objects.
Fixed	Intelligent Object	Typically used to represent an entire system being modeled (e.g., the plant), or component objects within a system that have a fixed location (e.g., machine, equipment, work cells). A Fixed object has a stationary location within the Facility Window. It is typically used to represent an entire system being modeled (e.g., the plant), or component objects within a system that have a fixed location (e.g., machine, equipment, work cells). Examples of fixed models include Source , Server , and Sink , in the Standard Object Library, as well as the Link and Node.
Agent	Intelligent Object	Adds behaviors for modeling objects that can be dynamically created and destroyed, are able to move in continuous space or discrete space (on a grid), and which can detect, chase, and intercept other objects. This type of model is particularly useful for agent-based modeling approaches in which a large number (perhaps many thousands) of independently acting agents interact to create the overall behavior of the system. In the current version of Simio, a user cannot add a new Agent Class object types. However, both Entities and Transporters are derived from the Agent Class object type.
Entity	Agent	Adds behaviors for modeling objects that can follow a work flow in the system, including the ability to use a network of links to move between objects, the ability to visit, enter, and exit locations within other objects through nodes, and the ability to be picked up, carried, and dropped off by transporter objects. An Entity object can be dynamically created and destroyed and can move into and out of Fixed objects. Entities can have multiple graphical symbols. An example of an Entity model is the default ModelEntity that is automatically added with the first Fixed model of the Project.
Transporter	Agent	A transporter object is a special type of entity that can pickup entity objects at a location, carry those entities through a network of links or free space, and then drop the entities off at a destination. A transporter object also has the ability to move off of a network while maintaining association with a node on the network (i.e., "park" at a node

		in a network). Examples of a transporter model are the Vehicle and the Worker in the standard library.
Link	Fixed	<p>Adds behaviors for modeling fixed objects that are pathways for entity/transporter movement. A link object has a length which may be separated into equally spaced locations (cells), must have a start node and end node, and is a member of one or more networks. A Link object transfers an Entity over a pathway defined by a polyline connecting two Node objects in the Facility Window. A Node is highlighted red when a Link is being drawn and it is close enough to that highlighted Node that if the user clicks, the link will be anchored at that Node. Examples of Link models include the Connector, Path, TimePath, and Conveyor in the Standard Library.</p> <p>Simio tracks the leading and trailing edge of each Entity on a link as well as provides events to manage collisions and passing. Link control and position is determined by the leading edge of an Entity. As soon as an Entity's leading edge moves onto a link, the Entity and its speed are controlled by the logic built into that Link. When the Entity's leading edge moves off the link (perhaps onto a connected link), the Entity is no longer controlled by the initial Link even if the trailing edge of the Entity remains on the initial Link. Control transfers to the subsequent Link as the Entity's leading edge transfers.</p> <p>A traveling entity is considered Accumulated on a link if the entity has reached the end of the link and has been stopped there without being engaged to the link, or if the entity's leading edge has collided with the trailing edge of an entity in front of it on the link, and the entity has accumulated behind that entity without being engaged to the link. Once flagged as Accumulated, the entity will continue to be considered Accumulated until either its leading edge leaves the link or the collision situation is cleared.</p>
Node	Fixed	<p>Adds behaviors for modeling fixed objects that are intersection points between link objects or the entry/exit points for visiting an object. Entities may be picked up/dropped off by a transporter at a node. Users can extend/customize the crossing logic through a node to model network flow and entity pickup/dropoff points. Examples of Node models include the BasicNode and TransferNode in the Standard Library.</p> <p>A Node defines a point in space -- it may constrain movement but it does not represent any physical space itself. So, for example, a transporter that stops at a node physically remains with its full length on the incoming link until it leaves the node. The Standard Library object nodes have an associated parking area, but by default this also does not represent a physical location. Internally parking is represented as a station, so it may have constrained capacity and may be animated.</p> <p>The following key strokes can be used to manipulate Nodes within the Facility Window:</p> <ul style="list-style-type: none"> • Click – Selects node and displays properties in the Property window. • Click and drag – Moves the Node to a different screen location. Note that it is still “attached” to its associated object if any (for example, a Server), and if that associated object is moved, the Node position will also change. • Ctrl+Shift+Click and move – Initiates the creation of a Link between Nodes. The Link type may be preselected in the library panel or it will be prompted for when the Link is terminated by clicking on a Node. <p>A node with its <i>Crossing Capacity</i> property set to 'Infinite' is not seized or released and therefore cannot be queried for Capacity.Allocated.</p>

Copyright 2006-2025, Simio LLC. All Rights Reserved.

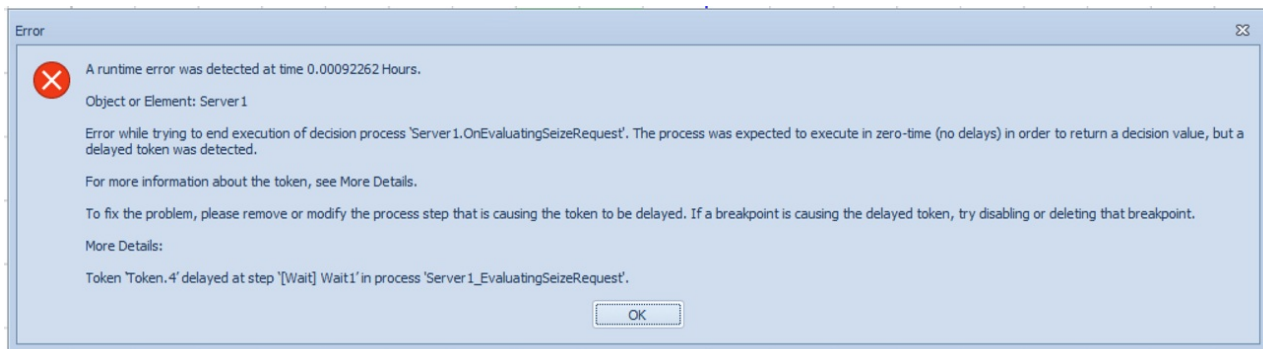
Send comments on this topic to [Support](#)

Processes

A Process in Simio is comprised of [Steps](#), [Elements](#), and [Tokens](#). A Process is sequence of actions (e.g. assign state, delay by time, seize a resource, etc.) that may span time and change the state of the model. Tokens flow through a Process executing Steps that alter the state of one or more Elements. Simio provides an auto-layout function creating Process flows. Processes are created and modified in the [Processes Window](#), found by clicking on the Processes tab. Note: A process cannot be modified while the model is running.

A Process is either enabled or disabled. The process property *InitiallyEnabled* allows the user to specify whether a process is enabled when the system is initialized. A process has a state called Enabled that indicates whether the process is currently enabled. A disabled process ignores any attempt to execute the process. A process can be enabled by using an Assign step to set the state ProcessName.Enabled, where ProcessName is the name of the process, to 1. A process can be disabled by using an Assign step to set the state ProcessName.Enabled to 0. Any attempt to execute a disabled process (e.g., Execute step) will be ignored. Any triggering events for the process are canceled when it is disabled. However, when a process is disabled, if there are active tokens currently running in the process, then those tokens will continue and finish their processing. A disabled process (or a process that has no logic) is going to simply return its ReturnValue if asked (e.g., if an OnEvaluatingXXXX process is asked for a value by the engine). ReturnValue is a state associated with a process that returns the default token return value for the process.

There are certain processes within the standard Simio objects that are known as Decision Processes. They are referred to as Decision processes because the execution engine needs to ask the model for some value and it runs this particular process to find the answer. These processes should run to completion without the delay of the executing token. If for any reason the token gets delayed while executing the process, then an error message will be displayed, as shown below. The error details will inform the user which exact token was detected as being delayed. If a Decision Process contains an Execute step, the same applies with an error being issued with any delayed token(s).



Decision Process must run to completion without any interruption; however, the user can set a Breakpoint within the process to step through the decision logic. Process suspensions are not allowed. Decision processes within Simio include OnEvaluatingMoveRequest, OnEvaluatingRiderAtPickup, OnEvaluatingRiderReservation, and OnEvaluatingSeizeRequest.

Processes

Processes are used to either customize the behavior of an existing object, or to create new object definitions. If the user would like slightly different behavior from what is provided with the Standard Library objects, the modification of the object behavior is done with Simio Processes. The second major role for process modeling in Simio is for [creating new object](#) definitions. An object definition is simply a model for how instances of that object should behave.

There are several types of Processes in Simio. Standard Processes are automatically run by Simio at specific points in the logic. Decision processes are 0-time Standard Processes that compute a return value used for making a decision. Event-triggered Processes are executed whenever a user-specified Event occurs. Add-on Processes may be inserted into an object at selected points in the logic and are discussed below.

Standard Processes

There are nine (9) standard processes that are available for a given fixed class object/model. There are different/additional processes within other class type objects (Entity, Transporter, Node, etc.). Within the Processes window, any of these may be selected for logic to be added. Some of the processes below are only available if the Fixed Class object property *Resource Object* is set to 'True' (thus the object is used as a resource with capacity.) These processes include:

- **OnCapacityAllocated** - This is executed when a resource's capacity is allocated (seized). The token executing the process will be associated with the object that seized the capacity. This process is used within Standard library objects such as Resource, Worker and Vehicle.
- **OnCapacityChanged** - This is executed when a resource's capacity has changed. The token executing the process will be associated with the object that changed the capacity.
- **OnCapacityReleased** - This is executed when a resource's capacity is released. The token executing the process will be associated with the object that released the capacity. This process is used within Standard library objects such as Resource, Worker and Vehicle.
- **OnCapacityReservationCancelled** - This is executed when a capacity reservation for a resource has been cancelled, either due to a reservation timeout or because the entity owning the reservation was destroyed. The token executing the process will be associated with the object that cancelled the reservation. This process is used within Standard library objects such as Resource, Worker and Vehicle.
- **OnEvaluatingSeizeRequest** - This is executed when determining whether or not the object's capacity should be allocated (seized). The token executing the process will be associated with the object that is trying to seize the capacity. This process is used within many of the Standard library objects.
- **OnNewSeizeRequest** - This is executed **once** for each candidate resource when a token first arrives to a Seize step (i.e., when a new seize request occurs). The token executing the process will be associated with the object that is trying to seize the capacity. Note: When the token arrives to Seize step, it calls the process (optionally) for every possible candidate. The OnEvaluatingSeizeRequest process is then what is used to actually reject or accept and that process may be called multiple times whenever a queue search is done.
- **OnRunEnding** - If running in interactive mode, then this is executed when the Stop or Reset button action occurs. If in experiment mode, then when the ending time has been reached or calendar events have been exhausted such that the model run is now ending and final statistics about to be reported. Additional events can be put onto the event calendar, but they will not be processed. For example, if you place a Delay step in this process, the delay will be scheduled, but the run will end before the end of the delay. The delay will not extend the run time. Similarly, if you have an Execute Step in this process and the *Action* property is set to 'WaitUntilCompleted', anything after that step will not be processed. This is because the "WaitUntilCompleted" action schedules another event on the event calendar and the event calendar is not revisited.
- **OnRunInitialized** - This is executed when this object/model is first initialized. It's important to note that the

OnRunInitialized processes are executed last after all model element states and statistics have been initialized. Thus, all Timer elements in a model have already been initialized by the time the OnRunInitialized process is being executed. This includes any Timer elements that are embedded within objects, such as the Timer element for entity arrivals within a Source. Thus, any state variables referenced within the Source object should be initialized to their desired value, as an assignment within the OnRunInitialized process will not override the Timer element initialization.

- **OnRunWarmUpEnding** - If running in experiment mode, if a warm-up period has been specified and is ending, this is executed right before all statistics are cleared. Typical uses of this process may be to do some custom warm-up statistics clearing. For example, you may want to clear a state variable's value that is being used as a custom statistic or take a snapshot of some statistic values at the end of the warm-up period before Simio clears them. See "OnRunEnding" description above for more information about how the event calendar works with this Process.

Add-On Processes

When you place an object from the Standard Library into your Facility Window, you can have the object run Add-On Processes at specific points in the logic. An [Add-On Process](#) is simply a logical Process that you have created to perform some specific action such as assigning a state variable, seizing a resource, referencing information in a table, etc.

Process Scope

In general a process can only reference items that are at its own scope. Specifically, an object does not "know" anything about the model in which it is placed, so any processes defined inside that object have the same limitation. For example, a process defined inside a Server object can only reference things that have been defined within that Server object. One way objects and their process can interact with the other objects is by the object defining properties which are used to pass in information from outside the object. For example, Server has a Property called *Process Time* which may reference or "pass in" things like tables and functions that are defined outside of Server.

Add-on processes may appear that they are being defined at the scope of the referencing object (e.g. a Server), but they are not. An add-on process is actually part of the containing model and has that scope. So for example, if the Processing add-on process is defined for Server1 that is placed in model Brewery, the scope of that add-on process is Brewery. This is important to note because if the same code that is defined in an add-on process is later moved into the object definition, it may no longer work because the scope would change from the containing model to that of the object definition itself.

Simplifying Logic within Processes

A process can be structured such that all the logic that is to be done at a given point in the simulation is done within that particular process. However, users may find that some of the logic at different points in the model is similar to other areas of logic. The Execute step may be used to move from one given process into another process. When the Execute step is used, a new token enters that process and continues through the steps. Within the original process (where the Execute step is called), the original token may continue or it may wait for the executing process to be completed. See the [Execute](#) step for additional information.

If you use a custom token with custom states, the states of the receiving token in the called process get set to match, much like passing function arguments. If you set the ReturnValue of the token in the executed process, that value will be set (returned) to the token that started the execute, much like a function return value. This allows the user to pass information between the processes, much like you would with a function. See the [Tokens](#) page for more information.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Costing

Costing in Simio

Activity-based costing (ABC) is a method of assigning costs to products or services based on the resources that they consume. ABC identifies activities within the business and records the cost required to fulfill each activity. An activity cost driver is any factor that affects the costs associated with an activity including capital costs, resource consumption costs, and waiting or inventory holding costs. Activity-based costing allows managers to determine the costs to perform an activity as well as non-allocated costs such as those associated with unallocated (idle) resource time.

Simio's costing capability includes the ABC features described above plus the ability to specify costs in many different currencies and use a user-specified exchange rate to express cost statistics in the currency of choice.

Cost Centers & Cost Hierarchy

Cost centers are used to define identifiable areas of responsibility where costs are incurred or allocated. Cost centers may be linked in a hierarchical structure for cost accounting and reporting purposes, whereby costs are automatically rolled up through the cost center hierarchy. For example 'MechanicLabor' and 'ElectricianLabor' may both be allocated to the 'MaintenanceLabor' cost center. In turn costs associated with the 'MaintenanceLabor' cost center may be combined with costs in the 'MaintenanceSupplies' cost center and allocated to the 'MaintenanceTotal' cost center.

Each Standard Library object that supports costing has a *Parent Cost Center* property that can be specified. If none is specified, the costs associated with that object will be allocated to the parent object (which may be the base 'model') containing the object. Cost centers may also be defined within the Elements panel of the Definitions window. A cost center may have an initial cost value in addition to an initial cost rate.

When cost hierarchy is used, it is important to note what each cost center represents to avoid double counting specific costs. To continue the above example, it would be inappropriate to add together the costs from 'MaintenanceLabor' and 'MaintenanceTotal' because that would be double counting the 'MaintenanceLabor' costs.

Cost Drivers

Capital Costs

Capital costs are the initial one-time setup costs of adding an object to a system. Capital costs are included in the following Standard Library objects: Source, Sink, Server, Combiner, Separator, Workstation (Deprecated), Resource, Vehicle, Worker, and Conveyor.

For Vehicles and Workers, the capital cost is incurred for each object in the population that is added to the system. For example, if the Vehicle named ForkTruck has an *Initial Number in System* of '2', capital cost will be incurred for each of the two ForkTruck vehicles.

For statistics reporting, capital costs are allocated to both the object and the parent cost center associated with the object. They are shown on the Results page under the Category 'Costs' and Data Item 'Capital Costs'. In the example shown below, capital costs are specified for four objects, including Source, Server, Sink and Resource, each with its own cost value. As you can see, each object itself incurs the capital cost, and then all costs are 'rolled up' to the parent cost center, which is the Model.

Average						
						Drop Column Fields Here
Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	[Object]	Costs	TotalCost	Total (USD)	81.4689
Server	Server1	[Object]	Costs	CapitalCost	Total (USD)	15.0000
				TotalCost	Total (USD)	19.2689
		InputBuffer	Costs	TotalCost	Total (USD)	2.2689
		OutputBuffer	Costs	TotalCost	Total (USD)	2.0000
Sink	Sink1	[Object]	Costs	CapitalCost	Total (USD)	2.0000
				TotalCost	Total (USD)	4.0000
		InputBuffer	Costs	TotalCost	Total (USD)	2.0000
Source	Source1	[Object]	Costs	CapitalCost	Total (USD)	2.0000
				TotalCost	Total (USD)	48.2000
		OutputBuffer	Costs	TotalCost	Total (USD)	2.2000
Worker	Worker1[1]	[Object]	Costs	CapitalCost	Total (USD)	10.0000
				TotalCost	Total (USD)	10.0000

Buffer Costs

Buffer costs are the costs associated with holding an entity in an input or output buffer at a given object, such as a Server. The Standard Library objects that include buffer costs are Source (output buffer costs only), Sink (input buffer costs only), Server / Separator / Combiner / Workstation (Deprecated) (both input and output buffer costs).

Buffer costs include both a Cost per Use and Holding Cost Rate.

Cost per use for a buffer is the cost to hold an entity in a buffer, irrespective of the waiting time. It is important to note that this cost will be incurred at the associated object each time an entity passes through the buffer, even if there is no waiting time. Cost per use is allocated to the entity moving through the buffer, the buffer's associated location and the Parent Cost Center.

Holding cost rate for a buffer is the cost per unit time to hold an entity in the buffer. The holding cost is then allocated to the associated object, as well as to the Parent Cost Center. Holding costs are also allocated to the entity that is in the buffer.

For statistics reporting, buffer costs are shown on the Results page under the Category 'Costs' and Data Item 'CostPerItem' for the associated DataSource 'InputBuffer' or 'OutputBuffer' (or 'Population' for Entities). In the example shown below, buffer costs are specified for three objects, including Source, Server, and Sink, each with its own buffer cost data. Cost per use was specified for all objects, and holding cost rate was specified for the input buffer of the Server. As you can see, each object itself incurs the buffer costs, the entities that go through the system incur the cost (which will be different for each entity, given the waiting times at the buffers are different) and then all costs are 'rolled up' to the parent cost center, which is the Model.

Average						
						Drop Column Fields Here
Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	[Object]	Costs	TotalCost	Total (USD)	52.4689
ModelEntity	DefaultEntity	[Population]	Costs	CostPerItem	Average (USD)	2.4032
					Maximum (USD)	2.4144
					Minimum (USD)	2.4000
Server	Server1	[Object]	Costs	TotalCost	Total (USD)	4.2689
		InputBuffer	Costs	CostPerItem	Average (USD)	0.1032
					Maximum (USD)	0.1144
					Minimum (USD)	0.1000
		OutputBuffer	Costs	CostPerItem	Average (USD)	0.1000
					Maximum (USD)	0.1000
					Minimum (USD)	0.1000
Sink	Sink1	[Object]	Costs	TotalCost	Total (USD)	2.0000
		InputBuffer	Costs	CostPerItem	Average (USD)	0.1000
					Maximum (USD)	0.1000
					Minimum (USD)	0.1000
Source	Source1	[Object]	Costs	TotalCost	Total (USD)	46.2000
		OutputBuffer	Costs	CostPerItem	Average (USD)	0.1000
					Maximum (USD)	0.1000
					Minimum (USD)	0.1000

Resource Costs

Resource costs are the costs associated with utilizing a resource. Resource costs in Simio are included in the Server / Combiner / Separator / Workstation (Deprecated) fixed objects, as well as the Resource / Worker / Vehicle objects.

These costs are broken down into three categories: Cost per Use, Idle Cost Rate, and Usage Cost Rate.

A resource *cost per use* is incurred with each entity that utilizes the resource. Reporting the resource usage cost could be confusing though. It is important to understand that a *resource is a cost driver* and its usage costs are charged, or accrued, to the *cost of the users/owners* of the resource (i.e., charged to the tasks using the resource and then rolled up at those locations where the task is performed).

So, for example, if an entity goes into a Server and utilizes a Worker as a secondary resource during processing, then the Server will incur costs of its internal resource (shown as Data Item 'UsageCostsCharged' under the DataSource 'Resource'). Additionally, a 'CostPerItem' for the DataSource 'Processing' includes the cost per use of the Server resource, as well as the Worker resource. These costs are then rolled up to the parent cost center's Total Cost as well. Costs are incurred to the entity that 'owned' the resources as well, as shown below.

Average						Drop Column Fields Here
Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	[Object]	Costs	TotalCost	Total (USD)	6.0000
ModelEntity	DefaultEntity	[Population]	Costs	CostPerItem	Average (USD)	6.0000
					Maximum (USD)	6.0000
					Minimum (USD)	6.0000
					Total (USD)	6.0000
Server	Server 1	[Object]	Costs	TotalCost	Total (USD)	6.0000
		[Resource]	Costs	UsageCostCharged	Total (USD)	5.0000
		Processing	Costs	CostPerItem	Average (USD)	6.0000
					Maximum (USD)	6.0000
					Minimum (USD)	6.0000
				TotalCost	Total (USD)	6.0000
Worker	Worker 1[1]	[Resource]	Costs	UsageCostCharged	Total (USD)	1.0000

Worker has a Cost Per Use of 1

Server has a Cost Per Use of 5

Both are included in CostPerItem

Idle cost rate for resources is cost per unit time that is charged to the cost of the parent object for each unutilized (idle) scheduled capacity unit. This cost is also allocated to the cost of the parent cost center. This cost is NOT allocated to any entities in the system.

Usage cost rate for resources is the cost per unit time to use the resource. This cost rate will apply only during the time that the resource is in a utilized state and will be charged to the cost of the entity using the resource. This cost will also be charged to the resource and included in the 'UsageCostCharged' (along with the CostPerUse value). As all other costs, this is also 'rolled up' to the parent cost center's Total Cost. It's important to note that changeover times using Sequence dependent setup (resource state of 'Setup' or 'OffShiftSetup') are not included in utilized time or cost calculations.

In the example below, there was only Idle Cost Rate and Usage Cost Rate, both of 1 USD per hour. The Server was used for 6 minutes of the hour (10%). Notice how the Server incurs both the IdleCost and UsageCostCharged under the 'Resource' Data Source. However, only the usage costs are incurred to the cost per item of both the entity and processing area of the Server. The Server's total cost (both idle and usage) are then rolled up to the parent cost center.

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	[Object]	Costs	TotalCost	Total (USD)	1.0000
ModelEntity	DefaultEntity	[Population]	Costs	CostPerItem	Average (USD)	0.1000
					Maximum (USD)	0.1000
					Minimum (USD)	0.1000
Server	Server 1	[Object]	Costs	TotalCost	Total (USD)	1.0000
		[Resource]	Costs	IdleCost	Total (USD)	0.9000
				UsageCostCharged	Total (USD)	0.1000
		Processing	Costs	CostPerItem	Average (USD)	0.1000
					Maximum (USD)	0.1000
					Minimum (USD)	0.1000
				TotalCost	Total (USD)	0.1000

In the above example, the UsageCostCharged for the resource is allocated to the Server because the resource is utilized within the Server object. The UsageCostCharged is the cost charged to the users of the resource, not directly to the resource. Only idle costs are charged directly to the resource. Usage costs are not charged in the resource's reported total cost unless the entity is actually located inside the resource (such as a Server) causing the costs to be rolled up into the Server's costs.

It is important to keep in mind that for Workers and Vehicles that are 'moving' resources, the usage costs (both cost per use and usage cost rate) are only incurred for 'non-transport' tasks. For these objects, an additional category of costs, Transport Costs, is used for transport tasks, as described below.

Workstation (Deprecated) Usage Cost Rate - For workstations (deprecated), the usage cost rate is broken down into three separate cost rates, including Setup Cost Rate, Processing Cost Rate and Teardown Cost Rate, based on which delay the entity is incurring.

Transport Costs

Transport costs are included with the Worker and Vehicle objects. Both a Cost Per Rider and Transport Cost Rate may be specified.

Cost per rider is the cost to load and transport an entity using a transporter of this type, irrespective of the transportation time. This cost is allocated to an entity when it is loaded onto the transporter.

Transport cost rate is the cost per unit time to transport an entity using a transporter of this type. This includes only the time when the entity is on the vehicle and does not include any time when the vehicle is moving to the entity's location for pickup.

Both cost per rider and transport cost rate are incurred to the worker / vehicle, as well as to the entity. Total cost of the

vehicle includes all of the 'CostPerItem' values and is then rolled up to the parent cost center.

In the below example, two entities have required transport from one object to another. The worker has both a cost per rider (1 USD) and a transport cost rate (10 USD per minute). Note that the 'CostPerItem' of the worker is shown under the 'RideStation' Data Source. Total cost is then calculated and 'rolled up' to the parent cost center (Model).

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	[Object]	Costs	TotalCost	Total (USD)	2.2523
ModelEntity	DefaultEntity	[Population]	Costs	CostPerItem	Average (USD)	1.1262
					Maximum (USD)	1.1262
					Minimum (USD)	1.1262
Worker	Worker1[1]	[Object]	Costs	TotalCost	Total (USD)	2.2523
		RideStation	Costs	CostPerItem	Average (USD)	1.1262
					Maximum (USD)	1.1262
					Minimum (USD)	1.1262
					Total (USD)	2.2523

Material Costs

Material costs are specified with the Material element and include the Cost Per Unit. This cost per unit of material is charged if a quantity of the material is consumed.

There is no material cost directly specified within the Financials section of any of the Standard Library objects. The Workstation (deprecated) object includes the option to specify material consumption. If materials are consumed at the workstation and the cost per unit property is specified for the material (via the Material element), then that cost will be associated with the workstation.

In the below example, the workstation includes a capital cost and consumes material that has an associated Cost Per Unit. The entity using the material incurs a 'CostPerItem' charge. The 'CostPerItem' shown under the Processing Costs for the Workstation also include those material costs. Capital cost is then allocated to the workstation, along with the material cost for a TotalCost of the object. This cost is then 'rolled up' to the parent cost center. Note that within the parent cost center, the TotalCost for the object (Model) is displayed, as well as the MaterialCostCharged. Material costs are included in the TotalCost.

Average						Drop Column Fields H
Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	[Object]	Costs	TotalCost	Total (USD)	217.0000
		Material1	Costs	MaterialCostCharged	Total (USD)	217.0000
ModelEntity	DefaultEntity	[Population]	Costs	CostPerItem	Average (USD)	1.0000
					Maximum (USD)	1.0000
					Minimum (USD)	1.0000
Workstation	Workstation1	[Object]	Costs	TotalCost	Total (USD)	217.0000
		Processing	Costs	CostPerItem	Average (USD)	1.0000
					Maximum (USD)	1.0000
					Minimum (USD)	1.0000
					Total (USD)	217.0000

Currency Support

Simio's costing features includes a default currency which can be specified in any number of country's currency units. The default is USD (United States dollars). Exchange rates can also be specified between two countries' currency units. More information about default currency and exchange rates can be found on the Financials page of the help.

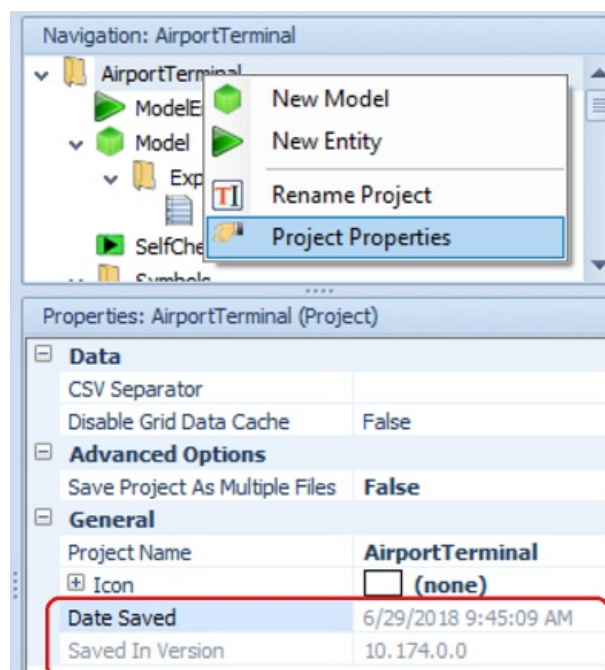
Projects And Models

Projects

A **Project** can contain just one **Model** or multiple **Models**. When a new project is created, a default model (of object class type Fixed) will be automatically added to the project. Only one single project may be open in Simio at a time. A project may be saved to a project file, and that file will contain all of the elements in the project. Thus, a user will be able to distribute an entire project by distributing a single project file.

A project containing models can be opened in two ways. First, a project may be opened as a project file where all models can be edited. Second, a project can be opened a read-only library that provides definitions used in placing objects into the model. The Load Library button on the Project Home tab will open a given project *.spfx file as a library. The project and all models within the project will then appear in a library on the left side of the Simio window with the Standard and Flow Libraries. Right clicking in the Library area will also allow users to load and unload libraries.

By right-clicking on the project name, the project can be renamed and the properties of the project can be viewed/modified.



Listed below are the properties of a **Project**:

Property	Description
CSV Separator	The separator character used when exporting data to a CSV file. If blank, Simio will use the "List Separator" character specified by the computer's current regional settings. To set this character in Windows, open Control Panel, and then depending on your operating system, look for something similar to "Date, Time, Language, and Regional Options", "Change the format of numbers, dates, and times", "Regional and Language Options", "Clock, Language, and Region", "Region and Language", "Change the date, time, or number format", and change the "List Separator" value.
Disable Grid Data Cache	Grid Data Providers may choose to persist information from one invocation of the provider to the next. In most cases, this is the desired behavior, and this value should be set to 'False'. In severely memory-constrained situations, it may be useful to prevent this caching by setting this value to 'True'.
Save Project As	Indicates if upon saving to a multi-file storage (like the simproj format) the project logic should

Multiple Files	all be contained in a single file, or be broken apart into individual files (such as a file per object definition and a file per table).
Support Source Control	This setting affects how Simio writes its project files. The default value is False and is appropriate for most cases where a single user edits the project file. Simio will digitally sign the file, which is necessary to allow certain lower editions of Simio to open files created by other editions. Simio will also include other optional information in the file, such as window layouts, view settings, and the project's save history. It will also create backup files when saving. In some cases, however multiple users can work on the same Simio project, typically by saving the project file using Simio's .simproj format and storing it in a version control system. Setting Support Source Control to True can help avoid conflicts when users merge their changes. In this case, Simio will not sign files and will exclude certain other information from the project file. When the project is ready to be distributed, Support Source Control can be set to False, and the project saved again (usually to a new name or location, using Simio's .spfx file format), thereby enabling normal project behavior.
Suppress Warning About Opening Newer File	Indicates if the "Opening Newer File" warning will be displayed on project load. Warning: File version discrepancies could have serious repercussions. Note: With <i>Suppress Warning About Opening Newer File</i> set to 'True', the ability to Save this project is still disabled. If saving is desired, use Save As and provide a different file name.
Show Documentation on Load	When this project loads, associated documentation for it (a pdf file of the same name in the same directory) should be opened as well.

Models

A model describes the logic, external interfaces, data structure, and presentation/animation for a class of objects. A model may be added to a library and then instances of that model embedded in another model. Thus, a user will be able to easily create a library that is a collection of models developed for some particular application domain or modeling effort.

The first model in your project is typically your main model. When you create your main model Simio also creates a default model entity with no logic, and a corresponding default entity object. This default entity object is created but not yet placed. The first entity object that you place in your model is this default entity object. You can use this entity model across multiple models, or add additional entity models as needed. You can also add custom logic to the entity model.

Model properties can be found within [The Facility window](#) page under the Model Level Properties section.

You can create sub-models by adding additional models to your project. You can then add logic to your sub-model, define the user view, properties, processes, and then drag objects for this model into your main model.

You can also subclass versions of the Standard Library or other Library objects. This allows you to start with an object and then make changes to customize the logic for your own needs. To add additional models to your project, select the Project in the Navigation window to open the Project window. You can then add a new models to your project either from scratch, by subclassing another model, or by copying a model from a library. See the [Project Window](#) and [Creating New Objects](#) help pages for additional information on how to add models. You can then add to or modify the logic of your subclassed model to create your own custom version of the model. When you select your main model back in the Navigation window, you'll see the new models in your Project Library on the bottom left side of the UI in the Libraries panels, and you can then drag these objects into your main model.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Queues

Queues in Simio

Queues, or lines of entities waiting to be processed, can be found in many different places around Simio. This provides tremendous flexibility in allowing the user to queue entities up at different points throughout the model. It also creates ease in modeling blocking. When placed in the Facility Window, the standard library objects all have their Input Buffer queue, their Processing queue and their Output Buffer queue automatically animated.

Entities can be removed from certain queues with the use of the [Remove Step](#).

Animating Queues in Simio

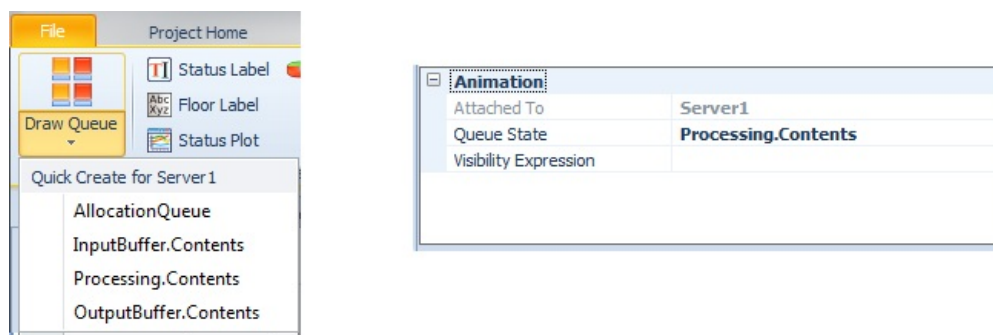
You can animate a queue in Simio with either a Detached Queue that is not tied to any particular object, or animate with a Queue that is attached an object. An Attached Queue will move around the window with the object. The icon used to animate a detached queue is found on the Animation Tab of the ribbon.

To draw a detached queue, do not have any object selected and select the Detached Queue icon in the Animation Tab of the ribbon. Left click in the Facility Window to add the vertices of the queue and right click to add the final vertex.

To draw an attached queue, first select the object that you wish to attach the queue with and then select the Queue icon from the ribbon of the Symbols Tab. Left click in the Facility Window to add the vertices of the queue and right click to add the final vertex. Once the Queue is drawn in the Facility Window, when the Queue is still selected, choose the appropriate Queue State in the Properties Window, found in the Definitions Tab. The Queue States are listed alphabetically by the Object they are associated with.

When animating an attached queue, Simio provides a short cut. When the object is selected, instead of clicking on the center of the Queue icon in the ribbon, select the down arrow to display the Quick Create drop down. This menu provides some appropriate queue choices for the user to select for animating this queue. The choices are the queues that are available for the selected object. If the user selects something from this Quick Create menu, and then draws the queue in the Window, the properties of the queue are automatically populated. For example, if the user selects InputBuffer.Contents from the drop down shown below and then draws a queue in the Window, the *Attached To* property will be the name of the object that was selected and the *Queue State* property will be InputBuffer.Contents and this will animate the entities that are currently in the InputBuffer of the selected object.

Queue Quick Create Menu



Note that the orientation properties for a queue, if specified, take precedence over any other given orientation for the queue (for example, the direction specified by an "Oriented Point" queue).

Listed below are the properties of **Queue (Animated)**:

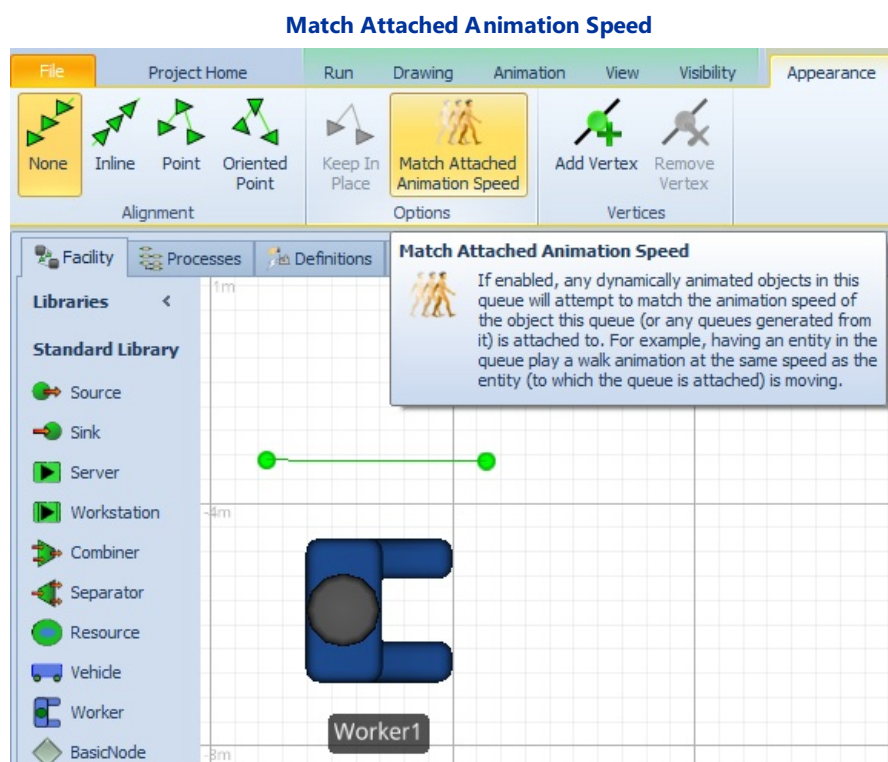
Property	Description
Attached To	Only visible for queues that are attached to an object, such as a Server, Worker, Vehicle or Node. The object to which the queue is attached.
Queue State	Name of the queue state holding entities that are displayed by this object.

Visibility Expression	<p>An expression that is evaluated per object in the queue to determine if that object is visible or not. To reference an object in the queue for which the expression is being evaluated, use the "Candidate" keyword.</p> <p>For example, the expression "Candidate.MyEntity.MyState > 1" assumes all objects in the queue will always be of type 'MyEntity' and will only show those that have a 'MyState' value greater than 1.</p>
X Direction	An expression indicating the direction an object should be oriented along the X axis.
Y Direction	An expression indicating the direction an object should be oriented along the Y axis.
Z Direction	An expression indicating the direction an object should be oriented along the Z axis.
Roll	An expression indicating the degrees of clockwise roll from upright an object should be oriented.

There is a property found by right clicking on each individual Queue called *Queue Line Externally Visible* which is intended give the behavior of the Visibility Ribbon at the object level. Turning this property off will "hide" the queue line but will still show the instances in the animation.

Animating Queues on Transporter-type objects, such as Workers and Vehicles

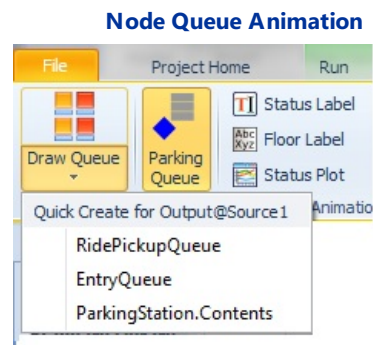
For queues attached to Workers and Vehicles (or queues in an external view of a Worker or Vehicle), a button will be enabled on the Appearance ribbon to indicate if objects in the queue should match their movement animation speed with the attached object. For example, if a Worker is escorting a Patient, the patient walks if the Worker walks, and runs if the Worker runs. This feature is applicable for 'animated' objects, otherwise the entity in the queue will appear to glide instead of walk.



Animating Queues on Nodes, such as Parking Queues

When a node is placed in Simio, its Parking Queue is automatically animated, even though nothing is visible within the Facility window. This is helpful when using Transporters and eliminates the need for the user to draw a Parking Queue in order to see the Vehicle when parked. If you select a node (even Input and Output nodes attached to standard objects), you will see the Parking Queue icon highlighted in the ribbon. If you click on this icon, it will turn off the automatic animation of the Parking queue for this node. Simio still provides the Draw Queue option, which allows the user to draw an attached

queue for this selected node, including a Parking Queue, if the user does not like the default location of the automatic Parking Queue.



**Queue Animation Options
available when a Node is
selected in the Facility window**

Changing the Appearance of Queues

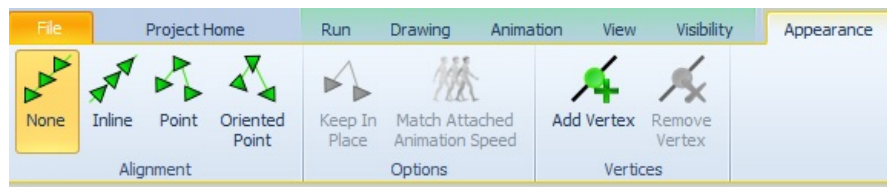
The appearance of a queue, by default, is a line queue. This means that the objects will appear to line up within the queue along the line that is drawn. The **None** and **Inline** options on the Appearance ribbon show two variations of the Line queue in terms of the object orientation within the queue. With the None option, objects align to always point to the left, no matter what the queue orientation. With the Inline option, the objects align to always point forward along the line of the queue. With both of these line queues, when an object leaves the queue, all other objects move up within the queue. The number of objects graphically displayed within a line queue is based on the size of the line and the size of the objects.

A line queue may be changed to a point queue by highlighting the queue and selecting the **Point** or **Oriented Point** buttons from the Appearance ribbon. With both these type of point queues, objects are graphically shown in the queue on the vertices of the queue line. When you change a queue to a point queue, you will notice that small orientation arrows will be associated with each of the vertices of the queue. With a Point queue, when you select the queue, one of the end orientation arrows becomes a vertex and, when selected, may be moved around to change the orientation of all the vertices on the line. With an Oriented Point queue, when you select the queue, each of the vertices along the line has an associated orientation vertex that can be aligned to different directions per vertex.

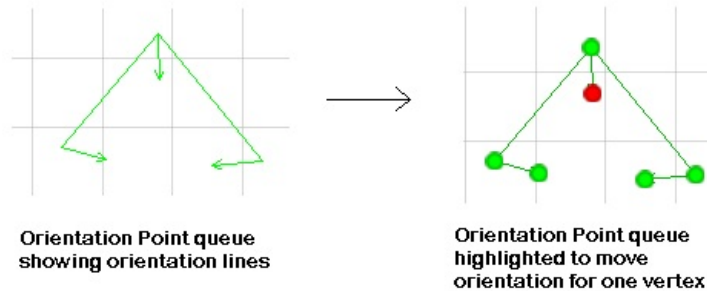
For both Point and Oriented Point queues, the user has the option to select the Keep in Place button. This option will allow the objects in the queue to remain at the point they were inserted while in the queue, instead of sliding forward to the next available position when another object is removed from the queue.

With all types of queues, additional vertices may be added or removed by using the **Add Vertex** and **Remove Vertex** buttons on the Appearance ribbon. When a queue is selected and the Add Vertex button is pressed, the queue will get an extra vertex added to the end of the queue. The user may click to place the vertex anywhere in the Facility window. If the new vertex is placed on the line between two other vertices, it will be added to the middle section of that line. If the new vertex is placed outside of the line, it will be added as a vertex at the end of the queue. To remove a vertex, simply click on the vertex to remove and select the Remove Vertex button and the queue line will be redrawn.

Queue Appearance



Appearance ribbon for Changing the Alignment of Queues



Queues that can be found in Simio:

Network Element

VisitRequestQueue – An entity is inserted into this queue of the model's 'Global' network (by the Ride Step) when it is waiting to reserve a ride on a Transporter in the system. When a Transporter (i.e., the standard library Vehicle object) becomes idle, it searches the 'Global' network's *VisitRequestQueue* (using the PlanVisit Step) to possibly select a reservation request from that queue. This queue on any other network element is not currently used.

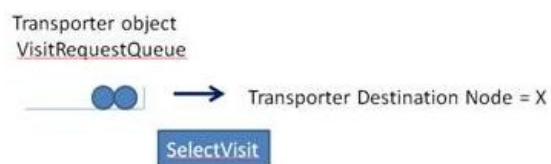
Entity Object

VisitRequestQueue - This queue is on the Entity object but is currently only used by a Transporter object (which is a subclass of entity). This queue is holding all the entity objects whose pickup reservations the Transporter has selected and accepted to perform. Visit requests are initiated from either a Ride or Move step. Thus, when a Transporter object selects a pickup reservation request from the 'Global' network's *VisitRequestQueue* using the PlanVisit Step, that request is moved from the network *VisitRequestQueue* to the individual Transporter object's *VisitRequestQueue*. The Transporter then can actually set its destination to one of the pickup reservations in its *VisitRequestQueue* using the SelectVisit Step. Examples include: Vehicle1.VisitRequestQueue

Details on the Network's VisitRequestQueue and a Transporter's VisitRequestQueue



A transporter uses the PlanVisit step to search the 'Global' network's *VisitRequestQueue* to select a pickup reservation and move that pickup reservation into its own *VisitRequestQueue*.



When a transporter is ready to actually do a pickup somewhere, it uses the SelectVisit step to search its own *VisitRequestQueue* and set its destination to one of its pickup reservation destinations.

BatchQueue - If an Entity is the parent of a batch, then all of its batch members are held in this queue owned by the Entity.

Examples include: ModelEntity.BatchMembers

Any Object That Is Declared As A Resource Object

AllocationQueue - Objects that are waiting to seize the Object wait in this queue. This queue's static ranking rule as well as the resource object's dynamic selection rule determine the next object to seize the object. Examples include: Server1.AllocationQueue, Resource1.AllocationQueue

Station Element

EntryQueue - Entities waiting to enter the Station, if capacity is not available, wait in this queue owned by the Station element. Examples include: Server1.Processing.EntryQueue

Note: Entities will never go into the EntryQueue of the ParkingStation of a node, because a node's ParkingStation always has a capacity of 'Infinity'. **Note:** Entities will never go into the EntryQueue of the RideStation of a Transporter, because the Pickup step used by the Transporter doesn't initiate a transfer into the Ride station unless ride capacity is available.

Contents - Where entities exist while they are in the Station. This queue is owned by the Station element. Examples include: Server1.Processing.Contents, Vehicle1.RideStation.Contents

Material Element

AllocationQueue - Objects waiting to consume a quantity of the material wait in this queue owned by the Material Element.

Node Object

RidePickupQueue - Entities waiting to get picked up by a Transporter at this location wait in this queue. Thus, if the ReservationMethod on the Ride Step was not 'FirstAvailableAtLocation', then an entity is going to be in two queues at once while waiting for a ride pickup, as a reservation request in a network/transporter VisitRequestQueue and as a location-specific pickup request in a node RidePickupQueue. Examples include: MemberOutput@Separator1.RidePickupQueue, TransferNode1.RidePickupQueue

BatchLogic Element

ParentQueue - Entities wait here to collect a batch of members. This queue is owned by the BatchLogic element.

MemberQueue - The queue of entities waiting to be added as a member of a batch. This queue is owned by the BatchLogic element.

RoutingGroup Element

RouteRequestQueue - Entities waiting to be routed to a destination in the routing group wait in this queue. It is owned by the routing group element.

Storage Element

Queue - Objects in the Storage are held and ranked in this queue owned by the Storage Element.

Note: Currently, the Insert and Remove steps can only be used to insert/remove objects from a storage element queue. Examples include: Storage1.Queue

Link Object

EntryQueue - Entities waiting to enter the Link wait in this queue, which is owned by the Link object. Examples include: Conveyor1.EntryQueue

Contents - Entities that are physically located on the link. Entities are ranked in the queue based on the order of entry onto the link (* It is important to note that on paths that allow passing and entities with varying speeds, this may not indicate the order of those entities currently on the link, but only how they entered the link). Examples include: Conveyor1.Contents

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Tokens and Entities

Tokens

A Token executes the Steps in a process flow. A Token may have one or more user-defined States that carry information from Step to Step. A Token may also reference an associated object such as an Entity that is visiting the parent object. A Token lives inside of a process - it is created at the beginning of a process and it is destroyed at the end of that same process. Therefore, a Token is created whenever a process is first executed. A new Token is also created out of the Create segment of a Create Step and out of the Found segment of a Search Step. The type of Token that is created inside of that process is determined by the *Token Class Name* property of the Process. The function *ProcessName.TokensInProcess.NumberItems* returns the number of tokens currently executing a given process.

Simio provides you with a default Token named 'Token' that you can use for most of your processes. You only need to add a new Token if you require a Token with one or more custom states. See the [Tokens](#) page for more information.

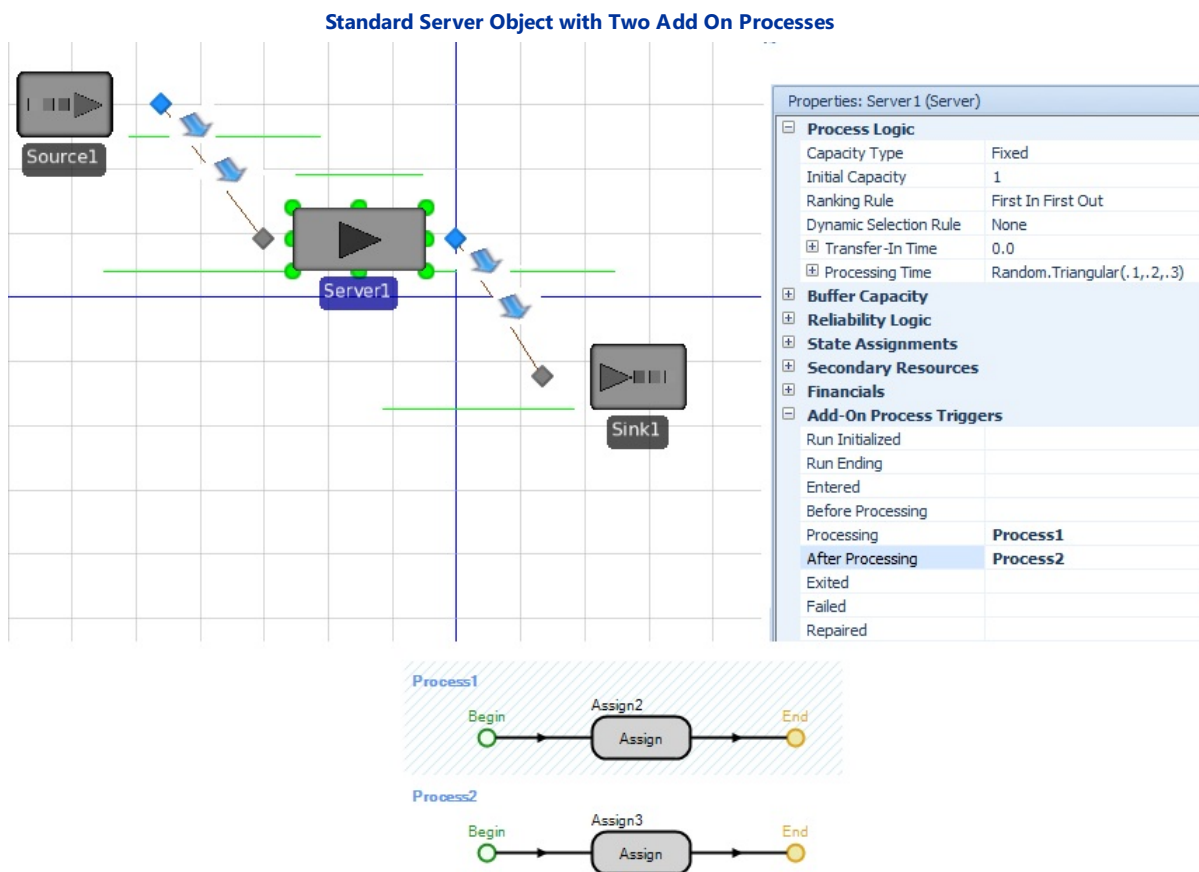
Entities

In Simio, Entities are part of an object model and can have their own intelligent behavior. They can make decisions, reject request, decide to take a rest, etc. Entities have object definitions just like the other objects in the model. Entity objects can be dynamically created and destroyed, move across a network of Links and Nodes, move through 3D space, and move into and out of Fixed objects. Examples of Entity objects include customers, parts or work pieces. Entities do not flow through processes, as Tokens do. The movement of Entities into and out of objects may trigger an event, which might execute a process. When the process is executed, a Token is created that flows through the steps of a process.

Entities have a physical location within the Facility Window and they can reside in either Free Space, in a Station, on a Link or at a Node. When talking about the location type of an Entity, we are referring to the location of the Entity's leading edge. Examples of [Stations](#) within Simio are: a Parking Station of a Node (TransferNode1.ParkingStation), the Inputbuffer of a Separator object (Separator1.InputBuffer), Processing Station of a Server object (Server1.Processing).

Examples of the Different Roles of an Entity and a Token

Example 1 - Processes via Server with Add On Processes

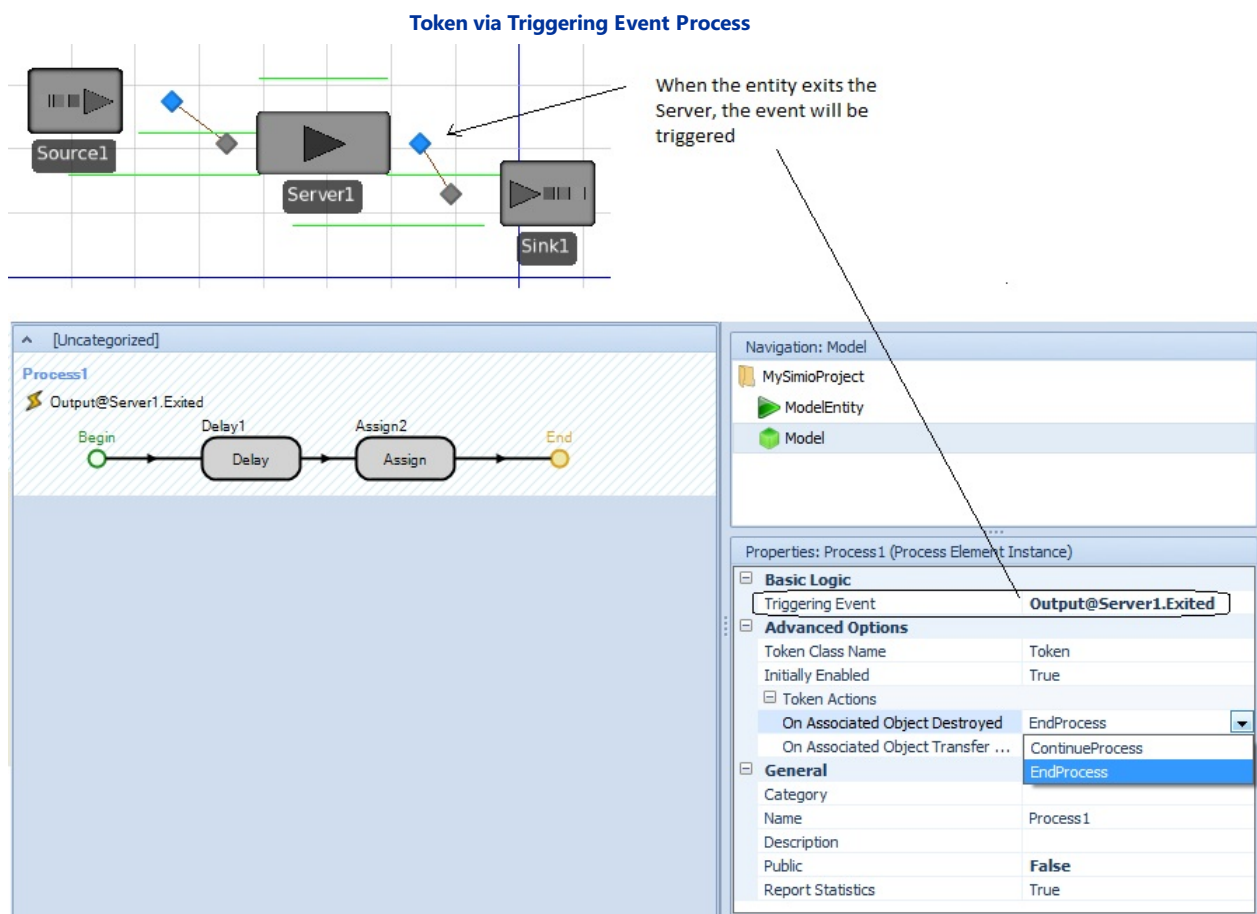


If we walk through the simple example of an entity that enters a standard Server object, we can demonstrate the role of an Entity and a Token within Simio. In this first example, there are processes that are executed by both the Processing and

After Processing Add On Process triggers of the Server object. The Entity instance that exists within the Facility Window arrives to the Input Node of a Server object. It exits the link, crosses the Input Node and then immediately is transferred into the InputBuffer station of the Server, assuming there is capacity available in the Input Buffer. If there is available capacity in the Server, the capacity is allocated and then the Processing trigger executes the Process1 process. A Token that is associated with the entity is created in Process1, which flows through the Assign step. After the Process1 is complete, the Entity object moves to the Processing station of the Server. When the Server finishes its processing time delay, but before the capacity of the Server is released, the Process2 process is executed. The Entity object remains in the Processing station of the Server object and a new Token is created within the Process2 process. This Token flows through the Assign Step that exists within this process. The Token finishes its job when it reaches the End of the Process2 process. At this point, the Entity then moves from the Processing station of the Server to the OutputBuffer station of the Server, assuming there is capacity available in the Output Buffer. And finally, the Entity is transferred to the Output Node of the Server object where it is routed out with the routing out logic of the Output node. As you can see, the Entity travels physically in the Facility Window between stations and nodes and the Tokens that are associated with the entity are created to flow through the Processes and perform the logic of the Steps.

It is important to note that within the Server logic, an Execute step is used for each of the Add On Process Trigger processes. Within each Execute step, the *Action* is 'WaitUntilCompleted'. Thus, when the token associated with the entity is created within a process, the entity will remain physically where it is for the entire process, even when delay steps are involved.

Example 2 - Processes via Triggering Event



In this second example, the process will be triggered using a Triggering Event. Triggering events may be a user defined Event or Timer.Event, or may be related to an action within an object, such as Output@Source1.Exited or Input@Server1.Entered. If the process is triggered by a user defined event (i.e., Event1) or monitor event (i.e., Monitor1.Event), the token going through the process is associated with the 'model' and not a specific entity. Therefore, as long as the model is running, the token will continue through all the steps in the process. If, however, the triggering event is based on an entity entering or exiting another object (i.e., Output@Server1.Exited), the token going through the process is associated with a particular entity. If that associated entity is destroyed for whatever reason while the token is still in the process (i.e., in a Delay step), the token will also be destroyed and will not continue through the remaining process steps.

This example shows the happenings of the entity and associated token with a small example of Source, Server, Sink. Entities are created at Source1, move to Server1 and are processed, then move to Sink1 and are destroyed. Suppose when an entity exits Server1, you would like to have a token delay for a period of time and then increase a state variable. As shown above, Process1 is triggered by an entity leaving the Server. The *Triggering Event* is 'Output@Server1.Exited'. Therefore, the token that goes through Process1 is associated with the entity that exited and not the model. When the entity leaves Server1 and moves to Sink1, it is then destroyed. If the token is still in the process's Delay step when its associated entity is destroyed, it will also be destroyed by default. If you would like to have the token continue processing, enter the Advanced Options section and click on the '+' sign of Token Actions. The *On Associated Object Destroyed*

property allows users to specify what happens to the token when an associated entity is destroyed. By default, the value for this property is 'EndProcess'. However, by changing this value to 'ContinueProcess', the token will not be destroyed when its associated entity is and will instead continue its process until the end.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

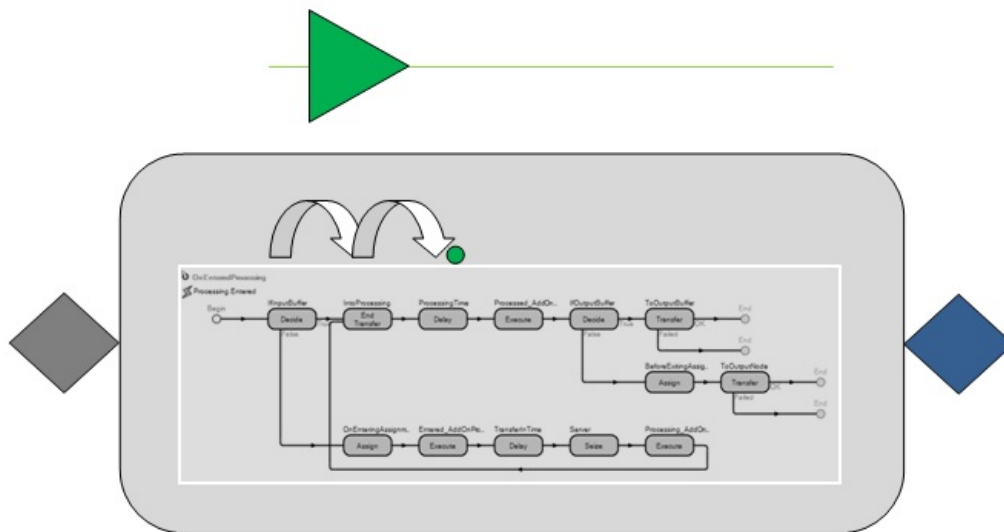
Send comments on this topic to [Support](#)

Associated Vs Parent Object

Example 1

This example shown below details the process named 'OnEnteredProcessing'. The process is triggered when an Entity is transferred into the Processing Station of a Server. At the same time as the Entity enters the Station, a Token is created at the Begin endpoint shown below in the process. It acts on behalf of the Entity. Next, the Token moves to the first process Step and performs some action or logic. In this example, the first Step is a Decide Step. Tokens travel from Step to Step in zero time.

Token flows through 'OnEnteredProcessing' Process of Standard Server



What is an Associated Object?

The Associated Object is the object that triggered a specific process to be executed. In this example, the Associated Object is the Entity because it trigger this process when it transferred into the Processing Station of the Server. The Entity object will remain in the Processing Station until its Token has completed the all of the Steps in the process. In other words, a Delay step is actually delaying the Token from being passed on to the next Step, it is not delaying the Entity from leaving the Station. The Entity must wait for the delayed Token.

What is a Parent Object?

The Parent Object is the object where the process was created and resides. Because the OnEnteredProcessing Process is part of the Server object, the Server is OnEnteredProcessing's Parent Object.

An easy way to figure out what object is the Parent Object is to think about what Object would need to be selected in order for you to see the process. To see the details or edit the OnEnteredProcessing process, you would have to subclass a standard Server and go into the Processes window of that Object.

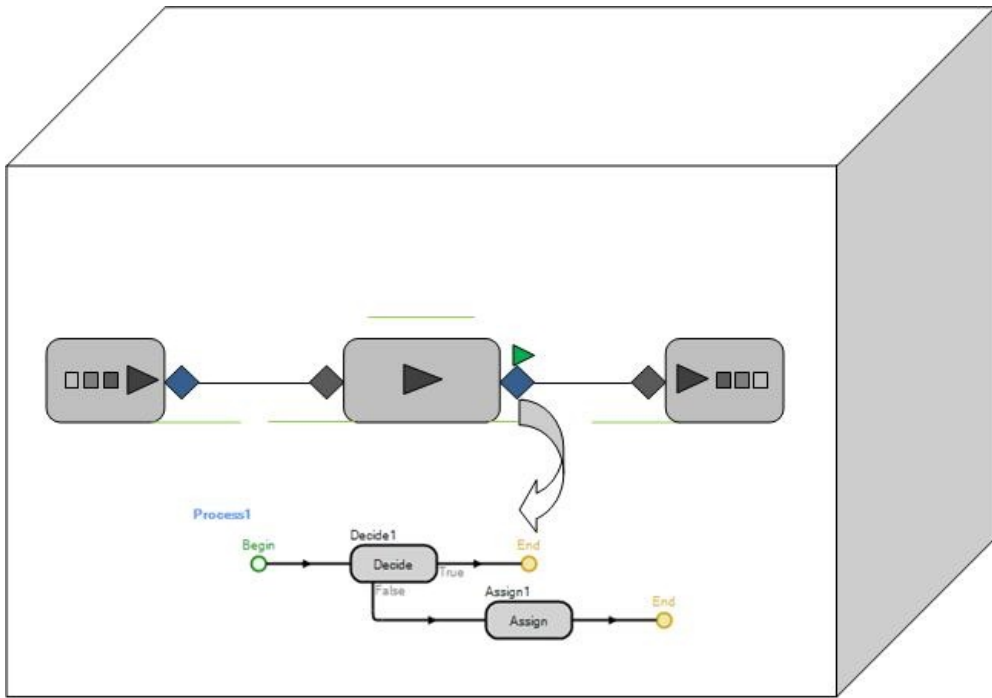
Therefore, for all Add-On Processes, the Parent Object is most commonly the main Model, since the Add On Processes are created within the main Model and they can be viewed and edited in the main Model's Processes window.

Example 2

This second example shows an Add-On Process that was triggered by an Entity entering the Output Node at a Server. For this process below named Process1:

- The Parent Object is the main Model
- The Associated Object is the Entity

Add On Process in OutputNode Spawns a Token to Execute 'Process1'



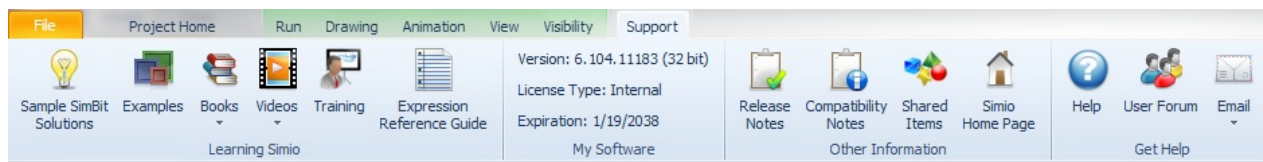
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Learning Aids and Support

The Learning Aids and Support Available for Simio

There are a number of different places where a user can go to learn more about modeling in Simio. The Support ribbon contains links to numerous small models and examples, as well as on-line resources available for helping you to learn Simio.



A **Sample SimBit Solution** is a small Simio model which provides an example of how to use certain features of the software. Each SimBit .spfx file has a matching .pdf file that provides a detailed description of the model. The files can be found in C:\Program Files\Simio LLC\Simio\SimBits. The **SimBits** help page provides a list of all the available SimBits in various categories. SimBits can be searched by clicking on the **Sample SimBit Solutions** button in the Support ribbon of the Simio software, which will take you to a page where Basic and Advanced searches may be performed.

There are a number of **Example** models that are available with the installation of the software. Example models can be found in C:\Program Files\Simio LLC\Simio\Examples. See the **Examples** page for a listing of each and accompanying documentation. The Examples button in the Support ribbon opens a dialog for selecting one of our example models.

The **Books** button will take you to <https://textbook.simio.com/> where you can learn about a variety of Simio publications as well as view them.

Simio features and concepts are demonstrated in a number of **Videos** found on the Simio website. The Introductory Topics include a 20-minute video about Simio and the 11 module Introduction to Simulation and Scheduling videos. The Intermediate Topics link includes the 8-hour Introduction to Simio video set that introduces the basics of Simio. A matching e-book is also available under the Books button. Both of these video sets include certification opportunities. Special topics, such as WonderWare MES Integration can also be found in this section. The Simio YouTube Channel option provides links to a number of videos available for a range of topics, from Simio and Feature Overviews to Student Projects and Example Models.

The **Training** button provides links to our training course schedule and various learning resources, as well as teaching resources for academics and an interactive map of universities with Simio grants. Simio offers public training classes through our worldwide network of training partners, as well as customized and on-site training. Our teaching resources include information about products available, grant application link, teaching materials and our semi-annual student competition.

The **Expression Reference Guide** button opens an html document which will be opened in the default browser. This document lists the Simio built-in functions and keywords, states and elements. It also lists the functions, states, events and elements for those objects that have been placed in your model.

The **My Software** section of the Support ribbon contains information specifically related to your Simio license. It includes the **Version** of Simio which you are currently running, in the format Simio 5.80.***. The **License Type** displays the type of license within our **Simio Product Family** you are currently running. Finally, the **Expiration** date displays the date of maintenance agreement expiration.

The **Release Notes** icon opens the Simio Release Notes.pdf file (found in in the install folder (by default \Program Files\Simio LLC\Simio)). This file provides information regarding new features in each sprint release.

The **Compatibility Notes** icon opens the Simio Compatibility Notes.pdf file (also found in in the install folder (by default \Program Files\Simio LLC\Simio)). This file lists any known problems and work-arounds as well as any compatibility issues converting between releases.

The **Shared Items** link opens the SI Shared Items area of the Simio User Forum at <https://www.simio.com/shared-items/>. This area of the forum provides access to objects, libraries and other items that users have shared.

The **Simio Home Page** link opens the <http://www.simio.com/index.php>. Start here to find out anything about Simio.

The **Help** icon provides a link to the searchable help on all aspects of Simio.

The **User Forum** icon takes the user to the Simio User Forum where you may search the various forums for answers, post problems and suggestions and interact with Simio employees and the world-wide Simio user base. The direct link to the Simio User forum can be found at <http://www.simio.com/forums/>. Users can become a **Simio Insider** to have full access to the User Forums where other users share information about how they use the software and the models they are creating. This is an opportunity for the global Simio community to communicate and collaborate.

The **Email** button allows you to **Email a Question** to Simio Support, **Email This Project** to Simio Support or **Email Sales** for additional product information.

SimBits

SimBits - General Information

A SimBit is a small Simio model which provides an example of how to use certain features of the software. Each SimBit .spfx file has description that explains the details of the model. The model *.spfx files can be found in C:\Program Files\Simio LLC\Simio\SimBits.

SimBits can be searched by clicking on the [Sample SimBit Solutions](#) icon in the Support ribbon. By default, the model will open in a new Simio window. Alternatively, if you'd like to open the SimBit in the currently active Simio window, hold down the Shift key when you click onto the name of the SimBit that you'd like to open.

Below is a categorization of the SimBits based on their functionality. Some models can be found under multiple categories.

Add-On Process Logic

[AnimatePathDensity](#)

[ChangingQueuesWhenServerFails](#)

[CheckingNextEntityAheadOnLink](#)

[ChooseAlternateSequence](#)

[CreateDiscreteEntitiesBasedOnFlow\(from project Flow Concepts\)](#)

[CustomUnbatching](#)

[DashboardsWithinExperiments](#)

[DisableFailureTimers](#)

[DiscreteLookupTable](#)

[ElectricVehicle](#)

[FillingEmptyingTank\(from Project Flow Concepts\)](#)

[Financials](#)

[FindAMinimumStateValue](#)

[HourlyStatistic](#)

[InventoryReplenish](#)

[KeepQueueTimeForLast10Entities](#)

[MergingConveyorsControlledByGate](#)

[MultipleServerFailures](#)

[NotifyStep](#)

[PickupDropOffFlow](#)

[ResourceSelectionConcepts](#)

[ResourceStatesWhenOffShift](#)

[RotatingVehicleExample](#)

[SequentialProcessingByBatchSpecifiedInTable](#)

[ServerBlockingApproaches](#)

[ServerUsingTaskSequenceWithWorkers](#)

[SimpleTank](#)
[SingleWorkerCompletesProcessAndMovesToNode](#)
[TankCleanInPlaceTrigger](#)
[TrackingTankContentsByProductTypeUsingTableStates](#)
[TransferLine](#)
[TravelWithSteeringBehavior](#)
[UsingaMonitor](#)
[UsingAStorageQueue](#)
[VehicleDoingSimultaneousLoading](#)
[WorkerUsesWorkSchedule_InterruptWorkingOffShift](#)

Animation

[AnimatePathDensity](#)
[AnimatedPeople](#)
[SimpleElevatorObject](#)

Arrival Logic

[AppointmentArrivals](#)
[DisableFailureTimers](#)
[LeveledArrivals](#)
[RandomValueFromTable](#)
[SearchTables](#)
[ScheduledMaterialArrivalsWithDeviation](#)
[SourceWithRateTable](#)
[SourceWithCustomEntityTypeProperty](#)
[UsingButtonsToAffectSystem](#)
[UsingTheStipulateStepForNodeLists](#)
[UsingTheStipulateStepForResourceAllocation](#)

Buffering

[AddAndRemoveServerCapacity](#)
[ConstraintLogic](#)
[CONWIP](#)
[HourlyStatistic](#)
[KeepQueueTimeForLast10Entities](#)
[MultiServerSystemWithJockeying](#)
[NotifyStep](#)
[OneQueueForMultipleServers](#)
[OverflowWIP](#)
[ServerBlockingApproaches](#)
[ServerQueueWithBalkingAndReneging](#)

[SimpleLeastSlackSelectionRule](#)

[SourceWithBalkingIfBlocked](#)

[UserDefinedListState](#)

Building New Objects / Hierarchy

[CombinerNode](#)

[CommunicationBetweenObjects](#)

[ElectricVehicle](#)

[FacilityModelWithinModel](#)

[HierarchyWithTables](#)

[HourlyStatistic](#)

[MassFlowProcessing](#)

[MoveableOperator](#)

[ProcessModelWithinModel](#)

[ResourceSelectionConcepts](#)

[SourceWithCustomEntityTypeProperty](#)

[UpdateStateInModelFromObject](#)

[VisitAllServersInAnyOrder](#)

[WorkersArriveLateToShift](#)

Changeover Logic

[ServerWithSequenceDependentSetup](#)

Combining and Separating Entities

[BatchingProcessUsingScanOrWaitStepToControlBatchSize](#)

[CombineMatchingMembers](#)

[CombineMultipleEntityTypesOntoPallets](#)

[CombinerNode](#)

[CombinerReleasingBatchEarly](#)

[CombineThenSeparate](#)

[CustomUnbatching](#)

[ReferenceBatchedEntity](#)

[RegeneratingCombiner](#)

[SeparatorMakesCopies](#)

[UsingAStorageQueue](#)

Constraint Logic Element

[ConstraintLogic](#)

Continuous Systems

[InfectionPropagationUsingContinuousAndFlow](#)

Conveyor Systems

[EntitiesEnteringAlongConveyor](#)

[ExamplesOfConveyors](#)

[MergingConveyorsControlledByGate](#)

[TransferLine](#)

[SortingConveyorSystem](#)

Custom Reports

[DashboardReportInteractiveLogs](#)

Custom Statistics

[DashboardReportInteractiveLogs](#)

[DashboardReportTallies](#)

[DashboardsForSchedulingExamples](#)

[HourlyStatistic](#)

[KeepQueueTimeForLast10Entities](#)

[RecordDistanceTraveled](#)

[TallyStatisticsInTables](#)

[UsingaMonitor](#)

[UserDefinedListState](#)

[WorkersArriveLateToShift](#)

Dashboard Reports

[DashboardReportInteractiveLogs](#)

[DashboardReportTallies](#)

[DashboardsForSchedulingExamples](#)

[DashboardsWithinExperiments](#)

Data Tables

[CONWIP](#)

[EntityFollowsSequenceWithTable](#)

[EntityFollowsSequenceWithRelationalTables](#)

[HierarchyWithTables](#)

[ImportExportTables](#)

[InitializeObjectPropertiesFromATable](#)

[LeveledArrivals](#)

[RandomValueFromTable](#)

[ResourceSelectionConcepts](#)

[RoutingWithoutPaths](#)

[RelationalTablesInRepeatingProperty](#)

[ScheduledMaterialArrivals](#)

[ScheduledMaterialArrivalsWithDeviation](#)

[SearchTables](#)

[SearchTableUponEnteringObject](#)

SequentialProcessingByBatchSpecifiedInTable
ServersUsingTaskSequenceWithDataTables_FlowLine
ServersUsingTaskSequenceWithDataTables_JobShop
ServersUsingTaskSequenceWithDataTables_LoopbackBranches
TableReferenceInRepeatingProperty
UsingAddRowAndOutputTable
UsingRelationalTables
UsingRelationalTablesToDefineNodeLists
UsingRelationalTablesToDefineTaskResourceLists
UsingTheStipulateStepForNodeLists
UsingTheStipulateStepForResourceAllocation

Decision Logic – Paths

BidirectionalPaths
ChangingQueuesWhenServerFails
CustomRoutingGroup
EntityFollowsSequence
EntityFollowsSequenceMultiple
EntityFollowsSequenceWithTable
EntityStopsOnLink
FindAMinimumStateValue
LogicBasedOnEntityProperty
LogicBasedOnEntityState
ObjectReferenceOnEntity
PathSelectionRule
SelectServerWithShortestLine
ServerBlockingApproaches
ServerWithTransferInConstraints
SimpleElevatorObject
TravelWithSteeringBehavior
VisitAllServersInAnyOrder
VehicleFleetManagement

Decision Logic – Processing

AddAndRemoveServerCapacity
DiscreteLookupTable
CustomRoutingGroup
EntityFollowsSequenceWithTable
FindAMinimumStateValue
InterruptibleOperator

[InterruptingAcrossMultipleServers](#)
[InterruptingServerWithMultipleCapacity](#)
[ObjectReferenceOnEntity](#)
[RemoveFromAllocationQueue](#)
[RoutingWithoutPaths](#)
[ServerBlockingApproaches](#)
[SourceServerSinkApproaches](#)
[UsingRelationalTablesToDefineNodeLists](#)
[UsingTheStipulateStepForNodeLists](#)
[UsingTheStipulateStepForResourceAllocation](#)
[WorkersArriveLateToShift](#)

Discrete States

[CommunicationBetweenObjects](#)
[FindAMinimumStateValue](#)
[LogicBasedOnEntityState](#)
[StringStates](#)
[CommunicationBetweenObjects](#)
[VisitAllServersInAnyOrder](#)
[UserDefinedListState](#)
[WorkersArriveLateToShift](#)

Entity Characteristics

[AnimatedPeople](#)
[CustomRoutingGroup](#)
[DefineEntityProperties](#)
[FindAMinimumStateValue](#)
[LogicBasedOnEntityState](#)
[ObjectReferenceOnEntity](#)
[RequestRideFromSameTransporter](#)
[SeizingSameResourceFromList](#)
[SelectEntityTypeFromTable](#)
[TurnaroundMethod](#)
[VehicleFleetManagement](#)

Failures

[MultipleServerFailures](#)

File Management

[DbReadWrite](#)
[ExcelReadWrite](#)
[FileSecurity](#)

[WritingToAFile](#)

Flow Library

[ConstraintLogic](#)

[FlowConcepts](#)

[InfectionPropagationUsingContinuousAndFlow](#)

[TankCleanInPlaceTrigger](#)

[TrackingTankContentsByProductTypeUsingTableStates](#)

Functions

[ExamplesOfFunctions_DynamicObjects](#)

[ExamplesOfFunctions_StaticObjects](#)

[InfectionPropagationUsingContinuousAndFlow](#)

[StringStates](#)

Input Analysis

[InputAnalysis](#)

Level States

[InfectionPropagationUsingContinuousAndFlow](#)

[MassFlowProcessing](#)

[SimpleTank](#)

Lookup and Rate Tables

[DiscreteLookupTable](#)

[LearningCurveWithLookup](#)

[SimpleLeastSlackSelectionRule](#)

[Materials](#)

Constraint Logic Element

[ConstraintLogic](#)

[InventoryReplenish](#)

[ServerWithMaterialConsumption](#)

[ServerWithMaterialConsumptionAndReplenish \(from Project ServerWithMaterialConsumption\)](#)

Movement In Free Space

[FreeSpaceMovement](#)

MultiTask Server

[DashboardsWithinExperiments](#)

[MultiTaskServer_SpecificTime \(from Project ServerUsingTaskSequence\)](#)

[MultiTaskServer_ProcessName \(from Project ServerUsingTaskSequence\)](#)

[MultiTaskServer_Submodel \(from Project ServerUsingTaskSequence\)](#)

[ServersUsingTaskSequenceWithDataTables_FlowLine](#)

[ServersUsingTaskSequenceWithDataTables_JobShop](#)

[ServersUsingTaskSequenceWithDataTables_LoopbackBranches](#)

[ServerUsingTaskSequenceWithWorkers](#)

[UsingRelationalTablesToDefineTaskResourceLists](#)

Resources

[MultipleServerFailures](#)

[ResourceSelectionConcepts](#)

[ResourceStatesWhenOffShift](#)

[SelectingResourceFromList](#)

[SelectingResourcesAsAGroup](#)

[SingleWorkerCompletesProcessAndMovesToNode](#)

[UsingRelationalTablesToDefineTaskResourceLists](#)

RPS

[UsingTheStipulateStepForNodeLists](#)

[UsingTheStipulateStepForResourceAllocation](#)

Schedules / Changeovers

[ResourceSelectionConcepts](#)

[ResourceStatesWhenOffShift](#)

[ResourcesWithWorkSchedules](#)

[ServerFollowsCapacitySchedule](#)

[ServerFollowsDailySchedule](#)

[ServerFollowsOddSchedule](#)

[WorkerUsesWorkSchedule](#)

[WorkerUsesWorkSchedule_InterruptWorkingOffShift](#)

Sequence Tables

[ChooseAlternateSequence](#)

[EntityFollowsSequence](#)

[EntityFollowsSequenceMultiple](#)

[EntityFollowsSequenceWithTable](#)

[SortingConveyorSystem](#)

[UsingRelationalTablesToDefineNodeLists](#)

Supply

[InventoryAndMaterials](#)

Vehicles

[DynamicallyCreatingVehicles](#)

[ElectricVehicle](#)

[ExamplesOfFunctions_DynamicObjects](#)

[RecordDistanceTraveled](#)

[RotatingVehicleExample](#)

[SeizingVehicle](#)

SelectSpecificVehicle
SingleVehicleUsage
VehicleDoingSimultaneousLoading
VehicleFinishesAndParksWhenOffShift
VehicleFixedRoute
VehicleFleetManagement
VehicleMovingBackward
VehiclesPassingOnRoadway
VehicleStopsWhenServerOffShift
VehicleVisitsServiceCenter
VehicleWithVariableRideCapacity

Worker

DashboardsWithinExperiments
MoveASeizedObject
RelationalTablesInRepeatingProperty
RequestRideFromSameTransporter
ServerUsingTaskSequenceWithWorkers
SingleWorkerCompletesProcessAndMovesToNode
TableReferenceInRepeatingProperty
WF_AdditionalResource
WF_Authorization
WorkersArriveLateToShift
WorkerUsedForMultipleTasks
WorkerUsesWorkSchedule
WorkerUsesWorkSchedule_InterruptWorkingOffShift

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AddAndRemoveServerCapacity - SimBit

Problem:

I would like to make an additional server "active" whenever the current servers have a certain number of entities waiting. In other words, there are multiple servers and if any server with capacity reaches a "maximum input buffer level", I want to "open" a new server.

Categories:

Buffering, Decision Logic -- Processing

Key Concepts:

AssociatedStationLoad, Active Symbol, Add-On Process, Assign Step, Condition Based, Contents, Current Capacity, Decide Step, InProcess, InputBuffer, Load, Monitor, NodeList, On Exited, Processing, Status Label

Assumption:

Assume that capacity can be added instantaneously with no set-up/transition time.

Technical Approach:

We want the maximum number of entities in a server's input buffer to be six. When a seventh entity arrives, a new server should come on-line or be activated. There are monitors that watch all the Server Input Buffers and when the seventh entity arrives, an event is fired that triggers a process to change the server capacities. When the server has completed processing, capacity is reduced to zero again until it is needed in the future.

Details for Building the Model:

Simple System Setup

- Place a Source, 6 Servers in parallel, and a Sink in the Facility Window.
- Name the Servers: a1 through a6.
- Connect the Source to each Server and each Server to the Sink with Paths.

Setting Up the Selection Logic

- From within the Definitions window, create a Node List that contains the Input Nodes for all 6 Servers.
- In the Output@Source1 node, change the *Entity Destination Type* property to 'Select From List' and select the Node list that was just created.
- Set the *Selection Goal* to 'Smallest Value' and use the default *Selection Expression* 'Candidate.Node.AssociatedStationLoad'. This expression will select the least congested possible destination. Please refer to the documentation for more details on the AssociatedStationLoad [function](#).
- Set the *Selection Condition* property to 'Candidate.Server.CurrentCapacity > 0'. This ensures that Entities do not get routed to a Server with a capacity of zero.

Creating the Monitor Elements:

- In the Definitions window, select the Elements panel and add 5 Monitor elements (we do not need to monitor server a6).
- Set up the properties of each Monitor Elements as follows:
 - *Monitor Type* is 'CrossingStateChange'
 - *State Variable Name* is 'a1.InputBuffer.InProcess' (or a2, a3,a4, a5)
 - *Threshold Value* is '6'
 - *Crossing Direction* is 'Positive'
- This means that when the Input Buffer threshold value of 6 is crossed in the positive direction (going from 6-7) the Monitor will be triggered. (If the Crossing Direction would be 'Negative' it would be triggered when it went from 6-5, which is not what we want to have happen.)

Increasing Server Capacities:

- From within the Processes window, create a new Process (Create Process ribbon button) and rename it

'UpdateServerCapacities'

- Place a single Assign step. Open the Repeating Property Editor in Assignments (More).
 - For the first assignment, set the *State Variable Name* to 'a6.CurrentCapacity' and the *New Value* to '(Input@a6.AssociatedStationLoad > 0 || ((a1.Capacity > 0) && (a2.Capacity > 0) && (a3.Capacity > 0) && (a4.Capacity > 0) && (a5.Capacity > 0)))'
 - Add another assignment: *State Variable Name* is 'a5.CurrentCapacity' and *New Value* is '(Input@a5.AssociatedStationLoad > 0 || ((a1.Capacity > 0) && (a2.Capacity > 0) && (a3.Capacity > 0) && (a4.Capacity > 0)))'
 - Similarly, add: *State Variable Name* 'a4.CurrentCapacity' with *New Value* '(Input@a4.AssociatedStationLoad > 0 || ((a1.Capacity > 0) && (a2.Capacity > 0) && (a3.Capacity > 0)))'
 - And finally: *State Variable Name* 'a3.CurrentCapacity' and *New Value* '(Input@a3.AssociatedStationLoad > 0 || ((a1.Capacity > 0) && (a2.Capacity > 0)))'

These expressions are checking 2 things: (1) If it is processing an Entity, an Entity is waiting to be processed, or an Entity is routing to be processed, then let it remain operating OR (2) all the Servers before that one are working. Because we used a logical expression it will return a 1 if true, and a 0 if false – which is what we want to set our capacities to.

Note: It is important that these expressions are in this exact order - descending order. Otherwise, the logic will not evaluate correctly.

- Now that the Process has been created, we have to trigger it – this is done back in the Monitor Element. From the Definitions window, set the *On Change Detected Process* to 'UpdateServerCapacities' for each Monitor element. Therefore, once the Monitor is triggered (i.e. when the seventh Entity arrives) it will fire this process, which checks all of the conditions and then sets the capacities accordingly.

Decreasing Capacities:

- From within the Facility window, create a process in a3's Exited Add-On Process Trigger
- From the Processes window, place a Decide step in this new process. Set *Decide Type* to 'ConditionBased' and *Expression* to 'Input@a3.AssociatedStationLoad == 0'.
- On the True segment, place an Assign step. Set the *State Variable Name* to 'a3.CurrentCapacity' and leave the *New Value* '0.0'.
- This process will check to see if there are any entities processing, waiting to be processed, and routing to the server for processing. If there are not, then it assigns the Server's capacity to 0.
- Repeat this for Servers a4, a5, and a6.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AnimatedPeople - SimBit

Problem:

I would like to use animated people (i.e. walking people) and change the animation of the person in the model.

Categories:

Animation, Entity Characteristics

Key Concepts:

Active Symbol, Assign Step, Current Animation Index

Assumption:

This model assumes the user is using the animated people symbols that are available in the Simio Project Symbol library available with the Simio installation (Library\People\Animated). To ensure you only get symbols of people that move, click on the Apply Symbol button. Then change the Action filter to Animated.

Technical Approach:

This model demonstrates how to use the animation features available with some of the animated symbols provided in the Simio Symbol Library. Once an animated symbol is placed in the Facility window, right click onto the symbol and select *List Animations of Active Symbol* to see what options are available for animation of this symbol.

When an entity is selected, there are two properties on the entity which control the animation for the symbol. The *Current Animation Index* returns the numeric index or string name of the current animation of the current active symbol. By default, the value of this property is 'ModelEntity.Animation', which is a String State on the ModelEntity object.

The other property on the ModelEntity that controls animation is the *Default Animation Action* property. This indicates if and how Simio will animate objects that don't have an explicit animation set via the *Current Animation Index* expression. The possible values for this property are None, Moving and MovingAndIdle. 'None' means that no action will be taken. 'Moving' indicates actively moving entities will be animated with any available forward movement animations from the active symbol. 'MovingAndIdle' indicates actively moving entity and idle objects will be animated with any available forward movement or idle animations from the active symbol.

Details for Building the Model:

Simple System Setup

- Place four Source objects and four Sink objects. Connect the first three Source objects to the first three Sink objects with a Path.
- Place a TransferNode between Source4 and Sink4 and connect Source4 to the TransferNode with a Path and connect the TransferNode to Sink4 with a Path.

Randomly Assign Animation Example (Source1)

- Drag an entity into the Facility window from the Project Library. Rename this entity 'Person1'.
- With this entity selected, select one of the Male icons from the Project Symbols drop down menu in the ribbon (under Library\People\Animated).
- With the male entity selected, right click and select *List Animations of Active Symbol* to see what options are available for animation of this symbol. Notice the options for "Walk" and "Run". Click OK to close the window.
- Select Source1 and open the *Before Exiting Repeat Property Editor* window. Click Add and enter the *State Variable Name*, 'ModelEntity.Animation'. Set the *New Value* to 'Random.Discrete("Walk", .6, "Run", 1) so that 60% of the entities will walk and the other 40% will run.

Assign Numeric Value to Current Animation Index Property Example (Source2)

- Place another entity into the Facility window from the Project Library. Rename this entity 'Person2'.
- With this entity selected, select one of the Female icons from the Project Symbols drop down menu in the ribbon.
- With the Female entity selected, right click and select *List Animations of Active Symbol* to see what options are available for animation of this symbol. Notice that number 13 is "Walk Pushing". Click OK to close the window.
- With the Female entity selected, expand the Animation property category and find the *Current Animation Index*

property. This property will accept either an expression that returns the numeric index or string name of the current animation of the active symbol. Put the value of '13' into this property so that the entities will "Walk Pushing".

- Click onto Source2 and change the *Entity Type* property to 'Person2' so that it produces this new entity type.

Simio Determines Animation Based on Speed Example (Source3)

- Place an entity into the Facility window from the Project Library. Rename this entity 'Person3'.
- With this entity selected, select a Male from the Project Symbols drop down menu in the ribbon.
- Click onto Source2 and change the *Entity Type* property to 'Person3' so that it produces this new entity type.
- With Source3 selected, open the *Before Exiting* Repeat Property Editor window. Click Add and enter the *State Variable Name*, 'ModelEntity.DesiredSpeed'. Set the *New Value* to 'Random.Uniform(.1,3)' and *Units* to 'Meters per Second' so that some entities travel slow and others fast. Simio will determine which animation to use, based on the speed. Notice that some entities will walk, while others will run.
- Change the *Allow Passing* property on the path between the Source and Sink to be 'False', so that slower walking people cause others to slow down as well.

Change Animation in Logic Example (Source4)

- Select Source4 and open the *Before Exiting* Repeat Property Editor window. Click Add and enter the *State Variable Name*, 'ModelEntity.Animation'. Set the *New Value* to 'Random.Discrete("Run", .5, "Dance", 1) so that 50% of the entities will run and the other 50% will dance.
- Click onto TransferNode1 and create a new State Assignment in the *On Entering* property.
- In the Repeat Group, set the *State Variable Name* to 'ModelEntity.Animation' and the *New Value* to '"Walk"'. When the entity enters this TransferNode, it will change from its current animation of "Dance" or "Run" to "Walk". Note that you can do this using an Add-On Process Trigger with an Assign step as well.

Embellishments:

Change the active animation to try different options, such as a sitting female symbol or a male symbol that is texting.

***IMPORTANT NOTE:** Animation options, such as "Walk", "Run", "Dance", etc. are case-sensitive.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AnimatePathDensity - SimBit

Problem:

I would like to animate the travel density of each path by changing the width of each path so that the width is equal to the percentage of travelers that have traveled this path.

Categories:

Add-On Process Logic, Animation, Building New Objects/Heirarchy

Key Concepts:

Add-On Process, Assign Step, Custom Object, NumberEntered, On Entered, Size.Width, State Assignments

Technical Approach:

The model demonstrates how to animate the travel density of a path by changing the width so that the width is equal to the percentage of travelers that have traveled this path. The model demonstrates how to do this using the standard Path object and an Add On Process and also by customizing the Path object and using this subclassed Path to accomplish the same goal.

The first demonstration contains three standard paths connecting a Source and a Sink object. Each path has a different value set for its Selection Weight property, indicating the percentage of entities that should travel on each path. Upon entering each path, the Path.Size.Width state is set to the percentage of travelers that have entered into this Path divided by the total traffic in the system.

The second demonstration is the same as the first, except that a custom path object is used that contains the width update inside its own logic instead of having the width updated with an Add On Process in the model.

Enhanced Technical Approach:

Facility Window:

- Place a Source object and a Sink object and connect them with three standard Paths.
- In the properties window for Path1, under Routing Logic, set *Selection Weight* to '6'.
- In the properties window for Path2, under Routing Logic, set *Selection Weight* to '1'.
- In the properties window for Path3, under Routing Logic, set *Selection Weight* to '3'.

Definitions Window:

- Go to the States panel and create a new Real State named TotalTraffic_Example1.

Processes Window:

- Create a new process and name it Path_Entered.
 - Place an Assign Step and set the State Variable Name to 'TotalTraffic_Example1' and the New Value to 'TotalTraffic_Example1 + 1'
 - Place another Assign Step. Set the State Variable Name to 'Path1.Size.Width' and the New Value to 'Path1.NumberTravelers.Entered/TotalTraffic_Example1'
 - Add two more assignments to this Assign Step, assigning the same values to Path2 and Path3. For example: Path2.Size.Width should be set to 'Path2.NumberTravelers.Entered/TotalTraffic_Example1'

Facility Window:

- Select each Path (hold CTRL and click onto each Path). Under the Add-On Process Triggers property category, enter the name of the process (Path_Entered) into the Entered property. Therefore, each time a Modelentity enters any path, this new process is executed.

To create a custom Path object (functionality not available with Simio Express Edition):

- Go to the Definitions Window and create a second Real State, named 'TotalTraffic_Example2'
- In the Facility window, right click onto the Path in the Standard Library and select Subclass.
- Click onto the new Path object in the Navigation Window.
 - Go into the Definitions window of the MyPath object. Go to the Properties panel and create a new State

Property by selecting this from the Standard Property drop down in the Ribbon. Name this new property 'TotalTraffic'.

- Go into the Processes window of the MyPath object
 - Find the OnEntered process and with this selected, click onto Override in the Ribbon to override the default logic.
 - Place an Assign Step after the EntityMovementRate Assign Step and before the IfPassingAllowed Decide step. Set the *State Variable Name* to 'TotalTraffic' and the *New Value* property to 'TotalTraffic + 1'
 - Place another Assign Step after this. Set the *State Variable Name* to 'Size.Width' and the *New Value* property to 'NumberTravelers.Entered/TotalTraffic'
- When you use this new MyPath object in the main model to build the second example, you will need to put the name of the state variable 'TotalTraffic_Example2' into the State Property on each instance of the MyPath that you place into the Facility window. This will tell this object which State Variable to update in order to keep track of the total traffic in this system.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AnimatingQueuesWithVehiclesAndWorkers - SimBit

Problem:

I would like to show entities in different queues based on an expression and match the animation speed of an animated entity in an attached queue with the Vehicle/Worker.

Categories:

Animation, Data Tables, Entity Characteristics, Vehicles, Worker

Key Concepts:

Animated Symbol, Before Creating Entities, Bidirectional Path, Data Table, Entity Property, Initial Desired Speed, Initial Node (Home), Match Attached Animation Speed, ModelEntity, Queue, Queue State, People, Priority, RandomRow, Real Property, Ride On Transporter, RideStation.Contents Queue, Sink, Source, Symbol, Table Row Referencing, Vehicle, Visibility Expression, Worker

Technical Approach:

The Vehicle picks up multiple entities and they are shown in different queues based on the Visibility Expression on the queues. A Worker transports an animated entity, and the animation speed is automatically set to match the Worker's speed.

Details for Building the Model:

Top Model

- Place a Source (Source1), Sink (Sink1), and Vehicle (Vehicle1) on the Facility window.
- Use a Path to connect Source1 to Sink1, and use a second path to connect Sink1 to Source1.
- Place two ModelEntity instances from the Project Library on the bottom left into the Facility window.
- Select DefaultEntity and change its *Initial Priority* to '2'. Leave the other Entity's *Initial Priority* as '1.0'.
- Double click DefaultEntity and change its name it to 'Red'. Change the other Entity's name to 'Blue'.
- Select the Red Entity, open the Color dropdown on the Symbol Ribbon and select the Red box. Then click the top of the Red ModelEntity. Click the Red box in the Color dropdown again and select the side of the Red ModelEntity (must be in 3D mode to view side - press '3' key). Change the Blue entity's symbol color to blue.
- Open the Data Tab and click Add Data Table (Table1). Add an Entity Property Column (Object Reference dropdown) and a Real Property Column (Standard Property dropdown). Change the column *Name* to 'WhichType' and 'HowMuch', respectively.
- In column Which Type input 'Blue', 'Red' and in column How Much input '60', '40'.
- On Source1, set *Entity Type* to 'Table1.WhichType'. Under Table Row Referencing-> Before Creating Entities set *Table Name* to 'Table1' and *Row Number* to 'Table1.HowMuch.RandomRow'.
- On Output@Source1, set *Ride On Transporter* to 'True' and *Transporter Name* to 'Vehicle1'.
- Select Vehicle1 and change its *Initial Ride Capacity* to '4', *Initial Desired Speed* to '.5 meters per second', and *Initial Node (Home)* to 'Input@Sink1'.
- Select the Vehicle1 RideStation.Contents queue. Set the *Visibility Expression* to 'Candidate.ModelEntity.Priority == 1'.
- Select Vehicle1. In the Draw Queue dropdown (in the Symbols Ribbon) select RideStation.Contents. To draw the queue, click and release in the model space just below the right side of Vehicle1, click and release in the space just below the left side of Vehicle1, and then right click to stop drawing the queue.
- Select that queue and set the *Visibility Expression* to 'Candidate.ModelEntity.Priority == 2'.
- To move the queue, first press the '3' key to go to 3D mode. Then select the queue and hold the 'Shift' key to move the queue vertically. Move the queues so both are directly above the vehicle. Press '2' to go back to 2D mode.

Bottom Model

- Place a Source (Source2), Sink (Sink2), Worker (Worker1), and ModelEntity on the Facility window.
- Connect the Source to the Sink using a Path and set its *Type* to 'Bidirectional'.
- Select the ModelEntity, then click Apply Symbol in the Symbols Ribbon. Scroll down to the 'Animated' categories and select a person.

-
- Double click the ModelEntity and rename it to 'Person'.
 - On Source2, set *Entity Type* to 'Person' and *Interarrival Time* to 'Random.Exponential(3)'.
 - On Output@Source2, set *Ride On Transporter* to 'True' and *Transporter Name* to 'Worker1'.
 - On Worker1, set *Initial Desired Speed* to '.2 meters per second' and *Initial Node (Home)* to 'Input@Sink2'.
 - Select Worker1 and click Apply Symbol in the Symbols Ribbon. Scroll down to the 'Animated' categories and select a person.
 - Select Worker1's RideStation.Contents queue. Hold the Shift key to lower the queue. Then press the '2' key to switch to 2D mode. Use the green circle ends of the queue to adjust its placement. By default, the queue option for Match Attached Animation Speed is turned on. This will cause the entity in the queue to be animated (walking) with the worker instead of gliding.
-

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AppointmentArrivals - SimBit

Problem:

I would like to model arrivals that occur at a specific time, such as scheduled appointments.

Categories:

Arrival Logic

Key Concepts:

Arrival Mode, Arrival Table, Arrival Time Deviation, Data Table, Floor Label, Status Plot

Technical Approach:

Scheduled arrivals are put into a Data Table in a DateTime property column. In this example, appointments are scheduled every 15 minutes from 8:00am until 4:30pm. The Source object reads the arrival data table to see when to create an entity. However, since the Source's Arrival Time Deviation property has a non-zero value, there will be some deviation from when the entities are actually created from the time listed in the table. This simulates that some people arrive early to their appointment and some people arrive late. Because the Source's Arrival No-Show property is set to .05, there is a 5% chance of each arrival being a no-show, which simulates the scenario when someone doesn't show up at all for their scheduled appointment. The plots and floor labels show how the status of the Average Number of entities waiting to be served and the Average Time that each entity waits to be served.

Details for Building the Model:

Simple System Setup

- Click on the Data tab and create a new Data Table by clicking on Add Data Table in the Schema ribbon and name this new table DailyAppointments. Create a Date Time column by selecting DateTime in the Property drop down in the Schema ribbon. Name this new column AppointmentTime.
 - Enter data into the table, beginning at 8:00am on the day the simulation begins and every 15 minutes for the rest of the day, with the last appointment occurring at 4:30pm.
- Place a Source, Server and Sink into the Facility window and connect them with Paths. Alternatively, go to the Project Home ribbon and select the Source, Server, Sink option from the Actions (under Add-Ins) drop down.
- Select the Source object and set the following properties:
 - *Arrival Mode* to 'Arrival Table'
 - *ArrivalTimeProperty* to 'DailyAppointments.AppointmentTime' – pointing to the DateTime column you created.
 - *ArrivalTimeDeviation* to 'Random.Uniform(-5,5)' and the Units to 'Minutes'
 - *Arrival No-Show Probability* to '.05'
- Select the Server object and set the *Processing Time* property to 'Random.Triangular(10, 16, 22)'
- Click onto the Run ribbon and change the starting time of the simulation run to be 8:00am and the ending time to be 5:00pm on the same day. The date should match the date that was entered into the Data Table.

Embellishments:

Place a Status Plot that displays statistics of interest, such as the expression 'Server1.InputBuffer.Contents.AverageNumberWaiting' or 'Server1.InputBuffer.Contents.AverageTimeWaiting'.

Arrival Tables can also use a numeric property column that lists the amount of time the arrival should occur after the start of the simulation run. For example, if the Arrival table had a Numeric column and had the entries of .25, .5, an arrival would be generated at .25 hours after the start of the simulation run and again at .5 hours after the start of the simulation run. This is just an alternate way to model scheduled arrivals.

AutoCreatePathsWithVertices - SimBit

Problem:

I want to create some links with vertices from a Data Table.

Categories:

Data Tables

Keywords:

Auto-Create, Data Tables, Links

Technical Approach:

Using Data Tables and auto-creation, create objects and links from data.

Details for Building the Model:

To start this model, add three data tables to the Data Tables view. Set their Names to 'ObjectTable', 'LinkTable', and 'VerticesTable', respectively.

Object Table Setup

- Add an Object Type column
- Set *Name* to 'Object Type'
- Add a Real column
- Set *Name* to 'X'
- Add a Real column
- Set *Name* to 'Z'
- Add an Object column
- Set *Name* to 'ObjectName'
- Set *Object Type* to 'Object Type'
- Set *Default Value Instantiation* to 'AutoCreateInstance'
- Set *X (Initial Object Offset)* to 'X'
- Set *Z (Initial Object Offset)* to 'Z'

Link Table Setup

- Add a Node column
- Set *Name* to 'Start Node'
- Add a Node column
- Set *Name* to 'End Node'
- Add an Object column
- Set *Name* to 'LinkName'
- Set *Object Type* to 'Path'

- Set *Default Value Instantiation* to 'AutoCreateInstance'
- Set *Link Starting Node* (Link Nodes) to 'StartNode'
- Set *Link Ending Node* (Link Nodes) to 'EndNode'
- Set *Vertices X* (Link Vertices) to 'VerticesTable.VertexX'
- Set *Vertices Z* (Link Vertices) to 'VerticesTable.VertexZ'
- Set Column As Key

Vertices Table Setup

- Add a Foreign Key Property column
- Set *Table Key* to 'LinkTable.LinkName'
- Set *Name* to 'LinkName'
- Add a Real column
- Set *Name* to 'VertexX'
- Add a Real column
- Set *Name* to 'VertexZ'

Filling Out the Tables with Data

- Return to the Object Table and select the "Object Type" column
- Add a Source, BasicNode, Server, TransferNode, and Sink rows to the table.
- The "Object Name" column will automatically fill out names.

Set X and Z columns with the following data.

X	Z
-8.75	-3.75
-4.5	-2
0.125	0
4.875	-2
9.5	-2.5

Return to the Link Table and set the "Start Node" and "End Node" columns with the following data. The "Link Name" column will automatically fill out its rows.

Start Node	End Node
Output@Source1	BasicNode1
BasicNode1	TransferNode1
TransferNode1	Input@Server1
Output@Server1	Input@Sink1

Return to the Vertices Table and set the "Link Name", "Vertex X", and "Vertex Z" with the following data.

Link Name	Vertex X	Vertex Z
-----------	----------	----------

Path1	0.5	-7
Path2	-0.25	3.75
Path3	2.375	-4.25
Path4	6.5	2

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

BusinessGraphicsAnimation - SimBit

Business Graphics Animation – SimBit

The SimBit project includes three models providing examples of Simio's Facility window Charts and Label animations.

1. **ExpressionDrivenAnimation:** Demonstrates how to create Charts and Status Labels based on expressions.
2. **DataTableDrivenAnimation:** Demonstrates how to create Charts and Status Labels based on Data Tables.
3. **OutputTableDrivenAnimation:** Demonstrates how to create charts based on Output Tables.

Model 1: Expression Driven Animation

Problem:

I want to use different Charts, Plots, and Status Labels in the Facility window to display real time data in my Model.

Categories:

Animation

Keywords:

Bar Chart, Pie Chart, Status Label, Time Plot

Assumptions:

One Source will create all Entities.

Technical Approach:

Plots, Charts, and Status Labels are used to display real-time data in the Facility window. Expressions and List States drive the data in these animations.

Details for Building the Model:

Facility Window: Objects

- In the Facility window, place a Source, Server, and Sink, as shown in the Model. Connect the Source to the Server and the Server to the Sink with Paths.
- Add a ModelEntity Instance.

Facility Window: Animation

- In the Facility window, go to the Animation ribbon and select Time Plot. Create a new Time Plot by clicking where you want to place the top right corner of the plot, dragging the mouse until the plot is as large as desired, and then clicking again.
- With the Time Plot selected, set the following values in the Appearance ribbon:
 - *Title* = 'Number In System'
 - *X Axis* = 'Hour'
 - *Y Axis* = 'Number Entities'
 - *Time Range* = 'Two Hours'
- With the Time Plot selected, set the following properties:
 - *Data Source* = '[Expressions]'
- Under *Expressions*, open the Expressions Repeating Property Editor by clicking the three dots on the right. Add an

item by clicking the add and set the following properties:

- *Series* = 'NIS'
- *Value* = 'DefaultEntity.Population.NumberInSystem'
- *Color* = blank
- Go to the Animation ribbon and select Bar Chart. Create a new Bar Chart, as shown in the Model.
- With the Bar Chart selected, set the following values in the Appearance ribbon.
- *Title* = 'Number Entities Created and Destroyed'
- *X Axis* = 'Entity Type'
- *Y Axis* = 'Number Entities'
- With the Bar Chart selected, set the following properties:
- *Data Source* = '[Expressions]'
- Under *Expressions*, open the Expressions Repeating Property Editor by clicking the three dots on the right.
- Add an item by clicking the add button and set the following properties:
- *Series* = '"Number Created"'
- *Argument* = '"Default Entity"'
- *Value* = 'DefaultEntity.Population.NumberCreated'
- *Color* = 'Color.Green'
- Add a second item and set the following properties:
- *Series* = '"Number Destroyed"'
- *Argument* = '"Default Entity"'
- *Value* = 'DefaultEntity.Population.NumberDestroyed'
- *Color* = 'Color.Red'
- Go to the Animation ribbon and select Status Label. Create a new Status Label to the right of the Server, as shown in the Model.
- With the Status Label selected, set the *Expression* to 'Server1.InputBuffer.Contents'.

Facility Window: Objects, Animation

- With the Server selected, go to the Attached Animation ribbon. Click the arrow below Pie Chart and select the Resource State option under Quick Create for Server1. Add the Server1.ResourceState Pie Chart, as shown in the Model.
- With the Server selected, go to the Attached Animation ribbon and select Status Label. Create a new Status Label under the Server, as shown in the Model.
- With the Status Label selected, set the *Expression* to 'InputBuffer.Contents'.
- With the DefaultEntity instance selected, go to the Attached Animation ribbon and select Status Label. Create a new Status Label above the Default Entity, as shown in the Model.
- With the Status Label selected, set the *Expression* to 'TimeCreated'.

Discussion:

Run this Model and see how the visualizations of the system change over time. Experiment with different expressions, chart types, and styles to see how the animations change.

Notes:

For the Time Plot, you will notice the *Color* property is left blank. If the *Color* property is not given a value, a default color palette will be used.

For *Series* and *Argument* names in the Status Plot and Bar Chart, you need to use quotation marks around the string if you want a space between words, as displayed in the Model. If you do not have quotation marks, there will be no space between words.

For the Attached Animation Status Label that is attached to Server1, the value will be the same as the Status Label that is unattached. They are both displaying the number of Entities waiting in the Server's Input Buffer. The only differences are that if Server1 is moved, the Attached Animation Status Label will move in relation to the Server and the expression in the Attached Animation Status Label must be related to Server1.

For the ModelEntity, the Attached Animation Status Label will be created for each Actualization of the Instance and the expression of the label will be evaluated by the Actualization, rather than the DefaultEntity1 Instance. The Entity's label will move with dynamic Actualization as it goes through the system.

Model 2: Data Table Driven Animation**Problem:**

I want to use data tables as inputs for Charts, Plots, and Status Tables in the Facility window.

Categories:

Animation, Data Table

Keywords:

Data Table, Pie Chart, Status Table

Assumptions:

One Source will create three Entity types with the Entity Mix defined in a Data Table with a random interarrival time.

Technical Approach:

Charts, Plots, and Status Tables are used to display real-time data in the Facility window. A Data Table drives the data in the animations.

Details for Building the Model:

Facility Window: Objects

- Start with Model1.
- Add two additional ModelEntities. Name the Entity instances 'Red', 'Green', and 'Blue'. While an instance is selected, you can choose to color the symbol with the Color option in the Symbols ribbon.

Data Window: Tables

- In the Data window, add a Data Table named "EntityTable".
- For the EntityTable Schema, add the following:
 - Entity Property
 - *Name* = 'EntityType'
 - Real Property
 - *Name* = 'Mix'
 - Integer State Variable Property
 - *Name* = 'NumberCreated'
- For the EntityType column, add the following data: Red, Green, Blue
- For the Mix column, add the following data: 50, 20, 30

Process Window: Processes

- Create a new Process. With the Process selected, change the *Name* to 'CreatedEntity'.
- Add an Assign step and set the following properties:
- *State Variable* = 'EntityTable.NumberCreated'
- *New Value* = 'EntityTable.NumberCreated + 1'
- *Name* = 'IncreaseNumberCreated'

Facility Window: Objects

- Select the Source and set the following properties:
- Entity Arrival Logic
- *Entity Type* = 'EntityTable.EntityType'
- Table Row Referencing > Before Creating Entities:
- *Table Name* = 'EntityTable'
- *Row Number* = 'EntityTable.Mix.RandomRow'
- Add-On Process Triggers
- *Created Entity* = 'CreatedEntity'

Facility Window: Animation

- Go to the Animation ribbon and select Pie Chart. Create a new Pie Chart, as shown in the Model.
- With the Pie Chart selected, set the *Title* to 'Entity Type Mix' in the Appearance ribbon.
- With the Pie Chart selected, set the following properties:
- *Data Source* = 'EntityTable'
- *Argument Data Field* = 'EntityTable.EntityType'
- *Value Data Field* = 'EntityTable.Mix'
- Go to the Animation ribbon and select Status Table. Create a new Status Table above the Source, as shown in the Model. Set the *Table Name* property to 'EntityTable'.

Discussion:

Run this Model and see how the Status Pie is static during the run while the NumberCreated column in the Status Table is dynamic. State columns cannot be used with Plots and Charts but can with Status Tables. Since the *Data Source* for the Pie Chart is a Data Table, it will be constant throughout the run.

Model 3: Output Table Driven Animation

Problem:

I want to use Output Tables as inputs for Charts and Status Labels in the Facility window.

Categories:

Animation, Data Table, Output Table

Keywords:

Bar Chart, Data Table, Output Table, Time Plot

Assumptions:

One Source will create three Entity types with the Entity Mix defined in a Data Table with a random interarrival time.

Technical Approach:

Plots and charts are used to display real-time data in the Facility window. An Output Table drives the data in the animations.

Details for Building the Model:

Facility Window: Objects

- Start with Model 2.

Data Window: Tables

- In the Data window, add an Output Table named 'EntityOutput'
- For the EntityTable Schema, add the following:
 - DateTime State
 - *Name* = 'TimeDestroyed'
 - Real State
 - *Name* = 'Priority'
 - Entity Reference State
 - *Name* = 'EntityType'
 - Real State
 - *Name* = 'TIS'
 - Real State
 - *Name* = 'Color'

Process Window: Processes

- Select the 'CreatedEntity' Process and add an Assign step named "ChangePriority" with the following properties:
 - *State Variable Name* = 'ModelEntity.Priority'
 - *New Value* = 'Random.Discrete(1, .1, 2, .4, 3, .6, 4, .85, 5,1)'
- Create a new Process named "UpdateOutputTable"
- Add an AddRow step named "AddRowInOutputTable" and set the following properties:
 - *Table Name* = 'EntityOutput'
- Add an Assign step named "AssignValuesInOutputTable" with 5 items in the Assignments Repeating Property Editor with the following properties:
 - *State Variable Name* = 'EntityOutput.TimeDestroyed'
 - *New Value* = 'TimeNow'
 - *State Variable Name* = 'EntityOutput.Priority'
 - *New Value* = 'ModelEntity.Priority'
 - *State Variable Name* = 'EntityOutput.EntityType'
 - *New Value* = 'ModelEntity.EntityType'
 - *State Variable Name* = 'EntityOutput.TIS'
 - *New Value* = 'ModelEntity.TimeInSystem'

- *State Variable Name* = 'EntityOutput.Color'
- *New Value* = 'Math.If(Is.Red, Color.Red, Math.If(Is.Green, Color.Green, Color.Blue))'

Facility Window: Objects

- Select the Sink and set the following property:
- Add-On Process Triggers
- *Entered* = 'UpdateOutputTable'

Facility Window: Animation

- Go to the Animation ribbon and select Bar Chart. Create a new Bar Chart to the top left of the Source, as shown in the Model.
- With the Bar Chart selected, set the following values in the Appearance ribbon.
- *Title* = 'Last Entity's Priority'
- *X Axis* = blank
- *Y Axis* = 'Priority'
- *Text Scale* = '0.75'
- With the Bar Chart selected, set the following properties:
- *Data Source* = 'EntityOutput'
- *Series Data Field* = 'EntityOutput.EntityType'
- *Value Data Field* = 'EntityOutput.Priority'
- *Color Data Field* = 'EntityOutput.Color'
- *Summary Function* = 'None'
- *Aggregation Interval* = 'None'
- With the Bar Chart Selected, go to the Appearance Tab -> Change Chart Type and select 'Stacked Bar Chart'.
- Go the Animation ribbon and select Bar Chart. Create a new Bar Chart to the bottom left of the Source, as shown in the Model.
- With the Bar Chart selected, set the following values in the Appearance ribbon.
- *Title* = 'Average Entity Priority by Type Per Hour'
- *X Axis* = 'Hour'
- *Y Axis* = 'Priority'
- *Text Scale* = '0.75'
- With the Bar Chart selected, set the follow properties:
- *Data Source* = 'EntityOutput'
- *Series Data Field* = 'EntityOutput.EntityType'
- *Value Data Field* = 'EntityOutput.Priority'
- *Color Data Field* = 'EntityOutput.Color'
- *Summary Function* = 'Average'
- *Aggregation Interval* = 'Hour'

- With the Bar Chart Selected, go to the Appearance Tab -> Change Chart Type and select 'Stacked Bar Chart'.
- Go to the Animation ribbon and select Time Plot. Create a new Time Plot to the top right of the Source, as shown in the Model.
- With the Time Plot selected, set the following values in the Appearance ribbon.
- *Title* = 'Time in System per Entity by Priority'
- *X Axis* = 'Hours'
- *Text Scale* = '0.75'
- *Time Range* = '6 Hours'
- With the Time Plot selected, set the follow properties:
- *Data Source* = 'EntityOutput'
- *Series Data Field* = 'EntityOutput.Priority'
- *Argument Data Field* = 'EntityOutput.TimeDestroyed'
- *Value Data Field* = 'EntityOutput.TIS'
- *Color Data Field* = blank
- *Summary Function* = 'None'
- *Aggregation Interval* = 'None'
- Go to the Animation ribbon and select Time Plot. Create a new Time Plot below the Sink, as shown in the Model.
- With the Time Plot selected, set the following values in the Appearance ribbon.
- *Title* = 'Time in System per Entity by Priority'
- *X Axis* = 'Hours'
- *Text Scale* = '0.75'
- *Time Range* = '6 Hours'
- With the Time Plot selected, set the follow properties:
- *Data Source* = 'EntityOutput'
- *Series Data Field* = 'EntityOutput.Priority'
- *Argument Data Field* = 'EntityOutput.TimeDestroyed'
- *Value Data Field* = 'EntityOutput.TIS'
- *Color Data Field* = blank
- *Summary Function* = 'Maximum'
- *Aggregation Interval* = 'Hour'

Discussion:

In the Model, test out changing the fields of the charts, particularly the *Summary Function* and *Aggregation Level* to build different charts. Watch in the Facility how changing the fields changes the charts. When an *Aggregation Level* is set, the *Summary Function* will default to 'Average' if it is not set.

Notes:

The AddRow step is only available in Professional and RPS editions. For more information on this, see the "AddRow" and "Output Tables" Simio Help pages.

For more information on the different fields in the charts, see the “Status Labels, Plots, Gauges and Buttons” Help page.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

BatchingProcessUsingScanOrWaitStepToControlBatchSize - SimBit

Problem:

I want to model a single server that must process entities in batches. For example, an oven that cures a quantity of arriving items all at once.

Categories:

Combining and Separating Entities

Key Concepts:

Batching, Scan Step, Wait Step, Monitor, Polled Waiting, Event Driven Waiting

Assumptions:

A system consists of a single server that must process entities in batches. For example, an oven that cures a quantity of arriving items all at once. The time between entity arrivals to the system is sampled from an exponential distribution, EXPO(1.1) minutes.

The maximum size of a batch that can be processed depends on the server's capacity. In this example, that capacity is set to 10, with a minimum batch size requirement of 8 items before the process can be started. Thus, the processing of the next batch cannot be started unless there are at least that many entities waiting in the server's input buffer. The process delay itself is batch size dependent and is 3 minutes plus 0.6 minutes for each item in the batch. Once processing of a batch is completed, the individual entities that were processed together then exit the server.

Details for Building the Model:

Facility Window Setup

- Add a Source, Server, and Sink to the Facility window. Connect the Source to the Server and the Server to the Sink using Connector links.
- Place a ModelEntity object from the Project Library into the Facility window. The name may be left as 'DefaultEntity'.

Source Properties

- Specify the *Interarrival Time* as 'Random.Exponential(1.1)' minutes.

Server Properties

- Specify the *Initial Capacity* as '10'. This is the maximum batch size constraint.
- Specify the *Processing Time* as '3 + 0.6 * Server1.Capacity.Allocated' minutes.
- Go to Add-On Process Triggers and create add-on processes to be executed for *Run Initialized* and *Evaluating Seize Request*.

Definitions-> States

- Add a global boolean state variable named 'Server1HoldForBatching' (initial value of 'True'). This variable will be used in conjunction with the server's *Evaluating Seize Request* add-on process to indicate whether entities arriving to the server can currently seize it.

Definitions-> Elements

- If using a Wait step approach (event driven waiting) to model the batch size control at the server, then add a Monitor element to the model named 'Server1StatusChanged' that fires an event whenever discrete state changes are detected for the state variables Server1.AllocationQueue or Server1.ResourceState.

Processes Window -> The server's Evaluating Seize Request triggered process

- Refer to the example model for guidance. The purpose of this process logic is to reject an entity's attempt to seize the server (by assigning the executing token's Return Value state to False) if the value of the boolean state variable Server1HoldForBatching is 'True'.

Processes Window -> The server's Run Initialized triggered process

- Refer to the example model for guidance. The purpose of this process logic is, when the server is initialized, to create a single control token that waits at either a Scan or Wait step until the minimum acceptable number of entities is waiting to seize the server and the server is idle. Once that condition is true, the token is released from the Scan or Wait step and momentarily sets the Server1HoldForBatching flag to 'False' before then executing an Allocate step to allocate the server's capacity (up to a maximum of 10). The token then sets the Server1HoldForBatching flag to 'True' and loops back to the Scan or Wait step to wait until it can start the processing of the next batch.

BidirectionalPaths - SimBit

Problem:

I have bidirectional links in my system, but my entities get stuck at a node trying to get from one link to the other in opposite directions.

Categories:

Decision Logic -- Paths

Key Concepts:

Allow Passing, BasicNode, Bidirectional Path, Deadlock, Path, Prefer Desired Direction, Traffic Direction Rule

Assumptions:

Entities are able to pass each other at 'bypass' areas along various bidirectional paths.

Technical Approach:

The path network is set up with multiple bidirectional paths. The paths are separated at various points along the path with a small triangle of unidirectional paths so that entities may sit/wait for passing in the opposite direction to occur.

Details for Building the Model:

Simple System Setup

- Place a Source (Source1) and Sink (Sink1) at the left and right sides, respectively, of the Facility window.
- Then, place a second Source (Source2) at the right side, below Sink1. Place a second Sink (Sink2) on the left, below Source1.
- Place two ModelEntity objects. Change the names to 'Red' and 'Green' and change the symbol color of Red to match the name. Change *Entity Type* of Source1 to 'Green' and *Entity Type* of Source2 to 'Red'.

Specifying the Destination Locations

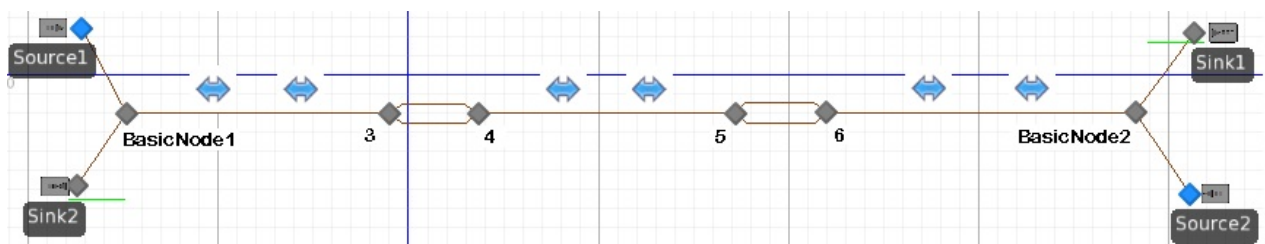
- Within Source1's output node (Output@Source1), change the *Entity Destination Type* to 'Specific' and the *Node Name* to 'Input@Sink1'. Similarly, within Source2's output node (Output@Source2), change the *Entity Destination Type* to 'Specific' and the *Node Name* to 'Input@Sink2'. These destinations need to be specified because with the network of paths, the option 'Continue' for *Entity Destination Type* could cause entities to continually loop around the paths.

Adding Nodes, Paths and a Bidirectional Path

- Place a BasicNode1 to the right of Source1/Sink1. Place a second BasicNode2 to the left of Source2/Sink2.
- Using Paths, connect Source1 to BasicNode1 and Source2 to BasicNode2. Then connect BasicNode1 to Sink1 and BasicNode2 to Sink2. For the Paths connecting the Sources to the BasicNodes, set Allow Passing to 'False'.
- Add a path connecting BasicNode1 and BasicNode2. Change the path's Type to 'Bidirectional', Allow Passing to 'False' and Speed Limit to '1'.

Positioning the Nodes

- Place 6 basic nodes in the middle of the Facility window between the Source1/Sink2 and Sink1/Source2. BasicNode1 and BasicNode2 are closest to the Sources/Sinks, while two are near each other about 1/3 of the way between BasicNode1 and BasicNode2. The other two are near each other about 2/3 of the way between BasicNode1 and BasicNode2. See the diagram below for placement.



Placing the Paths

- Use a unidirectional path to connect Source1 to BasicNode1. Do the same to connect Source2 to BasicNode2.
- Use a unidirectional path to connect BasicNode1 to Sink2 and to connect BasicNode2 to Sink1.
- Use bidirectional paths to connect BasicNode1 to BasicNode3, BasicNode4 to BasicNode5, and BasicNode6 to BasicNode2. This is done by connecting the nodes with a path and changing the *Type* property to 'Bidirectional'.
- Use unidirectional paths to connect the BasicNode3 to BasicNode4 and BasicNode4 back to BasicNode3 (in two different directions). Do the same with BasicNode5 and BasicNode6. This will provide paths for the entities to reside on while entities moving in the opposite direction are on the bidirectional paths coming into the bypass.
- Change the *Allow Passing* property to 'False' on all paths so that the entities don't stack up on the paths.
- For the bidirectional links, click on the '+' by the *Type* property. Change the *Traffic Direction Rule* on all three bidirectional links to 'Prefer Desired Direction'. This will allow entities that are waiting to enter a bidirectional link to move onto the links more efficiently. See notes below.

Embellishments:

Changing the Initial Traveler Capacity on the unidirectional paths within the bypasses will limit the amount of traffic in those areas; however, if the rate of entities through these areas is too great, you could have deadlocking again within the triangles. See the example MiningExample.spfx for another version of bypass modeling.

Notes:

Please note that bypass areas have to be size to support the maximum traffic that could pass. If they are not adequately sized for the number of incoming entities, deadlocking can still occur.

With bidirectional links, the *Traffic Direction Rule* property may be used to determine how traffic flows from multiple directions onto a link. 'First In Entry Queue' means the first ranked entity in the link's EntryQueue will always be selected to enter the link next. The queue's ranking rule might be a FIFO, LIFO, SVF, or LVF. 'Prefer Desired Direction' means an entity can only get onto the link in a non-desired direction if nobody is waiting to go in the preferred direction. The preferred direction is changed each time an entity moves onto the link and starts to move. Therefore, if an entity is traveling right to left on the link as two additional entities enter nodes at opposite ends, the entity on the right can start moving from right to left immediately (since the direction of traffic at that time is right to left). Refer to SimBit [PathSelectionRule](#) for an example.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ChangingQueuesWhenServerFails - SimBit

Problem:

A system consists of two parallel servers. If one of the servers fails, then entities waiting to get processed at that server are redirected to the other server if that server is operational.

Categories:

Buffering

Key Concepts:

Buffer Logic, Balking, Reneging, Jockeying, Reliability Logic, Failure, Processing Count Based

Technical Approach:

Balking and reneging behavior is specified for the input buffer of each server. An arriving entity balks entering the input buffer of a server if the server is currently failed and the other server is operational. The balked entity is then redirected to the operational server. Also, if a server fails, then any entities already waiting in its input buffer are automatically removed from the queue (reneged) and redirected to the other server if that server is operational.

Details for Building the Model:

Facility Window Setup

- Add a Source, two Servers, and a Sink from the Standard Library. Connect the Source to each of the Servers and the Servers to the Sink using Paths. Place a BasicNode next to each Server. Connect the BasicNode at Server1 to the input node at Server2 and the BasicNode at Server2 to the input node at Server1 using Paths.
- For Server1, go to Reliability Logic and specify the server's *Failure Type* as 'Processing Count Based', the *Count Between Failures* as 'Random.Uniform(20,30)' and the *Time To Repair* as 'Random.Uniform(5,10)' minutes.
- For Server2, go to Reliability Logic and specify the server's *Failure Type* as 'Processing Count Based', the *Count Between Failures* as 'Random.Uniform(30,45)' and the *Time To Repair* as 'Random.Triangular(5,10,15)' minutes.
- For Server1, go to Buffer Logic -> Input Buffer -> Balking & Reneging Options and specify the *Balk Decision Type* as 'Conditional'. Specify the *Balk Condition Or Probability* as 'Server1.Failure.Active && !Server2.Failure.Active'. This will cause an arriving entity to balk entering the input buffer of Server1 if that server is failed but Server2 is operational. Send balked entities to the *Balk Node Name* that redirects them to Server2. Then open the *Reneg Triggers* repeat group and add event-based triggers that will remove entities waiting in Server1's input buffer and redirect them to Server2 if Server1 is failed but Server2 is operational. Refer to the example model for guidance on specifying the renege trigger properties.
- For Server2, go to Buffer Logic -> Input Buffer -> Balking & Reneging Options and specify similar balking and reneging behavior that was described above for the input buffer of Server1. Refer to the example model for guidance on specifying the renege trigger properties.
- Place a ModelEntity instance from the Project Library and add an additional symbol from the Symbols ribbon. Select the new symbol (value 1) and change the color from the default green to red.
- For both Server1 and Server2, go to State Assignments and open the *On Balking* and *On Reneging* repeat groups. Assign the ModelEntity.Picture of an entity that is balking at entering or reneging from the input buffer of the server to the value '1' (i.e., the red colored symbol). Refer to the example model for any further guidance on specifying these animation related state assignments.

Defining the Node Selection List from the Source

- Within the Definitions window, select the Lists panel. Create a Node list named 'ServerNodeList' and add 'Input@Server1' and 'Input@Server2' to the list.
- Go back to the Facility window and change the Source's *Interarrival Time* to 'Random.Exponential(.15)' and *Maximum Arrivals* (Stopping Conditions section) to '200'.
- For the Source's output node, change the *Entity Destination Type* to 'Select From List', the *Node List Name* to 'ServerNodeList', and the *Selection Condition* to '!Candidate.Server.Failure.Active'. This will send new entities created by the Source to the first of the parallel servers that is not currently failed.

See also:

ServerQueueWithBalkingAndReneging.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CheckingNextEntityAheadOnLink - SimBit

Problem:

I want to know the traffic that is in-front of an entity on a link.

Categories:

Add-On Process Logic

Key Concepts:

Add-On Process, Decide Step, Delay Step, Floor Label, NextEntityAheadOnLink, Path, TransferNode

Technical Approach:

A process is created which triggers when an entity enters a path. In the process, entity is delayed to maintain the desired fixed distance from the next entity ahead of it on the link. A floor label is used to display information on the ModelEntity.

Details for Building the Model:

Simple System Setup

- Add a Source, TransferNode and Sink from the Standard Library. Connect the Source to TransferNode and TransferNode to Sink using Paths.
- In the Properties window for Path2, under Add-On Process Triggers, click *Entered* and select '[Create New]'. This will create a new process named Path2_Entered within the Processes window.
- Place an instance of ModelEntity from the Project Library named 'DefaultEntity'. Highlight the Entity and attach a Floor Label from the Symbols ribbon. Edit the Floor Label from the Appearance ribbon. Add 'Entity {ID} is following {NextEntityAheadOnLink} at {NetworkDistanceTo.NextEntityAheadOnLink}' as *Label Text*.

Defining the Process

- Under Path2_Entered, add a Decide step into the process. Set the *Decide Type* to 'ConditionBased' and *Expression* to 'ModelEntity.NetworkDistanceTo.NextEntityAheadOnLink < 10 & & ModelEntity.NetworkDistanceTo.NextEntityAheadOnLink != 0'. This will evaluate the entity, if any, in front of the current entity and will return a 'True' if the distance is less than 10 (meters, which is the default distance units).
- From the Decide step's True exit, add a Delay Step with the *Delay Time* of '2' and *Units* of 'Seconds'. The Delay step should then connect back to the Decide step for re-evaluation.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ChooseAlternateSequence - SimBit

Problem:

I have a model that normally process parts in the sequence Server1, Server2, Server3, Inspect, Sink but based on the result of the inspection, it sometimes must be diverted to Adjust and then restart at Server2.

Categories:

Add-On Process Logic, Sequence Tables

Key Concepts:

Sequence Table, By Sequence, Entity Destination Type, Add-On Process, SetRow Step

Assumption:

The source produces one type of entity. The entity will follow a primary sequence of: Source – Server1 – Server2 – Server3 – Inspect – Sink.

But if it fails inspection (30% of the time) it will divert to an alternate sequence of Adjust and then resume its normal sequence from Server2.

Technical Approach:

Separate Sequence Tables are created for the entity to follow its normal sequence and sometimes divert to another route. We will create one main sequence table which is identical to the first sequence specified above. We will create an additional sequence to switch to if the entity fails and has to go to the Adjust server. In order to make decisions at the inspection, add-on processes will be created.

Details for Building the Model:

Simple System Setup

- Place a Source and a Sink in the Facility window. In between them, place four Servers that are relatively next to each other and place Server5 parallel to Server4.
- Connect an Output Node of Source to an Input Node of Server1, an Output Node of Server1 to an Input Node of Server2, an Output Node of Server2 to an Input Node of Server3, an Output Node of Server3 to an Input Node of Server4, and an Output Node of Server4 to the Sink with a Path. An Output Node of Server 4 to an Input Node of Server5 and an Output Node of Server 5 to an Input Node of Server2 must be connected with a Path also.
- Rename Server4 to Inspect and Server5 to Adjust. Place one Model Entity object in a Facility window. Rename Model Entity to 'Part'.

Setting up the Sequence Tables

- In the Data Window, select the Tables panel and add two Sequence Tables. Name them 'NormalSequence' and 'AdjustmentSequence'.
- For the NormalSequence, set the following sequence: Input@Server1 – Input@Server2 – Input@Server3 – Input@Inspect – Input@Sink1
- For the AdjustmentSequence, set the sequence: Input@Adjust.

Setting up Add-On Process

- In the Facility window, by clicking Inspect object and under Add-On Process Triggers by double clicking *After Processing*, a process will be created by default in a Processes window.
- Repeat these steps with Adjust object in order to create *After Processing* Add-On Process as well.

Setting up Process Logic

- In a Processes Window in the Inspect_AfterProcessing process, place a Decide step to represent the inspection and the likelihood of passing. Change its *Decide Type* is going to be 'ProbabilityBased' with *Expression* set to '0.7'. The True exit represents "Passed" and requires no action – we will leave that unchanged. The False exit represents "Failed" and requires using SetRow step to change to an alternate Sequence (set *Table Name* to 'AdjustmentSequence' and leave *Row Number* at its default (the start of the sequence)). Here we have also added an Assign step to set the

ModelEntity.Picture to '1' so we can identify it as a red failed part in the model.

- In the Adjust_AfterProcessing process, we need to restore the part to its normal look and processing. Place SetRow step with *Table Name* of 'NormalSequence'. Set the *Row Number* to '2' to allow entity to resume with Server2 object and follow its normal sequence. Also add an Assign step to set the ModelEntity.Picture back to '0' so it will again be green.

Finalizing the Model

- Change the *Initial Sequence* of the Entity to 'NormalSequence'.
- Add Additional Symbol to the entity and color it red.
- Change the *Interarrival Time* on the Source to 'Random.Exponential(.3)' to give a more balanced system.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CombineMatchingMembers - SimBit

Problem:

I want to batch multiple entities together for subsequent processing. Only entities with identical state values can be batched together.

Categories:

Combining and Separating Entities

Key Concepts:

BatchMembers Queue, Combiner, Matching Rule, MemberInputBuffer, ModelEntity, ParentInputBuffer, Priority, Queue

Assumptions:

We will batch 4 parts together into each container and as many as 5 batches can be worked on concurrently.

Technical Approach:

Use the existing state named 'ModelEntity.Priority' and use an Add-On Process in the Source to assign it. Use a Combiner of capacity 5 to do the batching of 4 Parts into a Container.

Details for Building the Model:

Simple System Setup

- Add two Sources, a Combiner, a Server and a Sink to the Facility Window.
- Connect the Sources to the Combiner, the Combiner to the Server and Server to the Sink with Paths.

Adding Multiple Entity Types

- Drag two ModelEntity objects from the Project Library into the Facility Window and change the *Name* of each. Specify two different entity types, 'Parts' and 'Containers'.
- Designate each Source object to generate a different part type by changing the *Entity Type* property on one to be 'Parts' and the other to be 'Containers'.

Changing the Entity Priority

- Within the 'Parts' Source object, enter the State Assignments, *Before Exiting* repeating editor and add a *State Variable Name* 'ModelEntity.Priority' with a *New Value* of 1 or 2 using the 'Random.Discrete(1, .5, 2, 1)' distribution function.

Using the Combiner

- The Combiner is batching four members (Parts) to one parent (Container) and only batches members together that have matching priority value (e.g. each batch will have either four entities with priority of 1 or four entities with priority of 2). This is done by specifying the *Batch Quantity* to '4', *Matching Rule* to 'Match Members' and *Member Match Expression* to 'ModelEntity.Priority'.
- As mentioned in Assumptions, the Combiner has a *Capacity* of '5', meaning up to 5 batches may be processing at a time, with *Processing Time* set to 'Random.Uniform(1,5)'.

Enhancing the Animation

- Move the queue that displays parents waiting for a match.
- Add a queue that displays members waiting for a match by selecting a Detached Queue from the Drawing tab and changing the *Queue State* to 'Combiner1.MemberInputBuffer.Contents'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CombineMultipleEntityTypeOntoPallets - SimBit

Problem:

Use a Combiner object to pack quantities of multiple entity types onto pallets.

Categories:

Combining and Separating Entities

Key Concepts:

Combiner, Batch Quantity, Matching Rule

Assumptions:

The following quantities and entity types are required to be packed onto each pallet:

Entity Type	Batch Quantity
A	1
B	3
C	2
D	1

Once the target batch quantities have been collected and placed on a pallet, a processing time of `Random.Triangular(1,2,3)` minutes is required to finish the packing process.

Technical Approach:

A Combiner object is used to collect the required batch quantities and attach the member entities to parent pallet entities.

Details for Building the Model:

Simple System Setup

- Place 5 Sources, 1 Combiner, and 1 Sink in the Facility window.
- Place 5 ModelEntity objects into the Facility window, one next to each Source.
- Use Paths to connect Source1 to the ParentInput@Combiner, the 4 other Sources to the MemberInput@Combiner, and the Combiner to the Sink.
- Rename the Entities: 'Pallet', 'A', 'B', 'C', and 'D'.
- Change *Entity Type* of Source1 to 'Pallet' and *Entity Type* of the other Sources to 'A', 'B', 'C', 'D' respectively, so each entity type is created by a different Source.
- For Source1 that is creating the pallet entities, specify the *Arrival Mode* as 'On Event', *Initial Number Entities* as '1', and the *Triggering Event Name* as 'Output@Combiner1.Exited'. This will cause the source to automatically create a new pallet whenever the previous pallet has left the combiner.
- Select the Pallet entity, and in the Symbol ribbon, click the Draw Queue drop down and select BatchMembers. Click near the Pallet entity, then click somewhere else to create the queue. Right-click to end queue building. Make it long enough to show 7 entities.
- Multi-select Entities A—D by holding Ctrl when selecting the entities. Then change the *Dynamic Label Text* to 'ModelEntity.EntityType.Name'.

Combiner Setup

- In the Batching Logic properties, specify the following *Batch Quantity* and *Matching Rule* information:

Batch Quantity	Matching Rule	Member Match Expression	Parent Match Expression
1	Match Members And Parent	ModelEntity.EntityType	A
3	Match Members And Parent	ModelEntity.EntityType	B
2	Match Members And Parent	ModelEntity.EntityType	C
1	Match Members And Parent	ModelEntity.EntityType	D

- Then, in the Process Logic properties, specify "Random.Triangular(.1,,3)" minutes as the *Processing Time*.

Embellishments:

Change the symbol or color of each Entity Type.

Change the arrival rates at the member entity Sources to see how that affects queue length in the member input buffer of the Combiner.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CombinerNode - SimBit

Problem:

I want to create a user-defined node that can batch entities based on entity priority.

Categories:

Building New Objects / Hierarchy, Combining and Separating Entities

Key Concepts:

Batch Step, BatchLogic Element, BatchMembers Queue, Expression Property, Decide Step, Discrete State, EndTransfer Step, Node Class, OnEntered, OnEnteredParking, Path, Park Step, Search Step, State Variable, Station Element, Transfer Step, UnPark Step

Technical Approach:

A node object is created that has process logic which stops entities at the node to batch them into the user-defined Batch Size and then sends them out through the node. The process logic first checks if the entity needs to be batched. If so, the entity is transferred to the ParkingStation until the batch is complete. If a parent already exists in the batch logic for that priority, the entity becomes a member. After all the members of the batch arrive, the batched entity is transferred from the ParkingStation back to the beginning of the node process. Since the entity is batched, the batched entity is transferred to the outbound link.

Details for Building the Model:

Creating the Node Object and Defining the Properties, States and Elements

- On Project Home ribbon, click the New Model dropdown list and select Node Class. A new Node will be created in the Navigation window.
- In the Node Model Properties (right click on the object and select Properties), change the *Model Name* to 'CombinerNode'.
- In the Definitions Window, select Properties window. From the Standard Property list, select Expression. In the Properties window, change the *Name* to 'BatchSize', set *Default Value* as '2', *Display Name* to 'Batch Size', *Category Name* to 'Process Logic', and *Description* to 'Total number in group, including parent.'
- Click Properties (Inherited). Select CapacityType. Change *Visible* to 'False'. Also do this for InitialCapacity, WorkSchedule, RankingRule, and DynamicSelectionRule. Changing the visibility will hide these properties in the CombinerNode Properties window since we will not be using them.
- Click the States button on the Definitions panel, and select Real from the States ribbon to add a new state. Change the *Name* to 'Batched' and the *Initial State Value* to '0'.
- Select the Elements button from the Definitions panel, and add a BatchLogic element from the Elements ribbon. Set *Batch Quantity* to 'BatchSize - 1'. BatchSize typically represents the number of 'member' entities to batch with a single parent entity. Set *Matching Rule* to 'MatchMembersAndParent', *Member Match Expression* to 'ModelEntity.Priority', and *Parent Match Expression* to 'ModelEntity.Priority'.

Creating Process Logic for the CombinerNode

- In the Processes window, click Select Process and select OnEntered.
- Add the following steps in order to the OnEntered process: Decide, Park, Search, Decide, Batch, Assign and UnPark. Batch, Park, and UnPark are under All Steps.
- Select the first Decide Step; set *Decide Type* to 'ConditionBased' and *Expression* to 'Batched==0'. When all entities first enter the node, this condition will be true and they will continue out the True exit (you will see shortly when the condition will be false).
- The Park step is used so that entities will sit and graphically wait at the node until the batch is made (otherwise, they will be shown at the node on top of each other). No property changes are necessary for this step.
- Set the Search step *Collection Type* to 'QueueState', *Queue State Name* to 'BatchLogic1.ParentQueue', *Match Condition* to 'ModelEntity.Priority==Candidate.ModelEntity.Priority', *Search Expression* to '1'. This step is used to determine if there is already a 'parent' entity (one entity of that same priority value). The value of the Search Expression is returned in the Token.ReturnValue for the process. If no items are found (no parent yet), the return value will be 0.

- For the second Decide step, set *Decide Type* to 'ConditionBased' and *Expression* to 'Token.ReturnValue<1' to check the above returned information.
- Within the Batch step, set *Batch Logic Name* to 'BatchLogic1' and *Category* to 'Parent'.
- Set the Assign step *State Variable Name* to 'Batched' and *New Value* to '1'.
- The UnPark step is used once the batch is made to move the newly batched entity from the parking area of the station (the entity then goes back 'into' the node and goes through the OnEntered process, which is why we set the Batched variable to 1, so that newly batched entities will exit the False exit of the Decide step, as shown below).
- Add an Assign step to the False branch of the first Decide step. Set the *State Variable Name* to 'Batched' and *New Value* to '0'.
- Add a Transfer step after the Assign step; set *From* to 'CurrentNode' and *To* as 'OutboundLink'. This will cause only newly batched entities to leave the node on the outbound link.
- Add a Batch step to the False branch of the second Decide step. Set *Batch Logic Name* to 'BatchLogic1' and *Category* to 'Member'. All entities except the first entering 'parent' will go here and be batched with the parent entity.
- Click on Select Process on the Process ribbon and select OnEnteredParking. Add an EndTransfer step so that when the entities are parked, their transfer into the parking station is complete.

Simple System Setup

- Click Model in the Navigation window. Add two Sources and a Sink from Standard Library. From the Project library (below Standard and Flow Libraries on the left), place a CombinerNode in the Facility window. Connect the Sources to the CombinerNode and the CombinerNode to the Sink using Paths.
- Place two instances of ModelEntity from the Project Library. Select one Entity instance and in the Symbol ribbon, click Draw Queue and select BatchMembers. Draw a line near the Entity and do the same for the other Entity. Set one entity's *Initial Priority* to '1' and the other's *Initial Priority* to '2'. Select one entity, click the Color dropdown in the Symbols Ribbon, select Red, and click the top of the entity.
- Set Source1 *Entity Type* to 'DefaultEntity' and Source2 *Entity Type* to 'ModelEntity1'.
- Select the CombinerNode and set *Batch Size* to '5'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CombinerReleasingBatchEarly - SimBit

Problem:

Use Combiner to pack items onto pallets, but with a max wait time to collect the full batch quantity. A pallet is released early with a batch quantity less than the target quantity if the max wait time expires.

Categories:

Combining and Separating Entities

Key Concepts:

Combiner, Batch Quantity, Release Batch Early Triggers

Assumptions:

At the Combiner, a target quantity of 5 items is to be packed onto each pallet, but with a max wait time of 1 minute to collect the full batch quantity. If the max wait time expires, then a pallet is released early with whatever batch quantity was collected during the waiting time.

Technical Approach:

A Combiner object is used to collect the required batch quantities and attach the member entities to parent pallet entities.

Details for Building the Model:

Simple System Setup

- Place 2 Sources, 1 Combiner and 1 Sink in the Facility window.
- Place 2 ModelEntity objects into the Facility window, one next to each Source.
- Use Paths to connect Source1 to the ParentInput@Combiner1 and Source2 to MemberInput@Combiner1, as well as to connect the Combiner's output node to the input node of the Sink.
- Rename the ModelEntity objects: 'Pallet' and 'Item'.
- Change *Entity Type* of Source1 to 'Pallet' and *Entity Type* of Source2 to 'Item'.
- For Source1 that is creating the pallet entities, specify the *Arrival Mode* as 'On Event', *Initial Number Entities* as '1', and the *Triggering Event Name* as 'Output@Combiner1.Exited'. This will cause the Source to automatically create a new pallet whenever the previous pallet has left the Combiner.
- Select the Pallet entity, and in the Symbols ribbon, click Apply Symbol and select the Pallet symbol to make the entity look like a pallet. Then click the Draw Queue drop down and select BatchMembers. Click near the Pallet entity, then click somewhere else to create the queue. Right-click to end queue building. Make it long enough to show 5 entities.

Combiner Setup

- In the Batching Logic properties, specify the *Batch Quantity* as '5'.
- Got to the Batching Logic -> Other Batching Options -> *Release Batch Early Triggers* repeat group and add a time-based trigger with a *Wait Duration* of '1' minute.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CombineThenSeparate - SimBit

Problem:

I have two different entity types entering the system. I need to batch them together, process the combined batch and then un-batch them before exiting the system.

Categories:

Combining and Separating Entities

Key Concepts:

Combiner, Separation Mode, Separator, Split Batch

Assumptions:

Parts are batched, then processed through a Server and then separated.

Technical Approach:

Two sources create the two different entity types and then lead into a Combiner object that batches two entities together. The batched entities are processed by a Server and then sent to a Separator to be un-batched again before exiting the system.

Details for Building the Model:

Simple System Setup

- Place two Sources in the Facility Window. Also place a Combiner, a Server, a Separator and two Sinks into the Facility Window.
- Connect the Sources to the Combiner, the Combiner to the Server, the Server to the Separator and the Separator to each Sink using a Path.

Defining Multiple Entity Types

- Place two ModelEntity objects into the Facility Window by dragging them from Project Library window. Change the Name property to 'Entity1' and 'Entity2', respectively.
- Within each Source, change the Entity Type property to generate either 'Entity1' or 'Entity2'.

Using the Combiner

- The Combiner Batch Quantity should be set to '1' so that the entity at the Parent input node waits for 1 entity at the Member input node and then batches together.

Separating Entities

- The Separation Mode property in the Separator object is set to 'Split Batch' and the Split Quantity is set to '1'.

Embellishments:

The same requirements could be modeled without a Server object. The processing time of the Server from the above approach could instead be put in the Processing Time of the Combiner object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CommunicationBetweenObjects - SimBit

Problem:

One object wants to “watch” another’s behavior and react to it.

Categories:

Building New Objects / Hierarchy, Discrete States

Key Concepts:

Assign Step, ContinueProcess, Monitor, Process Triggering Event, Real State Variable, State Property, Status Label, Timer Element

Assumptions:

We have two fixed objects (the Subject and the Observer), each of which has a state (SubjectRate and ObserverRate). The Subject changes SubjectRate randomly. The Observer wants to change ObserverRate to match SubjectRate.

Technical Approach:

Both of the two fixed objects are process models (e.g. no facility logic or objects). The Observer is defined with a State Property, ObserverRate, that allows a State Property from another object, SubjectRate, to be passed in. The Observer then uses that state property for both setting up a monitor and getting the value for ObserverRate.

Details for Building the Model:

Defining the Subject

- Define a new fixed object *Subject* with a discrete state variable with *Name* ‘SubjectRate’.
- In the Definitions Window, within the Elements panel, add a Timer element with *Name* ‘Timer1’ to fire an event with *Interval* of ‘1’ hour.
- In the Processes Window, add a process with *Name* ‘SubjectUpdatesValue’ whose *Triggering Event* is ‘Timer1.Event’. Add an Assign step where the *State Variable Name* is ‘SubjectRate’ and the *New Value* is ‘Random.Triangular(0, 1, 2)’.

Defining the Observer

- Define a new fixed object *Observer* with a discrete state variable with *Name* ‘ObserverRate’.
- In the Definitions Tab of Observer, select the Properties panel and add a State Property with *Name* ‘SubjectRate’. This will allow the value of this property, when placed in a Facility Window of a model, to be assigned to the SubjectRate of the Subject.
- In the Definitions Window, within the Elements tab, add a Monitor element to fire a process when the *State Variable Name* ‘SubjectRate’ changes.
- In the Processes Window, add a process with *Name* ‘ObserverFollowsSubject’ whose *Triggering Event* is ‘Monitor1.Event’.
- Add an Assign step where the *State Variable Name* is ‘ObserverRate’ and the *New Value* is ‘SubjectRate’.

Defining a new model and using Subject and Observer objects

- Open a new model and place one Subject and one Observer within the Facility Window.
- In the properties window for the Observer, enter ‘Subject1.SubjectRate’ in the *SubjectRate* property.

Discussion:

If you watch the trace you will see that every hour the following sequence will occur:

- *Timer1* triggers an event which triggers the process *SubjectUpdatesValue*
- *SubjectRate* changes value which triggers the monitor on *SubjectRate*
- The monitor triggers the process *ObserverFollowsSubject*
- *ObserverRate* changes value to match *SubjectRate*

Send comments on this topic to [Support](#)

ConstraintLogic - SimBit

This SimBit project includes three models demonstrating the use of Constraint Logic Elements for modeling constraint-driven situations.

1. ResourceAvailabilityConstraintType_RoutingToParallelServers: Use Constraint Logic Element to check and hold an Entity until a Worker is available.
2. MaterialAvailabilityConstraintType_MultipleBOMs: Use Constraint Logic Element to restrict an Entity from progressing to a Material consumption location when the material is not available.
3. ConditionExpressionConstraintType_RoutingToFillers: Use Constraint Logic Element to restrict a ContainerEntity from filling at a Tank until the Tank is full.

Model 1: ResourceAvailabilityConstraintType RoutingToParallelServers

Problem:

I have a system that requires Entities to stay at their current nodes until the resources required for processing at the next destination are available.

Categories:

Buffering, Constraint Logic Element

Keywords:

Blocked, Blocked Destination Rule, Buffering, Buffer Logic, Candidate, Constraint Logic, Constraint Logic Element, Node List, Secondary Resources, Worker

Technical Approach:

Create a Constraint Logic Element to reevaluate the listed constraints and determine when the Entity can leave the Output Buffer.

Details for Building the Model:

Facility Window Setup

- Create two ModelEntities, one named 'OrderTypeA' and another named 'OrderTypeB'.
- Place two Sources, two Servers, and two Sinks to form parallel lines, and. Connect each Source to its Server, and each Server to its Sink with Paths.
- Rename one Source to 'SourceTypeA' and change its *Entity Type* property to 'OrderTypeA'. Change the *Name* of its connected Server to 'WorkTableA'.
- Rename the second Source to 'SourceTypeB' and change its *Entity Type* property to 'OrderTypeB'. Change the *Name* of its connected Server to 'WorkTableB'.
- Place three basic nodes, one next to each server and one off to one side for the Worker's *Initial Node*. Change the *Name* of the first two nodes to 'WorkLocationA' and 'WorkLocationB', and the *Name* of the last node to 'WorkerHome'.
- Place one Worker. Change the Worker's *Park While Busy* property to 'True', its *Initial Node* property to 'WorkerHome', its *Idle Action* property to 'Park At Home', and its *Initial Number In System* to '3'.
- Change the *Interarrival Time* property on each Source to 'Random.Exponential(0.75)'.
- On each Server, set the *Initial Capacity* to '2', the *Processing Time* to '1.5', and the *Input Buffer Capacity* to '0'.
- Under the For Processing Subgroup in the Secondary Resources Property Group of the Server, change the *Repeat Group* property to 'False'. Set the *Resource Name* property to 'Worker1', the *Request Move Property* to 'To Node', and the *Destination Node* property to the name of the closest Work Location Node, 'WorkLocation1' or 'WorkLocation2'.

Constraint Logic Element and Node List Setup

- In the Element View of the Definitions tab, select the Elements ribbon and create one Constraint Logic Element.
- Change the *Name* of the Constraint Logic Element to 'WorkerAvailable'.
- In the 'WorkerAvailable' Constraint Logic Element,
- Open the *Constraints* Repeating Property Editor
- Click Add once to create one new constraint
- Select the constraint and change the *Constraint Type* to 'ResourceAvailability' and the *Resource Name* to 'Worker1'.
- In the List View of the Definitions tab, click Node twice in the List Data ribbon to create two new Node Lists.
- In one list enter 'Input@Server1' and in the other enter 'Input@Server2'.

Applying the Constraint Logic Element

- In the Output Node at one Source, change the *Entity Destination Type* to 'Select From List' and change the *Node List Name* to the name of the corresponding Node List for that Source, 'NodeList1' or 'NodeList2'.
- At the same Node, expand the Other Routing Out Property Subgroup in the Routing Logic Property Group and add the 'WorkerAvailable' Constraint Logic Element.
- Complete the same steps at the other Source's Output Node.

Model 2: MaterialAvailabilityConstraintType MultipleBOMs

Problem:

I have a system that requires only Entities whose materials are available to enter the Processing Station of my Server.

Categories:

Buffering, Materials, Server, Constraint Logic Element

Keywords:

Bill of Materials, Constraint Logic Element, Constraint Logic, Consume Step, Data Table, Input Buffer, Material Element, Processing

Technical Approach:

Adding a Constraint Logic Element to the Server to ensure that Entities whose Materials are not available will not begin processing at the Server.

Details for Building the Model:

Facility Window Setup

- Place a Source, Server, and Sink. Rename the Server to 'Assemble' and connect the three objects by paths.
- Create two ModelEntities named 'Stool' and 'Table'. Change one of their colors to highlight the difference.

Material Elements Setup

- In the Element View in the Definitions tab, create three new Material Elements.
- Rename one to 'Leg' and set its *Initial Quantity* to '7'.
- Rename one to 'Tabletop' and set its *Initial Quantity* to '1'.
- Rename one to 'Seat' and set its *Initial Quantity* to '1'.
- Create two Bill Of Material Elements.
- Rename one to 'TableBOM'. Open the *Components* Repeating Property Editor and click Add twice to add two new components.

- On the first component, change the *Material Name* to 'Leg' and the *Quantity* to '4'.
- On the second component, change the *Material Name* to 'Tabletop' and the *Quantity* to '1'.
- Rename one to 'StoolBOM'. Open the *Components* Repeating Property Editor and click Add twice to add two new components.
- On the first component, change the *Material Name* to 'Leg' and the *Quantity* to '4'.
- On the second component, change the *Material Name* to 'Seat' and the *Quantity* to '1'.

Data Table Setup

- In the Tables View in the Data tab, create a new table names 'Products'.
- Create three columns, one Entity property column called 'Product', one Material Element Property column called 'BOM', and one Real property column called 'Mix'.
- In the 'Product' column, change the *Auto-set Table Row Reference* property to 'True' and make this column the primary key.
- In the 'BOM', change the *Auto-set Row Reference* property to 'True'.
- Enter the data below into the table.

Product	BOM	Mix
Stool	StoolBOM	1
Table	TableBOM	1

Constraint Logic Element Setup

- In the Elements View of the Definitions tab, add a new Constraint Element.
- Rename the Constraint Logic Element to 'MaterialsAvailable' and open the *Constraints* Repeating Property Editor.
- Add one new Constraint. Change the *Consumption Type* to 'BillOfMaterials', change the *Material Name* to 'Products.BOM', and set the *Constraint Scope* to 'RequestedItem'.

Incorporate Tables and Elements in Facility View

- On the Source, change the *Entity Type* property to 'Products.Product' and the *Maximum Arrivals* to '4'. In the Table Row Referencing property group, set the *Table Name* property to 'Products' and the *Row Number* to 'Products.Mix.RandomRow'.
- On the Server, in the Other Processing Options subgroup of the Process Logic property group, open the *Seize Constraint Logic* repeating property editor and click Add to add a new Constraint Logic Element. Change the *Constraint Logic Name* to 'MaterialsAvailable'.
- In the Add-On Process Triggers property group of the Server, double click *Processing* to create a new process. In the new process, add a Consume step, setting the *Consumption Type* to 'BillOfMaterials' and the *Material Name* to 'Products.BOM'.
- Add Floor Labels as shown in the model to display Material availability.

Model 3: ConditionExpressionConstraintType RoutingToFillers

Problem:

I have a system with a constraint which cannot be expressed as a Material Availability or Resource Availability constraint.

Categories:

Buffering, Constraint Logic Element, Flow Library

Keywords:

Buffer Logic, Candidate, Condition, Condition Expression, Constraint Logic Element, Constraint Logic, Container Entity, Filler, Flow Connector, Flow Node, Flow Source, Lists, Monitor, Node List, Select From List, Tank, Transfer Node

Technical Approach:

Apply Condition Expression constraints to a Transfer Node to create a pull system for Fillers.

Details for Building the Model:

Facility Window Setup

- From the Flow Library, place a ContainerEntity. Change its *Initial Volume Capacity* property to '1'.
- Place a Source object and change its *Entity Type* property to 'ContainerEntity1', its *Entities Per Arrival* property to '10', and its *Maximum Arrivals* property to '1'.
- From the Flow Library, place two Flow Sources, two Tanks, and two Fillers into two parallel lines. Connect the Flow Nodes of the Flow Sources to the Tanks, then from the Tanks to the Fillers.
- Change the *Initial Volume Capacity* of one Tank to '1' and the other to '1.5'. Change the *Input Buffer Capacity* of each Filler to '0'.
- Change the *Initial Maximum Flow* property of the Output Flow Nodes of the Flow Sources to '20'.
- Place one Sink object near the output Nodes of the Fillers.

Element Setup

- In the Elements View of the Definitions tab, create two new Monitor Elements.
- Name one 'Tank1FillStatusMonitor' and change its *State Variable Name* property to 'Tank1.FillStatus'.
- Name the remaining Monitor Element 'Tank2FillStatusMonitor' and change its *State Variable Name* property to 'Tank2.FillStatus'.
- In the Lists View of the Definitions tab, create a new Node List named 'FillerDestinationNodes'.
- Enter Nodes ContainerInput@Filler1 and ContainerInput@Filler2 into the List.

Constraint Logic Element Setup

- In the Elements View of the Definitions tab, add a new Constraint Element named 'FullTank'.
- Open the *Constraints* Repeating Property Editor and create two new Constraints.
- With the first constraint
 - Change its *Constraint Type* to 'ConditionExpression' and its *Condition Expression* to 'Tank1.FillStatus == Tank.List.FillStatusName.Full'.
 - Change its *Monitoring Event Name* property to 'Tank1FillStatusMonitor.Event', its *Constraint Scope* to 'RequestedItem', and its *Requested Item Condition* to 'Candidate.Node == ContainerInput@Filler1'.
- With the second constraint
 - Change its *Constraint Type* to 'ConditionExpression' and its *Condition Expression* to 'Tank2.FillStatus == Tank.List.FillStatusName.Full'.
 - Change its *Monitoring Event Name* property to 'Tank2FillStatusMonitor.Event', its *ConstraintScope* to 'RequestedItem' its *Requested Item Condition* to 'Candidate.Node == ContainerInput@Filler2'.

Note

Constraints contain three properties which customize how they are applied: *Constraint Scope*, *Requested Item Condition*, and *Skip Constraint If*. In this example, the constraints which verify that the appropriate Tank is full rely on the *Requested Item Condition* property. This property checks if the constraint is applicable for each requested item. In this instance, it is not desirable to have each constrained Entity check if both Tanks are full because each Entity only needs to visit one Filler. To avoid this, the *Requested Item Condition* is set to 'Candidate.Node == [DestinationNode]' such that the constraint is only applicable if the destination node is the Container Input Node for the Filler that the Entity is visiting.

The *Requested Item Condition* property defaults to referencing the destination Node; however, the *ObjectClass* of the Parent Object can also be used in the expression.

Applying the Constraint Logic Element

- Select the Output Node of the Source. Change its *Entity Destination Type* to 'Select From List' and its *Node List Name* to 'FillerDestinationNodes'.
- Expand the Other Routing Out Options property subgroup and open the Route Constraint Logic repeating property editor.
- Add one constraint and change its *Constraint Logic Name* to 'FullTank'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ControllingEntityOrientationWithQueue

This SimBit project includes two models providing use cases and illustrations of entity orientation in a queue.

1. **EntityQueueOrientation:** Demonstrates keeping the orientation of an entity on a Conveyor the same throughout the model with the use of a Storage element.
2. **VehicleQueueOrientation:** Demonstrates a discrete orientation of an entity to rotate continuously or fixed towards a specified point.

Model 1: EntityQueueOrientation

Problem:

I want to keep an entity facing a certain direction regardless of the directional heading of the Link.

Categories:

Animation, Add-On Process Logic, Custom Object

Keywords:

Add-On Process, Assign Step, Create Step, Insert Step, Storage Element, Entity Property, Object Reference State

Assumptions:

The Source (Arrivals) produces one type of entity (Base). On creation of a Base entity, a Microwave entity will be created and inserted into a Storage queue on the Base entity.

The Microwave is then tested for defects at the DefectTest Server. If a defect is found (10% of the time), the Microwave is sent to the DefectiveParts Sink to be discarded. If no defects are found (90% of the time), the Microwave is sent to the premium, standard, or no rush shipping area. Customers ask for premium shipping 40% of the time, standard shipping 48% of the time, and no rush shipping 12% of the time.

Technical Approach:

Two separate entities are used to demonstrate keeping the orientation of an entity the same on a Link. One entity will act as the "top" of the two and will be the visible throughout the model. The other entity will act as the base of the two and will be transparent throughout the model. On this base entity, a Storage element will be created. A Source will create the base entity. With the use of Processes, a top entity will be created when a base entity is created and placed on the Storage queue of the base entity. The queue's *Orientation* expressions determine the direction the top entity should orient along the X, Y, or Z axis.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server, two TransferNodes, and four Sinks in the Facility window. Rename the Source to 'Arrivals', the Server to 'DefectTest', and the four Sinks to 'DefectiveParts', 'PremiumShipping', 'StandardShipping', and 'NoRushShipping'. Place the three Shipping Sinks next to each other.
- Use Conveyers for all following connections. Connect the Output@Arrivals to the Input@DefectTest, the Output@DefectTest to the Input@DefectiveParts, the Output@DefectTest to TransferNode1, TransferNode1 to the Input Node@PremiumShipping, TransferNode1 to TransferNode2, TransferNode2 to the Input@StandardShipping, and TransferNode2 to the Input@NoRushShipping.
- Place one ModelEntity object in the Facility window. Rename ModelEntity to 'Microwave'. Right-click in the Navigation pane and click 'New Entity'. Rename this new entity to 'Base'. Drag a Base entity from the Project Library to the Facility window.
- On the Output@DefectTest and on TransferNode1 and TransferNode2, change the *Outbound Link Rule* to 'By Link Weight'. Change the *Selection Weight* of the following conveyers to the following values:
 - Output@DefectTest to the Input@DefectiveParts to '0.1'
 - Output@DefectTest to TransferNode1 to '0.9'

- TransferNode1 to the Input@PremiumShipping to '0.4'
- TransferNode1 to TransferNode2 to '0.6'
- TransferNode2 to the Input@StandardShipping '0.8'
- TransferNode2 to the Input@NoRushShipping '0.2'

Setting Up the Entity Definitions

- In the Navigation pane, select the Base entity and navigate to the Definitions tab. In the Elements view, add a Storage element by clicking on Storage in the General section. Navigate to the Properties view and add an Entity Property by clicking on the Object Reference drop down and selecting Entity. Name this property 'Base_TopType'.
- In the Navigation pane, select ModelEntity and select the Definitions tab. In the States view, add an Object Reference State Variable by clicking on the Object Reference drop down and selecting Object. Name this state variable 'MyBottom'.

Setting Up Process Logic

- In the Navigation pane, select the Base entity. In the Processes tab, click the Select Process drop down and select OnCreated. Drag a Create step to the right of the Assign step and set *Entity Type* to 'Base_TopType'. Drag an Insert step to the right of this Assign step and set *Queue State Name* to 'Storage1.Queue'.
- Navigate back to the Facility view of the model. Click on the Base1 entity and click on Draw Queue in the Attached Animation ribbon. Draw a horizontal line across the Base1 entity by clicking once to start the line, clicking again to set the ending point of the line, and right-clicking to exit drawing. Change this queue's *Queue State* to 'Storage1.Queue'. Click the Base1 entity and change the *Base_TopType* to 'Microwave'.

Finalizing the Model

- Set the Path Decorator of each Conveyor object to the Conveyor in the Link Tools > Edit Ribbon. Nodes where multiple Conveyers feed into may need to be moved slightly to avoid collision of Conveyers.
- Select the Arrivals Source and apply the Delivery symbol. Select the DefectTest Server and apply the Analyzer2 symbol. Select the DefectiveParts Sink and apply the Trash symbol. Select all three shipping Sinks and apply the Box2 symbol. Select the Microwave entity and apply the Oven3 symbol. Lines have been added to the Microwave symbol to help visualize the orientation of the entity, but this is optional.
- Select the Base entity definition in the Navigation pane and open the Definitions tab. In the External view select "Rectangle" in the Drawing ribbon. Draw a rectangle slightly bigger than the Microwave entity. Select the "Color" drop down and select More Colors. Set the *Opacity* to '0' and click OK. Click the Color button then click on the drawn rectangle to apply the new color setting. This will make the Base entity transparent so it will not be seen in the model. The Base entity determines the physical space it will take up in the model. Creating a Base with a similar size to the Microwave entity allows the entities to appear properly spaced on the Links.

Model 2: VehicleQueueOrientation

Problem:

I want to animate a Transporter that shows additional moving parts. This Transporter should display riders that will load and unload from the Vehicle but also exhibit a dynamic component that will stay with the Transporter. This addition will change its orientation either by spinning continuously or spinning to orient towards a fixed point.

Categories:

Animation, Add-On Process Logic, Custom Object, Level States, Vehicles

Keywords:

Add-On Process, Assign Step, Create Step, Transfer Step, Station Element, Entity Property, Object Reference State, Subclass, Monitor

Assumptions:

The Source (Loading) produces one type of entity (IceCream). Ten IceCream entities are created per arrival at the Source. All ten of the entities then ride on the Transporter (IceCreamTruck) to the Sales Sink where all ten IceCream entities are sold and leave the system. The IceCreamTruck takes five seconds to load all ten IceCream entities and ten seconds to unload all ten IceCream entities.

Technical Approach:

We will use one entity to demonstrate a discrete orientation of an entity on a queue. This entity will be transferred to one of two Stations on top of an ice cream truck. One Station will be used to have the entity on top spin continuously. The other Station will be used to have the entity always be oriented to a fixed point, in this case the origin. Other entities will load on a Transporter, ride it, and unload at a Sink.

Details for Building the Model:

Simple System Setup

- Place a Source, and a Sink in the Facility window. Place the Source and the Sink horizontal to each other and spaced out. Rename the Source to 'Loading' and the Sink to 'Sales'.
- Connect the Output@Loading to the Input@Sales with a Path. Change the *Type* of the Path to 'Bidirectional'.
- Create two ModelEntity instances in the Facility window. Rename the ModelEntity objects to 'IceCream' and 'TopIceCreamCone'.
- In the Navigation pane, click the project. Click on the Subclass From Library drop down and select Vehicle. Rename this vehicle to 'IceCreamTruck'. In the Facility view, add an IceCreamTruck object from the Project Library.

Setting Up the Entity and Vehicle Definitions

- In the Navigation pane, select ModelEntity and select the Definitions tab. In the States view, add an Object Reference State Variable by clicking on the Object Reference drop down and selecting Object. Name this state variable 'MyBottom'. NOTE: If EntityQueueOrientation (Model 1) has been completed, then this step has been implemented already.
- In the Navigation pane, select IceCreamTruck and select the Definitions tab. In the Lists tab, select String from the Create section. Name this 'RuleSelection'. Below set one entry to 'FixedPoint' and another to 'ContinuousRotation'. Set the *Category* for each property to 'Ice Cream Truck'. In the Elements view, add two Station elements by clicking on Station in the General section. Name one Station element to 'ContinuousStation' and the other to 'FixedStation'. Move to the Properties pane and add an Entity property by clicking on the Object Reference drop down and selecting Entity. Name this property 'Top_Type'. Add a List property by clicking on the Standard Property drop down and select List. Name this List property 'IceCreamConeOrientationRule'. Set the List's *Name* to 'RuleSelection'. Move to the States view and add an Object Reference State Variable by clicking on the Object Reference drop down and selecting Object. Name this state variable 'Myself'. Add a Level State Variable by selecting 'Level' in the Continuous section and name this 'Degrees'. Set the *Initial Rate Value* to '10000'. Navigate back to the Elements View and add a Monitor element. Set the State Variable's *Name* to 'Degrees', the *Monitor Type* to 'CrossingStateChange', and the *Initial Threshold Value* to '360'. Next, double-click on the Triggered Process Name property. Drag an Assign step into the Process and set the *State Variable Name* to 'Degrees'.

Setting Up Process Logic

- In the Navigation pane, select the IceCreamTruck Transporter definition. In the Processes tab, click the Select Process drop down and select OnRunInitialized. Right-click this process and select Override. Drag an Assign step at the beginning and set *State Variable Name* to 'Myself' and *New Value* to 'Object'. Drag a Create step to the right of the Assign step and set *Entity Type* to 'Top_Type'. Drag an Assign step below the Create step and set its *State Variable Name* to 'ModelEntity.MyBottom' and *New Value* to 'Myself'. Drag a Decide step to the right of the Assign step and set its *Condition Or Probability* to 'IceCreamConeOrientedRule'. Add a Transfer step after the 'True' branch of this Decide step and set its *To* property to 'Station' and its *Station Name* to 'ContinuousStation'. Place a Transfer step after the 'False' branch of the Decide step and set its *To* property to 'Station' and its *Station Name* to 'FixedStation'. Drag all loose End branches into the Execute process.
- Navigate back to the Facility view of the model. Select the IceCreamTruck1 Vehicle and click on Draw Queue in the Attached Animation ribbon. Draw a horizontal line across the IceCreamTruck1 Vehicle by clicking once to start the line, clicking again to set the ending point of the line, and right-clicking to exit drawing. Change the queue's *Queue State* to 'FixedStation.Contents'. Set the *X Direction* to $-(\text{Candidate.ModelEntity.MyBottom.Location.X} - \text{Candidate.ModelEntity.MyBottom.Orientation.Direction.X} * 2)$, the *Y Direction* to $-(\text{Candidate.ModelEntity.MyBottom.Location.Y} - \text{Candidate.ModelEntity.MyBottom.Orientation.Direction.Y} * 2)$, and the *Z Direction* to $-(\text{Candidate.ModelEntity.MyBottom.Location.Z} - \text{Candidate.ModelEntity.MyBottom.Orientation.Direction.Z} * 2)$. This will make the Ice Cream Cone entity on the Ice Cream Truck to spin to always be oriented toward the origin. Move this queue to the top of the Vehicle by holding Shift and dragging the queue up. A better view of this can be seen in 3D by pressing '3'.
- Repeat the same process to create another Station, but set the *Queue State* to 'ContinuousState.Contents', the *X Direction* to $\text{Math.If}(\text{Degrees} == 90 \parallel \text{Degrees} == 270, 0, \text{Degrees} > 90 \&\& \text{Degrees} < 270, -1, 1)$, and the *Z Direction* to $\text{Math.If}(\text{Degrees} > 90 \&\& \text{Degrees} < 270, 1 / \text{Math.Tan}((\text{Degrees} - 90) / 180 * \text{Math.PI}),$

$\text{Math.Tan}(\text{Degrees} / 180 * \text{Math.PI}))'$. This will make the Ice Cream Cone entity on the Ice Cream Truck spin continuously. The rate of how fast the Ice Cream Cone spins can be changed by clicking on IceCreamTruck in the Navigation pane, selecting the Definitions tab, States view, and changing the *Initial Rate Value* on Degrees. Click the IceCreamTruck1 Vehicle and change the Top_Type to 'TopIceCreamCone'.

Finalizing the Model

- Select the Path between the Loading Source and the Sales Sink and select "Single Lane" from the Path Decorators.
- Select the IceCreamTruck1 Vehicle and apply the Panel symbol. Select the IceCream and TopIceCreamCone entities and apply an ice cream cone symbol. A symbol from the 3D Warehouse was used for this SimBit. Make the size of the IceCream entity small and the size of the TopIceCreamCone large.
- Click on IceCreamTruck1 and scroll down in the Properties pane to IceCreamOrientationRule. This can be set to Fixed Point if you would like the Ice Cream Cone on top of the Truck to spin to always be oriented to the origin. This can also be set to Continuous Rotation if you would like the Ice Cream Cone on top to spin continuously.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Conwip - SimBit

Problem:

I would like to model a CONWIP system (constant level of WIP). The system should begin with 10 entities in the system and as soon as one entity leaves the system, another should arrive.

Categories:

Buffering, Data Tables

Key Concepts:

Create Step, Data Table, Limit, Numeric Property, OnRun Initialized Process, Real State Variable, Search Step, Source, Status Label, Table Station Element Property, Transfer Step

Assumptions:

This model contains 5 servers and the initial 10 entities are placed evenly at each server (i.e. 2 entities at each Server to begin). When a new entity arrives, it will be randomly sent to one of the 5 servers.

Technical Approach:

Upon initialization, the model looks in a Data table to find any initial WIP that should exist in the system and It creates those entities and transfers them to the appropriate station. The Source is set to create an arrival whenever an entity enters the Sink.

Details for Building the Model:

Simple System Setup

- Place a Source and a Sink into the Facility window. Also place a Model Entity from the Project Library into the Facility window, which will be called DefaultEntity.
- Place 5 Servers in between the Server and Sink. These Servers are in parallel.
- Connect everything with Paths.
- Click on the Source object and set the *Arrival Mode* property to 'OnEvent'. Set the *Event Name* property to 'Input@Sink1.Entered'. This tells the Source object to only produce arrivals when the event Input@Sink1.Entered occurs. This creates a pull system where arrivals occur whenever an entity enters the Sink is about to exit the system.

Create Data Table

- The Data Table will contain the number of entities that should be in the system upon initialization and where they should be located.
- Go to the Data window and click the Add Data Table icon in the ribbon to add a new table.
- Add a column that is an Integer property by selecting Integer property from the Standard Property drop down in the ribbon. The column can be renamed to InitialWIP.
- Add another column that is a Station property by selecting Station from the Element Reference drop down in the ribbon. The column can be renamed to Station.
- Populate this table by putting the value of 2 in each row of the first column and then selecting Server1.InputBuffer for the first row, Server2.InputBuffer for the second row, etc – for the second column.

Add Process Logic

- From within the Processes window, create a new process by selecting OnRunInitialized from the Select Process drop down in the ribbon.
 - Place a Search step into this process, followed by a Create step in the Found segment of the Search Step, and finally, a Transfer step in the Created segment of the Create step.
 - In the Search step, set the *Collection Type* property to 'TableRows' and select the name of the Data Table that you created above. Within the Advanced Options section of the Search step, change the value '1' in the *Limit* property to 'Infinity'. This will allow the Search to find all rows in the table.
 - In the Create step, the *Create Type* property should be set to 'NewObject' and the *Object Instance Name* should be set to 'Default Entity'. The *Number of Objects* property should be set to 'InitialWIP.InitialWIP', where InitialWIP is the name of the Data Table and InitialWIP is the name of the Integer column in the Data Table.

This tells the Create Step to create the number of entities that are listed in the Integer column of the Data Table.

- In the Transfer step, set *From* to 'FreeSpace' (since entities are created in Free Space by default) and the *To* property to 'Station'. The *Station Name* property should be set to 'InitialWIP.Station', where InitialWIP is the name of the Data Table and Station is the name of the Station property in the table.

Embellishments:

If you would like to see the current number of entities in the system, you can add a Status Label to the Facility window and set the expression to DefaultEntity.Population.NumberInSystem.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CounterElement - SimBit

Counter Element

This SimBit project includes three models providing examples of using the Counter Element.

1. **CountNumberInQueueAndExpectedProcessingTime:** Demonstrates using the Counter Element to count types of entities in a queue and count the expected processing time of the entities waiting in the queue.
2. **CountNumberEnrouteRoutingLogic:** Demonstrates using the Counter Element to count the entities enroute to a location. Shows Routing Logic used to pick a destination based on the smallest Counter Element value.
3. **CountNumberEnrouteRoutingLogicEnhanced:** Demonstrates using the Counter Element to count the entities enroute to a location. Shows Routing Logic used to pick a destination based on the smallest Counter Element value and specific entity characteristics.

Model 1: CountNumberInQueueAndExpectedProcessingTime

Problem:

I want to count the number of each type of entity waiting at a Server. I also want to estimate the expected processing time for all the entities waiting at the Server.

Categories:

Element

Keywords:

Counter, Element

Assumptions:

There will be three different entity types and entities will arrive randomly to the system.

Technical Approach:

The entities will have a table reference, which provides information on their different processing times. Counter Elements will be used to count information about these entities at Input@Server1.

Details for Building the Model:

Facility Window: Objects and Properties

- See the model for approximate placement of all objects.
- Place a Source.
- Place three ModelEntity instances in the Facility next to the Source.
- Rename these entities "A", "B", and "C".
- Change the colors of these entities with the Color button in the Symbols ribbon to the following:
 - A will remain default green.
 - B is red.
 - C is yellow.
- Place a Server.
- Place a Sink.
- Create a Path and connect Output@Source1 to Input@Server1.

- Create a Path and connect Output@Server1 to Input@Sink1.

Data Window: Tables

- Create a new Data Table with the 'Add Table' button.
- Name the Table "Entity Table" and create the following columns:
- Entity Property named "Entity"
- *Auto-set Table Row Reference* = 'True'
- Real Property named "Mix"
- Expression Property named "ProcessingTime"
- Populate the Data Table with the following information in each row:
- Row 1:
- *Entity* = 'A'
- *Mix* = '0.3'
- *ProcessingTime* = 'RandomExponential(1)'
- Row 2
- *Entity* = 'B'
- *Mix* = '0.45'
- *ProcessingTime* = 'RandomExponential(3)'
- Row 3
- *Entity* = 'C'
- *Mix* = '0.25'
- *ProcessingTime* = 'RandomExponential(2)'

Definitions Window: Elements

- Create a Counter Element named "CounterNumberWaiting".
- On CounterNumberWaiting, change the following properties:
- *Queue State Name* = 'Server1.AllocationQueue'
- *Key Expression* = 'ModelEntity.EntityType'
- Create a Counter Element named "CounterExpectedProcessTime".
- On CounterExpectedProcessTime, change the following properties:
- *Queue State Name* = 'Server1.AllocationQueue'
- *Key Expression* = 'ModelEntity.EntityType'
- *Value Expression* = 'Math.ExpectedValue(EntityTable.ProcessingTime)'

Facility Window: Properties

- On the Source1 instance, update the following properties:
- Entity Arrival Logic
- *Entity Type* = 'EntityTable.Entity'

- *Interarrival Time* = 'Random.Exponential(1)'
- Table Row Referencing
- *Table Name* = 'EntityTable'
- *Row Number* = 'EntityTable.Mix.RandomRow'
- On the Server1 instance, update the *Processing Time* property to 'EntityTable.ProcessingTime'.

Facility Window: Animation

- Create a Time Plot in the Facility by using the button in the Animation ribbon. Title the Plot "Number In Queue". Provide the following information in the *Expressions* Repeating Property Editor:
 - Item 1:
 - *Series* = 'A'
 - *Value* = 'CounterNumberWaiting.CountForKey(A)'
 - *Color* = 'Color.Green'
 - Item 2:
 - *Series* = 'B'
 - *Value* = 'CounterNumberWaiting.CountForKey(B)'
 - *Color* = 'Color.Red'
 - Item 3:
 - *Series* = 'C'
 - *Value* = 'CounterNumberWaiting.CountForKey(C)'
 - *Color* = 'Color.Goldenrod'
- Create a Time Plot in the Facility by using the button in the Animation ribbon. Title the Plot "Expected Total Processing Time in Queue". Provide the following information in the *Expressions* Repeating Property Editor:
 - Item 1:
 - *Value* = 'CounterExpectedProcessTime.CountForKey(A) + CounterExpectedProcessTime.CountForKey(B) + CounterExpectedProcessTime.CountForKey(C)'

Discussion:

As the model runs, watch how the Time Plots change with the entities in the Input Buffer of Server1.

The Time Plots in the Facility use the Counter Element's function 'CountForKey' to group certain counts by an attribute. In this case, the attribute we are counting by is the EntityType which distinguishes the different Entity instances.

The expected processing time is counted by changing the Counter Element's *Value Expression* to be the expected value of each entity's processing time. Now, the Counter Element is not incrementing by a static '1' count.

Notes:

The Counter Element will increment when the entity first enters the associated queue and decrement when it leaves the queue. If the entity's information that the Counter Element is using changes while the entity is waiting in the queue, the Counter will not automatically update.

Model 2: CountNumberEnrouteRoutingLogic

Problem:

I want to count the number of shipments and the number of parts on that shipment enroute to a destination. Subsequent shipments should choose their destination based on the location with the smallest number of parts already enroute to that

destination.

Categories:

Element

Keywords:

Counter, Element

Assumptions:

Shipments and parts will arrive randomly. Each shipment will randomly contain one to five parts. Shipments should choose a destination based on the smallest number of parts already enroute to that location.

Technical Approach:

Counter Elements will be used to keep track of the entities and batched entities traveling to each Server. Routing Logic will check the Counter Elements to determine the next destination to select.

Details for Building the Model:

Facility Window: Objects and Properties

- See the model for approximate placement of all objects.
- Place two Sources, one below the other.
- Place two ModelEntity instances, one behind each Source. Name the top ModelEntity "Shipment" and the bottom "Part".
- Color the Part instance a different color by using the Color button in the Symbols ribbon.
- With Shipment selected, go to Attached Animation tab and use the drop-down on the Queue button to select BatchMembers. Draw the queue by clicking the Facility near the Shipment entity. Move the mouse to draw the queue and click again to create the end of the queue. Right click to stop drawing the queue.
- Ensure the first Source is creating the *Entity Type* 'Shipment' and the second Source is creating the *Entity Type* 'Part'.
- Place a Combiner.
- Change the *Batch Quantity* property to 'Random.Discrete(1, 0.1, 2, 0.4, 3, 0.6, 4, 0.8, 5, 1)'.
- Place 3 Servers in a stack.
- Place a Sink.
- Place a Server.
- Create Connectors to connect the following Nodes:
 - Output@Source1 to ParentInput@Combiner1
 - Output@Source2 to MemberInput@Combiner1
- Create TimePaths to connect the following Nodes:
 - Output@Combiner1 to Input@Server1
 - Output@Combiner1 to Input@Server2
 - Output@Combiner1 to Input@Server3
- On all the TimePaths, change the *Travel Time* to '5' minutes.
- On the Output Nodes of the three Servers, change the following properties:
 - *Entity Destination Type* = 'Specific'
 - *Node Name* = 'Input@Sink1'

Definitions Window: Elements

- Create the following Counter Elements and update their properties:
- "CounterNumberShipmentEnroute_Server1"
- *Counter Type* = 'NumberEnroute'
- *Node Name* = 'Input@Server1'
- *Key Expression* = 'ModelEntity.EntityType'
- "CounterNumberShipmentEnroute_Server2"
- *Counter Type* = 'NumberEnroute'
- *Node Name* = 'Input@Server2'
- *Key Expression* = 'ModelEntity.EntityType'
- "CounterNumberShipmentEnroute_Server3"
- *Counter Type* = 'NumberEnroute'
- *Node Name* = 'Input@Server3'
- *Key Expression* = 'ModelEntity.EntityType'
- "CounterNumberPartEnroute_Server1"
- *Counter Type* = 'NumberEnroute'
- *Node Name* = 'Input@Server1'
- *Key Expression* = 'ModelEntity.EntityType'
- *Value Expression* = 'ModelEntity.BatchMembers.NumberWaiting'
- "CounterNumberPartEnroute_Server2"
- *Counter Type* = 'NumberEnroute'
- *Node Name* = 'Input@Server2'
- *Key Expression* = 'ModelEntity.EntityType'
- *Value Expression* = 'ModelEntity.BatchMembers.NumberWaiting'
- "CounterNumberPartEnroute_Server3"
- *Counter Type* = 'NumberEnroute'
- *Node Name* = 'Input@Server3'
- *Key Expression* = 'ModelEntity.EntityType'
- *Value Expression* = 'ModelEntity.BatchMembers.NumberWaiting'

Data Window: Tables

- Create a new Data Table with the 'Add Table' button.
- Name the Table "NodeCounterTable" and create the following columns:
- Node Property named "InputNodeName"
- *Auto-set Table Row Reference* = 'True'
- Expression Property named "ShipmentCount"

- Expression Property named "PartCount"
- Populate the Data Table with the following information in each row:
- Row 1
- *Input Node Name* = 'Input@Server1'
- *Shipment Count* = 'CounterNumberShipmentEnroute_Server1.CountForKey(Shipment)
- *Part Count* = 'CounterNumberPartEnroute_Server1.CountForKey(Shipment)'
- Row 2
- *Input Node Name* = 'Input@Server2'
- *Shipment Count* = 'CounterNumberShipmentEnroute_Server2.CountForKey(Shipment)
- *Part Count* = 'CounterNumberPartEnroute_Server2.CountForKey(Shipment)'
- Row 3
- *Input Node Name* = 'Input@Server3'
- *Shipment Count* = 'CounterNumberShipmentEnroute_Server3.CountForKey(Shipment)
- *Part Count* = 'CounterNumberPartEnroute_Server3.CountForKey(Shipment)'

Facility Window: Properties

- On the Output@Combiner1, update the following properties:
- Routing Logic
- *Entity Destination Type* = 'Select From List'
- *Node List Name* = 'NodeCounterTable.Node'
- *Selection Goal* = 'Smallest Value'
- *Value Expression* = 'NodeCounterTable.PartCount'

Facility Window: Animation

- In the Animation tab, add a Status Table to the Facility. Set the Status Table's *Table Name* to 'NodeCounterTable'.

Discussion:

When the entity leaves the Combiner, it will route using the list of Nodes which comes from the Data Table. In the Data Table, the Nodes remember their row, so when routing and considering each candidate Node location, the Node's Data Table reference can be used in the Routing Logic's *Selection Goals*. The *Selection Goal Value Expression* will look at the Candidate Node's associated Counter Element expression. The ShipmentPartCount Counter Element expression is counting the number of batched entities on each Shipment entity.

The model is set up to use the ShipmentPartCount in the Routing Logic. The other column in the Data Table, ShipmentCount, could be used instead to count the number of Shipments entities enroute.

Notes:

Data Tables can be used as a list in Simio, but they may not appear in the drop-down to automatically select the option. You may need to manually type the Data Table reference into the property.

Model 3: CountNumberEnrouteRoutingLogicEnhanced

Problem:

I want to count the number of shipments and the number of parts on that shipment enroute to a destination. Subsequent shipments should choose their destination based on the location with the smallest number of parts per specific Shipment Type already enroute to that destination.

Categories:

Element

Keywords:

Counter, Element

Assumptions:

Shipments and parts will arrive randomly. Each shipment will randomly contain one to five parts. Shipments can be type "A" or type "B" and should choose a destination based on the smallest number of parts per the same Shipment Type already enroute to that location.

Technical Approach:

A ModelEntity State Variable will keep track of the Shipment's Type. Counter Elements will be used to keep track of the entities and batched entities traveling to each Server and the Counter Count will be segmented by the Shipment Type. Routing Logic will check the Counter Elements to determine the next destination to select.

Details for Building the Model:

Initial Setup

- Use the "CountNumberEnrouteRoutingLogic" model as a starting point. You may right-click the model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

ModelEntity Definition

- Navigate to the ModelEntity in the Navigation pane. Go to Definitions tab > States.
- Add a String State Variable called "ShipmentType".

Facility Window: Properties

- Update the Source1 Properties:
- State Assignments
- *Before Exiting*
- *State Variable* = 'ModelEntity.ShipmentType'
- *New Value* = 'Random.Discrete("A", 0.5, "B", 1)'

Definitions Window: Elements

- Update all 6 Counter Element properties to use *Key Expression* = 'ModelEntity.ShipmentType'.

Data Window: Tables

- Update the Expressions in the NodeCounterTable so that every CountForKey expression now uses the 'ModelEntity.ShipmentType' as its key.
- Create a new Data Table with the 'Add Table' button.
- Name the Table "CounterStatusTable" and create the following columns:
- Node Property named "InputNodeName"
- Four Expression Properties
- "AShipmentCount"
- "AShipmentPartCount"
- "BShipmentCount"
- "BShipmentPartCount"

- Populate the CounterStatusTable with the following information in each row:
- Row 1
- *Input Node Name* = 'Input@Server1'
- *A Shipment Count* = 'CounterNumberShipmentEnroute_Server1.CountForKey("A")'
- *B Shipment Count* = 'CounterNumberShipmentEnroute_Server1.CountForKey("B")'
- *B Part Count* = 'CounterNumberPartEnroute_Server1.CountForKey("B")'
- Row 2
- *Input Node Name* = 'Input@Server2'
- *A Shipment Count* = 'CounterNumberShipmentEnroute_Server2.CountForKey("A")'
- *A Part Count* = 'CounterNumberPartEnroute_Server2.CountForKey("A")'
- *B Shipment Count* = 'CounterNumberShipmentEnroute_Server2.CountForKey("B")'
- *B Part Count* = 'CounterNumberPartEnroute_Server2.CountForKey("B")'
- Row 3
- *Input Node Name* = 'Input@Server3'
- *A Shipment Count* = 'CounterNumberShipmentEnroute_Server3.CountForKey("A")'
- *A Part Count* = 'CounterNumberPartEnroute_Server3.CountForKey("A")'
- *B Shipment Count* = 'CounterNumberShipmentEnroute_Server3.CountForKey("B")'
- *B Part Count* = 'CounterNumberPartEnroute_Server3.CountForKey("B")'

Facility Window: Animation

- Select the Shipment Entity Instance. In the Attached Animation ribbon, use the State Label button to draw an attached label with the *Expression* 'ShipmentType'.
- Change the Status Table in the Facility to look at the CounterStatusTable.

Discussion:

Now, the count for entities enroute to each Node is separated by the Shipment's State Variable called ShipmentType. When an entity routes at Output@Combiner1, the Counter's CountForKey expression will be filled in with the ShipmentType to dissect and get a count for only that grouping. Shipment Type "A" will check the count of other Shipment Type "A" enroute and not consider other Shipment Types. When routing, Shipment type "A" will choose the destination with the smallest number of parts among the other Shipment Type "A" also enroute. Shipment "A" and "B" will then be able to route based on *Selection Goal* values independent of the other Shipment Type counts.

Consider how the approaches from the models in this project could be used together. When routing and deciding a next location, the Counter Element could be used to check the expected working time at the next destination based on the entities in the queue, as seen with the first model, CounterNumerInQueueAndExpectedProcessingTime. The next destination could be picked based on the smallest expected working time.

The Counter Element can be used to help make decisions in a model based on checking count values. The Counter Element might also be used with Neural Network models to record pertinent system values as inputs for the Neural Network.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

CustomRoutingGroup - SimBit

Problem:

I would like arriving entities from two different locations to select between servers based on the closest server available. Entities from one of the sources have priority over the other, however, when a busy server becomes available.

Categories:

Decision Logic – Paths, Decision Logic - Processing, Entity Characteristics

Key Concepts:

Buffer Capacity, Entity Destination Type, InputBuffer, Lists, NodeList, Picture, Priority, RoutingGroup Element, State Assignments

Technical Approach:

In this example, we will have two Sources feeding entities into one of three Servers. The buffer capacities of the Servers are zero, so entities will wait at the Sources to proceed forward. To ensure that entities from the Source1 object have priority over those from Source2, we will create a custom RoutingGroup that both Sources will reference. The RoutingGroup will have a Route Request Ranking Rule of 'Smallest Value First' based on the entity priority value.

Details for Building the Model:

Simple System Setup

- Place two Sources, three Servers and a Sink. Connect each of the Sources to each of the Servers with Paths (6 paths), then connect each of the 3 Servers to the Sink.
- Place a ModelEntity from the Project Library into the Facility window. While the entity is highlighted, click on the Add Additional Symbol button to add a symbol. Change the symbol for the value 1 to be the color blue.

Create Node List and Routing Group Element

- From within the Definitions window, click on the Lists panel and add a Node type list (NodeList1) to include the three Servers – Input@Server1, Input@Server2, Input@Server3.
- Also from within the Definitions window, click on the Elements panel and add a RoutingGroup element named 'RoutingGroup1'. Specify the *Node List Name* to be 'NodeList1'. Under the Advanced Options, change the *Route Request Ranking Rule* to be 'Smallest Value First' and the *Route Request Ranking Expression* leave as 'Entity.Priority'. When multiple entities are waiting to move from this routing group, they will be ranked based on their priority instead of first in first out.

Editing the Source Objects

- Within each of the Sources, change the Interarrival Time to 'Random.Exponential(0.18)' minutes.
- Within Source1, add State Assignments of *State Variable Name* 'ModelEntity.Picture' to '0' and *State Variable Name* 'ModelEntity.Priority' to '1'.
- Within Source2, add State Assignments of *State Variable Name* 'ModelEntity.Picture' to '1' and *State Variable Name* 'ModelEntity.Priority' to '2'.
- Within the output TransferNode of each of the Sources, change the *Entity Destination Type* to 'Use Custom Routing Group'. Specify the *Routing Group Name* as 'RoutingGroup1'. Change the *Selection Goal* to 'Smallest Distance'.
- Note that by using the same RoutingGroup element for both Source objects, if an entity has to wait for a Server object to become available, the entities from BOTH Source objects are evaluated together and will be ranked based on their priority values. Therefore, waiting entities from Source1 (green color) will move to a Server before waiting entities at Source2 (blue color).

Editing the Server Objects

- Within each of the Servers, change the *Input Buffer* under Buffer Capacities to '0'. Therefore, no entities will wait at the Server for processing; they will wait at the Source objects. This is because of the buffer capacity in conjunction with the *Blocked Destination Rule* at the Source output nodes as 'Select Available Only'.

CustomUnbatching - SimBit

Problem:

I have two entity types (P1 and P2) batched to a parent entity type (Crate). I need to unbatch the P2 entities, change their dimensions and send them to a different sink.

Categories:

Add-On Process Logic, Combining and Separating Entities

Key Concepts:

Add-On Process, Assign Step, Combiner, Decide Step, Match Condition, Priority, Selection Weight, Sink, Source, TransferNode, Transfer Step, UnBatch Step

Technical Approach:

A mix of five P1 and P2 entities are batched to the Crate entity at the Combiner. When the batched entities reach TransferNode1, the P2 entities are unbatched, transferred to TransferNode1 from Free Space, and resized using Add-On Process logic. Then the P2 entities are sent to a separate Sink using Path Selection Weight.

Details for Building the Model:

Simple System Setup

- Place three Sources in the Facility window. Also place a Combiner, a TransferNode and two Sinks into the Facility window.
- Using Paths, connect Source1 to the ParentInput@Combiner1 node. Connect Source2 and Source3 to the MemberInput@Combiner1 node. Then connect the Combiner to the TransferNode and the TransferNode to both Sinks.
- Place three ModelEntity objects into the Facility window from Project Library window. Change the ModelEntity *Name* to 'Crate', 'P1' and 'P2', respectively.
- Select the Entity named 'Crate' and in the Symbol Ribbon, click Draw Queue dropdown and select BatchMembers. Then draw a line above the 'Crate' entity. Right click to stop drawing the queue. This line will show all entities batched to the 'Crate' entity.
- Change the P2 Entity *Initial Priority* to '2' in the ModelEntity properties.
- Change the *Entity Type* of Source1 to 'Crate', *Entity Type* of Source2 to 'P1' and Source3 to 'P2'. Change the *Interarrival Time* of Source1 to 'Random.Exponential (.6)' minutes.
- Set the Combiner *Batch Quantity* to '5'.
- Select the path between the TransferNode and the Sink1. Change its *Selection Weight* to 'ModelEntity.EntityType == P2'.
- On the path between the TransferNode and the Sink2, set its *Selection Weight* property to 'ModelEntity.EntityType != P2'.

UnBatching Entities Add-On Process Logic

- Create a process on the *Entered* Add-On Process Trigger in the TransferNode1 properties. This process will be triggered whenever an entity enters this TransferNode.
 - Place a Decide step. Set the *Decide Type* property to 'ConditionBased' and set the *Expression* to 'ModelEntity.BatchMembers == 0'. Crate entities will have 5 batched members and P2 entities will have 0 batched members once it is unbatched and re-enters the node.
 - On the True branch leaving the Decide step, place an Assign step. Set the *State Variable Name* to 'ModelEntity.Size.Width' and the *New Value* to 'ModelEntity.Size.Width * 2'.
 - On the False branch of the Decide step, place an UnBatch step. Set the *Desired Quantity* property to 'Entity.BatchMembers' and the *Match Condition* to 'Candidate.ModelEntity.Priority==2'.
 - In the Member segment leaving the UnBatch step, place a Transfer step. Set the *From* property to 'FreeSpace', the *To* property to 'Node' and the *Node Name* property to 'TransferNode1'. This will send all unbatched P2 entities to the same TransferNode and trigger this process again for re-sizing the entity.

Embellishments:

Select the path between the TransferNode and the Sink1 and change the *Allow Passing* property to 'False'. That will show all unbatched entities separated through the path instead of all of them in just one entity.

Change the path *Speed Limit* to '0.5' for the path between the Combiner and the TransferNode and the paths between the TransferNode and Sink1 and Sink2.

Note: The UnBatching Process Logic in this model can be replaced with a Separator object, but the goal of this SimBit is to show the features of the UnBatch step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DashboardReportInteractiveLogs - SimBit

Only Simio RPS Edition will have the full set of features demonstrated in this SimBit. The data that drives the creation of the Dashboards is only available in RPS Edition. For an example of a Dashboard available in all editions, please see the model and documentation for Dashboard Report Tallies SimBit, Dashboards Within Experiments SimBit, or the Hospital Emergency Department Example. For questions on editions and features and functionality, please contact sales@simio.com.

Problem:

For an existing project, I want to be able to graphically display the resource utilization as well as display a vehicle's pickup and drop off times in chart format. We will accomplish this by creating a Dashboard Report from an existing SimBit project VehicleVisitsServiceCenter.

Categories:

Dashboard Reports, Custom Statistics

Key Concepts:

Dashboard Reports, Interactive Logging, Log Observations, Resource State Log

Assumption:

This SimBit will illustrate how to create a Dashboard Report of log files generated within the VehicleVisitsServiceCenter project. In order to create a Dashboard Report for Interactive Logs, you must have the Simio RPS Edition.

Technical Approach:

A dashboard with a grid of vehicle pickup times and a pie chart of resource utilization percentages will be created.

Details for Building the Model:

System Setup

- Load the VehicleVisitsServiceCenter Simio project.
- For Server1, Server2 and MyVehicle1, under the Advanced Options change the *Log Resource Usage* property to 'True'.
- From the Run ribbon, under Advanced Options, make sure Enable Interactive Logging is turned on.
- Fast Forward the model to completion.

Creating the Dashboard Report

- Select Results, Logs, Resource Usage Log and Transporter Usage Log and make sure there is a log table that is populated with data. If not, verify the above steps.
- Select Results, Dashboard Reports, Dashboard Report Create.
- In the Add Dashboard window, select an name for the dashboard (ex: Pick Ups).
- Select Transporter Usage Log from the pull down list on the left of the window.
- Click on Grid in the Dashboard ribbon. This will create a Grid of items in the Transporter Usage Log and insert it into the dashboard.
- From the Transporter Usage Log, drag Resource over top of the Columns data item to the right. Follow that by dragging From Node, Start Time, To Node, End Time and Entity onto the Columns data items. This identifies which columns will appear on the grid.
- For the Start Time and End Time items, click on the right of the item to change the modifier to Date-Hour-Minute-Second.
- Right click in the Grid area, select Edit Names and change the Dashboard item name to Pick Ups.
- Now select Resource State Log from the pull down list on the left side of the window.
- Click on Pies in the Dashboard ribbon. This will create a pie chart of items in the Resource State Log and insert it into the dashboard.
- From the Resource State Log, drag Duration over the Value data item, State over the Argument data item and Resource over the Series data item.
- Right click in the Pie area, select Edit Names and change the Dashboard item name to Resource States.
- Select the Pie item and drag it below the Grid item.

-
- Click the Save button in the upper left corner of the ribbon to save the dashboard. Close the DashboardDesignerForm.
 - Various Dashboard Reports can be displayed in the results tab by selecting a dashboard from the Select pull down list.

Embellishments:

Add a Pivot grid using the data in the Resource State Log using the Duration as the Value data item, State as the Column data item and Resource as the Row data item. This will display a chart of the amount of time the resource was in each state.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DashboardReportTallies - SimBit

Problem:

For an existing project, I want to be able to graphically display the time in the system for each entity in a model, once the model has completed. We will accomplish this by creating a Dashboard Report from an existing SimBit TallyStatisticsInTables.

Categories:

Dashboard Reports, Custom Statistics

Key Concepts:

Arguments, Bar, Chart, Data Items, Data Source, Edit Filter, Ignore Empty Points, Line, Log Observations, Series, Stacked Bar, Tally Observation Log, Tally Statistic Element, TallyStatistic, Values, Dashboard Reports

Assumption:

The SimBit TallyStatisticsInTable must be opened with a Simio Professional or RPS license to create the Dashboards. This model uses 4 Tally Statistics to record the Time In System for passed and failed PartsA and PartsB.

Technical Approach:

Create a Dashboard with 3 charts:

1. The total number of each part type that passed and failed.
2. The number of parts that passed each hour for each part type and total. Filter out failed parts.
3. The Time in System for each part type over time.

Details for Building the Model:

System Setup

- Load the TallyStatisticsInTables SimBit.
- Under Definitions tab> Elements view> TallyStatistic PartA_Passed, change the Log Observations property (under Advanced Options) to 'True'. Do the same for the other 3 Tally Statistics.
- Fast Forward the model to completion.
- In Results window, Logs view, Tally Observation Log and make sure there is a log table that is populated with data. If not, verify that Log Observations is set to 'True' on all the Tally Statistics.

Creating the Dashboard Report

- In Results select Dashboard Reports view, and click on Dashboard Report in the Dashboards ribbon.
- In the Add Dashboard window, name the dashboard (ex:Tallies).
- **Chart1** (Top left)
 - In the DashboardDesignerForm, click on Chart within the Home ribbon. This will create a Chart dashboard item and insert it into the dashboard.
 - Select Tally Observation Log from the Data Source drop down list on the left of the window.
 - From the Tally Observation Log list of options, drag Value over top of the Value data item to the right.
 - Click on the down arrow that appears when you hover over Values (Sum) and select Count. Count counts the number of rows for each argument in a series.
 - Drag Time over top of the Argument data item.
 - Drag DataSource onto the Series data item.
 - Right Click in the Chart Item and select Edit Names. Change the Dashboard Item Name to 'Number Finished'.
 - To remove the X axis value go to Design ribbon and click X-Axis Settings (make sure the Chart item is still selected). Uncheck 'Show X-axis'.
 - To change the Y axis title click Y-Axis Setting in the Design ribbon. Make sure 'Show Title' is selected, then select 'Custom Text'. In the textbox change the title to 'Count'.
 - To add the point labels above each bar, select the graph icon next to Value(Count) in Data Items section. Go to the Point Label Options tab. In *Content* select 'Value' and select OK.
- **Chart2** (Top Right)

- Click on Chart within the Home ribbon. This will create a Chart dashboard item and insert it into the dashboard.
 - Select Tally Observation Log from the Data Source drop down list on the left of the window.
 - From the Tally Observation Log list of options, drag ObjectType over top of the Value data item to the right. It should automatically be set to Count. (Any column can be used for Count)
 - Drag Time over top of the Argument data item.
 - Drag DataSource onto the Series data item.
 - On the Time Argument, change the time from Year to Date-Hour in the Time dropdown (2nd More).
 - Click on the icon to the right of the ObjectType(Count) data item. Change the chart type to Stacked Bar in the Series Type tab.
 - To filter the chart to only passing parts, right click the chart and select Edit Filter. Click the plus sign to add a filter. Click 'enter a value' and open the dropdown. Select 'PartA_Passed'. Click the plus sign again and set the 'enter a value' to 'PartB_Passed'. In the top left, click on 'And' and select 'Or'.
 - Change the chart name in the Design ribbon > Edit Names. Change the Name to Number Passed.
 - To remove the Y-axis title go to Y-axis Settings in the Design ribbon and uncheck 'Show Title'.
 - Change the format of the x-axis values in the Time Argument dropdown (Data Items section) by selecting 'TimeOnly' in the Format (Default) dropdown.
- **Chart3** (Bottom)
 - Click on Chart within the Home ribbon. This will create a Chart dashboard item and insert it into the dashboard. To change the dashboard layout, hold the Title bar of the 2nd chart and move it to the right of only the 1st chart (a blue vertical line will appear on right).
 - Select Tally Observation Log from the Data Source drop down list on the left of the window.
 - Select the 3rd chart. From the Tally Observation Log list of options, drag Value over top of the Value data item to the right.
 - Click on the icon to the right of the Value(Sum) data item. Change the chart type to Line in the Series Type tab. In the Common Options tab check Ignore Empty Points. This option will draw the lines even if the Time stamps are different.
 - Drag Time over top of the Argument data item. Click on the dropdown on the Time data item (right side) and change the modifier from Year to Exact Date. Change the X-axis format in the Time Argument dropdown>Format>Hours>Short. Make sure the arrow to the left of the Time Data Item is pointing up so that our timeline is in the right order. (you can click once on the arrow to change from up to down or down to up).
 - Drag DataSource onto the Series data item.
 - Right Click in the Chart Item and select Edit Names. Change the Name to Time In System.
 - Change the Y axis title in the Design ribbon>Y-Axis Settings. Select Custom Text under Show Title and set the title to 'Hours In System'.
 - Click the Save button in the upper left corner of the ribbon to save the dashboard. Close the DashboardDesignerForm.
 - Various Dashboard Reports can be displayed in the results tab by selecting a dashboard from the Select pull down list.

Embellishments:

When you Filter a chart, only columns in DATA ITEMS sections can be used. If you want to filter on a column but not show it on the chart, drag the column name to HIDDEN DATA ITEMS.

Add a Range Filter to the dashboard. This will allow you to apply filtering to your dashboard items. A Range Filter displays a chart with selection thumbs that allow you to filter out values displayed on the argument axis.

DashboardsForSchedulingExamples - SimBit

Only Simio RPS Edition will have the full set of features demonstrated in this SimBit. The data that drives the creation of the Dashboards is only available in RPS Edition. For an example of a Dashboard available in all editions, please see the model and documentation for Dashboard Report Tallies SimBit, Dashboards Within Experiments SimBit, or the Hospital Emergency Department Example. For questions on editions and features and functionality, please contact sales@simio.com.

Problem:

Show results in graphical format using Dashboards with an RPS version. The model is based on the Scheduling Discrete Part Production Example.

Categories:

Custom Statistics, Dashboard Reports

Key Concepts:

Cards, Chart, Constraint Log, Cross-Data-Source-Filtering, Dashboard Reports, Grid, Interactive Logging, Log Observations, Material Log, Multiple Master Filter, Pies, Resource State Log, Resource Usage Log, Single Mater Filter

Assumptions:

Familiar with the Resource and Material Logs and what information is reported for each column. (Sometimes easier if you open another Simio with the logs shown because you see the logs when you open the Dashboard Editor.)

Technical Approach:

Create 3 Dashboards:

1. A dispatch list for each Resource and a graph with the percent breakdown in each state. Includes the ability to show a single or multiple Resources.
2. Graph the Stock Level of each Material over the simulation run, but only show 1 Material at a time.
3. A detailed list of the resources used (and when) for each Order and a graph of the different constraint types for each order. Includes the ability to show between 1 and all the orders.

Details for Building the Model:

Simple System Setup

- Load the Discrete Part Production Example.
- Make sure all Resources and Servers have *Log Resource Usage* set to 'True'.
- Turn on Interactive Logging in Run ribbon> Advanced Options> click **Enable Interactive Logging**.
- Fast Forward through the model. If the model already has results, then a preview of the dashboard items will show as you create the Dashboard.

Create the Dashboards

- Go to Results>Dashboard Reports View.

Dispatch List Dashboard

- In the Dashboard ribbon, click the *Dashboard Report* button in the Create section. In the Add Dashboard popup window, name the Dashboard 'Dispatch List Sim' and press OK.
- In the Dashboard Designer Form Window, on the Home ribbon click on the Cards, Pies and Grid buttons to add those items to the Dashboard. Move an item by holding the top title bar of the item and dragging to desired location in relation to other items (dark blue highlighted area).
- **Cards**
 - Select the Cards item on the dashboard. In the *Data Source* dropdown (left side of window) select the 'Resource Usage Log'.
 - Drag 'Duration (Hours)' to the Actual spot under DATA ITEMS (Cards section). Drag 'Resource' to *Series*.
 - Click the Gear symbol to the right of 'Duration (Hours)' in the DATA ITEMS section. In the Layout Options tab, select one of the templates and uncheck 'Actual Value'. This will show only the Resource Name on the card. The

example Dashboards use the 'Compact' template with the *Min Width* to '140' and *Max Width* to 'Auto'. Click OK.

- On the Data ribbon, select *Multiple Master Filter* and *Cross-Data-Source-Filtering*. Multiple Master Filter sets the Cards to control the data shown in the other dashboard items. Cross Data Source Filtering will apply the Master Filter to Dashboards Items that use a different Log but a full name of the data source field matches.
- Right click in the Cards item and select Edit Names. Change the Dashboard Item Name to 'Resource'.

- **Pies**

- Select the Pies item. In the *Data Source* dropdown select 'Resource State Log'.
- Drag 'Duration' to *Value*, 'State' to *Argument*, and 'Resource' to *Series*.
- On the Design ribbon, click Edit Names and change the Dashboard Item Name to 'Resource State'.

- **Grid**

- Select the Grid item and in the *Data Source* select 'Resource Usage Log'.
- Drag 'Resource' to *New Column*, then 'StartTime' to the 2nd *New Column*, then 'EndTime', 'Entity', and 'Duration'.
 - If the column was added in the wrong order, under DATA ITEMS select and hold the column and move to desired location. A yellow bar will appear between the other columns you are placing the selected column between.
- To change the format of the datetime columns, hover on the StartTime item under DATA ITEMS and open the right side drop down arrow. Open the 2nd More and select *Date-Hour-Minute*. Do the same for EndTime.
- On the Design ribbon, click Edit Names and change the Dashboard Item Name to 'Operations'.

- Click the Save button on the Home ribbon and Close the Dashboard creator window.

Materials Dashboard

- In the Dashboard ribbon, click the *Dashboard Report* button in the Create section. Name the Dashboard 'Materials Sim'.
- In the Dashboard Designer Form Window Home ribbon add a Cards item and Chart item. Move an item by holding the top title bar of the item and dragging to desired location in relation to other items. Resize each item's by moving the edges.
- **Cards**
 - Select the Cards item on the dashboard. In the *Data Source* dropdown (left side of window) select the 'Material Usage Log'.
 - Drag 'Quantity' to the *Actual* spot under DATA ITEMS (Cards section). Drag 'Material' to *Series*.
 - Click the Gear symbol to the right of 'Duration (Hours)' in the DATA ITEMS section. In the Layout Options tab, select one of the templates and uncheck 'Actual Value'. This will show only the Resource Name on the card. The example Dashboards use the 'Compact' template with the *Min Width* to '140' and *Max Width* to 'Auto'. Click OK.
 - On the Data ribbon, select *Single Master Filter*. Single Master Filter will filter the data in other items based on the Card selected. Only 1 card can be selected at a time.
 - Right click in the Cards item and select Edit Names. Change the Dashboard Item Name to 'Material'.
- **Chart**
 - Select the Chart item and in the *Data Source* select 'Material Usage Log'.
 - Drag 'Stock Level' to *Value*, 'Time' to *Argument*, and 'Material' to *Series*.
 - Under DATA ITEMS, on 'Stock Level' dropdown (hover on Stock Level, right side arrow), select Max. If there are multiple entries to the Material Log at the same time grouping for the same resource, the Graph will show the Max value.
 - On the graph symbol next to 'Stock Level' select the Step Line graph in Series Type tab.
 - On the Time Argument dropdown, change the format/grouping to Date-Hour-Minute-Second in the 2nd More dropdown.
 - Another option for datetime is to choose Exact Date from the Time dropdown for the grouping and choose a format from the Format dropdown below ExactDate.
 - On the Design ribbon, click Edit Names and change the Dashboard Item Name to 'Stock Level'.
- Click the Save button on the Home ribbon and Close the Dashboard creator window.

Order Details Dashboard

- In the Dashboard ribbon, click the *Dashboard Report* button in the Create section. In the Add Dashboard popup window, name the Dashboard 'Order Details Sim' and press OK.

- In the Dashboard Designer Form Window, Home ribbon click on the Cards, Chart, and Grid buttons to add those items to the Dashboard. Move an item by holding the top title bar of the item and dragging to desired location in relation to other items (dark blue highlighted area).
- **Cards**
 - Select the Cards item on the dashboard. In the *Data Source* dropdown (left side of window) select the 'Resource Usage Log'.
 - Drag 'Duration' to the Actual spot under DATA ITEMS (Cards section). Drag 'Entity' to *Series*.
 - Click the Gear symbol to the right of 'Duration (Hours)' in the DATA ITEMS section. In the Layout Options tab, select one of the templates and uncheck 'Actual Value'. This will show only the Resource Name on the card. The example Dashboards use the 'Compact' template with the *Min Width* to '140' and *Max Width* to 'Auto'. Click OK.
 - *If you do not want Duration to show on the Cards, you could have used any of the columns from the Resource Usage Log for the Actual Cards.
 - On the Data ribbon, select *Multiple Master Filter* and *Cross-Data-Source-Filtering*.
 - On the Design ribbon, click Edit Names and change the Dashboard Item Name to 'Orders'.
- **Chart**
 - Select the Chart item. In the *Data Source* dropdown select 'Constraint Log'.
 - Drag 'Duration' to *Value*, 'Constraint Type' and 'Facility Location' to *Argument*, and 'Entity' to *Series*. The Facility Location is always Model for this model, but does change for other scheduling models.
 - On the Design ribbon, click Edit Names and change the Dashboard Item Name to 'Constraints'.
- **Grid**
 - Select the Grid item and in the *Data Source* select 'Resource Usage Log'.
 - Drag 'Entity' to *New Column*, then 'StartTime', 'EndTime', 'Resource', and 'Duration'.
 - To change the format of the datetime columns, hover on the StartTime item under DATA ITEMS and open the right side drop down arrow. Open the 2nd More and select *Date-Hour-Minute*. Do the same for EndTime.
 - Right click in the Grid item and select Edit Names. Change the Dashboard Item Name to 'Operations'.
- Click the Save button on the Home ribbon and Close the Dashboard creator window.

Embellishments:

In the Dashboard window, Automatic updates should be selected (it is by default).

Change the Dashboard by going to Results>Dashboard View and selecting the Dashboard in Select dropdown and clicking Edit.

On the Dispatch List dashboard, when you select a Card, the Pie and Grid data is filtered to only the selected Resource. Hold the Ctrl key to select multiple Resource Cards. To reset the selection, right click the Cards items and select Clear Master Filter.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DashboardsWithinExperiments - SimBit

Problem:

Within an existing model, I'd like to experiment on the number of workers required and display experiment graphical information within a Dashboard Report. This will be accomplished by creating an Experiment and corresponding Dashboard Report from an existing SimBit ServerUsingTaskSequenceWithWorkers.

Categories:

Add-On Process Logic, Dashboard Reports, MultiTask Server, Worker

Key Concepts:

Active Symbol, Add-On Process, Assign Step, BasicNode, Dashboard Reports, Decide Step, Experiment, InputBuffer, ObjectList, Picture, Process, Process Type, Request Move, Server, Started Task, Status Label, Task Sequence, Worker

Assumptions:

The SimBit ServerUsingTaskSequenceWithWorkers must be opened with a Simio Professional or RPS license to create the Experiment Dashboards. The original SimBit was modified slightly to work with the controls in the experiment. Those changes are noted below. This model uses 3 Controls (model properties) and includes 4 Responses within the Experiment.

Technical Approach:

Create 3 Dashboards:

1. The average response result values across all scenarios. This will include all 4 responses on same graph.
2. The individual replication values and average value for each response and each scenario.
3. Pie charts showing resource states across all scenarios, with data from the experiment statistic summaries.

Details for Building the Model:

System Setup

- Load the ServerUsingTaskSequenceWithWorkers SimBit.
- Within the Definitions window, delete the Transporter list of Nurses, as we will use the population of one of the nurses for experimentation. Within the Facility window, delete Nurse2 and change the Nurse1 name to Nurse.
- Enter the Server1, Server2 objects, *Processing Tasks* and change the first 3 tasks (that require Nurse) from *Object Type* 'Select From List' to 'Specific'. Enter the second task (LabWork) and change the *Condition or Probability* to be a property named 'LabWork_Probability' (right click, Set Referenced Property > Create New Referenced Property).
- Within the Doctor object, go to the *Initial Number in System* property, right click, Set Referenced Property > Create New Referenced Property to 'NumberOfDoctors'. Do the same with the Nurse object with a property named 'NumberOfNurses'.
- Finally, add a Sink named LeaveWithoutBeingSeen. Then, enter Source1 and specify a *Renegé Triggers* under the Buffer Logic to include a *Wait Duration* of '40' minutes and *Renegé Node Name* 'Input@LeaveWithoutBeingSeen'. Entities will leave the system early if the wait is too long (this will be a response in experiment as well).

Experimentation

- Within the Model properties, there will now be 3 model properties, NumberOfDoctors, NumberOfNurses and LabWork_Probability that can be changed.
- Go to the Project Home ribbon and select New Experiment. Those 3 properties will automatically appear under Controls.
- Add 4 Responses to the experiment. This will include:
 - *Name* is 'DoctorUtilization' – *Expression* is 'Doctor.Population.Capacity.ScheduledUtilization'
 - *Name* is 'NurseUtilization' – *Expression* is 'Nurse.Population.Capacity.ScheduledUtilization'
 - *Name* is 'PatientTimeInSystem' – *Expression* is 'DefaultEntity.Population.TimeInSystem.Average'
 - *Name* is 'LeaveWithoutBeingSeen' – *Expression* is 'LeaveWithoutBeingSeen.InputBuffer.NumberEntered'
- Add various scenarios to the experiment, including combinations for 1 or 2 Doctors, 1 or 2 Nurses and a couple with no lab work (assume the hospital is analyzing having lab work done afterwards at separate area). Run the experiment with default Analysis parameters for confidence level, etc.

Dashboard Reports

- Click on the Dashboard Reports tab and click Create to create a new report. Note that there are 4 Data Sources available for reporting. We will use the **Response Results Summaries** (includes Min, Max, Median, Average, etc. if desired), **Response Results Details** and **Experiment Statistic Summaries**.
- For the first report (Response Averages By Scenario), select the **Response Results Summaries** Data Source.
 - Within the Home ribbon, select a Chart and a Filter Elements > List Box.
 - Within the Data Items for the Chart, place the ResponseName in the Series area. This will provide a bar in the chart then for each of the Responses in the experiment. Place the ScenarioName in the Arguments area which will then list the Scenarios along the X-axis. Place the Mean within the Values section. This will graph the mean of each of the responses along the Y-axis. Users could alternatively select Minimum or Maximum for the Value, depending on the information desired.
 - Within the Data Items for the List Box, place the ResponseName in the Dimensions.
 - Save the first dashboard.
- For the second report (Response Details Per Scenario), select the **Response Result Details**.
 - Within this report, we'd like to see the individual values per scenario for each response, along with the average value. Because we don't have the average value within the Response Results Details, we will add a new Calculated Fields entry. This is done by right clicking on the Response Result Details name and selecting Add Calculated Field. Name the new field AverageValue and use the expression builder to generate the *Expression* 'Aggr(Avg([Value]))'.
 - Within the Home ribbon, select a Chart and two Filter Elements > List Box.
 - Within the Data Items for the Chart, place the ScenarioName in the Series. Place the Replication field in the Arguments, as the replication value will be shown along the X-axis (instead of the Scenario as in the report above). Place both the Value field and the new AverageValue calculated field within the Values. This will then give us a data point for each replication value as well as the average over the # of replications run.
 - To customize the chart further, go to the Design ribbon, select the placement of the Legend (for value/averagevalue) and change the Series Type to be a Line.
 - Within the Data Items for the first List Box, place the ResponseName in the Dimensions. Go to the Design ribbon and select Radio item type. This will allow the end user to only select one of the responses at a time (instead of checking multiple ones). Right click to Edit Name to 'Response'.
 - Within the Data Items for the second List Box, place the ScenarioName in the Dimensions. Change this one to a Radio item type as well. Right click to Edit Name to 'Scenario'.
 - Save the second dashboard.
- For the third report (Resource States by Scenario), select the **Experiment Statistic Summaries**.
 - Within this report, we'd like to review the resource states information within the pivot grid. Users may wish to review the pivot grid formatting (Object Type, Object Name, Data Source, Category, Data Item, etc.) to determine the exact fields to display and use as filters.
 - Within the Home ribbon, select a Pie and Filter Elements > List Box.
 - Within the Data Items for the Pie, place the Scenario in the Series. This will then display a separate pie for each scenario. Place the Data Item within the Argument. Data Item includes the values for Time Busy, Time Idle, Time Starved, Time Processing, etc. based on the type of resource. Place the Average value in the Values data item.
 - o IMPORTANT: Still within the Pie, place the Category and Statistic within the Hidden Data Items under Dimensions. These categories will not be shown (hidden) but will be used to filter the experiment statistics to show only the Resource State category and Percent statistic. With the Pie graphic selected, go to the Data ribbon and select Edit Filter (or right click and select Edit Filter). Add a Filter that is '[Category] Equals Resource State' and another that is '[Statistic] Equals Percent'. This will filter the entire report to only show Resource State information, which by default, will include the Data Items needed.
 - Within the Data Items for the List Box, place the Object Name in the Dimensions. Go to the Design ribbon and select Radio item type. This will allow the end user to only select one of the responses at a time (instead of checking multiple ones).
 - Follow the same step as shown above (IMPORTANT:...) by placing the Category under the Hidden Data Items and adding the same filter (Statistic filter not required).

DbReadWrite - SimBit

Problem:

I want to use SQL Server Express to read data into Simio and to write data from Simio.

Categories:

File Management

Key Concepts:

DBConnect Element, DBRead Step, DBWrite Step, DBQuery Step, DBExecute Step, Database, SQL Server Express

Assumptions:

Database connectivity must already be setup, or this model will not run! See Appendix for example.

There are two approaches described below, one model using DbRead, and DbWrite, and another model using DbQuery and DBExecute.

Approach Using DbRead and DbWrite

Technical Approach:

DBConnect Element is used to connect with the SQL Server Express database. The OnRunInitialized process is used to delete existing data from the database table. An Add-On process is used within an object where data is read using DBRead step. The position is defined where data must be written in the database using Assign step. Then using DBWrite step, data is written into database.

Details for Building the Model:

Simple System Setup

- Within the Facility window, place a Source, a Server and a Sink and connect them using Path objects.
- Within the Sink's input node, double-click on the *Entered* Add-On Process to create a new process named 'Input_Sink1_Entered'. We will add steps within that process shortly.

Defining the States for the Model

- Within the Definitions window, click on the States panel and add a String state with the *Name* 'StringState1'. Set the *Initial State Value* to 'DefaultEntity.11.'
- Add a Real state with the *Name* 'RealState1'. Change the *Initial State Value* to '-1'.
- Add two DateTime states with the *Name* of 'DateTimeState1' and 'DateTimeState2'. Set their *Initial State Value* properties to '10/1/12' and '11/1/12', respectively.
- Add two Integer states with the *Name* of 'IntegerState1' and 'RowID'. Leave the *Initial State Value* of both to the default '0'.
- Add a Boolean state with the *Name* 'SimpleDbSteps' that has an *Initial State Value* of 'True'.

Defining the DBConnect Element

- Within the Definitions window, click on the Element panel and select the DBConnect element from the User Defined button pull down.
- In the property window for DBConnect1, enter the *Connection String* as 'Server=localhost\SQLEXPRESS; Database=Test; Uid=username; Pwd=password'. Click on *Provider Name* and set it as 'SqlClient Data Provider'. The current settings in this property require that you rename the username and password value 'test' to the values you have configured in the SQL Server Express database.

Using the Database Steps in a Process

- Within the Processes window, click on Select Process in the Process ribbon and select the OnRunInitialized process. Add a DbExecute step from the User Defined panel to the process. Specify the *DbConnect* as 'DbConnect1', the *SQL Statement* as 'Delete From TestReadWrite where Id>@1'. Within the *Items* repeat group, enter the *Expression* 'RowID'.
- Next, within the Input_Sink1_Entered process, add the Decide, DbRead, Assign and DbWrite steps.
- Within the Decide step, keep *Decide Type* as 'ConditionBased' and add *Expression* as 'RowID > 0'.
- For the DbRead properties, the *DbConnect* is 'DbConnect1', created above. Assign the *Table Name* to 'TestReadWrite'. Click on *Columns* repeating property editor and add 5 items as shown below

Column – State

- String1 – StringState1
- Integer1 – IntegerState1
- Real1 – RealState1
- DateTime1 – DateTimeState1
- DateTime2 – DateTimeState2

Within the *Where* repeating property editor, add *Where Column* value 'Id' and *Where State* value 'RowID'.

- Within the Assign step, enter the *State Variable Name* as 'RowID' and *New Value* as 'RowID+1'.
- For the DbWrite properties, the *DbConnect* is 'DbConnect1' and the *Table Name* is 'TestReadWrite'. Click on *Columns* repeating property editor and add 6 items as shown below

Column – Expression

- Id -- RowID
- String1 – Entity.Name
- Integer1 – Entity.ID
- Real1 – TimeNow
- DateTime1 – String.FromDateTime(Entity.TimeCreated, "yyyy/MM/dd HH:mm:ss")
- DateTime2 – String.FromDateTime(TimeNow, "yyyy/MM/dd HH:mm:ss")

Adding Status Labels to the Facility Window

- From Animation ribbon, select and place 6 Status Labels in the Facility window.
- For first label, enter the *Expression* 'StringState1'. Repeat the same for the other 5 states defined above (not needed is the SimpleDbSteps Boolean state). In the case of the DateTime states, use the *Expression* 'DateTime.ToString(DateTimeState1)' and 'DateTime.ToString(DateTimeState2)'.

Approach Using SQL Query (DBQuery and DBExecute)

All the states and process information remain the same as above, but in the Processes window, use the DbQuery and DbExecute steps instead of DbRead and DbWrite steps.

- For the DbQuery step, the *DbConnect* is 'DbConnect1'. For the *SQL Statement*, enter 'Select String1, Integer1, Real1, DateTime1, DateTime2 from TestReadWrite where Id = @6'. Within the *States* repeating property editor, add 6 State values:

State

- StringState1
- IntegerState1
- RealState1
- DateTimeState1
- DateTimeState2
- RowID

- For the DbExecute step, the *DbConnect* is 'DbConnect1'. The *SQL Statement* value is 'Insert into TestReadWrite (Id, String1, Integer1, Real1, DateTime1, DateTime2) values (@1, '@2', @3, @4, '@5', '@6)'. Within the *Items* repeating property editor, add 6 Expression values:

Expression

- RowID
- Entity.Name
- Entity.ID
- TimeNow

- `String.FromDateTime(Entity.TimeCreated, "yyyy/MM/dd HH:mm:ss")`
- `String.FromDateTime(TimeNow, "yyyy/MM/dd HH:mm:ss")`

Appendix:

Instructions to Install SQL Server Express

- Go to the link below to install SQL Server 2017 Express edition.
- <https://www.microsoft.com/en-us/sql-server/sql-server-editions-express>
- Download 'SQLServer2017-SSEI-Expr.exe' version or higher and run the install.
- In this Appendix, assume that the full version of SQLManagementStudio_x64.exe has been downloaded and installed.

Instructions to Create table in SQL Server Express Edition

- Open Microsoft SQL Server Management Studio.
- Leave the defaults and then press Connect.
- Right click on the Databases folder and select New Database called Test.
- Create a new Login that has the appropriate privileges and desired password.
- Create a New Query clicking the New Query button or pressing (Ctrl + N).
- Enter the code below to create a table 'TestReadWriteTable'

```
USE [Test]
```

```
GO
```

```
SET ANSI_NULLS ON
```

```
GO
```

```
SET QUOTED_IDENTIFIER ON
```

```
GO
```

```
SET ANSI_PADDING ON
```

```
GO
```

```
CREATE TABLE [dbo].[TestReadWrite](
```

```
  [Id] [int] NOT NULL,
```

```
  [String1] [varchar](50) NULL,
```

```
  [DateTime1] [datetime] NULL,
```

```
  [Integer1] [int] NULL,
```

```
  [Real1] [real] NULL,
```

```
  [DateTime2] [datetime] NULL,
```

```
  CONSTRAINT [PK_TestReadWrite] PRIMARY KEY CLUSTERED
```

```
(
```

```
  [Id] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

```
) ON [PRIMARY]
```

```
GO
```

```
SET ANSI_PADDING OFF
```

```
GO
```

- Click Execute to create the table.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DefineEntityProperties - SimBit

Problem:

I have multiple kinds of entities going through a working system. Each type of entity has its unique processing time and rejection rates.

Categories:

Entity Characteristics

Key Concepts:

Expression Property, Numeric Property, Path, Selection Weight

Assumptions:

One Source produces only one type of entity. Reject rates are based on the ratio of the total number of entities that need to be reworked to the total number of entities coming out of the server. (So there is a possibility that some entities will never reach the sink.)

Technical Approach:

Add processing time and rejection rates as properties to ModelEntity, so each type of entity can define its unique processing time and rejection rates. The processing time of the Server is referencing the processing time property of ModelEntity, the weights of the paths represent rejection and success based on the rejection rate property of ModelEntity.

Details for Building the Model:

Simple System Setup

- Place two Sources, a Server and a Sink from the Standard Library into the Facility Window.
- Use Path to connect two Sources' output nodes to the input node of Server separately (Path1, Path2), connect the output node of the Server to its input node (Path3), and also the Server's output node to the Sink (Path4).
- Add two ModelEntity objects from the Project Library into the Facility Window, name one of them PartA, the other one PartB.
- Set Source1's *Entity Type* to 'PartA', Source2's *Entity Type* to 'PartB'. Set the *Interarrival Time* of both of these two Sources to 'Random.Exponential(.5)'.

Generating Entity Properties

- Go to Navigation Window, click on the current project *DefineEntityProperties*, and then go to *ModelEntity*. Open the Definitions Window and select the Properties panel.
- Click on *Standard Property* Ribbon, choose *Real* Data type, and create a Numeric Property. Name it 'RejectRate', its *Display Name* is 'Reject Rate', *Description* is 'Ratio of rejections (0-1)', and *Default Value* is '0.0'.
- Again, go to *Standard Property*, choose *Expression* Data type, and create an Expression Property. Name it 'ProcessingTime', its *Display Name* is 'Processing Time', and *Default Value* is '0.0', *Unit Type* is 'Time', *Default Units* is 'Minutes'.
- In order to make those created properties to be used by customer more conveniently, we can save them to a separate category of properties which we will call 'Custom'. For the property 'RejectRate', go to Properties Window, click on *Category Name*, and create a new category with the name of 'Custom', this will save 'RejectRate' to a new property category which is 'Custom'. Do the same thing for the property 'ProcessingTime'.

Using Entity Properties

- Go back to Model Window and open the Facility Window. Go to Server1, set *ProcessingTime* to 'ModelEntity.ProcessingTime'. This will let the Server's processing time use each Entity's specific processing time according to their entity type.
- Click on Path3, which connects the output node of the Server to its input node. Then go to the Properties Window, right click on *Selection Weight* and choose *Set Referenced Property*, and then choose *Create New Referenced Property*, and create a new referenced property with the name 'ModelEntity.RejectRate'. This will let entities be rejected and go through Server again based on their reject rates corresponding to their entity types.
- Click on Path4, which connects the output node of the Server to the Sink, then set its *Selection Weight* to '1-

ModelEntity.RejectRate'.

Specify Data on Each Entity Instance

- For entity PartA and PartB, click on either one of them and go to Properties Window, there will be two properties we just defined: *Reject Rate* and *Processing Time*.
- Here we can specify reject rate and processing time on each instance of different types of entities. Set PartA's *Reject Rate* to '0.1', *Processing Time* to '0.2' (Minutes); set PartB's *Reject Rate* to '0.5', *Processing Time* to '0.1' (Minutes).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DisableFailureTimers - SimBit

Problem:

I want my model to stop running when all entities have finished processing, but the failures in the system keep the simulation running.

Categories:

Add-On Process Logic, Arrival Logic

Key Concepts:

Calendar Time Based Failure, EventCount, Failure, Maximum Arrivals, Server

Assumption:

Only one Source is generating entities and once the maximum arrivals have been created, the simulation should stop running.

Technical Approach:

When the number of entities that have been created at the Source reaches the maximum value of entities to be created, the Timer state for the calendar time based failure associated with the server is disabled. This is done through an add-on process.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server and a Sink from the Standard Library into the Facility window. Use Paths to connect the Source to the Server and the Server to the Sink.
- Within the Source's Advanced Options, change the Maximum Arrivals to '25' so we can determine how long it takes to process 25 entities through the system.
- Change the Ending Type in the Run Ribbon to Unspecified (Infinite).

Server Failure

- Within the Server, under the Reliability Logic section of properties, change the Failure Type to 'Calendar Time Based'. Change the Uptime Between Failures to '10' and the Units to 'Minutes'. Change the Time to Repair to '1' and the Units to 'Minutes'.

Turning Off Failures When Entities Stop

- Within the Processes window, create a new process named Process1. Add a Decide step that will have the Decide Type as 'ConditionBased' and the Expression as 'Source1.EntityArrivals.EventCount == Source1.MaximumArrivals'. This will evaluate the entities exiting the source with the maximum arrivals value to see if all of the entities have been created.
- From the True exit of the Decide step, add an Assign step. Assign the State Variable Name 'Server1.CalendarTimeBasedFailures.Enabled' to the New Value of '0'. This means the failures are no longer enabled, but are disabled. This will ensure that no additional events continue on the event calendar to keep the simulation running after the entities have all completed processing.
- Within the Facility window, within the Source's output node, change the Exited Add-On Process Trigger to 'Process1' so that when an entity exits the node, this process is run.

Enhancements:

Within every Server, there are multiple types of failures that can be defined. In this example, we demonstrate turning off the Calendar Time Based Failures. If you review the 'MyServer' object in the Navigation window, you will see in the Definitions window, Elements panel, there are multiple Timer elements that are initially disabled. These are enabled if Failure Type corresponding to that Timer is used. Therefore, you can disable any of these types of failures by simply using the corresponding Timer name.

DiscreteLookupTable - SimBit

Problem:

The entities in my model have different processing times depending on a priority that is assigned to the entities in the middle of the model. The processing times and entity priorities are mapped in a table.

Categories:

Add-On Process Logic, Decision Logic – Processing, Lookup and Rate Tables

Key Concepts:

Lookup Table, Linear Interpolation, ModelEntity, Path, Priority, Selection Weight

Technical Approach:

The system has three servers in parallel, followed by one server in series. Depending on which server processes the entity, it is assigned a different value to the state variable, ModelEntity.Priority. The fourth server uses this state variable and reads a Lookup Table to determine the processing time.

Details for Building the Model:

Simple System Setup

- Place a Source, Sink and 4 Servers in the Facility Window. The first three Servers are in parallel and they all lead to the fourth Server that is in series.
- The three paths leading from the Source to the first three Servers each have a different link *Selection Weight* of '.5', '.3' and '.2'. This means that 50% ($.5/ (.5+.3+.2)$) of the entities go to one server, 30% go to another and 20% to another.

Changing the Entity Priority on Paths

- Within the State Assignments properties of the paths leading to Server4 (from Server1, Server2 and Server3), enter the *On Entering* repeating editor. Add a new assignment to specify the *State Variable Name* is 'ModelEntity.Priority', where the *New Value* is '1' for those departing Server 1 on path 4, '2' for those from Server2 on path 5 and '3' for those entities from Server3 on path 6.

Adding a Lookup Table

- Click on the Data tab and select the Lookup Tables panel. Click the Lookup Table button to create a new table.
- Change the *Name* of the table in the Properties Window to 'ProcessingTimes'.
- Enter three rows of data in the table, mapping the numbers 1, 2, 3 to the values 3,5,6.

Utilizing a Table within a Server for Processing Times

- In the *Processing Time* property of Server 4 in the Facility Window, use the following syntax to reference the table 'ProcessingTimes[ModelEntity.Priority]'.

DynamicallyCreatingVehicles - SimBit

Problem:

I have a system where a transporter carries parts from machine to machine. When the system gets moderately busy and I have more than 4 parts in the system, I would like to bring in another vehicle to help with the work. Similarly, if the system becomes extremely busy and there are more than 8 parts in the system, I would like to bring in a third vehicle.

Categories:

Vehicles

Key Concepts:

Add-On Process, Create Step, Decide Step, Label, Monitor, On Associated Object Destroyed, On Entered, Process Triggering Event, Real State Variable, Ride on Transporter, Sink, Source, Transfer Step, Vehicle

Assumptions:

A second vehicle is only created the first time the number of parts in the system goes above 4. And even if the number of parts becomes less than 4, the second vehicle remains part of the model, it is not destroyed. There is similar logic for the creation of the third vehicle.

Technical Approach:

Two different monitor elements watch a Model State that keeps track of the number of parts in the system. The first monitor fires an event when the number of parts in the system goes over 4. This event triggers a process that checks to see how many vehicles are currently in the system and if there is only one, it creates a new vehicle object. Similarly, the second monitor fires an event when the number of parts in the system goes over 8 and a third vehicle is created (if there are only 2 in the system at the time). The reason the process checks to see how many vehicles are in the system before it creates a new one is because the monitor element will fire the event each time the State positively crosses the threshold value (4 or 8). So there might be 5 parts in the system, but then it goes down to 4 and up again to 5. We do not want to create a new vehicle every time it goes above 4, just the first time.

Details for Building the Model:

Simple System Setup

- Add a Source, three Servers, a Vehicle and a Sink to the Facility Window.
- Connect the objects together with Paths. Create two paths between each Server object, each one traveling in opposite directions. Our example has a Bi Directional path between the input and output nodes of Server 3 so the transporter can travel around the object.
- In our example, the *Interarrival Time* of the Source object is set to 'Random.Exponential(.3)' minutes and the *Processing Time* of Server1 is set to 'Random.Triangular(.05, .06, .08)' minutes.
- The Vehicle's *Desired Speed* is set to '0.5' Meters per second and the *Initial Node (Home)* is set to 'Output@Server1'.

Keeping Track of the Number in System

- A new discrete model State is created, called 'Number In System'. This is done by clicking on the Definitions tab, selecting the States panel and adding a new state using the Discrete State button on the ribbon tab. This state is incremented with an Assign Step that is within a process which is triggered by an Add On Process trigger in the Source object, called *Created Entity*. This process is triggered right after an entity has been created by the Source.
- The state Number in System is decremented with an Assign Step that is within a process which is triggered by an Add On Process trigger in the Sink object, called *Entered*.

Creating the Monitor Elements

- Go to the Definitions tab and select the Elements panel. Click on the Monitor icon in the General Ribbon group. The Monitor Type is 'CrossingStateChange' and the State Variable Name is 'NumberInSystem', or whatever name you gave to your model State variable. The Threshold Value is '4' and the Crossing Direction is 'Positive'. This monitor will fire an event whenever the state variable crosses from 4 to 5.
- Add a second Monitor element which is very similar to the first. The only difference with this monitor is that the Threshold Value is set to 8.

Processes that Create the New Vehicle

- Select the Processes tab in the Project Model tabs. Click on the Create Process icon in the Process Ribbon group.
- In the Properties window of this new Process, set the *Triggering Event* to 'Monitor_Over4.Event' (exact name will depend on what you named your monitor). This indicates that when that event is fired, it will trigger this process to execute.
- Place a Decide step in the Process. The *Decide Type* is condition based and the *Expression* is set to 'Vehicle1.Population.NumberInSystem < 2', which will ensure that we only create a new vehicle if there is only one in the system.
- Place a Create step in the True segment leaving the Decide step. The *Create Type* is 'NewObject' and the *Object Instance Name* is 'Vehicle1' (the name of your vehicle).
- When a vehicle is created, it is created in Free Space. So it needs to be transferred to a node that is on the network. Place a Transfer step on the Created segment leaving the Create step. *From* is set to 'FreeSpace', *To* is set to 'Node' and *Node Name* is set to 'Output@Server1'.
- A similar process should be created for the monitor that is watching for more than 8 parts in the system. The only difference in this process is that the *Expression* in the Decide step is 'Vehicle1.Population.NumberInSystem < 3'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EfficientSearchStepPractices - SimBit

Efficient Search Step Practices

This SimBit project includes five models providing examples of using a Search step. The first and second model show the same objective with different approaches. The third, fourth, and fifth model show the same objective with different approaches.

1. **SearchStep1_MatchCondition:** Demonstrates using Search step using a *Match Condition* to search through the entire Data Table and find the best row.
2. **SearchStep1_KeyRelationships:** Demonstrates using a Search step with related rows to narrow down the rows searched to find the best row.
3. **SearchStep2_MinimizeExpression:** Demonstrates using the Search step to minimize an expression by looking through the entire Data Table.
4. *SearchStep2_ForwardSearch:* Demonstrates using the Search step to look through a Data Table in the forward direction so the whole Data Table is not searched.
5. *SearchStep2_ForwardSearchWithIndex:* Demonstrates using a Search step to look through a Data Table starting at a specific index and moving forward so a smaller section of the Data Table is searched.

Model 1: SearchStep1_MatchCondition

Problem:

Shipments arrive with different traits. A data table keeps track of the number of shipments sent to each storage area. Based on the traits of the shipment, I want to look through the data table to find what area has the fewest number of shipments with these traits and send the shipment to this location.

Categories:

Add-On Process Logic, Data Tables

Keywords:

Data Tables, Process Logic, Search Step

Assumptions:

The shipment will arrive randomly and be assigned a random trait in 4 categories. The main Data Table contains all combinations of traits and end locations that can be chosen.

Technical Approach:

A Search step will be used to find a row in the Data Table that matches the traits of the arriving entity. Once the row is found in the Data Table, the row will point to the Input Node of a Sink. The entity will set its destination to this Node.

Details for Building the Model:

Facility Window: Run Parameters

- In the Run ribbon > (Run Setup Group), set *Starting Type* = 'At start of run and use the current' 'Second'. This will allow you to use your computer's current time. You will be able to see in real time how long each model takes to run.

Facility Window: Objects

- Place a Source and 4 Sinks in the Facility.
- Name each respective Sink "Freezer", "Refrigerator", "Dry", "Other".
- Place a ModelEntity instance in the Facility.

ModelEntity Definition

- In the ModelEntity definition, navigate to the Definitions tab > States.
- Add 5 String State Variables with the following names:
- 'Trait1_Number'
- 'Trait2_Color'
- 'Trait3_Perishable'
- 'Trait4_Code'
- 'Area'

Facility Window: Animation

- Create a symbol and apply the symbol to the DefaultEntity instance.
- To create the Symbol, go to the Project Home ribbon and use the arrow below New Symbol. Select 'Create New Symbol'. A new Symbol should appear under the Symbols folder in the Navigation pane located on the upper right side. Use the tools in the Drawing ribbon to draw a rectangle. The color can be changed by using the Color Picker.
- Return to the Model's Facility window by selecting Model in the Navigation pane.
- Select the DefaultEntity instance in the Facility. In the Symbols ribbon, use the Apply Symbol button drop-down to apply the Symbol created.
- Attach a Floor Label to the ModelEntity. Add relevant expressions to show Trait State Variable values.
- '{ModelEntity.Trait1_Number} {ModelEntity.Trait2_Color} {ModelEntity.Trait3_Perishable} {ModelEntity.Trait4_Code} Area: {ModelEntity.Area}'
- Attach a Floor Label to the Model. This Floor Label will calculate the real time this model took to run in seconds by using this expression: 'Math.Round((DateTime.SystemNow - DateTime.FromString(DateTime.ToString(0))) * 3600, 2)'

Data Window: Tables

- Create 2 Data Tables to capture information about the system.
- "AreaTbl" captures each Area string name and its corresponding Sink InputNode Name.
- String Property
- *Name* = 'Area'
- Set Column As Key
- Node Property
- *Name* = 'SinkNode'
- "MainTbl" captures all combinations of Traits and Area locations. An Index column captures the row number of the table. A State Variable column is used to count the number of entities that are assigned to this Area as its end destination.
- Real Property column
- *Name* = 'Index'
- Four String Property columns
- *Name* = 'Number'
- *Name* = 'TagColor'
- *Name* = 'Perishable'
- *Name* = 'Code'

- Foreign Key Property column
- *Name* = 'Area'
- *Table Key* = 'AreaTbl.Area'
- Integer State Variable column
- *Name* = 'Count'

Processes Window: Process Logic

- Create a Process called "1_NewArrivalTraitsProcess" to assign the trait State Variables a value for the Entity created at the Source.
- The Assign step will randomly pick a value for each of the 4 traits and assign it to the ModelEntity's corresponding State Variable.
- Assign step "Traits"
- *State Variable Name* = 'ModelEntity.Trait1_Number'
- *New Value* = 'Random.Discrete("one", 0.1, "two", 0.2, "three", 0.3, "four", 0.4, "five", 0.5, "six", 0.6, "seven", 0.7, "eight", 0.8, "nine", 0.9, "ten", 1)'
- *State Variable Name* = 'ModelEntity.Trait2_Color'
- *New Value* = 'Random.Discrete("red", 0.15, "orange", 0.3, "yellow", 0.45, "green", 0.6, "blue", 0.75, "purple", 0.9, "pink", 1)'
- *State Variable Name* = 'ModelEntity.Trait3_Perishable'
- *New Value* = 'Random.Discrete("TRUE", 0.5, "FALSE", 1)'
- *State Variable Name* = 'ModelEntity.Trait4_Code'
- *New Value* = 'Random.Discrete("AF00123", 0.25, "BG00345", 0.5, "CH00456", 0.75, "DJ00789", 1)'
- Create a Process called "2_AreaDestinationProcess" to search the main table to find and assign the shipment's destination.
- The Search step will look through the Data Table to find the rows in the table that match the entity's traits. Out of those rows, we then want to select the area with the smallest count.
- Search step "MainTbl"
- *Collection Type* = 'TableRows'
- *Table Name* = 'MainTbl'
- *Search Type* = 'MinimizeExpression'
- *Match Condition* = 'ModelEntity.Trait1_Number == MainTbl.Number && ModelEntity.Trait2_Color == MainTbl.TagColor && ModelEntity.Trait3_Perishable == MainTbl.Perishable && ModelEntity.Trait4_Code == MainTbl.Code'
- *Search Expression* = 'MainTbl.Count'
- Off the Search step's Found branch, we want to use the table row for routing and increment the count of shipments that have gone to this area for this combination of traits. The row reference is saved to the Entity executing the process and the State in the Data Table is incremented.
- SetRow step "Area"
- *Table Name* = 'MainTbl'
- *Row Number* = 'MainTbl.Index'
- Assign step "AddToCount"

- *State Variable Name* = 'MainTbl.Count'
- *New Value* = 'MainTbl.Count + 1'
- *Assignments (More)*
- *State Variable Name* = 'ModelEntity.Area'
- *New Value* = 'MainTbl.Area'

Facility Window: Properties

- In the Facility window, update the following properties:
- On the Source:
- Add-On Process Triggers
- *Created Entity* = '1_NewArrivalTraitsProcess'
- On the Source's Output Node:
- Routing Logic
- *Entity Destination Type* = 'Specific'
- *Node Name* = 'AreaTbl.SinkNode'
- Add-On Process Triggers
- *Entered* = '2_AreaDestinationProcess'

Discussion:

Run this model with the Fast-Forward option. Note how long in seconds the model takes to run. As each shipment entity is created and goes to route, the Search step is looking through the entire Data Table to find where the entity's traits match. It needs to compare multiple State Variable values to the values in multiple column cells. Once it has found the rows that satisfy the Search's *Match Condition*, it must evaluate which row has the minimum Count State Variable value. Searching through the entire MainTbl each time is time consuming and computationally expensive.

Notes:

The ModelEntity is assigned a row reference to the MainTbl. The MainTbl does not have any object references so there is no indication as to where an Area is located or what Node to route to. By using the Foreign Key Relationship, "Area" column in MainTbl can then point back to its Key in the AreaTbl. The AreaTbl row has a Node location reference. When the entity needs to resolve the Routing Logic table reference 'AreaTbl.SinkNode', it can see the Key's row it should point to.

Model 2: SearchStep1_KeyRelationships

Problem:

Shipments arrive with different traits. A data table keeps track of the number of shipments sent to each storage area. Based on the traits of the shipment, I want to look through the data table to find what Area has the fewest number of shipments with these traits and send the shipment to this location. I do not want to have to search through the whole table to find this information.

Categories:

Add-On Process Logic, Data Tables

Keywords:

Data Tables, Foreign Key, Process Logic, Related Rows, Search Step

Assumptions:

The shipment will arrive randomly and be assigned a random trait in 4 categories. The main Data Table contains all combinations of traits and end locations that can be chosen.

Technical Approach:

Table Key Relationships will be used to narrow down the large Data Table into a subset of rows to Search through. Process Logic will set row references to Keys for an entity. The corresponding Foreign Keys will help Search only consider the related rows in the table. Once the row is found in the Data Table, the row will point to the Input Node of a Sink. The entity will set its destination to this Node.

Details for Building the Model:

Initial Setup

- Begin with the setup shown in the "SearchStep1_MatchCondition" section. You may right-click the "SearchStep1_MatchCondition" model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Data Window: Tables

- In the Data tab, add four Data Tables, one for each Trait.
- NumberTbl
 - String Property column
 - *Name* = 'Number'
 - Set Column As Key
- ColorTbl
 - String Property column
 - *Name* = 'Color'
 - Set Column As Key
- PerishableTbl
 - String Property column
 - *Name* = 'Boolean'
 - Set Column As Key
- CodeTbl
 - String Property column
 - *Name* = 'Code'
 - Set Column As Key
- The MainTbl has been updated so that the "Number", "TagColor", "Perishable", and "Code" columns are Foreign Key columns for the corresponding trait Data Table. To convert the column type to a Foreign Key, the 'Change Type' button in the Schema ribbon's Edit section can be used. Then specify the *Table Key* property to point back to the correct Key.

Processes Window: Process Logic

- Update the "2_AreaDestinationProcess" to include logic to set the Key to the trait Data Tables prior to Searching.
- The SetRow steps will use the ModelEntity's trait State Variable values to find the row with that same Key and save that explicit row reference to that ModelEntity.
- SetRow Step "Trait1_Number"
 - *Table Name* = 'NumberTbl'
 - *Row Number* = 'NumberTbl.Number.RowForKey(ModelEntity.Trait1_Number)'
- SetRow Step "Trait2_Color"

- *Table Name* = 'ColorTbl'
- *Row Number* = 'ColorTbl.Color.RowForKey(ModelEntity.Trait2_Color)'
- SetRow Step "Trait3_Perishable"
- *Table Name* = 'PerishableTbl'
- *Row Number* = 'PerishableTbl.Boolean.RowForKey(ModelEntity.Trait3_Perishable)'
- SetRow Step "Trait4_Code"
- *Table Name* = 'CodeTbl'
- *Row Number* = 'CodeTbl.Code.RowForKey(ModelEntity.Trait4_Code)'
- Update the MainTbl Search step and remove the *Match Condition*. The Search should only try to minimize the count per area.
- Search step "MainTbl"
- *Collection Type* = 'TableRows'
- *Table Name* = 'MainTbl'
- *Search Type* = 'MinimizeExpression'
- *Search Expression* = 'MainTbl.Count'
- Update the "Area" SetRow step to clear all row references for the entity prior to assigning the MainTbl row reference.
- *Unset All Rows First* = 'True'

Discussion:

Run this model with the Fast-Forward option. There should be a significant speed improvement from the first SearchStep1_MatchCondition model. By providing explicit Keys to the entity, when the Process Logic Searches the MainTbl Data Table, it will only see the rows relating to the entity's Keys. The Foreign Keys will narrow the subsection of rows the Search step needs to search through. If you look at Trace when the Search step is run, the number of items Search looks through is only 4 items. There are only 4 rows that share that combination of Foreign Keys. In the first model, SearchStep1_MatchCondition, the Search will look through all 2240 rows of the MainTbl. The Search step no longer needs to search each row and evaluate the *Match Condition*. Instead, the part of the *Match Condition* that was needed to find trait values in different columns is handled by the Foreign Keys.

Notes:

The RowForKey function returns the row number of a Key where the Key in that column matches the string passed into this function.

The "Area" Set Row step in the 2_AreaDestinationProcess unsets all rows prior to setting the row. Using the *Unset All Rows First* property removes all the row references previously set and for this entity, removes the references to the Keys. The reference to the Keys is removed because the *Row Number* in the "Area" SetRow step being provided is the absolute row index in the table. The Index column in the MainTbl is the absolute row value. If the entity still had a reference to the Keys and attempted to set a row to the table with the Key's related rows, we would only see the relative rows, so we would have to provide the relative row value instead.

To further explore and see how the number of rows available changes when there are Table Key relationships, we suggest looking at the Table's 'AvailableRowCount' function. Without a Key reference the 'AvailableRowCount' will provide the total number of rows available in a Data Table. When a Key reference is given, the Table's 'AvailableRowCount' will change to be the number of related rows for that Key. You might consider putting this function in a Notify step to see this information as Process Logic is run. Notify steps only read out String values, so make sure to convert the Real value to a String.

Another tool to check Table row references is the Watch window. The Watch window will display Table References and Potential Row information for an object.

Model 3: SearchStep2_MinimizeExpression

Problem:

Shipments arrive with different traits. An arrival schedule is provided which details when a certain shipment is expected. A

data table holds the arrival schedule information and traits of the expected shipment. I want to look through the data table to find when the next purple or orange color shipment is arriving, so I can prepare for that shipment type.

Categories:

Add-On Process Logic, Data Tables

Keywords:

Data Tables, Process Logic, Search Step

Assumptions:

The shipment will arrive based on the Arrival Table which specifies the shipment's traits.

Technical Approach:

On the shipment entity's arrival, a process will be executed with a Search step that looks through the Arrival Table to find the next shipment that has either a purple or orange color trait. The information on the next shipment of this type will be displayed in the Facility.

Details for Building the Model:

Initial Setup

- Begin with the setup shown in the "SearchStep1_MatchCondition" section and add the changes from the "SearchStep1_KeyRelationships" model. You may right-click the "SearchStep1_KeyRelationships" model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Data Window: Tables

- In the Data window, add one new Data Table. This table is the schedule of shipments expected.
- ArrivalTable
- Two Real Property columns
- *Name* = 'Index'
- *Name* = 'Arrival Time'
- Four String Property columns
- *Name* = 'Number'
- *Name* = 'Color'
- *Name* = 'Perishable'
- *Name* = 'Code'

Facility Window: Objects

- Update the Source object to use the new Arrival Table. On the Source, change the following properties:
- *Arrival Mode* = 'Arrival Table'
- *Arrival Time Property* = 'ArrivalTable.ArrivalTime'

Definitions Window: States

- Add Real State Variable named "NextArrival" to the model. Keep all default values.
- Add a String State Variable named "NextColor" to the model. Keep all default values.

Processes Window: Process Logic

- Update the "1_NewArrivalTraitsProcess" to include logic to Search the Arrival Table for the next purple or orange shipment and save off the information to Model State Variables.

- Change the Traits Assign step to use the information from the ArrivalTable to assign the shipment entity their State Variable values. Assignments are changed to the following:
- *State Variable Name* = 'ModelEntity.Trait1_Number'
- *New Variable* = 'ArrivalTable.Number'
- *State Variable Name* = 'ModelEntity.Trait2_Color'
- *New Variable* = 'ArrivalTable.Color'
- *State Variable Name* = 'ModelEntity.Trait3_Perishable'
- *New Variable* = 'ArrivalTable.Perishable'
- *State Variable Name* = 'ModelEntity.Trait4_Code'
- *New Variable* = 'ArrivalTable.Code'
- After the Traits Assign step, add a Search step to look through the Arrival Table. This step should find the rows in the table that either have a purple or orange color and have an arrival time greater than the current time. Arrivals that have already happened should not be considered. Out of those rows, we then want to select the row that has the smallest difference between the current simulation time and the time in the Table. The smallest difference indicates it is the next time this shipment will occur.
- Search step "NextOrangePurpleArrival"
- *Collection Type* = 'TableRows'
- *Table Name* = 'ArrivalTable'
- *Search Type* = 'MinimizeExpression'
- *Match Condition* = 'TimeNow < ArrivalTable.ArrivalTime && (ArrivalTable.Color == "orange" || ArrivalTable.Color == "purple")'
- *Search Expression* = 'ArrivalTable.ArrivalTime - TimeNow'
- On the Found branch of the Search step, add an Assign step that pulls the information from the found row in the table and assigns it to State Variables.
- Assign Step "NextArrivalTime"
- *State Variable Name* = 'NextArrival'
- *New Value* = 'ArrivalTable.ArrivalTime'
- *Assignments (More)*
- *State Variable Name* = 'NextColor'
- *New Value* = 'ArrivalTable.Color'

Facility Window: Animation

- Attach a Floor Label to the Model. This Floor Label will display when the next purple or orange arrival is expected. The text of this Floor Label will change based on the value of the NextColor State Variable. The NextArrival State Variable is also displayed as both the simulation model Hour and Date Time.

Discussion:

Run this model with the Fast-Forward option. Note how long in seconds the model takes to run.

For the Search step to have successfully minimized its *Search Expression*, each row in the Data Table will have to be considered. If each available row is not evaluated, there is no way of knowing that the correct minimum has been found. Even if the first row Searched does minimize the expression, the rest of the table will be Searched to ensure there is not a better option.

Notes:

Like the first set of models shown in this collection, the rows searched could have also been narrowed down by using Key Table relationships. But to demonstrate a different behavior, that approach was not included here. Additionally, to correctly find either the next purple or orange row, you would need to Search while the Key is pointing to 'purple', then change the row reference Key to point to 'orange' and Search again. Only one reference to a Key column can be held at a time by an Object or Token. You could not narrow down the table to purple and orange at the same time.

Enhancement: currently the full "1_NewArrivalTraitsProcess" is run each time an entity is created. To further prevent the Search step from being executed extraneously, prior to the Search step, a Decide step could be used to check if the entity currently running the process is an orange or purple entity. If the entity is not orange, purple, or the very first entity to execute the process, then we know the current "NextColor" and "NextArrival" information is still up-to-date based on the last entity to run this process. Therefore, there would be no need to Search again, so the process could end on the one end of the Decide step.

Model 4: SearchStep2_ForwardSearch

Problem:

Shipments arrive with different traits. An arrival schedule is provided which details when a certain shipment is expected. A data table holds the arrival schedule information and traits of the expected shipment. I want to look through the data table to find when the next purple or orange color shipment is arriving, so I can prepare for that shipment type. I do not want to have to search through the whole table to find this information.

Categories:

Add-On Process Logic, Data Tables

Keywords:

Data Tables, Process Logic, Search Step

Assumptions:

The shipment will arrive based on the Arrival Table which specifies the shipment's traits. The Arrival Table will be in chronological order.

Technical Approach:

On the shipment entity's arrival, a process will be executed with a Search step that looks through the Arrival Table to find the next shipment that has either a purple or orange color trait. The Search step will look forwards through the Data Table and will stop when it finds a match and meets its limit of number of items to find.

Details for Building the Model:

Initial Setup

- Use the "SearchStep2_MinimizeExpression" model as a starting point. You may right-click the "SearchStep2_MinimizeExpression" model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Processes Window: Process Logic

- Update the "1_NewArrivalTraitsProcess" to narrow the Search of the Arrival Table for the next purple or orange shipment.
- Change the Search step to look through the Arrival Table by using the 'Forward' *Search Type*. The 'Forward' option does not have a *Search Expression*, so this is removed.
- Search step "NextOrangePurpleArrival"
- *Collection Type* = 'TableRows'
- *Table Name* = 'ArrivalTable'
- *Search Type* = 'Forward'
- *Match Condition* = 'TimeNow < ArrivalTable.ArrivalTime && (ArrivalTable.Color == "orange" || ArrivalTable.Color == "purple")'

Discussion:

Run this model with the Fast-Forward option. Note how long in seconds the model takes to run. Compare this with the SearchStep2_MinimizeExpression model.

When the Search step uses the 'Forward' *Search Type*, the Search will start from the top of the Table and go row-by-row until a row matches the *Match Condition* and the *Limit* is satisfied. Since the *Limit* of our Search step is '1' by default, the first item we find that meets the criteria will stop the Search. This prevents all rows from being looked through each time the Search is executed.

In this scenario, the table must be in chronological order. If the Table is not in chronological order, the next matching row might not be the correct next arrival. As the simulation runs, the Table must look further and further down the table to find a match.

Model 5: SearchStep2_ForwardSearchWithIndex

Problem:

Shipments arrive with different traits. An arrival schedule is provided which details when a certain shipment is expected. A data table holds the arrival schedule information and traits of the expected shipment. I want to look through the data table to find when the next purple or orange color shipment is arriving so I can prepare for that shipment type. I do not want to have to search through the whole table to find this information.

Categories:

Add-On Process Logic, Data Tables

Keywords:

Data Tables, Process Logic, Search Step

Assumptions:

The shipment will arrive based on the Arrival Table which specifies the shipment's traits. The Arrival Table will be in chronological order.

Technical Approach:

On the shipment entity's arrival, a process will be executed with a Search step that looks through the Arrival Table to find the next shipment that has either a purple or orange color trait. The Search step will look forward through the Data Table starting at a specific index and will stop when it finds a match and meets its limit of number of items to find.

Details for Building the Model:

Initial Setup

- Use the "SearchStep2_ForwardSearch" model as a starting point. You may right-click the "SearchStep2_ForwardSearch" model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Processes Window: Process Logic

- Update the "1_NewArrivalTraitsProcess" to add information that helps narrow down the Search information.
- In the Search step, add in the *Starting Index* property found under the Advanced Options category.
- Search step "NextOrangePurpleArrival"
- *Collection Type* = 'TableRows'
- *Table Name* = 'ArrivalTable'
- *Search Type* = 'Forward'
- *Match Condition* = 'TimeNow < ArrivalTable.ArrivalTime && (ArrivalTable.Color == "orange" || ArrivalTable.Color == "purple")'
- *Starting Index* = 'ArrivalTable.Index'

Discussion:

Run this model with the Fast-Forward option. Note how long in seconds the model takes to run. Compare this with the SearchStep2_ForwardSearch model.

In the SearchStep2_ForwardExpression model, the Search would always start at the top of the table and evaluate each row until a match was found. Since the table was listed in chronological order, each time the Search was executed as the model

ran, it would have more rows to Search through till a match was found. By including the *Starting Index*, we can specify the row the Search step should start at. Since the process is executed by the entity that was just created, we have a reference to the row index of the most recent arrival. Again, since the Table is in chronological order, there is no reason to Search for rows before the most recent arrival. Therefore, the *Starting Index* can be the index of the entity that just arrived.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ElectricVehicle - SimBit

Problem:

I have an electric vehicle that it is used for the transportation of the entities in the system. The vehicle is sent to a charging station when the battery level decreases to a certain threshold.

Categories:

Add-On Process Logic, Building New Objects / Hierarchy, Custom Object, Vehicles

Key Concepts:

Assign Step, Bidirectional Path, CrossingStateChange, Decide Step, Dynamic Label Text, Execute Step, Level State Variable, Monitor, Movement, Movement.Rate, Off Shift, Override, Rate, Ride On Transporter, Subclass, Vehicle

Assumptions:

The electric vehicle battery consumption is directly related to the distance travelled, and it loses its charge at a rate of 1% per meter travelled. When the Battery level reaches 20% remaining, the vehicle will finish unloading is current entity and travel to the charging station. The vehicle charges at a rate of 5% per minute.

Technical Approach:

A Vehicle is subclassed from the Vehicle in the Standard Library and has additional process logic to track the battery level. A Monitor watches the battery level and triggers a process to make the vehicle go off shift and send it to the charging station when the battery level is low. When the Vehicle is done charging, the Vehicle goes back on shift and resumes transporting entities.

Details for Building the Model:

Simple System Setup

- In the Facility Window place a Source, a Server and a Sink. Also place one BasicNode above the Server, and a second BasicNode, renamed 'Charger', to the right of the first BasicNode.
- Place a ModelEntity from the Project Library into the Facility window.
- Using Paths to connect the Source to the Server and the Server to the Sink. Connect BasicNode1 to the Source, the Server, the Sink, and BasicNode 'Charger'. Also connect the Server's input node to Server's output node. Select all the paths change the *Type* property to 'Bidirectional'.
- Within the Source, change the *Interarrival Time* property to '10' minutes. Within the Server, change the *Processing Time* property to '5' minutes.

Creating a Sub-Classed Object MyVehicle

- Create a new vehicle called MyVehicle that is subclassed from Vehicle by right-clicking on Vehicle in the Standard Library and selecting Subclass.
- Select MyVehicle in the Navigation window and in its Definitions window, States panel, add a Level State Variable with *Name* 'BatteryRemaining' and change the *Initial State Value* property to '100'. This will track the current battery level, which will start with a full battery.
- In the Elements panel, add a Monitor element with the *Name* 'MovementRateChanged' and change the *State Variable Name* property to 'Movement.Rate' and the *Triggered Process Name* property to 'OnMovementRateChanged'. Every time the Vehicle starts and stops, the OnMovementRateChanged process is triggered.
- Add another Monitor element with the *Name* 'MonitorBatteryRemaining' and change the *State Variable Name* property to 'BatteryRemaining'. Change the *Monitor Type* property to 'CrossingStateChange'. Set the *Initial Threshold Value* property to '100' and change the *Triggered Process Name* to 'FullChargeDetected'. This will trigger the FullChargeDetected process every time the BatteryRemaining state increases to 100.
- Within the Processes window of the MyVehicle object, click Create Process to add a new process and change its *Name* to 'StartChargingBattery'.
 - Add a Decide step and change the *Condition* property to 'HomeNode==CurrentNode'.
 - On the True branch leaving the Decide step, place an Assign step. Set the *State Variable Name* property to 'BatteryRemaining.Rate' and its *New Value* to '300'. When the Vehicle is at the Charger node, the battery level

will increase at a rate of 300 per hour.

- Create a new process named 'FullChargeDetected'. This process is triggered when the BatteryRemaining variable reaches 100.
 - Add an Assign step and set the *State Variable Name* 'BatteryRemaining.Rate' to *New Value* of '0'.
 - Add another Assign step to the right of the previous Assign step, and set the *State Variable Name* to 'CurrentCapacity' and the *New Value* to '1'. This will change the current state of the Vehicle to 'Onshift'. Now that the vehicle is fully charged and on shift, it will travel to the next entity waiting for transport.
- Create a third process named 'OnMovementRateChanged'.
 - Add an Assign step, and change the *State Variable Name* property to 'BatteryRemaining.Rate' and the *New Value* to '- Movement.Rate'. This will decrease the battery level at the same rate the vehicle is travelling.
 - Add an Execute step for the *Process Name* of 'StartChargingBattery'.
- Right click the process "OnRiderUnloaded" and select Override to allow the process to be modified.
 - On the right side of the Fire step, add a Decide step and change the *Condition* property to 'BatteryRemaining < 20 && RideStation.Contents == 0'.
 - On the True branch leaving the Decide step, place an Assign step. Set the *State Variable Name* to 'CurrentCapacity' and the *New Value* to '0'. This will change the current state of the Vehicle to 'Offshift' if the BatteryRemaining is below 20 and there are no entities left on the Vehicle. The Vehicle is set to go to its Home Node (Charger) when it goes off shift.

Vehicle Setup

- Place a MyVehicle object from the Project Library to the Facility window and change the *Name* to 'ElectricVehicle'.
- Expand the Animation property and set the *Dynamic Label Text* property to 'Math.Round(MyVehicle.BatteryRemaining, 1)'. That will show a label next to MyVehicle object with the current BatteryRemaining state value.
- Change the *Initial Desired Speed* property to '0.5' Meters per Second.
- Under Routing Logic, change the *Initial Node (Home)* property to 'Charger'. Also change the *Idle Action* property to 'Remain to Place' and the *Off Shift Action* property to 'Park At Home'.

Transferring the Entity

- Within the Source output node, specify the *Entity Destination Type* of 'Specific' and the *Node Name* of 'Input@Server1'. Change the *Ride on Transporter* to 'True' and the *Transporter Name* to 'ElectricVehicle'.
- Within the Server output node, specify the *Entity Destination Type* of 'Specific' and the *Node Name* of 'Input@Sink1'. Change the *Ride on Transporter* to 'True' and the *Transporter Name* to 'ElectricVehicle'.
- These previous steps tell the entity where to go next and on what vehicle (otherwise, it may leave the node on any of the connected bidirectional paths).

Animation

- Change the color of the top of the Entity object in order to identify it when it is being transported by the vehicle.
- Change the symbol of the BasicNode Charger to identify where the charger is.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EntitiesEnteringAlongConveyor - SimBit

Problem:

I have multiple types of entities that enter onto a single conveyor from multiple merge points. Entities then travel down the single conveyor for later processing.

Categories:

Conveyor Systems

Key Concepts:

BasicNode, Conveyor

Assumptions:

There are three types of entities that enter the system. Each entity type has a unique entry area onto which it will later merge onto a single conveyor system. When arriving at its merge point, the entity must wait for adequate space on the conveyor before merging. At any merge point, entities enter based on a first in first out basis.

Technical Approach:

Use three Source objects to model the arriving entity types. Use three BasicNodes to model the merge points on the main conveyor system. Finally, use Conveyor objects to model the entry conveyors for each part type from the respective sources to the merge points, as well as for the main conveyor system.

Details for Building the Model:

Simple System Setup

- Place three Source objects and a Sink in the Facility Window.
- Place three BasicNodes into the Facility Window. These nodes will represent the merge points from each individual part type entering onto the main conveyor.

Defining the Conveyor System

- Connect each Source object to a BasicNode using the Conveyor object.
- Then, starting at the leftmost BasicNode, connect it to the next BasicNode, and then again to the next with a Conveyor. Connect the rightmost BasicNode to the Sink object. This will represent the entities leaving the conveyor system.

Specifying Conveyor Characteristics

- For each of the six (6) conveyor objects, the default Conveyor characteristics will be used. This includes the *Conveyor Speed* of '2', as well as the *Accumulating* property of 'True'. We would like all of our conveyors to be accumulating, therefore if a given merge point is blocked, entities will begin to accumulate on their respective entering conveyor.

Defining Multiple Entity Types

- Place three ModelEntity objects from the Project Library and change the *Name* them 'PartA', 'PartB' and 'PartC'.
- Change the entity symbols, so that you can graphically see the difference in the parts on the conveyors. We used the Simio symbol library to select various "box" symbols for each of the entity types.
- Within the three Source objects, specify the *Entity Type* as 'PartA', 'PartB' and 'PartC'. Therefore, each source will generate a different part type.

Changing the Conveyor Animation

- In order to graphically make the conveyor paths look like conveyors, click on the conveyor path and select one of the path decorators from the Path Decorators ribbon (i.e., conveyor, single lane, track, etc.).

EntityFollowsSequence - SimBit

Problem:

I have an entity that needs to be processed in a particular order by a series of machines.

Categories:

Decision Logic – Paths, Sequence Tables

Key Concepts:

By Sequence, Entity Destination Type, ModelEntity, Sequence Table, Table Transfer Node Property

Assumptions:

The source only produces one type of entity. This entity will follow the Sequence: Server 3 - Server 2 - Server 1 – Sink 1.

Technical Approach:

A Sequence Table is created and the entity is set to follow this Sequence in its Routing Logic. Each Output TransferNode is set to pass the entity to its next destination *By Sequence*.

Details for Building the Model:

Simple System Setup

- Add a Source, Sink and three Servers to the Facility Window. Update the *Processing Time* of each Server to be 'Random.Triangular(0.5, 0.8, 1.2)'.

Adding a Sequence Table

- In the Data Window, select the Tables panel and add a Sequence Table with *Name* 'MySequence'. Add the Destinations in the following order: Input@Server3 -> Input@Server2 -> Input@Server1 -> Input@Sink1.

Assigning the Sequence to the Entity

- In the Facility Window, place an entity from the Project Library into the model.
- Change the Routing Logic *Initial Sequence* property of the DefaultEntity to 'MySequence'.

Modifying the TransferNodes

- For each of the objects in the system, including Source and all three Servers, click on the TransferNode, change the *Entity Destination Type* property (under Routing Logic) to 'By Sequence'.
- Double-click on the Path to connect the Source object to all three Servers, and then connect all the Servers to each other and to the Sink object.

Embellishments:

There are many ways to make this model more specific. Try changing the Default entity's Travel Logic to a desired speed or the Source's Interarrival Time rate.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EntityFollowsSequenceMultiple - SimBit

Problem:

Enhancement of EntityFollowsSequence.spf

I have multiple sources; each producing their own specific entity type. Each entity type needs to be processed in a particular order by a series of machines.

Categories:

Decision Logic – Paths, Sequence Tables

Key Concepts:

By Sequence, Entity Destination Type, ModelEntity, Sequence Table, Table Transfer Node Property, TransferNode

Assumptions:

Each source only produces one type of entity.

- Source 1 produces Part A (Green) which follows the Sequence: Server 1 – Server 2 – Sink 1.
- Source 2 produces Part B (Red) which follows the Sequence: Server 3 – Server 2 – Server 1 – Sink 1.
- Source 3 produces Part C (Blue) which follows the Sequence: Server 2 – Sink 1.

Technical Approach:

A Sequence Table is created for each entity type and each entity is set to follow this Sequence in its Routing Logic. Each output TransferNode, except those at Source 2 and Source 3, is set to pass the entity to its next destination 'By Sequence'. The output TransferNodes at Source 2 and Source 3 have been excluded because an entity will travel from these nodes to the TransferNode at Source 1, which already has 'By Sequence' in its Routing Logic.

Details for Building the Model:

Simple System Setup

- Add three Sources, three Servers and a Sink to the Facility Window. Update the *Processing Time* of each Server to be 'Random.Triangular(0.5, 0.8, 1.2)'.

Adding a Sequence Table

- In the Data Window, select the Tables panel and add three Sequence Tables named 'SequenceA', 'SequenceB' and 'SequenceC'. Set the Destinations in each table in the following orders:

SequenceA : Input@Server1 -> Input@Server2 -> Input@Sink1

SequenceB : Input@Server3 -> Input@Server2 -> Input@Server1 -> Input@Sink1

SequenceC : Input@Server2 -> Input@Sink1

Assigning the Sequences to the Entities

- Place three DefaultEntity objects from the Project Library into the Facility Window. Change the *Name* of each to 'PartA', 'PartB' and 'PartC'.
- Change the Routing Logic *Initial Sequence* property of each PartA, PartB, and PartC to 'SequenceA', 'SequenceB', and 'SequenceC', respectively.

Modifying the TransferNodes

- For each output TransferNode, except those at Source2 and Source3, change the *Entity Destination Type* property to 'By Sequence'. Entities created at Source2 and Source3 will connect through the output TransferNode at Source1.
- Double-click on the Path to connect the Source1 object to all three Servers, and then connect all the Servers to each other and to the Sink object.

Embellishments:

There are many ways to make this model more specific. Try changing any entity's Travel Logic *Speed Limit* to a desired

speed or the Source's *Interarrival Time* rate.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EntityFollowsSequenceWithRelationalTables - SimBit

Problem:

Enhancement of EntityFollowsSequenceMultiple.spf

I have a single source producing multiple types of entities. Each entity type needs to be processed in a particular order by a series of machines. The service times at each server are dependent upon the entity type.

Categories:

Data Tables, Sequence Tables

Key Concepts:

Before Creating Entities, By Sequence, Data Table, Dynamic Object Property, Entity Destination Type, Expression Property, ModelEntity, Numeric Property, RandomRow, Relational Table, Sequence Table, Server, Source, String Property, Table Foreign Key Property, Table Key, Table Row Referencing, Table Transfer Node Property

Assumptions:

The Source produces three types of entities:

Part A (Green) which follows the Sequence: Server 1 – Server 2 – Server 3 – Sink 1.

Part B (Red) which follows the Sequence: Server 3 – Server 2 – Server 1 – Sink 1.

Part C (Blue) which follows the Sequence: Server 2 – Sink 1.

Technical Approach:

A single Sequence Table is created to keep track of the Sequences and Process Times for each entity type. A Data Table is created to specify product mix and point to the proper section of the Sequences table using the Foreign Keys feature. The Table Assignment properties of the Source will be used to select a random entity type from the Data Table while the entity is being created. Each TransferNode is set to pass the entity to its next destination 'By Sequence'.

Details for Building the Model:

Simple System Setup

- Add a Source, a Sink and three servers to the Facility Window. Also, place three ModelEntity objects from the Project Library into the window.

Setting up the Data Table

- Go to the Data tab, select the Tables panel and add a Data Table named 'JobTable' with the following Properties and in the following order:
(Entity Object Reference) PartType : PartA, PartB, PartC
(Integer) ProductMix : 10, 20, 30
(String) SequenceType : A,B,C
- When the SequenceType column is selected, make this the primary key of this table by clicking the "Set Column as Key" icon in the Ribbon.

Setting up the Sequence Tables

- Also the Data Window within the Tables panel, add a Sequence Table named 'Sequences'. For the order of Destinations in each table, please refer to EntityFollowsSequenceMultiple.pdf.
- Add a Foreign Key property to this Sequence Table by clicking on the Foreign Key icon in the Ribbon. The *Name* of this property should be set to 'SequenceType'. The *TableKey* property should be set to 'JobTable.SequenceType'.
- Add an Expression Property to the Sequence Table with the *Name* set to 'ProcessTime'. Set the *ProcessTime* according to the following:

SequenceA: Input@Server1 = Random.Uniform(0.5, 0.9), Input@Server2 = Random.Triangular(0.5, 1.1, 1.2),
Input@Sink1 = 0.0

SequenceB: Input@Server3 = Random.Triangular(0.5, 0.8, 1.2), Input@Server2 = 1.5, Input@Server1 = 1, Input@Sink1
= 0.0

SequenceC: Input@Server2 = Random.Triangular(0.5, 1.2, 1.6), Input@Sink1 = 0.0

Creating Multiple Entities from Single Source:

- In the Facility Window, Source object, expand the Table Row Referencing section.
- Set the Table Name property to 'JobTable', and the Row Number to 'JobTable.ProductMix.RandomRow'.

Finalizing the Model

- Change the Arrival Logic *Entity Type* property of the Source to 'JobTable.PartType'.
- Change all *Processing Times* of the Servers to 'Sequences.ProcessTime'.
- For each TransferNode, change the *Entity Destination Type* property to 'By Sequence'.

Embellishments:

There are many ways to make this model more specific. Try changing any entity's Travel Logic *Speed Limit* to a desired speed, the Source's *Interarrival Time* rate, or any Server's *Capacity*.

See Also:

EntityFollowsSequenceMultiple.spf, EntityFollowsSequenceWithTable.spf (Uses simple (not relational) table to accomplish the same thing) and UsingRelationalTables.spf

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EntityFollowsSequenceWithTable - SimBit

Problem:

Enhancement of EntityFollowsSequenceMultiple.spf

I have a single source producing multiple types of entities. Each entity type needs to be processed in a particular order by a series of machines. The service times at each server are dependent upon the entity type.

Categories:

Data Tables, Decision Logic – Paths, Decision Logic – Processing, Sequence Tables

Key Concepts:

Before Creating Entities, By Sequence, Data Table, Dynamic Object Property, Entity Destination Type, Expression Property, ModelEntity, Numeric Property, On Created Entity, RandomRow, Sequence Table, Server, Source, Table Row Referencing, Table Sequence Property, Table Transfer Node Property

Assumptions:

The Source produces three types of entities:

- Part A (Green) which follows the Sequence: Server 1 – Server 2 – Server 3 – Sink 1.
- Part B (Red) which follows the Sequence: Server 3 – Server 2 – Server 1 – Sink 1.
- Part C (Blue) which follows the Sequence: Server 2 – Sink 1.

Technical Approach:

A separate Sequence Table is created for each entity type to keep track of the Sequences and Process Times for each entity type. A Data Table is created to help the Servers determine how long to process an entity based on its type. The Data Table will be read by setting the Table Name property of the Source. The Source will also reference the Data Table after an entity has been created to get reference to the appropriate Sequence Table based on the assigned row from the Job Table. Each TransferNode is set to pass the entity to its next destination 'By Sequence'.

Details for Building the Model:

Simple System Setup

- Add a Source, a Sink and three servers to the Facility Window. Also, place three ModelEntity objects from the Project Library into the window.

Setting up the Sequence Tables

- In the Data Window, select the Tables panel and add three Sequence Tables named 'SequenceA', 'SequenceB', and 'SequenceC'. For the order of Destinations in each table, please refer to EntityFollowsSequenceMultiple.
- Add an Expression Property to each Sequence Table with the Name 'ProcessTime'. Set the *ProcessTime* according to the following:

SequenceA: Input@Server1 = Random.Uniform(0.5, 0.9), Input@Server2 = Random.Triangular(0.5, 1.1, 1.2),
Input@Sink1 = 0.0

SequenceB: Input@Server3 = Random.Triangular(0.5, 0.8, 1.2), Input@Server2 = 1.5, Input@Server1 = 1, Input@Sink1
= 0.0

SequenceC: Input@Server2 = Random.Triangular(0.5, 1.2, 1.6), Input@Sink1 = 0.0

Setting up the Data Table

- Add a Data Table named 'JobTable' with the following Properties and in the following order:
(Entity Object Reference) PartType : PartA, PartB, PartC
(Expression) ProcessTime : SequenceA.ProcessTime, SequenceB.ProcessTime, SequenceC.ProcessTime
(Integer) ProductMix : 10, 20, 30
(Sequence) PartSequence : SequenceA, SequenceB, SequenceC

Creating Multiple Entity Types from Source:

-
- In the Facility Window, expand the Table Row Referencing in the Properties Window of the Source object.
 - Under the *Before Creating Entities* subcategory, set the *Table Name* to 'JobTable' and the *Row Number* to 'JobTable.ProductMix.RandomRow'
 - Under the *On Created Entity* subcategory, set the *Table Name* to 'JobTable.PartSequence' and leave the *Row Number* empty.

Finalizing the Model

- Change the Arrival Logic *Entity Type* property of the Source to 'JobTable.PartType'.
- Change all *Processing Times* of the Servers to 'JobTable.ProcessTime'.
- For each TransferNode, change the *Entity Destination Type* property to 'By Sequence'.

Embellishments:

There are many ways to make this model more specific. Try changing any entity's Travel Logic *Speed Limit* to a desired speed, the Source's *Interarrival Time* rate, or any Server's *Capacity*.

See Also:

EntityFollowsSequenceMultiple.spf

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

EntityStopsOnLink - Simbit

Problem:

I have a system where the entities in the system must stop moving on their links when a particular event occurs.

Categories:

Decision Logic -- Paths

Key Concepts:

Add-On Process, Allow Passing, Calendar Time Based Failure, Current Symbol Index, Current Symbol Index, Event, Failure, Failure.Active, Fire Step, ModelEntity, On Failed, On Repaired, ParentObjectMovement, Path, Real State Variable, Resume Step, Server, Subscribe Step, Suspend Step, Time To Repair, Uptime Between Failures, Wait Step

Assumptions:

Entities that are currently moving on links will stop mid-stream when a particular event occurs. In this example, that event is a server failure. Once the failure has been fixed, all entities resume their movement.

Technical Approach:

The Subscribe step for the Model Entity is used to tie together the events of the model, such as failure and repair of the Server, with events for the entity itself, such as suspending and resuming movement on a link. Two properties for the entity are used to define the event names. When those events occur in the model, all currently moving entities have their movement stopped or started.

Details for Building the Model:

Simple System Setup

- Add a Source, Sink and Server to the Facility Window. Add a ModelEntity to the Facility Window. Connect the Source to the Server and the Server to the Sink using Paths. Change the *Allow Passing* property of each Path to 'False'.

Adding the Server Failure

- In the Reliability Logic section of the Properties window for Server1, add a failure by changing the *Failure Type* to 'Calendar Time Based'.
- Set the *Uptime Between Failures* property to '2' and *Units* to 'Minutes', and the *Time to Repair* property to '1' and *Units* to 'Minutes'.

Model Entity: Suspending and Resuming the Entity Movement

- In the Definitions Window of the Model Entity, add two new properties, 'SuspendMyselfEvent' and 'ResumeMyselfEvent'. These two properties will then be edited in the Model Entity's properties window to tie the events in the model to the model entity.
- In the Processes Window of the Model Entity, use the Create Process button to create two new processes called SuspendMyself and ResumeMyself. We will add steps to these processes shortly.
- Also in the Processes Window of the Model Entity, use the Select Process button to select the OnCreated process. Within that process, add two Subscribe steps. In the first Subscribe step, right click on *Event Name* and Set Referenced Property to 'SuspendMyselfEvent'. Set the *Process Name* to 'SuspendMyself'. In the second Subscribe step, right click on *Event Name* and Set Referenced Property to 'ResumeMyselfEvent'. Set the *Process Name* to 'ResumeMyself'.
- Within the SuspendMyself process, add a Suspend step and change the *Suspend Type* to 'ParentObjectMovement'. This will stop the movement of all entities currently created when this process is triggered.
- Within the ResumeMyself process, add a Resume step and change the *Resume Type* to 'ParentObjectMovement'. This will resume all entity movement of all entities when this process is triggered.

Model: Firing the Events to Stop Entity Movement

- Go to the Model and open the Processes window. Add two new processes called SuspendAll and ResumeAll. These processes will be triggered when the Server fails and is repaired.

- In the Definitions window, add two new events, 'SuspendEverybody' and 'ResumeEverybody'. These events will be fired from the Processes just defined.
- In the Processes window of the model, within the SuspendAll process, add a Fire step with the *Event Name* 'SuspendEverybody'. Within the ResumeAll process, add a Fire step with the *Event Name* 'ResumeEverybody'.
- Move to the Facility window and highlight Server1. Within the Add-On Process Triggers, change the *Failed* property to 'SuspendAll' and the *Repaired* property to 'ResumeAll'. This will fire the events to suspend and resume entities when the Server is failed and repaired, respectively.

Model: Specifying the Model Entity Properties for the Events

- Click on ModelEntity1 that is placed in the Model. In the General section of properties, you will notice the two new properties we added above called *SuspendMyselfEvent* and *ResumeMyselfEvent*. These are also referenced in the Subscribe steps for the ModelEntity.
- Change the *SuspendMyselfEvent* property to 'SuspendEverybody' and the *ResumeMyselfEvent* property to 'ResumeEverybody'. These will then feed into the Subscribe steps such that when those events are fired within the model, the associated processes in the ModelEntity will be triggered.

Model: Stopping Incoming Entities

- All of the above steps will stop entities that have been created and are moving in the system. To also stop incoming entities (as the Source will continue to generate entities), the following steps can be followed.
- In the model's Definitions window, States panel, add a new Discrete State with *Name* of 'MovementSuspended'. The *Initial State Value* property should remain as '0'. This state will be evaluated when incoming entities enter the system to determine if movement is suspended or not.
- Within the Facility window, highlight the transfer node of Source1 and add an Add-On Process Trigger to the *Entered* property by double-clicking on Entered. This will create a new process called 'Output_Source1_Entered'.
- Within the Process window, you will see the new process called 'Output_Source1_Entered'. Within this, add a Decide step where the *Decide Type* is 'ConditionBased' and the *Expression* is 'MovementSuspended==1'. From the True exit, add a Wait step, to wait for the *Event Name* 'ResumeEverybody'. From the False exit, there will be no steps.
- Finally, within the SuspendAll process (in this Processes window), add an Assign step (after the Fire step) where you assign the *State Variable Name* 'MovementSuspended' to the *New Value* of '1'. And similarly, in the ResumeAll process (in this Processes window), add an Assign step (after the Fire step) where you reset the *State Variable Name* 'MovementSuspended' to the *New Value* of '0'.

Embellishments:

Entities may be suspended and resumed based on any number of happenings in a model, including time and/or a state variable reaching a given value. You may also cause an event to occur by placing a Button in the Facility window and/or Console window.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExamplesOfConveyors - SimBit

Problem:

I would like to understand how various conveyor configurations behave.

Categories:

Conveyor Systems

Key Concepts:

Accumulating, Auto Align Cells, BasicNode, BasicNode, Calendar Time Based Failure, Cell Location, Conveyor, Cross Logic, Entity Alignment, Failure, Fixed Number Cells, Initial Capacity, Largest Value First, Number of Cells, On Event Arrival Mode, Time To Repair, Uptime Between Failures

Technical Approach:

A "Master Source" creates the Entity arrivals for all other sources. This allows all sets of conveyors to depict their behavior on the exact same arrivals. The sets of conveyors are then used to depict transfers between different conveyor configurations.

Details for Building the Model:

Standard – all defaults left untouched.

- Simply entities traveling on our standard conveyor

Accumulating – all defaults left untouched.

- Entities will form a queue on the first conveyor if Processing Station is not completely clear. A new entity will not be able to enter the Processing Station until the occupying entity's trailing edge has entered the exit conveyor.

Non-Accumulating – The entry conveyor has *Accumulating* set to 'False'.

- If an Entity cannot immediately gain access to the Server, the entire conveyor will temporarily stop movement. When the Entity in the Processing Station completely exits the Server, the next Entity will enter and the conveyor will begin moving again.

Changing Speeds from Fast to Slow – The second conveyor's *Desired Speed* has been set to '0.5 m/s'.

- The first entity will enter the second conveyor and begin moving at the speed of the slower conveyor. If Entities are created with a small enough interarrival time, the second entity will collide with the first entity and move on the first conveyor at the speed of the second conveyor.

Changing Speeds with a Non-Accumulating Conveyor – The first conveyor has *Accumulating* set to 'False' and the second conveyor's *Desired Speed* has been set to '0.5 m/s'.

- The first entity will enter the second conveyor and begin moving at the speed of the slower conveyor. If Entities are created with a small enough interarrival time, the second entity will collide with the first entity and the speed of the faster conveyor will be set to the speed of the slower conveyor. All other entities entering the faster conveyor will move at the speed of the slower conveyor until all queued entities have completely left the first conveyor.

Changing Speeds From Faster Cell Aligned to Slower Cell Aligned – Each conveyor has *Entity Alignment* set to 'Cell Location', *Cell Spacing Type* is set to 'Fixed Number Cells', *Number of Cells* is set to '8' and *Auto Align Cells* is set to 'No'

- Entities travel down the cell aligned conveyor on a chain dog. When the entity gets to the next cell aligned conveyor, it has to wait until a chain dog from that conveyor comes by to pick it up. Because the second conveyor is slower, this will cause entities to form a queue on the first conveyor and the entities will disengage from their chain dogs. Once a chain dog from the second conveyor picks up a waiting entity, the other waiting entities will step along the first conveyor, starting with the entity farthest along the conveyor, as chain dogs become available.

Changing Speeds From Slower Cell Aligned to Faster Cell Aligned – Each conveyor has *Entity Alignment* set to 'Cell Location', *Cell Spacing Type* is set to 'Fixed Number Cells', *Number of Cells* is set to '8' and *Auto Align Cells* is set to 'No'

- Entities travel down the cell aligned conveyor on a chain dog. When the entity gets to the next cell aligned conveyor,

it has to wait until a chain dog from that conveyor comes by to pick it up. Because the second conveyor is faster, there will never be any queuing because a chain dog will always arrive to pick up a waiting entity before the next entity arrives at the end of the link.

Faster Cell Aligned Non-Accumulating to a Slower Cell Aligned – Each conveyor has *Entity Alignment* set to 'Cell Location', *Cell Spacing Type* is set to 'Fixed Number Cells', *Number of Cells* is set to '8' and *Auto Align Cells* is set to 'No', and the first conveyor has *Accumulating* set to 'False'

- Entities travel down the cell aligned conveyor on a chain dog. When the entity gets to the next cell aligned conveyor, it has to wait until a chain dog comes by to pick it up and disengages itself from the first conveyor. If a second entity reaches the end of the conveyor before the first is picked up, it will stop the movement of the entire conveyor. When the waiting entity enters the slower conveyor, the entire first conveyor moves at the slower rate. After the entity is completely off of the first conveyor it will run at full speed.

Moving Onto Failing Cell Aligned from Cell Aligned – Each conveyor has *Entity Alignment* set to 'Cell Location', *Cell Spacing Type* is set to 'Fixed Number Cells', *Number of Cells* is set to '8' and *Auto Align Cells* is set to 'No'. The second conveyor has Calendar Time Based with an *Uptime Between Failures* of '0.1 minutes' and a *Time To Repair* of '0.1 minutes'

- This situation is similar to moving onto a slower cell aligned conveyor. Entities will form a queue and after the first entity completely exits the first conveyor, the waiting entities will step forward when chain dogs become available.

Merging Conveyors With Different Speeds – The two merging conveyors have their *Desired Speeds* set to '3 m/s' and '4 m/s', and they are merging onto a conveyor moving at 2 m/s.

- If there is a queue at the merge point, entities will take turns merging based on a FIFO Entry Ranking Rule.

Merging Cell Aligned Conveyors With Different Speeds – Same setup as above but Each conveyor has *Entity Alignment* set to 'Cell Location', *Cell Spacing Type* is set to 'Fixed Number Cells', *Number of Cells* is set to '8' and *Auto Align Cells* is set to 'No'

- Entities will merge based on a FIFO Entry ranking rule, but when it is an entities turn to merge, it must first wait for an available chain dog. Similar to other accumulating cell aligned conveyors, the remaining waiting entities will step their way to the end of the conveyor when chain dogs become available.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExamplesOfFunctions_DynamicObjects - SimBit

Problem:

I would like to understand some of the functions available to use with dynamic objects (entities, transporters, etc).

Categories:

Functions

Key Concepts:

Capacity.Allocated, CurrentLink, DestinationNode, Distance, Dynamic Object, Floating Label, FrontTraffic, Function, ID, IsRiding, Location.X, Math.Round(), NumberCreated, NumberDestroyed, NumberInSystem, Ride on Transporter, RideStation.Capacity.Remaining, CurrentTransporter, Round, SlipSpeed, TransferNode, Vehicle

Assumptions:

The functions shown in this model are not the entire list of available functions. To see all the available functions and to get additional information on all functions, see the [Functions](#) page.

Status Labels are used to display the expressions for the population functions. Status Labels are added to the Facility window from the Animation Ribbon. Floating Labels are used to label these functions. Floor Labels are used to display the titles of each section. Rectangles and Polylines were also used to create the displays in this model. Floating Labels, Floor labels, Rectangles and Polylines are all added to the Facility window from the Drawing Ribbon. The functions that are attached to the entity and the Transporter are attached Floor Labels that contain both text and expressions.

Technical Approach:

The attached Floor Labels were added to this model by first placing a Vehicle (from the standard library) into the Facility window. Also place a ModelEntity (from the project library) into the Facility window.

Click on each object and select Floor Label from the Attached Animation category in the Ribbon. Draw an attached Floor Label and then click on the Edit button in the Ribbon to edit the text. Text is typed into the Label Text window directly. An expression can be entered into this label by surrounding it with curved brackets { }. Because the expressions are attached to the object, the object qualifier does not need to be part of the expression. For example, to display Vehicle1.Capacity.Allocated in a label that is attached to Vehicle1, the expression should read {Capacity.Allocated}.

Details for Building the Model:

Examples of Functions Available for a Dynamic Object

- **Vehicle1.Capacity.Allocated** – Returns the current number of capacity units of Vehicle1 that have been allocated (have been seized) Note: These are resource capacity units, not ride capacity units.
- **Vehicle1.CurrentLink** – If Vehicle1's leading edge is currently on or at the end of a link, this function returns a reference to the link.
- **Vehicle1.DestinationNode** – Returns a reference to the Vehicle1's current destination node or returns 'Nothing' if it does not have a destination assigned.
- **Vehicle1.ID** – Returns the unique ID number of Vehicle1.
- **Vehicle1.RideStation.Capacity.Remaining** – Returns the current available carrying capacity of this transporter.
- **Vehicle1.SlipSpeed** – If the entity (or vehicle) is currently on a link, then this function returns the minimum of the link's current speed or the speed of the next entity immediately ahead on the same link. If there is no traffic ahead on the link, then this function simply returns the link's current speed.
- **DefaultEntity.ID** – Returns the unique ID number of this entity.
- **DefaultEntity.DestinationNode** – Returns a reference to the entity's current destination node or returns 'Nothing' if it does not have a destination assigned
- **DefaultEntity.FrontTraffic.Distance** – If the entity object is currently on a link, then this function returns the distance from the entity's leading edge to the trailing edge of the next entity immediately ahead on the same link. If there is no traffic ahead on the link, then this function returns the distance from the entity's leading edge to the end of the link. Note: The FrontTraffic function can be used alone and it will return True (or 1) if there is an entity ahead of it on

the link. In addition to the Distance function, there is also a FrontTraffic.ID function that will return the ID of any front traffic and FrontTraffic.Speed that will return the current speed of an entity that is traveling ahead on the link. See [Functions, States and Events for Link Objects](#) for more information.

- **DefaultEntity.IsRiding** – Returns 'True' (or '1') if the entity is currently riding on a Transporter.
- **DefaultEntity.CurrentTransporter** – If the entity object is currently riding on a transporter, then this function returns a reference to that transporter.
- **DefaultEntity.Location.X** – Returns the X coordinate of the object's current location. Therefore, this changes as the entity moves along the link. Location.X and Location.Y are also available.

Examples of Functions Available a Population of Dynamic Objects

- **DefaultEntity.Population.NumberInSystem** – Returns the current number of objects of this object's type that are currently in the system.
- **DefaultEntity.Population.NumberCreated** – Returns the total number of objects of this object's type that have been created.
- **DefaultEntity.Population.NumberDestroyed** – Returns the total number of objects of this object's type that have been destroyed.

See Also:

The SimBit titled [ExamplesOfFunctions_StaticObjects](#) for examples of functions that are available for static objects.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExamplesOfFunctions_StaticObjects - SimBit

Problem:

I would like to understand some of the functions available to use with static objects.

Categories:

Functions

Key Concepts:

AssociatedStationLoad, Average, AverageTimeWaiting, Capacity.Allocated, Capacity.Remaining, Contents, CurrentDirection, EventCount, InputBuffer, LastRecordedValue, Load, Location.X, NumberEntered, NumberOccurrences(), NumberTravelers, NumberWaiting, OutputBuffer, Path, PercentTime(), Processing, ResourceState, Server, Sink, TimeInSystem, TimeOnLink, TotalTime()

Assumptions:

The functions shown in this model are not the entire list of available functions. To see all the available functions and to get additional information on all functions, see the [Functions](#) page.

Status Labels are used to display the expressions. Status Labels are added to the Facility window from the Animation Ribbon. Floating Labels are used to label the functions. Floor Labels are used to display the titles of each section. Rectangles and Polylines were also used to create the displays in this model. Floating Labels, Floor labels, Rectangles and Polylines are all added to the Facility window from the Drawing Ribbon.

Details for Building the Model:

Examples of Functions Available for a Source

- **Source1.Location.X** – Returns the X coordinate of the Source's current location. The location function is available for all static and dynamic objects. Other options available are Location.Y and Location.Z
- **Source1.EntityArrivals.EventCount** - Returns the number of events that have been fired by the Timer since its last reset. The EventCount function is available with any Timer element.
- **Source1.OutputBuffer.Contents** – Returns the number of entities currently in the OutputBuffer station. There are other functions available to use with the Contents queue, such as AverageNumberWaiting and MaximumTimeWaiting. See the [Functions](#) page for information on Queue State functions.

Examples of Functions Available for a Path

- **Path1.CurrentDirection** – Returns the current direction of traffic on this link. The value returned is Enum.TrafficDirection.Forward, Enum.TrafficDirection.Reverse or Enum.TrafficDirection.None. (1, 2 or 3). This function is available for all Link objects.
- **Path1.TimeOnLink.Average** – Returns the average time the traveler was on this link. Maximum and Minimum are also available.
- **Path1.NumberTravelers** – Returns the current number of travelers on this link. There are other functions available to use with this function: NumberTravelers.Accumulated, NumberTravelers.Average, NumberTravelers.Minimum, NumberTravelers.Maximum, NumberTravelers.Entered, NumberTravelers.Exited. See [Functions, States and Events for Link Objects](#) Help page for additional information on these functions.

Examples of Functions Available for a Sink

- **Sink1.InputBuffer.NumberEntered** – Returns the total number of entity objects that have entered this station. NumberExited is also available.
- **Sink1.TimeInSystem.Average** – Returns the average of the observations recorded. In this example, this is the average time in system of the entity objects that enter into this Sink object. TimeInSystem is a [TallyStatistic](#) element and Average is a function available to all TallyStatistic elements. Other functions available for TallyStatistics are NumberObservations, HalfWidth, LastRecordedValue, Maximum and Minimum. The value is multiplied by 60 to convert from hours to minutes and is rounded to 1 significant digit using the Math.Round function.
- **Sink1.TimeInSystem.LastRecordedValue** – Returns the last recorded value for this TimeInSystem TallyStatistic within the Sink object. The TimeInSystem is referring to the TimeInSystem of the entities that enter this Sink object. TimeInSystem is a [TallyStatistic](#) element and LastRecordedValue is a function available to all TallyStatistic elements.

Other functions available for TallyStatistics are NumberObservations, HalfWidth, Average, Maximum and Minimum. The value is multiplied by 60 to convert from hours to minutes and is rounded to 1 significant digit using the Math.Round function.

Examples of Functions Available for a Server

- **Server1.Capacity.Remaining** – Returns the current unallocated capacity of this object. Other related functions are Capacity.Allocated, Capacity.Initial, Capacity.Maximum, Capacity.Minimum, Capacity.Previous, Capacity.Remaining, Capacity.AllocatedTo(owner) (Returns the number of capacity units of this object currently seized and owned by the executing token's associated object.)
- **Server1.Capacity.Allocated** – Returns the current number of capacity units of this object that are allocated (have been seized).
- **Server1.InputBuffer.Contents.AverageTimeWaiting** – Returns the average time that an entity waited in the Contents queue state of the InputBuffer(in hours). In other words, the average time that an entity spent in the InputBuffer of Server1. There are other functions available to use with the Contents queue, such as AverageNumberWaiting and MaximumTimeWaiting. The value is multiplied by 60 to convert from hours to minutes and is rounded to 1 significant digit using the Math.Round function. See the [Functions](#) page for information on Queue State functions.
- **Server1.InputBuffer.Contents.NumberWaiting** – Returns the current number of entities waiting in the InputBuffer of Server1.
- **Server1.Processing.Contents** – Returns the current number of entities that are in the Processing station of Server1.
- **Server1.ResourceState.PercentTime(1)** – Returns the percent time that this [Resource list state](#) had the value of 1. In other words, the percentage of the total simulation time that Server1 was in the Resource State of 1 (Processing). *
- **Server1.ResourceState.TotalTime(0)** – Returns the total time that this [Resource list state](#) had the value of 0. In other words, the total amount of time that Server1 was Starved or Idle. *
- **Server1.ResourceState.NumberOccurrences(3)** – Returns the number of times that Server entered into the [Resource list state](#) of 3. In other words, it returns the number of times that Server1 entered the Failed state. *
- **Server1.ResourceState == 1** - Returns the value of 0 if Server1 is not currently in ResourceState of 1 and returns a value of 1 when Server1's ResourceState is currently = 1. *
- **Input@Server1.AssociatedStationLoad** – For an external input node, this function returns the current 'load' on the station locations inside the node's associated object that may be entered using the node. The associated station 'load' is defined as the sum of current entities en route to the node intending to enter the stations, plus the current entities already arrived to the node but still waiting to enter the stations, plus the current entities occupying the stations.
- * NOTE: Default Server ResourceState Values: 0 – Starved, 1 – Processing, 2 – Blocked, 3 – Failed, 4 - OffShift, 5 - FailedProcessing, 6 - OffshiftProcessing, 7 - Setup, 8 - OffshiftSetup

See Also:

The SimBit titled [ExamplesOfFunctions_DynamicObjects](#) for examples of functions that are available for dynamic objects.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExcelReadWrite - SimBit

Problem:

I want to use MS Excel to read data into Simio and also to write data from Simio.

Categories:

File Management

Key Concepts:

Add-On Process, ExcelRead Step, ExcelWrite Step, Assign Step, ExcelConnect Element, String State Variable, Integer State Variable

Technical Approach:

ExcelConnect element is used to connect with the Excel sheet. An add-on process is used where row and column position are defined using Assign step from where data has to be read in Excel file. Then using ExcelRead step, data is read and using ExcelWrite step, data is written into Excel.

Details for Building the Model:

Simple System Setup

- Within the Facility window, add a Source, a Server and a Sink from Standard Library. Connect the objects using Paths.
- Within the Source object, change the Maximum Arrivals (under Stopping Conditions) to '30'.
- Within the output node of the Source, within the Add-On Process Triggers section, double-click on the Entered property to create a new process named 'Output_Source1_Entered'.

Adding States and Elements

- Within the Definitions tab, click on the States panel. Add a String State and three Integer State and name them 'StringState', 'RowIntegerState', 'ColumnIntegerState', 'IntegerState'.
- Next, click on Element panel and within the Element Ribbon, select on User Defined button to select ExcelConnect element. In the property window for ExcelConnect1 created, enter the Excel Workbook name 'ExcelReadWrite.xlsx'.

Creating the Writing Process

- Within the Processes window, in the Process named Output_Source1_Entered, add two Assign steps from Common Steps panel and ExcelRead and ExcelWrite Step from User Defined panel of steps.
- For the first Assign step, assign the State Variable Name 'ColumnIntegerState' and New Value 'ColumnIntegerState +1'. For the sSecond Assign step, assign State Variable Name 'RowIntegerState' and New Value 'RowIntegerState +1'.
- For the ExcelRead step, select 'ExcelConnect1' from the list for the ExcelConnect property. Click on Worksheet and enter 'Sheet1'. For Row , enter 'RowIntegerState' and for Starting Column, enter '1'. Then click on States to open repeating property editor. Add. a State 'StringState' and another State 'IntegerState'.
- For the ExcelWrite step, again select 'ExcelConnect1' for the ExcelConnect property. Add the Worksheet value of 'Sheet2'. For Row, enter 'RowIntegerState' and for Starting Column, enter 'ColumnIntegerState'.Then click on Items to enter the repeating property editor. Add an Expression named 'TimeNow', one named 'StringState' and one named 'IntegerState'.

Animating the State Information within the Facility Window

- Within the Facility window, click on the Animation ribbon. Select Status Label and draw two Status Labels in Facility window. For the first Status Label, enter the Expression 'StringState'. For second Status Label, enter the Expression 'IntegerState'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExperimentWithDataConfigurations - SimBit

Problem:

Data drives my Simio model. I want to experiment with different data inputs in the Data Tables to see how it affects the system.

Categories:

Data Tables, Experiment

Keywords:

Data Tables, Data Connector, Data Configuration, Experiment

Assumptions:

Parts will arrive based on a schedule. This schedule specifies the time, part type, and quantity to arrive.

Information in a Data Table represents the available resource capacity the system will have for a given 24-hour day. Work areas have varying processing times and may be staffed differently, giving the stations varying capacity.

External data is available and ready to be imported into Simio to represent the different part arrival schedules and resource capacity.

Technical Approach:

Data Tables drive inputs to the Source and Server objects. Data Connectors bind the Data Tables to an external data source which feed information into the model. Data Configurations are created for the Data Connectors that point to variations of that data. An Experiment is created to test how different Data Configurations effect the system.

Details for Building the Model:

Facility Window: Objects

- In the Facility window, place a Source, 3 Servers, and a Sink, as shown in the model.
- Create a NodeList that contains the Input Nodes of the 3 Servers. This can be done by right-clicking on a Node and selecting 'Add to Node List'. If a NodeList does not yet exist, you can create a new one from the menu option 'Create New Node List...'.
 - On the OutputNode of the Source, change *Entity Destination Type* to 'Select From List', *Node List Name* to 'NodeList1', *Selection Goal* to 'Smallest Value', and *Selection Expression* to `Candidate.Node.AssociatedStationLoad`
 - On the OutputNode of the 3 Servers, change *Entity Destination Type* to 'Specific' and *Node Name* to 'Input@Sink1'.
- Add two ModelEntity Instances and name them Red and Blue. While an instance is selected, you can choose to color the symbol with the Color option in the Symbols ribbon.

Data Window: Tables

- In the Data window, add two data tables named ArrivalTable and ServerTable.
- For the ArrivalTable schema, add the following:
 - DateTime Property
 - *Name* = 'ArrivalTime'
 - Entity Property
 - *Name* = 'PartType'
 - Real Property
 - *Name* = 'Qty'

- With the ArrivalTable selected, go to the Content ribbon and choose the Create Binding dropdown. Under the Create a New Importer Binding section choose Excel Data Importer.
- In the dialog that appears change the following:
 - *File Name* = 'ArrivalTable.xlsx'
 - *Name* = 'ArrivalTableImporter'
- Close the dialog and in the Properties pane on the right, specify *Worksheet or Named Range* as 'Schedule1'
- In the Content ribbon, use the Import Table button to populate the table with data.
- For the ServerTable schema, add the following:
 - Object Property
 - *Name* = 'ServerName'
 - *Auto-set Table Row Reference* = 'True'
 - Real Property
 - *Name* = 'InitialCapacity'
 - Real Property
 - *Name* = 'ProcessingTime'
 - *Unit Type* = 'Time'
 - *Default Units* = 'Minutes'
- With the ServerTable selected, go to the Content ribbon and choose the Create Binding dropdown. Under the Create a New Importer Binding section choose Excel Data Importer.
- In the dialog that appears change the following:
 - *File Name* = 'ServerTable.xlsx'
 - *Name* = 'ServerTableImporter'
- Close the dialog and in the Properties pane on the right, specify *Worksheet or Named Range* as 'ServerConfigBaseline'
- In the Content ribbon, use the Import Table button to populate the table with data.

Data Window: Data Connectors

- In the Data window, on the left side of the window, navigate to the Data Connectors view. The two Data Connector Importers should be visible in the Data Importers section.
- Select the ArrivalTableImporter and use the Add Configuration button to create a new configuration.
- Create the following configurations:
 - 'Schedule2'
 - 'Schedule3'
- These configurations will use the same *File Name* but point to different worksheets in the file. Use the small plus button next to the Configuration name to expand the information. Specify the following worksheets for each configuration:
 - [Default]
 - *Worksheet Or Named Range* = 'Schedule1'
 - Schedule2

- *Worksheet Or Named Range* = 'Schedule2'
- Schedule3
- *Worksheet Or Named Range* = 'Schedule3'
- Select the ServerTableImporter and use the Add Configuration button to create a new configuration.
- Create the following configurations:
 - 'ServerConfig1'
 - 'ServerConfig2'
 - 'ServerConfig3'
 - 'ServerConfig4'
- These configurations will use the same *File Name* but point to different worksheets in the file. Use the small plus button next to the Configuration name to expand the information. Specify the following worksheets for each configuration:
 - [Default]
 - *Worksheet Or Named Range* = 'ServerConfigBaseline'
 - ServerConfig1
 - *Worksheet Or Named Range* = 'ServerConfig1'
 - ServerConfig2
 - *Worksheet Or Named Range* = 'ServerConfig2'
 - ServerConfig3
 - *Worksheet Or Named Range* = 'ServerConfig3'
 - ServerConfig4
 - *Worksheet Or Named Range* = 'ServerConfig4'
- To switch between the configuration a Data Table is using, select the Data Importer and use the Select Configuration drop down in the Data Connectors ribbon. Make sure to use the Import All Matching Bindings in the same location or use the Import Table option in Content ribbon back in the Table view.

Facility Window: Properties

- In the Facility window, update the following properties on the Source:
 - *Entity Type* = 'ArrivalTable.PartType'
 - *Arrival Mode* = 'Arrival Table'
 - *Arrival Time Property* = 'ArrivalTable.ArrivalTime'
 - *Entities Per Arrival* = 'ArrivalTable.Qty'
- Update the following properties on the Servers: (Note, you can update multiple Servers at one time by multi-selecting them in the Facility (ctrl + click))
 - *Initial Capacity* = 'ServerTable.InitialCapacity'
 - *Processing Time* = 'ServerTable.ProcessingTime'
- Update the Run *Starting Type* -> *Specific Starting Time* to 1/1/2023 12:00:00M and the Run *EndingType* -> *Specific Ending Time* to 1/2/2023 12:00:00 AM

-
- Update the *Maximum Number In System Limit* to 'Infinity' for the Red and Blue Entities to prevent maximum number in system limit Warning

Experiment

- Create an Experiment either by using the New Experiment button in the Project Home or by right-clicking on the Model in the Navigation pane and selecting New Experiment.
- In the Experiment, a section for Importer Configurations should be seen with the ArrivalTableImporter and ServerTableImporter Data Connectors.
- Add the following Responses:
- PartsCompleted
- *Expression* = 'Sink1.InputBuffer.NumberEntered'
- RedTIS
- *Expression* = 'Red.Population.TimeInSystem.Average'
- BlueTIS
- *Expression* = 'Blue.Population.TimeInSystem.Average'
- Create a Scenario for each combination of Data Configurations.

Discussion:

In the Model, test out changing the Data Configurations. Be sure to import the data after changing the configuration. Watch in the Facility how the model changes. Add in additional animation such as labels and plots to verify the model is acting as you would expect.

As Scenarios are run in the Experiment, a different set of data will be imported into the Data Tables and change the inputs to the model, as verified above. The effects of changing the inputs are noticeable by comparing the Responses in the Experiment.

Notes:

For more information on Data Connectors and importing data, see the Help pages "DataConnectors" and "Importing and Binding To Tables".

In the Experiment, the Importer Configurations can be included in optimization runs as factors that the optimizer can change. To run the optimizer, you must first add the OptQuest Add-In*. The property option to *Include in Optimization* will then appear for the Importer Configurations, similar to how other Reference Property Controls can be added into the optimization. The *Objective Type* and *Primary Response* will need to be set for the Experiment. Ensure each Response has an Objective property set where appropriate. See the "OptQuest Add-In" Help page for more information on running the optimizer.

*OptQuest requires an additional license to use with Simio, please contact your sales representative or sales@simio.com to learn more about OptQuest.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExtrasLibraryCrane - SimBit

Crane_DualCraneSystem

Problem:

I have a system where entities are moved by one of two Cranes in a single bay. The Cranes can move into shared zones and may block each other. An idle Crane may need to be moved to allow the other Crane to complete its route.

Keywords:

Bay, Blocking, Composite Object, Crane

Categories:

Composite Object, Extras Library, Transporters

Assumptions:

The system has two flows of entities, each being moved by a dedicated Crane. Entities enter the system at two separate Sources, travel to a TransferNode serving as a pickup location, are moved through free space by a Crane to a BasicNode representing a drop-off location, and then travel on a network path to a Sink node. Each of the two entity flows have their own pickup and drop off locations, and each of the two entity flows utilize a dedicated Crane for that flow. However, the locations for the two entity flows overlap, therefore create a blocking situation between the two cranes.

Each Crane has a home node (i.e. *Initial Node (Home)*) that defines its starting point, as well as the return location for an idle Crane if the Crane's *Idle Action* is set to 'Go To Home'.

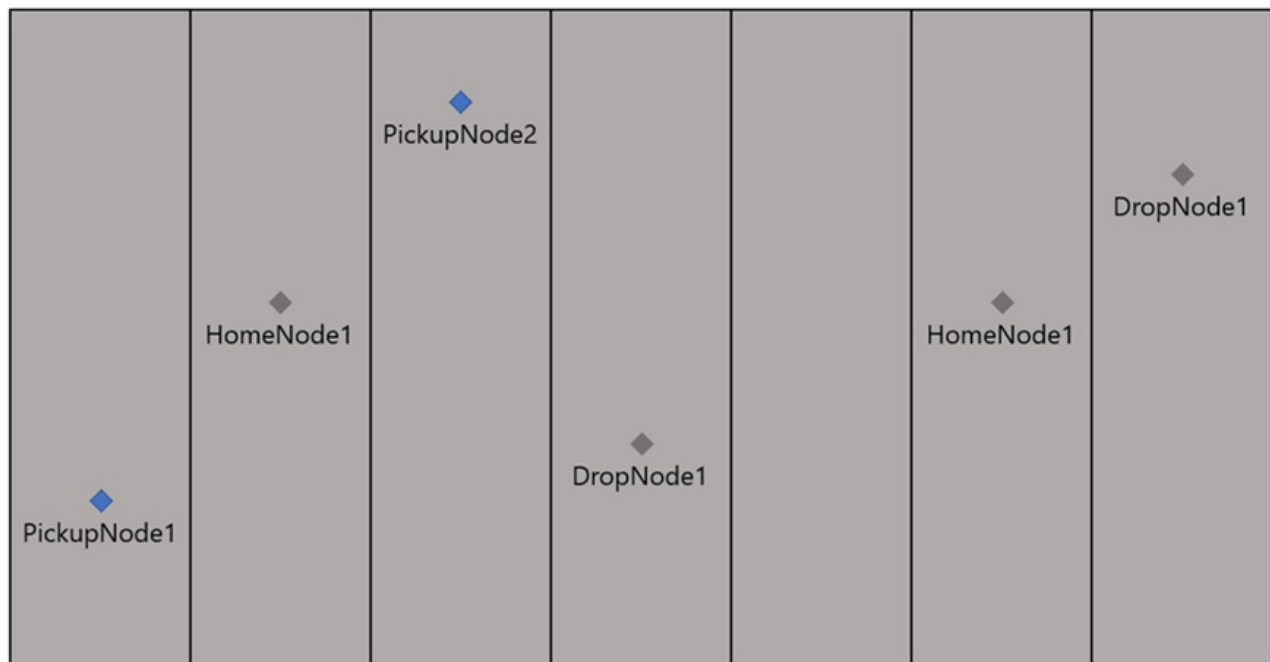
The home, pickup, and drop off locations for the two entity flows within the Bay are shown in Figure 1. Note because the Zones are dynamically generated, the divisions between zones are not shown until runtime. To move the Nodes while the Zones are visible, you can use the Step button to start and stop the model, then move the Nodes during runtime.

Technical Approach:

Create a Bay object divided into seven Zones to model the interaction between two Cranes. Before a Crane moves to a new location it will seize the destination Zone, and all intervening zones must be idle before this seize can take place. Then, set the *Blocking Action* for the Bay to 'Push Idle Bridges', causing an idle Bridge to be automatically moved out of the way when it is blocking a Crane move.

Note: The Crane is a composite object composed of the Crane (Transporter), Lift (Entity), Bridge (Entity), and Cab (Entity). To edit the Crane properties, you must click on the Crane label or the Cranes transporter symbol represented by the hook. In the 3D view, you can also independently click on the Bridge, Cab, and Lift objects to edit their properties. This is typically only done to change the traveling height of the Bridge and or Cab.

Figure 1: Home, Pickup, and Drop locations for the two entity flows



Details for Building the Model:

- Place a Bay in the center of the Facility View. Change its *Number of Zones* property to '7' and its *Blocking Action* to 'Push Idle Bridge'.
- Place two TransferNodes and four BasicNodes at the locations shown in Figure 1, named accordingly.
- Place a Crane object above HomeNode1, setting its *Initial Node* to 'HomeNode1' and its *Associated Bay* to 'Bay1'. Place a second Crane object above HomeNode2, setting its *Initial Node* to 'HomeNode2' and its *Associated Bay* to 'Bay1'. Change the Idle Action property of both Cranes to 'Go To Up Position'.
- Place a Source to the left of PickupNode1, changing its *Interarrival Time* property to 'Random.Exponential(1)'. Create a Path between the Output Node of Source1 to PickupNode1 and set its *Allow Passing* property to 'False'.
- Place a Source above PickupNode2 and change its *Interarrival Time* property to 'Random.Exponential(.5)'. Create a Path between the Output Node of the Source to PickupNode2 and set its *Allow Passing* property to 'False'.
- Create a Sink below DropNode1 and add Path connecting DropNode1 to the Sink's Input Node.
- Create a Sink to the right of DropNode2 and add a Path connecting DropNode2 to the Sink's Input Node.
- Select PickupNode1, then change its *Entity Destination* property to 'Specific', its *Node Name* property to 'DropNode1', its *Ride On Transporter* property to 'True', and its *Transporter Name* property to 'Crane1'.
- Select PickupNode2, then change its *Entity Destination* property to 'Specific', its *Node Name* property to 'DropNode2', its *Ride On Transporter* property to 'True', and its *Transporter Name* property to 'Crane2'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExtrasLibraryElevator - SimBit

This SimBit project includes two models providing examples and use cases for the Elevator object definitions included in the Extras Library.

1. Elevator_SingleBank - Demonstrates how to implement independent elevators to transfer Entities between levels.
2. Elevator_DualBank - Demonstrates how to implement a bank of multiple elevators to transfer Entities between levels.

Model 1: Elevator_SingleBank

Problem:

I have a system with two levels, requiring people to be shuttled between levels with a single elevator.

Keywords:

Elevator, Entity Destination Type, NodeList

Categories:

Extras Library, Transporter

Technical Approach:

Use an Elevator object from the Extras Library to model the flow of people between the two levels, adding a single Elevator Node per level representing the drop off and pickup locations.

Details for Building the Model:

Standard Library Setup

- Using the Rectangle tool in the Drawing ribbon, create a rectangle approximately 16 meters long in the X direction and 6 meters long in the Z direction.
- Using the Rectangle tool, create another rectangle approximately 16 meters long in the X direction and 6 meters long in the Z direction atop the first rectangle.
- In 3D perspective view, select one of the rectangles and hold the shift key to drag the rectangle up approximately 10 meters in the Y direction.
- In the Drawing tab, click the Color drop down and select a dark gray. Click the top platform to change its color. Repeat this for the bottom platform.
- Right-click on the top platform and click Lock Edits to ensure the platform is not moved. Repeat this for the bottom platform.
- Drag two ModelEntities into the Model, naming one "Floor1_Employee" and the other "Floor2_Employee". Add a different, easily distinguishable symbol to each ModelEntity by selecting the ModelEntity and clicking Apply Symbol in the Symbols ribbon, then selecting a symbol.
- Create a Sink on the upper-right side of the bottom platform. Change its *Name* to 'Floor1_Sink'. Create a Sink on the upper-right side of the top platform. Change its *Name* to 'Floor2_Sink'.
- Create a Source on the upper-left side of the bottom platform. Change its *Name* to 'Floor1_Source' and its *Entity Type* property to 'Floor1_Employee'. Select its output Node and change its *Outbound Travel Mode* property to 'Network Only', its *Entity Destination Type* to 'Specific', and its *Node Name* to 'Input@Floor1_Sink'.
- Create a Source on the upper-left side of the top platform. Change its *Name* to 'Floor2_Source' and its *Entity Type* property to 'Floor2_Employee'. Select its output Node and change its *Outbound Travel Mode* property to 'Network Only', its *Entity Destination Type* to 'Specific', and its *Node Name* to 'Input@Floor2_Sink'.

Elevator Setup

- Create an ElevatorNode in the middle of the bottom edge of the bottom platform. Create another ElevatorNode in the middle of the bottom edge of the top platform. Change the *Name* of the ElevatorNodes to 'Floor1_ElevatorNode' and 'Floor2_ElevatorNode'.
- In the Lists view of the Definitions tab, create a new Node List called "ElevatorNodeList". Add the Nodes Floor1_ElevatorNode and Floor2_ElevatorNode to this list.
- Place an Elevator adjacent to Floor1_ElevatorNode. Set its *Initial Ride Capacity* to '4', its *Minimum Dwell Time Type* to 'Specific Time', its *Minimum Dwell Time* to '0.2' minutes, its *Elevator Node List* to 'ElevatorNodeList', its *Initial Node* to 'Floor1_ElevatorNode', and its *Idle Action* to 'Remain In Place'.
- Select each ElevatorNode and change their *Associated Elevator* to 'Elevator1' and their *Elevator Destination Type* to 'EntityDestination'.
- Create two Paths, one from Floor1_Source to Floor1_ElevatorNode and another from Floor2_Source to Floor2_ElevatorNode. Set the *Allow Passing property* on each Path to 'False'.
- Create two Paths, one from Floor1_ElevatorNode to Floor1_Sink, and one from Floor2_ElevatorNode to Floor2_Sink. Set the *Allow Passing property* on each Path to 'False'.

Model 2: Elevator DualBank

Problem:

I have a system with two levels, requiring people to be shuttled between levels by a bank of elevators. This system's elevator bank contains 2 elevators.

Keywords:

Elevator, Entity Destination Type, NodeList

Categories:

Extras Library, Transporter

Technical Approach:

Use two Elevators and two sets of ElevatorNodes, adding an ElevatorSelectorNode on each floor to route entities to the preferred ElevatorNode.

Details for Building the Model:

Standard Library Setup

- Using the Rectangle tool in the Drawing ribbon, create a rectangle approximately 16 meters long in the X direction and 6 meters long in the Z direction.
- Using the Rectangle tool, create another rectangle approximately 16 meters long in the X direction and 6 meters long in the Z direction atop the first rectangle.
- In 3D perspective view, select one of the rectangles and hold the shift key to drag the rectangle up approximately 10 meters in the Y direction.
- In the Drawing tab, click the Color drop down and select a dark gray. Click the top platform to change its color. Repeat this for the bottom platform.
- Right-click on the top platform and click Lock Edits to ensure the platform is not moved. Repeat this for the bottom platform.
- Drag two ModelEntities into the Model, name one "Floor1_Employee" and the other "Floor2_Employee". Add a different, easily distinguishable symbol to each ModelEntity by selecting the ModelEntity and clicking Apply Symbol in the Symbols ribbon, then selecting a symbol.
- Create a Sink on the upper-right side of the bottom platform. Change its *Name* to 'Floor1_Sink'. Create a Sink on the upper-right side of the top platform. Change its *Name* to 'Floor2_Sink'.
- Create a Source on the upper-left side of the bottom platform. Change its *Name* to 'Floor1_Source' and its *Entity Type* property to 'Floor1_Employee'. Select its Output Node and change its *Outbound Travel Mode* property to 'Network Only', its *Entity Destination Type* to 'Specific', and its *Node Name* to 'Input@Floor1_Sink'.
- Create a Source on the upper-left side of the top platform. Change its *Name* to 'Floor2_Source' and its *Entity Type*

property to 'Floor2_Employee'. Select its Output Node and change its *Outbound Travel Mode* property to 'Network Only', its *Entity Destination Type* to 'Specific', and its *Node Name* to 'Input@Floor2_Sink'.

Elevator Setup

- Create two ElevatorNodes on the bottom edge of the bottom platform and change their names to 'Floor1_E1_ElevatorNode' and Floor1_E2_ElevatorNode.
- Create two ElevatorNodes on the bottom edge of the top platform and change their names to 'Floor2_E1_ElevatorNode' and Floor2_E2_ElevatorNode.
- In the Lists view of the Definitions tab, create a new Node List called "E1_ElevatorNodeList". Add ElevatorNodes Floor1_E1_ElevatorNode and Floor2_E1_ElevatorNode to the List.
- Create another Node List called "E2_ElevatorNodeList". Add ElevatorNodes Floor1_E2_ElevatorNode and Floor2_E2_ElevatorNode.
- Create another Node List called "Floor1_ElevatorNodeList". Add Floor1_Bank1_ElevatorNode and Floor1_Bank2_ElevatorNode to the List.
- Create another Node List called "Floor2_ElevatorNodeList". Add Floor2_Bank1_ElevatorNode and Floor2_Bank2_ElevatorNode to the List.
- Place an ElevatorSelectorNode in the center of the bottom platform, then change its *Name* to 'Floor1_ElevatorSelectionNode' and its *Elevator Node List* to 'Floor1_ElevatorNodes'.
- Place an ElevatorSelectorNode in the center of the top platform, then change its *Name* to 'Floor2_ElevatorSelectionNode' and its *Elevator Node List* to 'Floor2_ElevatorNodes'.
- Create an Elevator adjacent to Floor1_Bank1_ElevatorNode, then change its *Name* to 'Elevator1', its *Initial Ride Capacity* to '3', its *Minimum Dwell Time Type* to 'Specific Time', its *Minimum Dwell Time* to '0.2' minutes, its *Elevator Node List* to 'E1_ElevatorNodeList', and its *Initial Node* to 'Floor1_E1_ElevatorNode'.
- Create another Elevator adjacent to Floor1_E2_ElevatorNode, then change its *Name* to 'Elevator2', its *Initial Ride Capacity* to '3', its *Minimum Dwell Time Type* to 'Specific Time', its *Minimum Dwell Time* to '0.2' minutes, its *Elevator Node List* to 'E2_ElevatorNodeList', and its *Initial Node* to 'Floor1_E2_ElevatorNode'.
- On each ElevatorNode, change the *Associated Elevator* property to its corresponding Elevator and the *Elevator Destination Type* to 'Entity Destination'.
- Create Paths on each platform connecting the Source to the ElevatorSelectorNode, The ElevatorSelectorNode to the ElevatorNodes, and each ElevatorNode to the Sink. Set the *Allow Passing* property on each Path to 'False'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExtrasLibraryLiftTruckRack - SimBit

This SimBit project includes two models providing examples and use cases for the Rack and Lift Truck object definitions included in the Extras Library.

1. LiftTruckRack_StorageTimeRelease - Demonstrates the basics of creating Entities on a Rack and moving Entities into and out of the Rack using the Lift Truck.
2. LiftTruckRack_InterruptRelease - Demonstrates how to retrieve specific Entities stored on a Rack.

Model 1: LiftTruckRack StorageTimeRelease

Problem:

I have a system where boxes (Entities) are being moved from storage (Rack) to a staging area by a forklift (LiftTruck). The forklift must move its lift up and down to place and retrieve boxes on/off the shelves on each Rack.

Keywords:

Extras Library, Lift Truck, Rack, Storage and Retrieval

Categories:

Composite Object, Extras Library, Transporter

Assumptions:

The modeled system has two storage Racks, one for storage and one for staging. Each Rack contains five shelves, and each shelf has enough space to hold up to three boxes. The system is initialized with 15 boxes on the Storage Rack, and each box spends a random uniform time between 0 and 5 minutes on their shelf before requesting a ride via the forklift to the Staging Rack. Once placed on a shelf at the staging rack, each box waits a random uniform time between 1 and 3 minutes before departing the Rack and travelling along a Path to a Sink where it exits the system. The forklift that moves the boxes between the two Racks can carry one box at a time and always selects the lowest shelf with available capacity when placing the box at staging.

Technical Approach:

The system can be modelled using the Rack and LiftTruck objects from the Extras Library. The Storage Rack will be initialized with 15 boxes. Note that the Rack object is a composite object made up of a Rack frame (Fixed Object) and Rack Shelves (dynamically created Entities). Initially, the Rack instance has a single Shelf; however, when the model is initialized additional Shelves are created and added to the Rack based on the Rack's *Number of Shelves* property.

The LiftTruck is also a composite object made up of the Truck (Transporter) and the separate LiftDevice (Entity). The LiftDevice will automatically move up and down as necessary to store and retrieve entities on a Rack.

Details for Building the Model:

- Place a Rack approximately 10 meters left of the center and change its *Name* to 'Storage'. Set its *Number of Shelves* to '5', its *Shelf Capacity* to '3', its *Shelf Storage Time* to 'Random.Uniform(0, 5)' minutes, and its *Initial Quantity* to '15'. Move its InputNode to the left and place its OutputNode towards the left corner.
- Place a Rack approximately 10 meters to the right of the center and change its *Name* to 'Staging'. Set its *Number of Shelves* to '5', and its *Shelf Storage Time* to 'Random.Uniform(1, 3)' minutes.
- Place the DefaultEntity into the model. Select the DefaultEntity and use Apply Symbol to change its symbol to 'Box2'. This symbol can be found in the Library\Containers section.
- Place a LiftTruck in the model, then set its *Network Turnaround Method* to 'Rotate In Place' and its *Initial Node* to 'Input@Storage'.
- Select the Output@Storage Node, then set its *Ride On Transporter* property to 'True' and its *Transporter Name* to 'LiftTruck1'.
- Create a Path between Output@Storage and Input@Staging and change its *Type* to 'Bidirectional'.

- Create a Sink to the right of the Staging Rack.
- Create a Path connecting Output@Staging to Input@Sink1.

Model 2: LiftTruckRack InterruptRelease

Problem:

I have a system where two types of parts are stored on a single Rack until required for processing. When a third type of part arrives at the assembly station, the parts should be retrieved from the rack and transported to the station.

Keywords:

Extras Library, Interrupt Step, Lift Truck, Rack, Search Step, Storage and Retrieval

Categories:

Composite Object, Extras Library, Transporter

Technical Approach:

Use three different types of ModelEntities to represent the three different types of parts. Create a Rack to store Entities as they arrive in the system, and a LiftTruck to transport entities from the Rack.

Details for Building the Model:

Storage System

- Place a Rack in the center of the Facility view. Change its *Name* to 'Storage', its *Number of Shelves* to '4', its *Shelf Capacity* to '5', its *Height of First Shelf* to '0.5' meters, and its *Shelf Spacing* to '1.2' meters.
- Place 3 ModelEntities into the model.
- Change the *Name* of the first ModelEntity to 'PartA'.
- Change the *Name* of the second ModelEntity to 'PartB' and use the Color tool in the Symbols ribbon to change its color to red.
- Change the *Name* of the second ModelEntity to 'PartC' and use the Color tool in the Symbols ribbon to change its color to blue.
- Place a Source to the left of the Rack. Change its *Name* to 'PartB_Source', its *Entity Type* to 'PartB', and its *Interarrival Time* to 'Random.Exponential(1.5)' minutes.
- Place another Source to the left of the Rack. Change its *Name* to 'PartC_Source', its *Entity Type* to 'PartC', and its *Interarrival Time* to 'Random.Exponential(1.5)' minutes.
- Create Paths connecting the Source OutputNodes to the Rack's InputNode.

Assembly System

- Create new Source approximately 10 meters below and to the left of the Rack. Change its *Name* to 'PartA_Source', its *Entity Type* to 'PartA', and its *Interarrival Time* to 'Random.Exponential(1.35)' minutes.
- Place a Combiner to the right of PartA_Source. Change its *Name* to 'Assembly1', its *Must Simultaneously Batch* property to 'True', and its *Processing Time* to 'Random.Exponential(0.5)' minutes.
- Create a Path connecting Output@PartA_Source with ParentInput@Assembly1.
- Create another Combiner to the right of Assembly1. Change its *Name* to 'Assembly2', its *Must Simultaneously Batch* property to 'True', and its *Processing Time* to 'Random.Exponential(0.5)' minutes.
- Create a Path connecting Output@Assembly1 to ParentInput@Assembly2.
- Place a Sink to the right of Assembly2.
- Create a Path connecting Output@Assembly2 with Input@Sink1.

Routing and Lift Truck

- Create a BasicNode approximately 5 meters below the Rack and change its *Name* to 'HomeNode'.
- Place a LiftTruck into the model. Change its *Initial Node* to 'HomeNode' and its *Idle Action* to 'Go To Home'.
- In the Lists view of the Definitions tab, create a new NodeList. Change its *Name* to 'AssemblyNodeList' and add the "MemberInput@Assembly1" and "MemberInput@Assembly2" to the list.
- Select the Output@Storage TransferNode.
- Change its Entity Destination Type to 'Select From List' and its Node List Name to 'AssemblyNodeList'. Set its Selection Condition to 'Math.If(Is.PartB && Candidate.Node == MemberInput@Assembly1, True, Is.PartC && Candidate.Node == MemberInput@Assembly2, True, False)'.
- Note: This expression be checked for each Node in the NodeList, and only if it resolves to 'True' will an entity be allowed to that location. This expression causes all PartB entities to be routed to MemberInput@Assembly1 and all PartC entities to be routed to MemberInput@Assembly2.
- Change its *Ride On Transporter* property to 'True', and its *Transporter Name* to 'LiftTruck1'.

Add-on Process Logic

- In the Processes tab, create a new process named "RetrieveComponentB".
- Set the *Triggering Event Name* to 'ParentInput@Assembly1.Entered'.
- Add a new Search step. Change its *Name* to 'PartB', its *Entity Type* to 'PartB', its *Match Condition* to 'Candidate.ModelEntity.Location.Parent.Is.Shelf', and its *Save Number Found* property to 'Token.ReturnValue'.
- On the Found branch after the Search step, place an Interrupt step. Change its *Name* to 'StorageDelay', its *Process Name* to 'ModelEntity.Location.Parent.Shelf.EntityArrived', its *Selection Rule* to 'Smallest Value First', its *Filter Expression* 'Candidate.Entity.Name == ModelEntity.Name', and its *Interrupted Process Action* to 'EndDelay'.
- On the Original branch after of the Search step, place a Decide step. Change its *Name* to 'EntityFound' and its *Condition or Probability* to 'Token.ReturnValue > 0'.
- On the False branch of the Decide step, place a Wait step. Change its *Name* to 'EntityEnteredRack', and its *Event Name* to 'Input@Storage.Exited'.
- After the Wait step, place a Delay step. Change its *Name* to 'Epsilon' and its *Delay Time* to 'Math.Epsilon'.
- Connect the Delay step's exit to the "PartB" Search step.
- In the Processes tab, create a new process named "RetrieveComponentC".
- Set the *Triggering Event Name* to 'ParentInput@Assembly2.Entered'.
- Add a new Search step. Change its *Name* to 'PartC', its *Entity Type* to 'PartC', its *Match Condition* to 'Candidate.ModelEntity.Location.Parent.Is.Shelf', and its *Save Number Found* property to 'Token.ReturnValue'.
- On the Found branch after of the Search step, place an Interrupt step. Change its *Name* to 'StorageDelay', its *Process Name* to 'ModelEntity.Location.Parent.Shelf.EntityArrived', its *Selection Rule* to 'Smallest Value First', its *Filter Expression* 'Candidate.Entity.Name == ModelEntity.Name', and its *Interrupted Process Action* to 'EndDelay'.
- On the Original branch after of the Search step, place a Decide step. Change its *Name* to 'EntityFound' and its *Condition or Probability* to 'Token.ReturnValue > 0'.
- On the False branch of the Decide step, place a Wait step. Change its *Name* to 'EntityEnteredRack', and its *Event Name* to 'Input@Storage.Exited'.
- After the Wait step, place a Delay step. Change its *Name* to 'Epsilon' and its *Delay Time* to 'Math.Epsilon'.
- Connect the exit from the Delay step to the "PartC" Search step.

ExtrasLibraryRobot - SimBit

This SimBit project includes two models providing examples using the Robot object definition included in the Extras Library.

1. Robot_Transporter – Demonstrates how to use the Robot as a Transporter to transport entities between locations.
2. Robot_SecondaryResource – Demonstrates how to use the Robot as a secondary resource.

Model 1: Robot Transporter

Problem:

I have a system where entities are being moved by a Robot from one location to another.

Keywords:

Composite Object, Extras Library, Robot, Transporter

Categories:

Composite Object, Extras Library, Transporter

Assumptions:

Entities enter the system at a Source, travel to a TransferNode serving as a pickup location, then are moved through free space by a Robot to a BasicNode representing a drop-off location. The Entities then travel to an Input@Sink Node.

Technical Approach:

Place a Robot in the center of the model and a BasicNode nearby to serve as the Initial Node for the Robot Hand. Use a second BasicNode nearby to serve as the destination for Entities that are moved by the Robot. A TransferNode placed near the Robot can serve as the pickup point for Entities.

Note: The Robot is a composite object made up of a Robot Base (Fixed Object), Rotator (Entity), Lower Arm (Entity), Upper Arm (Entity), and Robot Hand (Transporter). To edit the basic Robot properties, click on the Robot Base. To edit the Transporter-related properties, click on the Robot Hand.

Details for Building the Model:

Facility Window Setup

- Using the 2D view, place a Robot in the Facility view with the center base acting as the center of the model.
- Place a BasicNode approximately located 2 meters to the right of the robot center and change its *Name* to 'HomeNode1'.
- Place a BasicNode located approximately 2 meters below the center of the robot and change its *Name* to 'DropNode1'.
- Place a TransferNode located approximately 2 meters above the center of the robot and change its *Name* to 'PickupNode1'. Set its *Ride On Transporter* property as 'True', its *Transporter Name* to 'RobotHand1', its *Entity Destination Type* to 'Specific', and its *Node Name* to 'DropNode1'.
- Select the base of the Robot. Set the Robot's *Initial Node* property to 'HomeNode1'.
- Select the RobotHand and set its *Idle Action* property to 'Go To Home'
- Refer to the Note in the Technical Approach section for more details regarding the Robot's composite design.
- Place a Source to the left of PickupNode1 and change its *Interarrival Time* to 'Random.Exponential(6)' minutes.
- Place a Sink to the right of DropNode1.
- Use Paths to connect Output@Source1 to PickupNode1 and DropNode1 to Input@Sink1. Set *Allow Passing* on both

Paths to 'False'.

Model 2: Robot_SecondaryResource

Problem:

I have a system where a Robot is required for processing at two different stations.

Keywords:

Add-On Process, Composite Object, EndTransfer Step, Extras Library, Robot, Secondary Resource

Categories:

Add-On Process Logic, Composite Object, Extras Library

Assumptions:

Entities arrive at two different lines, each corresponding to a different station. A single Robot completes the tasks at each station. Because the Robot completes tasks at more than one location, this Robot moves faster than standard robots. Its upper and lower arms move at an increased rate of 5 degrees per second, and the base rotates at 10 degrees per second.

Technical Approach:

Place a Robot adjacent to two Servers and require the Robot for processing at each Server. Adjust the Input Nodes, Output Nodes, and Processing Stations to enhance the Robot animation.

Note: The Robot is a composite object made up of a Robot Base (Fixed Object), Rotator (Entity), Lower Arm (Entity), Upper Arm (Entity), and Robot Hand (Transporter). To edit the basic Robot properties, click on the Robot Base. To edit the Transporter-related properties, click on the Robot Hand.

Details for Building the Model:

Facility Window Setup

- In the 2D view, place a BasicNode approximately 2 meters left of the center of the Facility view and change its *Name* to 'HomeNode'.
- Place a Robot in the center of the Facility view. Change its *Initial Node* to 'HomeNode'.
- Select its Rotator and change its *Rotation Rate* to '10'.
- Select its LowerArm and change its *Pitch Change Rate* to '5'.
- Select its UpperArm and change its *Pitch Change Rate* to '5'.
- Place a Server approximately 3.5 meters above the Robot, changing its *Name* to 'Station1' and its *Input Buffer Capacity* to '0'. In the *Secondary Resources* property group, set the *Repeat Group* property to 'False', the *Resource Name* to 'RobotHand1', the *Request Move* property to 'To Node', and the *Destination Node* to 'Server.Input'.
- Place a Server approximately 3.5 meters below the Robot, changing its *Name* to 'Station2' and its *Input Buffer Capacity* to '0'. In the *Secondary Resources* property group, set the *Repeat Group* property to 'False', the *Resource Name* to 'RobotHand1', the *Request Move* property to 'To Node', and the *Destination Node* to 'Server.Input'.
- Place a Source to the left of each Workstation. Change the *Interarrival Time* of each Source to 'Random.Exponential(1.5)' minutes.
- Place a Sink to the right of each Station Server.

Animation Enhancements

- Create Paths from each Source to the corresponding Station, and each Station to the Sink. Set each Path's *Allow Passing* property to 'False', its *Speed Limit* to '0.5' meters per second, and its *Width* to 1 meter.
- Note that the *Width* property of the Path is in the General property group in the Physical Characteristics section.
- Apply a Conveyor Path Decorator to each Path.
- Path Decorators are found in the Edit ribbon when the Path is selected.
- Move Station1's Input Node to 2 meters above the Robot, such that the Input Node is closer to the Robot than the

Server. Do the same for Station1's Output Node.

- Move Station1's Processing.Contents queue such that the queue is parallel to the Path and the left side of the queue is on top of the Input and Output Nodes.
- Move Station2's Input Node to 2 meters below the Robot, such that the Input Node is closer to the Robot than the Server. Do the same for the Station2's Output Node.
- Move Station2's Processing.Contents queue such that the queue is parallel to the Path and the left side of the queue is on top of the Input and Output Nodes. In the Processes tab, create a new process and change its *Name* to 'EndTransfer'.
- Place a single EndTransfer step in the EndTransfer process.
- In the Facility view, set the EndTransfer process to trigger at the *Entered* Add-On Process Trigger of each Workstation Server.

Note:

The final three steps in the Animation Enhancements section in which the EndTransfer process is created smooth the entities' transition from the Link to the Processing station. When an entity skips a Server's input buffer, the order of execution of the Server's OnEnteredProcessing process causes entities to briefly appear simultaneously at the end of the preceding Link and in the Server's Processing station. Creating an Add-On Process to immediately end the Transfer of Entities arriving from the Link eliminates this brief doubling effect.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FacilityModelWithinModel - SimBit

Problem:

You have a facility in which many of the operations are similar. You wish to make a single “object” that encompasses 3 serial processes into a single process for simplicity.

Categories:

Building New Objects / Hierarchy

Key Concepts:

BasicNode, Expression Property, External View, ExternalNode, Server, To Parent External Node, Bind To External Input Node, Bind To External Output Node, Externally Visible

Assumptions:

The similar processes that will be made into an object include a single Server (ProcessA), which then moves entities to one of 2 more Servers (Process B), followed by a fourth Server (ProcessC).

Technical Approach:

Within the first model, called ProcessABC, we will define the processing necessary for this single object. It includes ProcessA, ProcessB_1 and ProcessB_2, and ProcessC. All logic for these Servers will be assumed unchangeable except for the Processing Time property for each. Therefore, when this ProcessABC model is placed into the Facility Window of another model, the user must only specify the four processing times for the various Servers. The External Window is used to define the view of the model, including the input and output nodes.

Details for Building the Model:

Simple System Setup

- Rename Model to ProcessABC by clicking on MySimioProject, highlighting the Models panel and clicking on Model. Pressing F2 (or clicking again on Model) will put you in rename mode.
- Place 4 Servers within the Facility Window of ProcessABC and rename them to ProcessA, ProcessB_1, ProcessB_2 and ProcessC. Connect them with Paths – ProcessA to both ProcessB_1 and ProcessB_2, and then both ProcessB_1, ProcessB_2 connected to ProcessC.

Defining the Properties of a Hierarchical Model

- Highlight each Server and use the Processing Time property to indicate that the data for this processing time be given in the parent object. This is done by highlighting the Processing Time property (it will turn blue) and using the right click to select “Set Referenced Property” and then select “Create New Referenced Property”. This will open a dialog to specify the name of the property in the parent ProcessABC that the user will be able to edit (i.e., ProcessTimeA). Once you do this, the field will have the name of the property with a small green arrow in front of it. This means that the value will be “inherited” from the parent object ProcessABC when it is placed in a model. Do this will all four Servers.

Defining the External View of Model

- From the Facility window, right click on the Input Node of ProcessA and select *Bind To New External Input Node*. You will be prompted to name the new External Node. Keep the default name of Input_ProcessA. This is the node where entities will enter into this object. They will automatically enter this object and go directly into ProcessA.
- Similarly, right click on the Output Node of ProcessC and select *Bind To New External Output Node*. You will be prompted to name the new External Node. Keep the default name of Output_ProcessC. This is the node where entities will exit this object. They will automatically exit this object when they enter into the Output node of ProcessC.

Using the Model within Another Model

- Open a new model and within its Facility Window, place a ProcessABC object from the Project Library.
- Select the Source and Sink objects from the Standard Library and place them in the Facility Window.
- Connect the Source1 to ProcessABC and then ProcessABC to the Sink1 using Paths. Notice when you highlight ProcessABC, you have access to all the processing times, namely *ProcessTimeA*, *ProcessTimeB_1*, *ProcessTimeB_2*

and *ProcessTimeC*. When you run this new model, the logic behind ProcessABC then includes all logic specified within that model.

Discussion:

Notice that when you placed the Server objects into the ProcessABC model, they automatically appeared as part of the External View of that object. If you right click on each Server (ProcessA, ProcessB, etc) in the Facility window of model ProcessABC, you'll see that the option, *Externally Visible* is selected by default. This means that these objects will be visible in the External View of this object (will be seen when this object is placed inside another model). If you unselect *Externally Visible* from the right click menu of an object, it will no longer appear in the External View of that object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FileSecurity - SimBit

Problem:

I would like to password protect my model.

Categories:

File Management

Key Concepts:

Add-On Process, Data Table, Dynamic Object Property, Expression Property, Numeric Property, On Creating Entities, Password Protect, RandomRow, SetRow Step, Source, String Property

Assumptions:

The password to open the object named MyModel is 'simio'.

Technical Approach:

A password will be added to MyModel by using the Protection feature.

Details for Building the Model:

Adding a Password to MyModel

- Click on MySimioProject in the Navigation window and select the Models panel.
- Highlight the model that you would like to password protect and click on the Protect button in the Edit ribbon.
- Enter 'simio' or a desired password into the field.
- Upon reopening the project file, the correct password will have to be entered to gain access.
- You can later unprotect the model by selecting the Unprotect button if desired.

Embellishments

This method of protection can also be used for any model object in the Navigation window.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Financials - SimBit

Problem:

I would like my model to calculate costs, such as capital costs of objects, usage costs, holding costs and the cost of transporting an Entity on a Vehicle.

Categories:

Add-On Process Logic, Decision Logic – Paths, Entity Characteristics, Servers, Vehicles

Key Concepts:

Add-On Process, Assign Step, Bidirectional Path, Cost Center, Financials, InputBuffer, ModelEntity, Entered, Pivot Grid, Processing, Ride on Transporter, Selection Weight, Server, Vehicle

Technical Approach:

This model gives examples of how to use a few different costing features in Simio. One object demonstrates the Capital Cost property, another uses the Holding Cost property which calculates the cost to hold an Entity in the Input Buffer of this object. The model also shows how the user can manually apply costs with an Assign step. It also contains a Vehicle which has costs associated with each Entity that rides on the Vehicle. All the costs in this model are rolled up into one of four Cost Centers.

Details for Building the Model:

Simple System Setup

- Place a Source object and change the *Interarrival Time* property to 'Random.Exponential(.8)
- Place a Server object and change the *Name* of the Server to 'InspectionA', connect the Source to the Server with a Path.
- Place a Sink object and name it FailedInspection. Connect the Server 'InspectionA' to this FailedInspection Sink with a Path. Set the *Selection Weight* property to '.3'.
- Place a second Server object and *Name* it 'Paint'. Connect the Paint Server to the InspectionA Server with a Path and set the *Selection Weight* property to '.7'
- Place a Sink object and *Name* it 'Sink1'. Connect the Paint Server to this Sink with a Path. Change the *Type* property on the Path to 'Bidirectional'.
- Place a Vehicle object. Set its *Initial Home* property to 'Output@Paint' and change its *Name* to 'Car'.
- Click on the TransferNode of the Paint Server (Output@Paint) and within the Transport Logic section of properties, change *Ride on Transporter* to 'True' and *Transporter Name* to 'Car'.

Configure Costing of InspectionA

- Select InspectionA (Server) and open the Financials property category. View the drop-down selection list for the *Parent Cost Center* property by clicking on the down arrow that appears when this property is selected.
 - Create a new Cost Center by selecting 'Create New' from the drop-down. Name this new Cost Center 'InspectionCostCenter'. All the costs incurred at this Server will be rolled up to this new Cost Center.

Configure Costing of Paint

- Select Paint (Server) and change the *Processing Time* property to 'Random.Triangular(.2, .3, .4)'. Open the Financials property category. View the drop-down selection list for the *Parent Cost Center* property by clicking the down arrow that appears when this property is selected.
 - Create a new Cost Center by selecting 'Create New' from the drop down. Name this new Cost Center 'PaintCostCenter'. All the costs incurred at this Server will be rolled up to this new Cost Center.
 - Set the *Capital Cost* property to '1,000' USD. This is a one-time cost of \$1,000 to just own this Server.
 - Expand the Buffer Costs subcategory and expand the Input Buffer category. Set the *Holding Cost Rate* (under Input Buffer) to '200' USD per Hour. This is a \$200 cost for each hour that an Entity is held in the Input Buffer of this Server.

Configure Costing of Vehicle

- Select the Vehicle object and set the *Parent Cost Center* property to a new Cost Center named 'TransportationCosts'.

Expand the Transport Costs property category and set the *Cost Per Rider* property to '10' USD. This cost will be incurred each time a rider is loaded onto the Vehicle.

Configure Costing of Failed Inspections

- Create a new Cost Center element by going into the Definitions window/Elements panel. Click on the Cost Center element icon in the ribbon. Name this new Cost Center, 'CostOfFailedParts'.
- Select the Path that connects the Server InspectionA to the Sink called FailedInspection. Double-click on the Add-On Process trigger property *Entered* to create a new Add-On process.
 - Place an Assign Step in the process and set the *State Variable Name* to 'CostOfFailedParts.Cost' and the *New Value* to 'CostOfFailedParts.Cost + 150'. Therefore, each time an Entity enters this path and therefore executes this process, \$150 will be added to the Cost Center CostOfFailedParts. This is an example of how to assign cost manually within process logic.

Configure Costing of Production:

Production cost is comprised of costs incurred by Painting and Inspecting, so this Cost Center is a roll up of those two Cost Centers.

- Create a new Cost Center element by going into the Definitions window/Elements panel. Click on the Cost Center element icon in the ribbon. Name this new Cost Center 'ProductionCost'.
- Select 'InspectionCostCenter' and 'PaintCostCenter' and change their *Parent Cost Center* to 'ProductionCost'.

Configure Cost on the Entity:

- Place a ModelEntity object into the Facility window. Expand the Financials property category and set the *Initial Cost Rate* property to '100' USD per Hour. This is the cost per hour for this Entity.

Explanation of Results

- Set the *Ending Type* of the model to '10 hours', on the Run tab of the ribbon menu. Run the model until the end. Click on the Results Tab to view the Pivot Grid.
- If you view the Model object type category, you will see the breakdown of costs into each of the 4 cost centers; CostOfFailedParts, InspectionCostCenter, PaintCostCenter, TransportationCosts. You will also see a grand total of costs at the Model object level.
- To see details of the CostOfFailedParts cost, find the results for the Path2 (or the name of the Path where the Add On Process is located which assigns the cost). Note the Total Number Entered in the Throughput category. It is 235. Multiply this by the cost for entering that path (\$150), you get the total of \$35,250, which is the total cost of the CostOfFailedParts cost center.
- To see the details of the InspectionCostCenter, find the results for the InspectionA Server.
 - You'll first notice costs in the Input Buffer station of this object. This comes from the cost on the Entity object. The cost of the Entity, when it is located at this Server's InputBuffer, is rolled up into the InspectionCostCenter. Note the Average Number in Station (.0479) and multiply this by the Cost Rate on the Entity object (\$100) and also by the total simulation time (10 hrs). This is a total of \$47.8816, which is shown in the Total Cost of this InputBuffer.
 - Also notice that there are costs for the Processing station of this Server. This cost comes from the cost on the Entity object. The cost of the Entity, when it is located at this Server's processing station, is rolled up into the InspectionCostCenter. Note the Average Number in Station for processing (.2619) and multiply this by the Cost Rate on the entity object (\$100) and also by the total simulation time (10 hrs) and this gives \$261.9343, which is shown in the Total Cost of this processing station.
- To see the details of the PaintCostCenter, find the results for the Paint Server. We had put a \$1,000 value in the Capital Cost for this object and this is shown in the Capital Cost Data Item of this object.
 - We also put \$200 USD per Hour in the Holding Cost for the Input Buffer of this object. The cost of the Entity also needs to be included, which is \$100 USD. Note the Average Number in Station is .0270. Multiply \$300 times .0270 and then by the 10 simulation hours and you get the total cost of this Input Buffer station, \$80.8547.
 - There is also cost in the Processing Station, which is the cost of the Entity at this object. Note the Average Number in Station is .2777. Multiply that by 10 simulation hours and \$100 entity cost and you get the \$277.7135 shown in the total for this station.
- To see the details of the TransportationCosts, see the Vehicle object type results. There are two costs being rolled into this cost center.
 - The first is the Transport Cost that we defined, which is \$100 for every rider. Note the Number Entered in the Throughput category of the RideStation is 555. Multiply this by \$100 and you get a total of \$5550.00.
 - The other cost is the cost of the Entity. Note the Average Number In Station of this RideStation is .04622.

Multiply this by \$100 (Entity cost) and by the 10 simulation hours and you get 46.2. Once added to \$5550, the total cost of \$5596.25 (shown in the results) is reached.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FindAMinimumStateValue - SimBit

Problem:

I have 3 servers and want to keep track of the total processing time allocated to server (including entity at the server and those en route) with a state variable. I want to then find the minimum value to the various servers to even out the load to each throughout the simulation.

Categories:

Add-On Process Logic, Decision Logic -- Processing, Discrete States, Entity Characteristics

Key Concepts:

Add-On Process, Assign Step, Data Table, Find Step, SetNode Step, State Variable, Status Label

Assumptions:

We assume that the method for allocating entities to the various servers is based on the smallest processing time sent to any given server. Each entity will be assigned a processing time when it is created.

Technical Approach:

Use a state on the entity to assign the entity processing time upon its creation. Use a state variable (vector) that stores the total processing time for each given server based on the entity at the server and those en route to the server. Use the Find step to determine which of the three servers has had the least amount of processing time allocated to it. The first value within the state variable will refer to the first server, Server1, and so on. We will also store the input node for each of the three servers within a table, so that we can use the SetNode step to set the appropriate destination after the Find step.

Details for Building the Model:

Adding the ModelEntity ProcessingTime

- Within the Navigation window, click on the ModelEntity and go to the Definitions tab, States panel.
- Add a real state variable named ProcessingTime, which will be referred to as ModelEntity.ProcessingTime in the model.

Simple System Setup

- Within the Navigation window, click on the Model and go to the Facility window.
- Add a Source, three Servers and a Sink to the Facility window.
- Connect the Source to each of the three Servers and each Server to the Sink with Paths.

Modifying the Source

- Change the *Interarrival Time* to 'Random.Exponential(.05)'.
- Within the Before Exiting section of the State Assignments, add an assignment to the *State Variable Name* 'ModelEntity.ProcessingTime' a *New Value* of 'Random.Uniform(.1,.3)'.
- Click on the output node and go to the *Entered* add-on process. Double-click on *Entered* to add the process 'Output_Source1_Entered'.

Adding a State Variable Vector

- Within the Definitions window, States panel, add an integer type variable named 'Index'. This will be used within the Find step itself. This variable will remain as *Dimension Type* of 'Scalar'.
- Add a real state variable named 'ServerProcessingTime'. Change the *Dimension Type* to 'Vector' and the *Rows* to '3'. These values are then referenced ServerProcessingTime[1], ServerProcessingTime[2] and ServerProcessingTime[3].

Using the Find Step

- Within the Processes window, in the new process that was added above ('Output_Source1_Entered'), add a Find step. This step requires an index variable (which could be any name, but we have just created one named 'Index' above). Set the *Index Variable Name* to 'Index'. The value of this variable will be set to the item that is 'Found' (if any). Keep the *Starting Index* as '1' and change the *Ending Index* to '3'. This will then allow us to search the 3 vector values of the ServerProcessingTime variable.
- Change the *Search Type* to 'Minimize Expression'. Set the *Search Expression* to 'ServerProcessingTime[Index]'. This will allow the Find step to look at all three values of the ServerProcessingTime variable and find the minimum value. When the simulation first starts, all values are 0 and thus, the first one will be found.

Using the Index Value from the Find Step

- After the Find step, from the 'Found' exit, place an Assign step and a SetNode step. Within the Assign step, set the *State Variable Name* to 'ServerProcessingTime' and the *Row* to 'Index'. Input the *New Value* as 'ServerProcessingTime[Index] + ModelEntity.ProcessingTime'. Remember that the value 'Index' will be either 1, 2 or 3 (starting to ending index) based on the search type and expression specified. Therefore, with this Assign step, we are increasing the value of whichever minimum variable was found by the entity's processing time that was assigned in the Source.
- Within the SetNode, we will be leaving the *Destination Type* as 'Specific' and now need to determine how to specify the *Node Name* for the corresponding server to where we'd like the entity to move.

Using a Data Table to Store Input Nodes for Possible Destinations

- Because the variable 'Index' will return a value between 1-3, we will set up a data table that will have corresponding input nodes for the three Servers. We'll then index into the table within the SetNode step. To do this, go to the Data window, Tables panel and add a Data Table named 'Table1'. Add an Object Reference type column of type Node. Change the *Name* of the column to 'WhichServer'. Input the three input nodes for the servers, such as Input@Server1, Input@Server2 and Input@Server3. We can then reference the correct row by the variable 'Index'.
- Go back to the Processes window and within the SetNode step, set the *Node Name* to 'Table1[Index].WhichServer'. This will set the entity destination to the appropriate server. Because the *Entity Destination Type* on the Output@Source1 node is 'Continue', the entity will use its node destination to know where to go next.

Using ModelEntity.ProcessingTime in the Server

- Within each of the three servers, change the *Processing Time* property to 'ModelEntity.ProcessingTime' to use the value that was assigned to each individual entity.
- Within the Before Exiting section of the State Assignments of each server, we will decrease the state variable vector for the particular server. Set the *State Variable Name* to 'ServerProcessingTime'. For Server1, the *Row* would be '1'; for Server 2, the *Row* would be '2' and so on. The *New Value* should be 'ServerProcessingTime[1] - ModelEntity.ProcessingTime' (for Server1, with the value in brackets reflecting the server number). This will then decrease the state variable after the entity is processed.

Enhancements:

Add a Status Label next to each of the Servers that references the state variable vector for each (such as ServerProcessingTime[1], etc.). This way, you can see that the incoming entity is sent to the one with the minimum value of that vector. Additionally, on the DefaultEntity placed in the Facility window, attach a status label that shows the ModelEntity.ProcessingTime value as it moves through the system.

FlowConcepts - SimBit

This SimBit project includes eight models that demonstrate the use of the Simio Flow Library. Models included in this SimBit:

1. **SimpleFlow** – Demonstrates flow from a FlowSource object, through a FlowConnector object, to a FlowSink object.
2. **FillingEmptyingTank** – Demonstrates filling and emptying a tank object.
3. **TransferringFromOneTankToAnother** – Demonstrates transferring from one tank object to another.
4. **MergedFlow** – Demonstrates merging two different entity flows.
5. **SplitFlow** – Demonstrates splitting a single entity flow into two entity flows.
6. **CreateDiscreteEntitiesBasedOnFlow** – Demonstrates the use of monitors and events to create discrete entities when a tank reaches a certain flow level.
7. **SimpleFiller** – Demonstrates the use of the ContainerEntity and Filler objects to illustrate a filling operation of combining discrete containers with continuous flow.
8. **SimpleEmptier** – Demonstrates the use of the Emptier object to illustrate full container entities with multiple products emptying into multiple flow streams based on their product type to flow sink objects.

Model 1: SimpleFlow

Problem:

I want to create an entity flow from a FlowSource object, through a FlowConnector, to a FlowSink object. In addition, I want to monitor the flow volume and stop the flow at 5 cubic meters of total volume.

Categories:

Flow Library

Key Concepts:

FlowSource, FlowSink, FlowConnector

Assumptions:

Flow rate and stopping condition are in units of cubic meters.

Technical Approach:

Create entity flow using a FlowSource and send it directly to the FlowSink through a FlowConnector. Terminate the flow using the Maximum Volume stopping condition on the FlowSource.

Details for Building the Model:

Simple System Setup:

- Place a FlowSource and FlowSink from the Flow Library in the Facility window.
- Connect the source and sink using a FlowConnector.
- Set the *Maximum Volume* property of the FlowSource object to '5' cubic meters.
- Place a Status Label and set its *Expression* property to 'Output@FlowSource1.FlowRegulator.CurrentMaximumFlowRate'.
- Place a Status Label and set its *Expression* property to 'Output@FlowSource1.FlowRegulator.CurrentVolumeFlowOut'.
- Place a ModelEntity from the Project Library into the model and change the color to whatever color you'd like the flow to be.

Model 2: FillingEmptyingTank

Problem:

I want to fill a tank until the tank reaches the high mark. Once reaching the high mark, the tank should empty until it reaches the low mark, at which point it should repeat the fill-empty cycle.

Categories:

Key Concepts:

FlowSource, FlowSink, Tank, FlowConnector, Assign Step

Assumptions:

Flow is in units of cubic meters.

Technical Approach:

When the tank reaches the high point, stop the flow from the source and start the flow out of the tank. When the tank reaches the low point, start the flow from the source and stop the flow out of the tank.

Details for Building the Model:

Simple System Setup

- Place a FlowSource, at Tank, and a FlowSink from the Flow Library in the Facility window.
- Connect the FlowSource to the Tank input and the Tank output to the FlowSink.
- Set the *Initial Volume Capacity* of the Tank object to '10' cubic meters.
- Set the *Low Mark*, *Mid Mark*, and *High Mark* properties of the Tank object to '2', '5', and '8' cubic meters, respectively.
- Create an add-on process for the *Above High Mark* property of the tank. This process should use two Assign steps to assign 'Output@FlowSource1.FlowRegulator.Enabled' to 'False' (shutting off the flow though the output node of the FlowSource) and 'Output@Tank1.FlowRegulator.Enabled' to 'True' (enabling the flow out of the tank).
- Create an add-on process for the *Below Low Mark* property of the tank. This process should also use Assign steps to assign 'Output@FlowSource1.FlowRegulator.Enabled' to 'True' (enabling the flow though the output node of the FlowSource) and 'Output@Tank1.FlowRegulator.Enabled' to 'False' (shutting off the flow out of the tank).
- Add the Status Labels displaying the volume in the Tank and the maximum flow rate and flow enabled state of the output FlowNodes of the FlowSource and Tank objects. Keep in mind that 'False' is a value of 0, while 'True' is a value of 1 within the enabled state labels.

Model 3: TransferringFromOneTankToAnother

Problem:

I would like to fill a tank until it reaches its high point. Once this happens, I'd like to transfer the contents into another tank at a much higher rate. When the second tank reaches its high point, empty it.

Categories:

Flow Library, Add-On Process Logic

Key Concepts:

FlowSource, FlowSink, FlowConnector, Tank, Regulator, Assign Step

Assumptions:

Flow is in units of cubic meters and flow rates are in units of cubic meters per hour.

Technical Approach:

Add up two tanks and set the flow rates for the output node of the first tank and the input to the second tank to be higher than the input to the first tank (and the output from the source). Control the flow between the tanks and to the sink using the add-on processes for the tank levels.

Details for Building the Model:

Simple System Setup

- Place a FlowSource, two Tanks, and a FlowSink from the Flow Library in the Facility window.
- Connect the FlowSource to one of the Tank inputs, that Tank's output to the other Tank's input, and the second Tank's output to the FlowSink (all using FlowConnectors).
- Set the *Initial Volume Capacity* for both Tank objects to '10' cubic meters and set the first tank's low medium and high marks to '1', '5', and '9' cubic meters, respectively. Set the second tank's low, medium, and high marks to '0', '5', and '8' cubic meters, respectively.

- Set the *Initial Maximum Flow* rate parameters for the Output@Tank1 and the Input@Tank2 to be '1500' cubic meters per hour.
- Set the *Initial Maximum Flow* rate parameters for the Output@Tank2 and the Input@FlowSink1 to be '200' cubic meters per hour.
- Set the *Above High Mark* add-on process for Tank1 to enable flow to Tank2 by assigning the 'Output@Tank1.FlowRegulator.Enabled' to 'True' and set the *Below Low Mark* add-on process to disable the flow by setting the same property to 'False'.
- Set the *Above High Mark* add-on process for Tank2 to enable flow to the sink by assigning the 'Output@Tank2.FlowRegulator.Enabled' to 'True' and set the *Below Low Mark* add-on process to disable the flow by setting the same property to 'False'.
- Add status labels for the current volume in the tanks.
- Place 3 ModelEntity objects in the Facility window and change the color of each to be different. Within the output flow nodes of the tanks, change the *Initial Output Entity Type* to be 'ModelEntity2' and 'ModelEntity3', respectively, to show different flow colors on all the three connectors.

Model 4: MergedFlow

Problem:

I want to merge two flows into a single flow.

Categories:

Flow Library

Key Concepts:

FlowSource, FlowSink, FlowConnector, FlowNode

Assumptions:

All entity types may merge together. Flow will be merged proportionally based on the inflow rates from the various sources.

Technical Approach:

Create two separate entity flows and merge them into a single flow at a FlowNode.

Details for Building the Model:

Simple System Setup

- Place two FlowSource objects and a FlowSink object in the Facility window.
- Place a FlowNode between the two FlowSource objects and the FlowSink object.
- Connect each FlowSource object to the FlowNode using FlowConnectors.
- Connect the FlowConnector to the FlowSink.
- Set the *Initial Maximum Flow Rate* properties for FlowNode1 and the Input@FlowSink1 to be '200' cubic meters per hour.
- Set the *Flow Control Mode* for FlowNode1 to be 'Merge Flow'.
- Place three ModelEntity instances in the Facility window and specify that each of the two FlowSources use a different entity type by setting the *Entity Type* property.
- Set the *Initial Output Entity Type* property for FlowNode1 to be the third entity type.
- Add status labels for the flow volumes (FlowNode.FlowRegulator.CurrentVolumeFlowOut).

Model 5: SplitFlow

Problem:

I want to split a single flow into two flows and use the flow connector link weights to determine the proportion of flow set to each link.

Categories:

Flow Library

Key Concepts:

FlowSource, FlowSink, FlowConnector, FlowNode

Assumptions:

The link weights of the outgoing FlowConnectors to the FlowSinks will be used to proportionally allocate flow in each direction.

Technical Approach:

Use the Split Flow option on the FlowNode to allocate flow proportionally to a number of FlowSink objects.

Details for Building the Model:

Simple System Setup

- Place a FlowSource object and two FlowSink objects in the Facility window.
- Place a FlowNode between the FlowSource and the two FlowSink objects.
- Connect the FlowSource to the FlowNode and connect the FlowNode to the two FlowSink objects using FlowConnectors.
- Set the *Flow Control Mode* property for FlowNode1 to 'Split Flow' and change the *Split Allocation Rule* to 'Proportional Based On Link Weights'.
- Set the *Selection Weight* properties for FlowConnector2 and FlowConnector3 to '.65' and '.35', respectively.
- Create status labels for the flow rates using the *Expression*: 'Math.If(TimeNow > .05, FlowConnector2.CurrentVolumeFlowOut /TimeNow, "N/A")'.

Model 6: CreateDiscreteEntitiesBasedOnFlow

Problem:

I want to create a discrete entity based on the amount of flow coming out of a tank. This model is an enhancement of the FillingEmptyingTank model in this project.

Categories:

Flow Library, Add-On Process Logic

Key Concepts:

FlowSource, FlowSink, FlowConnector, Tank, Monitor

Assumptions:

A discrete entity is created for every 1 cubic meter of volume transferred out of the tank.

Technical Approach:

A monitor is used to detect changes in the volume transferred out of the tank. The monitor crossing value is changed every time a cubic meter is transferred, so that the crossing value is increased each time a discrete entity is created. The Source for discrete entities is based on an event triggered by the monitor to generate the entities at the appropriate times.

Details for Building the Model:

Simple System Setup

- Place a FlowSource, a Tank, and a FlowSink from the Flow Library in the Facility window.
- Connect the FlowSource to the Tank input and the Tank output to the FlowSink.
- Set the Tank properties similar to that of the Tank in the FillingEmptyingTank model (described above).
- Add the Tank1_AboveHighMark and Tank1_BelowLowMark add-on processes like the FillingEmptyingTank model.

Adding Discrete Entity Creation

- Add a Source and Sink from the Standard Library to the Facility window. Place an additional ModelEntity in the window as well and name it 'ModelEntity2'.
- Connect the Source1 to the Sink1 with a Conveyor. Change the Initial Desired Speed to '3' and the Units to 'Meters per Minute'.
- Within the Definitions window, Elements panel, add a Monitor element with the *Name* 'TransferMonitor'. This will monitor the tank based on the current volume flow out of the tank.

- Change the *Monitor Type* to CrossingStateChange. The *State Variable Name* that we will be monitoring is based on the function 'Output@Tank1.FlowRegulator.CurrentVolumeFlowOut'. Change the *Initial Threshold Value* to '1'. This monitor will automatically cause an event when the tanks output volume crosses the threshold in the positive direction. You'll see that we will be also changing the threshold value within the process.
- Within the Processes window, add a new process. The *Triggering Event* for the process will be the 'TransferMonitor.Event'. Within this process, add an Assign step, where the *State Variable Name* is 'TransferMonitor.CurrentThresholdValue' and the *New Value* is 'Math.ceiling(TransferMonitor.CurrentThresholdValue) + 0.999999'. This will increase the threshold of the monitor from its initial value of 1 (at the start of the simulation) by 1 each time a discrete entity is generated. This will cause the monitor to be fired for every volume out change of 1 cubic meter. Use 0.999999 instead of 1 to make sure the entity is created before the lower mark process is triggered.
- Within the Source in the Facility window, change the Entity Type to 'ModelEntity2', the *Arrival Mode* to 'On Event' and the *Triggering Event Name* to 'TransferMonitor.Event'. Therefore, a discrete entity will only be generated when the TransferMonitor.Event is fired.

Model 7: SimpleFiller

Problem:

I want to have liquid volume arrive to a filling operation and fill containers until they are full.

Categories:

Flow Library

Key Concepts:

ContainerEntity, Filler, FlowSource, FlowConnector

Assumptions:

Flow rate is in units of cubic meters and will fill containers until they are full.

Technical Approach:

Create entity flow using a FlowSource and send it directly to the Filler through a FlowConnector. Create ContainerEntities using a Source that move into the Filler to be filled with liquid volume.

Details for Building the Model:

Simple System Setup

- Place a FlowSource and Filler from the Flow Library in the Facility window and connect them using a FlowConnector.
- Place a ContainerEntity in the Facility window from the Flow Library as well.
- From the Standard Library, place a Source and connect the Source with the ContainerInput node of the Filler using a Path.
- Place a Sink to the right of the Filler and connect the output from the Filler with the Sink with a Path.
- Within the Source, change the *Entity Type* to 'ContainerEntity1' to create container type entities instead of model entities.

Model 8: SimpleEmptier

Problem:

I want to have full containers of a mixed liquid (2 product types) that I'd like to empty and send to their respective locations.

Categories:

Flow Library

Key Concepts:

ContainerEntity, Emptier, FlowSource, FlowConnector, Processing Count Based

Assumptions:

Flow rate is in units of cubic meters and product will flow from the emptier based on product type.

Technical Approach:

Full container entities are created using a Source and sent directly to the Emptier. Two flow streams exit the FlowNode of the Emptier to multiple FlowSinks. The Flow Control Mode of the Emptier's output FlowNode is Single Flow (No Splitting) such that the product is only split based on the Selection Weights of the flow connectors.

Details for Building the Model:

Simple System Setup

- Place a Source and a Sink from the Standard Library in the Facility window.
- Place an Emptier and two FlowSink objects from the Flow Library in the Facility window. Connect the Source to the Emptier and connect the ContainerOutput of the Emptier to the Sink using Paths.
- Connect the FlowOutput of the Emptier to each of the FlowSinks using FlowConnectors.
- Place a ContainerEntity in the Facility window from the Flow Library, as well as two ModelEntity objects from the Project Library.

ModelEntity and ContainerEntity Setup

- Change the *Name* of one of the ModelEntity objects to 'Red' and the other to 'Green'. Change the Red entity color triangle to the color red using the Symbols ribbon.
- Edit the ContainerEntity object Initial Contents by adding two rows to the repeating property editor. First, the *Entity Type* should be 'Green' with a *Quantity* of '.1'. The second initial content should be of *Entity Type* 'Red' with a *Quantity* of '.1'. Given that the *Initial Volume Capacity* is the default value of '0.2', this will fill the container when the entity is first created.
- Within the Source, change the *Entity Type* for arriving to be 'ContainerEntity1'.

Emptier Configuration

- Within the Emptier, we will also define Reliability Logic, such that the Emptier fails every x ContainerEntities that it processes. Change the *Failure Type* to 'Processing Count Based', with the *Count Between Failures* as '20' and the *Time to Repair* as 'Random.Uniform(.5,1)' and *Units* as 'Minutes'.
- On the FlowOutput node of the Emptier, specify the *Initial Maximum Flow Rate* of '60' (default *Units* of 'Cubic Meters per Hour'). In the Output Flow Control section, change the *Flow Control Mode* to 'Single Flow (No Splitting)' so that the flow will not be automatically split by volume between the two flow connectors. We'd like the flow out to be based only on product. This will then be done 'By Link Weight' as the *Outbound Link Rule*.
- Click on the FlowConnector to FlowSink1 and specify the *Selection Weight* as 'Is.Red'. If this expression evaluates to 'True' or a value of '1', then the product can flow through this connector. On the second FlowConnector to FlowSink2, change the *Selection Weight* to 'Is.Green'. Therefore, when the product is emptied from the containers at the Combiner, the Red product will flow to FlowSink1, while the Green product will flow to FlowSink2.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FreeSpaceMovement - SimBit

Problem:

I would like to learn how an Entity and a Vehicle can travel in Free Space.

Categories:

Add-On Process Logic, Movement In Free Space, Vehicles

Key Concepts:

Add-On Process, FreeSpace, Initial Network, ModelEntity, Transfer Step, Travel Step

Assumption:

Moving in Free Space means that an entity is moving within the Facility window but it is not moving on a Link.

Technical Approach:

This project contains three models and each model demonstrates a concept in free space movement.

The model named `FreeSpaceTravelWithNoDestinationSet` contains a `DefaultEntity` with its *Initial Network* property set to 'No Network (Free Space)'. The entity does not have a destination set when it travels so Simio will send the entity to the closest External Input Node. In this model there are three possible External Input Nodes for it to travel to; `Input@Server1`, `Input@Server2` or `Input@Sink1`. Moving these objects around before or during the run will show that the entity will always choose the node that is closest to its current location.

The model named `TransferIntoFreeSpace_UseTravelStep` contains a `DefaultEntity` with its *Initial Network* property set to the default value of 'Global'. This allows the entity to travel on the Path leaving the Source. When the entity enters `BasicNode1`, the Add On Process uses a Transfer Step to transfer the entity into Free Space and then the Travel Step tells the entity to move to the location where `BasicNode2` is currently located. After the entity arrives at that location, another Transfer Step transfers the entity from Free Space into `BasicNode2`. Since it is still on the Global network, it will then follow the Path leaving `BasicNode2` and travel to `Sink1`.

The model named `VehicleTravelInFreeSpace` contains a `DefaultEntity` and a `Vehicle` that both have their *Initial Network* property set to 'No Network (Free Space)'. This demonstrates how a vehicle can travel in free space.

Details for Building the `FreeSpaceTravelWithNoDestinationSet` Model:

- Place a Source, two Servers, and a Sink into the Facility window. Leave some space between the objects so you can see the entity moving in Free Space.
- Place a `ModelEntity` object from the Project Library into the Facility window.
- Select the `DefaultEntity` instance in the Facility window and change the *Initial Network* property to 'No Network (Free Space)'.

The entity does not have a destination set and because there are no links, the entity does not know where to go. In this situation, Simio will find the closest External Input Node and send the entity there. In this model, the External Input Nodes are `Input@Server1`, `Input@Server2` and `Input@Sink1`. Whichever node is closest to the entity will be selected as its destination. Move the objects around to see how the destination changes depending on the distance between the entity and the input nodes.

Details for Building the `TransferIntoFreeSpace_UseTravelStep` Model:

- Place a Source, a Sink and two Basic nodes into the Facility window. Connect the output node of the Source to `BasicNode1` with a Path. Connect `BasicNode2` to the input node of `Sink1` with a Path.
- Select `BasicNode1` and create a new Add On Process by selecting 'Create New' in the dropdown of the *Entered* Add On Process trigger property of this node. This will create a new process that will appear in the Processes window.
 - Place a Transfer Step into the process. Set the *From* property to 'CurrentNode' and the *To* property to 'FreeSpace'.
 - Place a Travel Step. Set the *Destination Type* to 'Specific Object', set the *Destination Object* to 'BasicNode2', and set the *Units* (under Maximum Movement Rate) to 'Meters per Second'.
 - Place another Transfer Step and set the *From* property to 'FreeSpace', the *To* property to 'Node' and the *Node Name* property to 'BasicNode2'.

By default, the DefaultEntity is set to travel on the Global network. This allows it to travel on all Links. Therefore, it travels on the Path leaving the Source and arrives at BasicNode1. At that point, we need to tell the entity to go into FreeSpace. We do that with a Transfer Step, transferring the entity from the current node into Free Space. The Travel Step is used only when an entity is in Free Space and it tells the entity where to move. When the entity arrives at BasicNode2, it is still in Free Space so it cannot travel on the Path into the Sink. So we need another Transfer Step that transfers the entity from Free Space into BasicNode2. At this point, it's still on the global network and therefore it travels on the Path into the Sink.

Details for Building the VehicleTravellnFreeSpace Model:

- Place a Source, a Sink and a Transfer node into the Facility window.
- Place a Vehicle into the Facility window.
 - Set the Vehicle's *Initial Network* property to 'No Network (Free Space)'. Set the *Initial Node (Home)* property to 'TransferNode1' and the *Idle Action* property to 'Go To Home'.
- Place a ModelEntity into the Facility window from the Project Library (bottom left of screen).
 - Set the *Initial Network* property to 'No Network (Free Space)'.
- Select the Output@Source1 node and set the *Entity Destination Type* property to 'Specific' and the *Node Name* property to 'TransferNode1'.
- Select TransferNode1 and set the *Entity Destination Type* property to 'Specific' and the *Node Name* property to 'Input@Sink1'. Set the *Ride On Transporter* property to 'True' and the *Transporter Type* property to 'Specific' and the *Transporter Name* property to 'Vehicle1'.

This is a simple model demonstrating that a Vehicle can travel in Free Space.

Embellishments

Instead of a Vehicle object, use a Worker object to show that a Worker can also travel in Free Space. A Travel Step can also be used with a Vehicle or Worker object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

HierarchyWithTables - SimBit

Problem:

I have a model that references data from tables and I would like to place this model inside of another model. Therefore, I would like to have a "submodel" reference data in a table.

Categories:

Building New Objects / Hierarchy, Data Tables

Key Concepts:

Add-On Process, BasicNode, Contents, Data Table, Expression Property, Expression Property, ExternalNode, Foreign Key, Key column, Numeric Property, Numeric Property, On Entered, Processing, Queue, Repeating Group Property, Resource, Search Step, Server, SetRow Step, String Property, Submodel, Token, TransferNode, Externally Visible, Bind To External Output Node, Bind To External Input Node

Technical Approach:

The model is placed into another model is a SimBit named "SearchTableUponEnteringObject". That model contains three Servers in parallel. An entity travels to one of the three Servers, seizes a certain number of Resources, delays for a certain amount of time and then releases the Resources. The number of Resources and the delay duration are different for each Server. This information is stored in a table. To find the appropriate row in the table, the token must search the table and find the row that contains the information for that particular Server. The search uses the ID of the Server object to find the appropriate row in the table.

In Simio, objects can only reference information from another object that is below them in the object hierarchy. The submodel cannot get information from the top-level model unless it is passed into the object via properties. The mechanism for passing in sets of data like found in Tables is to define a Repeat Group property. This Repeat Group is then used to store the information from the top level model. A Search step in the submodel can be used to search the Repeat Group property to find the appropriate row of information.

Details for Building the Model:

Update the Submodel

Facility window

- Delete the Source object and replace it with a Basic Node and connect this to the first Transfer Node with a connector. The Basic Node is the entry point into this model. We no longer need a Source object because entities will be created in the top level model and enter this submodel through the new Basic Node.
- Delete the Sink object.

Definitions window

- Create a new Repeating Group property from the Properties panel by clicking the 'Repeat Group' icon in the ribbon. Name this property 'Data'. Type 'Custom Properties' into the *Category Name* property of this Data property so this new property is in its own category, for easier reading.
- Ensure that this new property is selected/highlighted in the window and then selecting 'Expression property' from the Standard Property drop down in the ribbon. This will create a new Expression Property within the Repeat Group property. Name this Expression property 'Hours'.
- Click back onto the Data Repeat Group property in the window and select Object from the Object Reference property drop down to create an Object Reference property within the Repeat Group. Name this 'ServerName'.
- Add a final property to the Repeat Group and this property should be an Integer Property, found under the Standard Property drop down. Name this property 'NumberOfResources'.
- Now, using the Search feature on the Project Home tab, find all the references to the ServerData table and change these references to the new "Data" repeat group instead. You'll often need to right click on a property name, select 'Set Referenced Property' and then select Data from the list of properties that appear.
- The Data Table named ServerData is no longer needed so this can be deleted.

Create an External View

- By Default, all the objects in the Facility window are set to be *Externally Visible*, which means that they will be part of

the External View of this object. If you have placed any status labels or floor labels, you might consider not having these be part of the External View. To remove them from the External View, right click on the object in the Facility window and select *Externally Visible*, so that it is no longer highlighted. Similarly, consider not having the static Resource object be visible in the External View. Right click on this object and select *Externally Visible*.

- To create the entry point into this model, you will need to create an External Node. Select the Basic Node at the beginning of this model and right click. Select *Bind To New External Input Node*. Set the name to 'Input'.
- Similarly, select the Transfer Node at the end of this model and right click. Select *Bind To New External Output Node*. Set the name to 'Output'. This creates an exit point for entities to travel out of this model.
- Rename the model by right clicking on the object within the Navigation window in the top right of the Simio interface. Select Rename from the right click drop down menu and name the model 'SearchTable'.

Create the Main, Top Level Model

Create a new Model, either within the same project or in another project. If you create a new model in another project, you'll need to load this submodel into the project by using the "Load Library" icon on the Project Home tab.

Facility window

- Place a Source and a Sink and place two instances of the submodel in parallel between the Source and the Sink. Connect the Source and the Sink to both submodel instances with Paths. Change the *Interarrival Time* of the Source to 'Random.Exponential(3)'.

Data window

- Create two new Data Tables by clicking 'Add Data Table' from the ribbon. Name one table 'ServerData' and the other table 'ServerNames'.
- In ServerData, add three columns; a String Property column named "Name", an Integer Property column named "NumberOfResources" and an Expression Property column named "Hours".
 - Click on the Name column and select 'Set Column as Key' from the ribbon
 - Place the following data into the Table:
 - Server1,3,.1
 - Server2,2,.05
 - Server3,1,.02
- In ServerNames, add two columns; an Object Reference column named "ServerNames", and a Foreign Key column named ServerName (click Foreign Key from ribbon). Set the *Table Key* property of the Foreign Key column to 'ServerData.Name'.
 - Place the following data into the Table:
 - Server1, SearchTable1.Server1
 - Server2, SearchTable1.Server2
 - Server3, SearchTable1.Server3
 - Server1, SearchTable2.Server1
 - Server2, SearchTable2.Server2
 - Server3, SearchTable2.Server3

Facility window

- Back in the Facility window, click on one of the instances of SearchTable and find the new *Data* property. Open the Repeat Group by clicking of the ellipse on the far right and hit Add when the new window pops up.
 - Right click on *Hours* and set this to 'ServerData.Hours'.
 - Type in 'ServerData.NumberOfResources' into the *NumberOfResources* property.
 - Right click on *ServerName* and set this to 'ServerNames.ServerNames'.
 - After closing the Repeat Group editor window, right click onto the *Data* property, select 'Set Referenced Property' and select 'ServerNames'. This is telling the *Data* property to look to the ServerNames table.
- Repeat the above steps for the second instance of the submodel.

See Also:

Open the SimBit named "SearchTableUponEnteringObject.spfx" to see details of how that model was built.

HourlyStatistic - SimBit

Problem:

I want to know the average number of Entities in a Server's Input Buffer per hour of the simulation.

Categories:

Add-On Process Logic, Buffering, Building New Objects / Hierarchy, Custom Statistics

Key Concepts:

Add-On Process, Hourly Statistic, TallyStatistic, Tally Step, Subclass, Custom Object, Timer, MyServer

Assumptions:

We are only concerned with Hourly Statistics.

Technical Approach:

We will create a timer that will fire every hour and trigger a process to record the average value for that particular hour.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server, and a Sink in the Facility Window.
- Connect the Source to the Server, and the Server to the Sink with Paths.

Creating the States and Elements

- In the Elements panel of the Definition tab, add a TallyStatistic and a Timer.
- Name the TallyStatistic 'HourlyAverage' and the Timer 'HourlyTimer'.
- Keep the defaults for TallyStatistic.
- Change the *Time Offset* to '1 hour' in the Timer.
- In the States panel of the Definitions tab, add two Real States.
- Rename the states to 'AverageAtBeginning' and 'TimeAtBeginning'.

Creating the Assignment Process

- In the Processes tab, click 'Create Process' to add a new process to the model.
- Rename this Process CalculateHourlyAve.
- Set the *Triggering Event* to 'HourlyTimer.Event'.
- Place a Tally Step in the process.
 - Set the *Tally Statistic Name* to 'HourlyAverage'.
 - Leave the *Value Type* as 'Expression' and set the *Value* to

$$\frac{(\text{Server1.InputBuffer.Contents.AverageNumberWaiting} * \text{Run.TimeNow} - \text{AverageAtBeginning} * \text{TimeAtBeginning})}{(\text{HourlyTimer.TimeInterval})}$$

Server1.InputBuffer.Contents.AverageNumberWaiting returns the Average Number Waiting for the entire run, if you multiply by Run.TimeNow you will get the total number of Entities that had to wait in the Input Buffer for the entire run. Next we then subtract AverageAtBeginning (which we save each time) multiplied by the TimeAtBeginning to get the total number of Entities that had to wait in the Input Buffer BEFORE this interval started. By subtracting the total number waiting before this interval from the total number waiting, you get the total number waiting DURING this interval. Finally, to get the Average over that interval, we divide the Number Waiting during the interval by the TimeInterval of the timer (which in this case is 1 hour).

- Place an Assign Step in the process.
 - Assign the *State Variable Name* 'AverageAtBeginning' the *New Value* of

$$\text{Server1.InputBuffer.Contents.AverageNumberWaiting}$$
 - Add another assignment that assigns 'TimeAtBeginning' the *New Value* of 'Run.TimeNow'.

As seen in the previous Tally Step, these values will be used to determine the "old" number waiting for the entire run that will be subtracted from the "new" number waiting to get the number waiting for that interval.

Embellishments:

This methodology is easy to implement when dealing with a small number of servers, but could become quite tedious when trying to implement on a large scale – a separate TallyStatistic and AverageAtBeginning State would be needed for each Server.

To avoid having to do all of these manual assignments, we can move all of this logic (the Timer, Tally, States, and Process) into a sub-classed Server. Everything is essentially the same except instead of Tallying/Assigning 'Server1.InputBuffer.Contents.AverageNumberWaiting' we only need to use the expression 'InputBuffer.Contents.AverageNumberWaiting' because the process is referring to its Parent Object.

With all of the logic contained inside the MyServer, you just have to place the object in the Facility window and each MyServer will calculate the Average Number Waiting itself. This is demonstrated within the UsingCustomServer found in the Navigation window.

Also, the technique of Subtracting the old Total from the current Total Divided by a Time Interval will work for any length interval – it does not have to be 1 hour. That is why we divide by 'HourlyInterval.TimeInterval'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ImportExportTables - SimBit

Problem:

I would like to import data from a .csv file into a data table in Simio.

Categories:

Data Tables

Key Concepts:

Before Creating Entities, Connector, Data Table, Export Table, Expression Property, Import Data, Numeric Property, Random.Discrete(), Server, Source, Table Row Referencing

Assumptions:

This example model is a simple source, server, sink model and the server gets its processing time by reading the data in this imported table.

Technical Approach:

It is recommended that before you import a table that you create an empty table that has the appropriate properties (columns) and then export this table to a .csv file. After filling this table with data, the user can import this table into the empty table that was created in Simio.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and a Sink the Facility Window. Connect the Source to the Server and the Server to the Sink with Paths.

Assigning Table Row Number

- Within the Source object, within the Table Row Referencing / Before Creating Entities section, enter a *Table Name* of 'Table1' and the *Row Number* equal to 'Random.Discrete(1, .25, 2, .5, 3, .75, 4, 1)'. This will assign the token a row number from a discrete distribution to assign the values 1, 2, 3 and 4 with equal probability.

Adding a Data Table and Exporting Blank Table

- In the Data Window, select the Tables panel and click on the Add Data Table icon from the ribbon. The main part of the window will appear blank, but you will see a new tab created with the name of the new table (Table1, by default).
- To add a column to this table, select Expression from the Standard Property drop down. Change the *Unit Type* to 'Time' and the *Default Units* to 'Minutes' so that our processing times may be specified in minutes.
- Click on the Export icon from the Table tab in the ribbon. Save the .csv file to the computer.

Importing Data to a Data Table

- Open the .csv file in an editor, such as Excel, and add the appropriate data, keeping the column name untouched.
- To import the data, the spreadsheet must first be 'bound' to the model. Click on the 'Create Binding' button in the Content ribbon, select CSV file then select the appropriate File Name. Then, select Binding Options from the Content ribbon and select Manual to manually import the data into the table. The Import button will then be available to import the data from the spreadsheet into Simio.

Using the Data Table in the Server

- In the Facility Window, set the *Processing Time* property of the Server to read 'Table1.ExpressionProperty1' (assuming you kept the default name, ExpressionProperty1, of the expression property in the table).

InfectionPropagationUsingContinuousAndFlow

- SimBit

Problem:

I want to know how to model a continuous system or how to model System Dynamics. I want to know how to model a system of infection propagation.

Categories:

Continuous Systems, Flow Library, Functions, Level States

Key Concepts:

Assign Step, Decide Step, Delay Step, FlowConnector, Functions, Level State Variable, OnRun Initialized Process, System Dynamics, Tank

Technical Approach:

There are two models in this project, demonstrating two different ways to model this system. The model named, "UsingFlowObjects" demonstrates how to model the system with objects from the standard flow library. There are four Tank objects, connected together with Flow Connectors. The contents in the first tank represent the total population, which is susceptible. The second tank represents the number of people exposed to the infection. The flow nodes in between these tanks are set to allow flow at the rate of RateExposed, which is a function that calculates the rate of exposure. The third tank represents the number of people who are infectious. The flow nodes in between Exposed and Infectious allow flow at the rate of RateInfectious, which is a function that calculates the rate of infection. The fourth tank represents the number of people who have recovered. The flow nodes between Infectious and Recovered allow flow at the rate of RateRecovered, which is a function that calculates the rate in which people recover.

The model named, "UsingLevelStates" demonstrates how to model the system using Level State variables and one simple process. Instead of having Tank objects represent the number of people in each stage, this model uses Level State variables (continuous variables). There are four of these variables, representing Susceptible, Exposed, Infectious and Recovered. A process is called upon Run Initialized it assigns a new value to these level state variables and then delays for one hour, after which the rates of the variables are updated again. Similar to the model using Flow Objects, this model also uses Functions to calculate rates.

Details for Building the Model – UsingFlowObjects:

Adding Properties and Functions

- Go to the Definitions window and into the Properties panel. Create the following new Numeric properties, which will allow the user to input information into the model regarding the population size and infection rates.
 - TotalPopulation – set the *Default Value* to '10000', *Unit Type* to 'Volume / Cubic Meters'.
 - Infectivity – set *Default Value* to '6'.
 - ContactRateInfectious – set *Default Value* to '1.25'.
 - AverageIncubationTime – set *Default Value* to '10' and *Unit Type* to 'Time / Hours'.
 - AverageIllnessDuration – set *Default Value* to '15' and *Unit Type* to 'Time / Hours'.
- Click onto the Functions panel. Create three new functions.
 - RateExposed – set the *Expression* to 'Infectious.FlowContainer.Contents.Weight * ContactRateInfectious * Infectivity * Susceptible.FlowContainer.Contents.Weight / TotalPopulation'. Set the *Return Type* to 'Number' and the *Unit Type* to 'VolumeFlowRate'.
 - RateInfectious – set the *Expression* to 'Exposed.FlowContainer.Contents.Weight / AverageIncubationTime'. Set the *Return Type* to 'Number' and the *Unit Type* to 'VolumeFlowRate'.
 - RateRecovered – set the *Expression* set to 'Infectious.FlowContainer.Contents.Weight / AverageIllnessDuration'. Set the *Return Type* to 'Number' and the *Unit Type* to 'VolumeFlowRate'.

Adding Tanks to the Facility Window

- In the Facility window, place four Tank objects from the Flow Library into the Facility window. Rename Tank1 to 'Susceptible', Tank2 to 'Exposed', Tank3 to 'Infectious' and Tank4 to 'Recovered'. Connect Susceptible to Exposed with a Flow Connector, Exposed to Infectious together with a Flow Connector and Infectious and Recovered together with

- a Flow Connector.
- Select all four Tanks by holding down CTRL and clicking onto each Tank. Enter 'TotalPopulation' into the *Initial Volume Capacity* property of the tanks.
- Click onto the Susceptible Tank and click into the *Initial Contents* property to indicate that there are initial contents in this Tank. In the Repeating Property Editor, click Add to add a new row. The *Entity Type* property is 'DefaultEntity' and the *Quantity* property should be set to 'TotalPopulation - 1'.
- Click onto the Infectious Tank and click into the *Initial Contents* property to indicate that there are initial contents in this Tank. The *Entity Type* property is 'DefaultEntity' and the *Quantity* property should be set to '1'.
- Holding down the CTRL key, select the Output@Susceptible and the Input@Exposed and set the *Maximum Flow Rate Equation* to 'RateExposed'. This rate will be dynamically updated every hour because the *Update Interval* on these nodes is set to '1' hour, by default.
- Holding down the CTRL key, select the Output@Exposed and the Input@Infectious and set the *Maximum Flow Rate Equation* to 'RateInfectious'. This rate will be dynamically updated every hour because the *Update Interval* on these nodes is set to '1' hour, by default.
- Holding down the CTRL key, select the Output@Infectious and the Input@Recovered and set the *Maximum Flow Rate Equation* to 'RateRecovered'. This rate will be dynamically updated every hour because the *Update Interval* on these nodes is set to '1' hour, by default.

Details for Building the Model – UsingLevelStates:

Adding States, Properties and Functions

- Go to the Definitions window and into the States panel. Create the following new Level State Variables by clicking onto Level in the ribbon.
 - Pop_Susceptible
 - Pop_Exposed
 - Pop_Infectious
 - Pop_Recovered
- Click into the Properties panel within the Definitions window. Create the following new Numeric properties, which will allow the user to input information into the model regarding the population size and infection rates.
 - TotalPopulation – set the *Default Value* to '10000'.
 - Infectivity – set the *Default Value* to '6'.
 - ContactRateInfectious – set the *Default Value* to '1.25'.
 - AverageIncubationTime – set the *Default Value* to '10' and the *Unit Type* to 'Time / Hours'.
 - AverageIllnessDuration – set the *Default Value* to '15' and the *Unit Type* to 'Time / Hours'.
- Click onto the Functions panel. Create three new functions.
 - RateExposed – set the *Expression* to 'Pop_Infectious * ContactRateInfectious * Infectivity * Pop_Susceptible / TotalPopulation'. Set the *Return Type* to 'Number'.
 - RateInfectious – set the *Expression* to 'Pop_Exposed / AverageIncubationTime' and the *Return Type* to 'Number'.
 - RateRecovered – set the *Expression* to 'Pop_Infectious / AverageIllnessDuration' and the *Return Type* to 'Number'.

Adding Steps in Processes Window

- In the Processes window, open the Select Process drop down from the ribbon and select OnRunInitialized. This will create a new process that is triggered at the beginning of the run.
- Place an Assign Step into this process. This Assign step will have two assignments.
 - Set *State Variable Name* to 'Pop_Susceptible' and *New Value* to 'TotalPopulation - 1'.
 - Add another assignment in this step by opening the Repeat Group for Assignments (More) and set *State Variable Name* to 'Pop_Infectious' and *New Value* to '1'.
- Place another Assign Step in this process. This Assign step will have four assignments.
 - Open the Repeat Group for Assignments (More) and set *State Variable Name* to 'Pop_Susceptible.Rate' and *New Value* to '-RateExposed'.
 - Click Add to add another assignment. Set *State Variable Name* to 'Pop_Exposed.Rate' and *New Value* to 'RateExposed-RateInfectious'.
 - Click Add to add another assignment. Set *State Variable Name* to 'Pop_Infectious.Rate' and *New Value* to 'RateInfectious-RateRecovered'.
 - Click Add to add another assignment. Set *State Variable Name* to 'Pop_Recovered.Rate' and *New Value* to 'RateRecovered'.

-
- Place a Delay step after the second Assign Step and set the *Delay Time* to '1' (hour). Drag the segment leaving the Decide Step so that it connects to the beginning of the second Assign step. This will tell the process logic to delay for one hour and then update the Assignments in the second Assign step. It will therefore update the assignments every hour.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

InitializeObjectPropertiesFromATable - SimBit

Problem:

I have objects in my model that should get their initial property values from information in a Data Table.

Categories:

Data Tables

Key Concepts:

Auto-Set Table Row Reference, Data Table, Set Referenced Property

Technical Approach:

There are two Servers in this model and each Server will get some initial property values from a Data Table. The Data Table contains two rows; one for each Server that exists in the model. The Table contains four properties (columns): an object instance property which contains the name of each Server, an expression property which contains the Processing Time of each Server, another expression property which contains the Capital Cost of each Server, and another object instance property that contains the name of a Worker object that this Server should seize during processing. In order for each Server to know which row to look at in the Table, the first object instance property (column) that contains the name of the Server, must have its *Auto-set Table Row Reference* property set to 'True'. This tells Simio that the object contained in this column will automatically have a row reference set to this table and therefore we can reference the data contained in this table in the properties of each Server instance in the model.

Details for Building the Model:

Simple System Setup

- Place a Source and Sink into the Facility window. Place two Server objects, in parallel and connect the Source to both Servers with Paths and connect each Server to the Sink with paths.
- Place four Basic Nodes in the center of the model so that they form a diamond shape. BasicNode1 is right below Server1, BasicNode2 is right above Server2, BasicNode3 is at the left point of the diamond and BasicNode4 is on the right point of the diamond. Connect these nodes together with Paths so the Worker can travel from node to node in a circular pattern.
- Place two Worker objects into the Facility window. Set the *Initial Node* property of Worker1 to 'BasicNode3' and its *Idle Action* property to 'GoToHome'. Set the *Initial Node* property of Worker2 to 'BasicNode4' and its *Idle Action* property to 'GoToHome'.

Create the Data Table

- Go to the Data Window and click Add Data Table to create a new Table.
 - From the Object Reference drop down in the ribbon, select Object to create a new column. Rename this column 'ServerName'.
 - From the Standard Property drop down in the ribbon, select Expression to create a new column. Rename this column 'ProcessingTime'. In the Properties of this new column, set the *Unit Type* property to 'Time' and the *Default Units* property to 'Minutes'.
 - From the Standard Property drop down in the ribbon, select Expression to create a new column. Rename this column 'Capital Cost'.
 - From the Object Reference drop down in the ribbon, select Object to create a new column. Rename this column 'Worker'.
- Fill in the following data into the new Table:
 - Row 1: Server Name = 'Server1', ProcessingTime = 'Random.Triangular(.3, .4, .6)', CapitalCost = '2000', Worker = 'Worker1'.
 - Row 2: Server Name = 'Server2', ProcessingTime = '.5', CapitalCost = '1000', Worker = 'Worker2'.
- Click onto the first column, ServerName, so that you see its properties appear in the property window on the lower right side of the interface. Under the Advanced Options property category, set the *Auto-set Table Row Reference* property to 'True'.

Configure Server to Read Table

-
- Go to the Facility window and select Server1. Expand the Financials Property category and right click into the *Capital Cost* property of this Server and select 'Set Referenced Property'. From the options available, find Table1.CapitalCost and select this so that the *Capital Cost* property now gets its value from Table1.CapitalCost.
 - Right click into the *Processing Time* property of this Server and select 'Set Referenced Property'. From the options available, find Table1.ProcessingTime and select this so that the *Processing Time* property now gets its value from Table1.ProcessingTime.
 - Expand the Secondary Resources property category and expand the Resource for Processing sub-category. Right click into the *Object Name* property and select 'Set Referenced Property'. From the options available, find Table1.Worker and select this so that the *Object Name* property now gets its value from Table1.Worker.
 - Set the *Request Move* property to 'To Node' and set the *Destination Node* property to 'BasicNode1'. Visually seeing the Worker move to the Server will help validate that the Server is getting the correct information from the Table.
 - Exactly repeat the above four bullet points for Server2, except for the *Destination Node* property, set this to 'BasicNode2' instead so that the Worker moves to the node above Server2.

Embellishments:

Notice that this SimBit contains floor labels that display the property values of each Server to visually validate to the user that each Server is getting the correct information from the Data Table.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

InputAnalysis - SimBit

Problem:

Because I am lacking enough real-world data for my model, I want to use Input Analysis and Response Sensitivity to determine whether my models arrival rates or processing time have the biggest impact on the model responses (Time In System and Number In System).

Based on the limited number of real-world data inputs for Arrival Times and Processing Time in my model, I want to use Input Analysis and Sample Size Error to determine which inputs have the biggest contribution to my output errors on the model responses.

Categories:

Input Analysis

Key Concepts:

Input Parameters, Data, Analysis, Experiment, Response Sensitivity, Sample Size Error

Technical Approach:

There are two models in the project, each demonstrating a different aspect of Input Analysis. Both models include a simple 2 Source-Server-Sink logic to illustrate this input analysis features. Three Input Parameters are defined which will be used to measure the effect of the changes in Arrival Rates and Processing Time on the Number in System and Time in System.

In the first model, titled "ResponseSensitivity", the input parameters include the distribution type, but no information on the number of data samples for the data sets. In this model, the Response Sensitivity analysis within the experiment will be used.

The second model, titled "SampleSizeError", includes the same input parameters, but additional information regarding the sample sizes for the distributions. It is assumed that each of the Input Parameter distributions have been derived based on a sample size of 100. In this model, the Sample Size Error analysis within the experiment will be used.

Details for Building the Model - ResponseSensitivity:

Simple System Setup

- Place two Sources, a Server and a Sink object in the Facility window. Connect each Source to the Server and the Server to the Sink object using a Path.

Defining Input Parameters

- In the Data window, select the Input Parameters panel button and add 3 Distribution parameters. Name them 'Arrival1', 'Arrival2', and 'ProcessingTime' respectively. For Arrival1, change the *Distribution Type* to 'Exponential' with a *Mean* of '10'. Set the *Unit Type* to 'Time' and the *Units* to 'Minutes'. Do the same for Arrival2 but change the *Mean* to '12'.
- For ProcessingTime, set the *Distribution Type* to 'Triangular' with a *Minimum*, *Mode* and *Maximum* set to '8', '10' and '12' respectively. Change the *Unit Type* to 'Time' and change the *Units* to 'Minutes'.
- In the Facility window, on the Server object, change the *Processing Time* to 'ProcessingTime'. On Source1 change the *Interarrival Time* to 'Arrival 1'. On Source2 change the *Interarrival Time* to 'Arrival2'.

Defining the Experiment – ResponseSensitivity

- In the Navigation window, right click on Model and select 'New Experiment'.
- In the Experiment window, click on the Design ribbon and select Add Response.
- Name the Response 'TimeInSystem' and on this response add the *Expression* as 'DefaultEntity.Population.TimeInSystem.Average', change the *Unit Type* to 'Time' and the *Display Units* to 'Minutes'.
- Add another Response. Name this 'NumberInSystem'. Change the *Expression* to 'DefaultEntity.Population.NumberInSystem.Average'.
- In the Experiment Properties window, change the *Default Replications* to '100'.

Analyzing the Results – ResponseSensitivity

- From the Experiment window, Design ribbon, select Run. Once the runs have completed, click on the Input Analysis tab and then select Response Sensitivity on the panel.
- A Tornado Chart will be displayed which will show each input's sensitivity coefficient for each response. You can select which response to view from the pull-down list in the upper left corner. Select TimeInSystem from the pull-down list if it isn't already displayed. What can be inferred here is that an increase in the ProcessingTime has the most impact on the TimeInSystem of the entities.
- Select NumberInSystem from the pull-down list and you can infer the same. However, for the NumberInSystem, there is very little difference between an increase in ProcessingTime and a decrease in Arrival1. Click on the Bar Chart tab at the bottom of this window. Here you can see the percentage of impact each input has on each response.

Details for Building the Model - SampleSizeError:

Simple System Setup

- Place two Sources, a Server and a Sink object in the Facility window. Connect each Source to the Server and the Server to the Sink object using a Path.

Defining Input Parameters

- In the Data window, select the Input Parameters panel button and add 3 Distribution parameters. Name them 'Arrival1', 'Arrival2', and 'ProcessingTime' respectively. For Arrival1, change the *Distribution Type* to 'Exponential' with a *Mean* of '10'. Set the *Unit Type* to 'Time' and the *Units* to 'Minutes'. Change the *Number of Data Samples* to '100' and set *Include in Sample Size Error Analysis* to 'True'. Do the same for Arrival2 but change the *Mean* to '12'.
- For ProcessingTime, set the *Distribution Type* to 'Triangular' with a *Minimum*, *Mode* and *Maximum* set to '8', '10' and '12' respectively. Change the *Unit Type* to 'Time' and change the *Units* to 'Minutes'. Change the *Number of Data Samples* to '100' and set *Include in Sample Size Error Analysis* to 'True'.
- In the Facility window, on the Server object, change the *Processing Time* to 'ProcessingTime'. On Source1 change the *Interarrival Time* to 'Arrival 1'. On Source2 change the *Interarrival Time* to 'Arrival2'.

Defining the Experiment – SampleSizeError

- In the Navigation window, right click on Model and select 'New Experiment'.
- In the Experiment window, click on the Design tab and select Add Response.
- Name the Response 'TimeInSystem' and on this response add the *Expression* as 'DefaultEntity.PopulationTimeInSystem.Average', change the *Unit Type* to 'Time' and the *Display Units* to 'Minutes'.
- Add another Response. Name this 'NumberInSystem'. Change the *Expression* to 'DefaultEntity.Population.NumberInSystem.Average'.
- Change the *Default Replications* on this experiment to '100'.

Analyzing the Results – SampleSizeError

- From the Experiment window, select the Design tab and select Run.
- Once the runs have completed, select the Input Analysis tab and click on Sample Size Error panel button.
- You must first click on Run Analysis button within the Sample Size Error ribbon. Once the runs have completed, a chart will appear that will display the Contribution to Uncertainty for Scenario1. This shows which inputs have the biggest contribution to the response. What can be inferred here is that Arrival1's uncertainty has the biggest impact on the TimeInSystem. The SMORE plot below the chart illustrates the standard confidence interval for the mean (in tan) and the expanded half width due to input uncertainty in blue. As we can see from this graph we have more input uncertainty than experimentation uncertainty in this example.
- The second bar chart, Benefit of Additional Samples, shows the relative benefit of collecting additional data for each input parameter. In many cases, such as in this example, the bar chart will appear similar to the relative Contribution to Uncertainty bar chart. We would not expect this to be the case if the *Number of Data Samples* properties of the Input Parameters were widely different.

InterruptibleOperator - SimBit

Problem:

I would like to model a system with 2 Servers and a movable operator that needs to be present at the server for processing and be able to interrupt the processing of a lower priority job for a higher priority job.

Categories:

Decision Logic -- Processing

Key Concepts:

Add-On Process, Allow Passing, Assign Step, BasicNode, Bidirectional Path, Current Symbol Index, Filter Expression, Initial Desired Speed, Interrupt Step, Interrupted Process Action, Largest Value First, ModelEntity, Processing, After Processing, Path, Priority, Ranking Expression, Ranking Rule, Release Step, Request Move, ResumeDelay, Seize Step, Server, Time Offset, Value Expression, Vehicle

Assumptions:

The interrupted part is allowed to remain in the processing station until the operator returns. Upon processing for the second time the job only requires the remaining processing time, no set-up or tear-down time.

Technical Approach:

Upon the arrival of a high priority job, the server will attempt to seize the operator. If the operator is already seized, the job that is being processed is interrupted. The server releases the operator which is then seized by the high priority job's server. After the operator is finished processing all the high priority jobs, it is then released and re-seized by the low priority job. Once the operator returns, the only processing time remaining on that entity is the remaining processing time that was not executed before the interruption.

Details for Building the Model:

Simple System Setup

- Drag in 2 Sources, 2 Servers, and 2 Sinks to the Facility View. Arrange them in two Source-Server-Sink sets and place a Basic Node next to each Server.
- Connect the Source-Server-Sink sets with unidirectional Paths and change *Allow Passing* to 'False' in the Paths going from Source to Server. Connect the Basic Nodes with a bidirectional Path. Click on BasicNode1 and attach ParkingStation.Contents queue, because this will be the Initial Node.

Defining a Vehicle

- Drag a standard Vehicle into the Facility window. Change its *Desired Speed* to '0.1 Meters per Second'. Set the *Initial Node* to 'BasicNode1'. Set its *Ranking Rule* to 'Largest Value First', with a *Ranking Expression* of 'ModelEntity.Priority'.

Altering Entities

- Drag two ModelEntities into the Facility window and name them 'HighPriorityJob' and 'StandardJob'.
- Change the color of HighPriorityJob to red and set its *Initial Priority* to '3.0'.
- [Add an additional symbol](#) to StandardJob and change the second symbol's color to yellow. Set their *Desired Speed* properties to '0.2 Meters per Second'.

Defining Fixed Objects

- Change the *Entity Type* property in Source2 to 'StandardJob'. Set the *Interarrival Time* to 'Random.Exponential(30)' minutes and *Entities Per Arrival* to '2'.
- Change the *Time Offset*, as well as the *Interarrival Time*, in Source1 to 'Random.Exponential(50)'.
- Both Server1 and Server2 should have their *Initial Capacity* property set to 'Infinity', *Processing Time* property set to 'Random.Triangular(10,20,30)', and *Input Buffer Capacity* property set to '1'.

Adding Process Logic for Seizing and Releasing the Vehicle

- In Server2, create a process in the *Processing* add-on process trigger. Add a Seize step in this process. Have this step seize Vehicle1. Click on the "..." button to open the Repeating Property Editor. Click the Add button and choose

'Vehicle1' for the *Object Name*. Change *Request Move* from 'None' to 'ToNode' and the *Destination Node* to 'BasicNode2'.

- Also in Server2, create a process in the *After Processing* add-on process trigger. Add a Release Step that releases Vehicle1. Click the "..." button next to Releases in the Property window to open the Repeating Property Editor. Click the Add button and select 'Vehicle1' from the drop down list next to *Object Name*.
- Similarly, you can do the exact same thing for Server1 (or you can re-use the process named Server2_AfterProcessing and eliminate this step, if preferred). Add a Release Step that releases Vehicle1. Click the "..." button next to Releases in the Property window to open the Repeating Property Editor. Click the Add button and select 'Vehicle1' from the drop down list next to *Object Name*.

Adding Process Logic for Interruption

- To set the Interrupting logic, create a process in Server1's *Processing* add-on process trigger. Place an Interrupt Step to interrupt the activity taking place at the other server (Server2). Select the *Process Name* to 'Server2.OnEnteredProcessing' from the drop down list. Change the *Interrupted Process Action* to 'ResumeProcess'.
- On the Original segment leaving the Interrupt step, place a Seize step that seizes Vehicle1. Open the Repeating Property Editor in the Seizes property. In the Editor, select 'Vehicle1' from the drop down list next to *Object Name*. Change the *Request Move* to 'ToNode' with *Destination Node* 'BasicNode1'.
- On the Interrupted segment leaving the Interrupt step, place an Assign step that changes the picture of the interrupted entity. Set the *State Variable Name* to 'ModelEntity.Picture', the *New Value* to '1'.
- Add an additional Assignment to increase the interrupted Entity's priority by opening the Repeating Property Editor of the Assign Step. Add a new Item with *State Variable Name* set to 'ModelEntity.Priority' and set its *New Value* to '2'. This gives the interrupted entity higher priority than an uninterrupted StandardJob so that it can resume processing on this particular job but still lower Priority than a HighPriorityJob so it can be interrupted if need be.
- After the Assign on the interrupted segment, add a Release step and select Vehicle1 in the Repeating Property Editor. This releases the vehicle from Server2 and allows it to be seized by a HighPriorityJob.
- Place a Seize step, seize Vehicle1 in the Repeating Property Editor. Set *Request Move* to 'ToNode' with *Destination Node* set to 'BasicNode2'. This puts the interrupted server back in the vehicle's queue list of objects that are trying to seize it, so that as soon as it become free it will return to Server2 to complete processing.
- Finally, add an Assign step changing the picture back to the original green color to signify processing has resumed. To do this, set the *State Variable Name* to 'ModelEntity.Picture' with a *New Value* of '0'.

See Also:

[InterruptingAcrossMultipleServers](#) and [InterruptingServerWithMultipleCapacity](#).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Interrupting Across Multiple Servers - SimBit

Problem:

I have multiple entities with different priorities and multiple servers. If all servers are busy, I want to be able to interrupt the processing at a server processing a lower priority entity.

Categories:

Decision Logic -- Processing

Key Concepts:

Add-On Process, Allow Passing, BasicNode, Blocked Destination Rule, Candidate, Condition Based, Connector, Decide Step, Entity, Entity Destination Type, Filter Expression, Interrupt Step, Largest Value First, Last Seized, On Creating Entities, On Entered, OnEnteredProcessing, Path, Priority, Ranking Rule, Real State Variable, Release Order, Release Step, ResourceState, Save Remaining Time, Select Available Only, Selection Rule, Smallest Value First, Source, Transfer Step, TransferNode, Value Expression

Assumptions:

All entities wait at one spot for an available server. All interrupted entities wait in the same FIFO queue.

Technical Approach:

All entities go to a single node with routing logic to select an available server. If all servers are busy and the entity that is attempting to enter the server has a higher priority than at least 1 entity being processed, then the lower priority entity is interrupted (with its remaining processing time saved) and routed back to the entry node.

Details for Building the Model:

Simple System Setup

- Place 3 Sources, 3 Servers, 3 Entities, a Sink, a BasicNode and a TransferNode into the Facility Window. Position the Sources so that they are aligned vertically and align the Servers similarly.
- Rename the entities 'LowPriority', 'MediumPriority', and 'HighPriority'.
- Change the color of HighPriority to red and its *Initial Priority* to '3.0'. Change the color of MediumPriority to yellow and its *Initial Priority* to '2.0'. Also, add an additional symbol and change its color to a slightly different shade of yellow. For LowPriority, keep the *Initial Priority* at '1.0' and the color green, but add an additional symbol and change its color to a light green.
- Rename the TransferNode 'Dispatch' and position it in between the Sources and the Servers. All Sources will be sent here first and then to a Server.
- Rename BasicNode1 'ReRoute'. Position this node above the set of Servers. This node will act as the exit point for the interrupted jobs that have to be removed.
- Draw paths going from the Sources to Dispatch (node), as well as from ReRoute to the Dispatch. Set the *Allow Passing* property to 'False' for these paths to allow them to queue up at the Dispatch area.
- Connect Dispatch to the Servers using Connectors and connect the Servers to the Sink with Paths.

Modifying Library Objects

- Add a Discrete State to the ModelEntity called 'ProcessingTime'. To do this, click on ModelEntity in the Navigation window. Click on Definitions Tab, then States. Add a Discrete State, and then change the name to 'ProcessingTime'.
- Go back to the Navigation window and change to the Model for the following steps.
- To select nodes and servers from a list, go to the Definitions tab and then to Lists panel. Add a Node List with name of 'ServerInputNodes' with the nodes Input@Server1, Input@Server2, and Input@Server3.
- Add an Object List and name it 'ServerList' with the objects Server1, Server2 and Server3.
- Rename the Sources 'LowPrioritySource', 'MediumPrioritySource', and 'HighPrioritySource'.
- Modify the LowPriority Source so that it has an *Entity Type* of 'LowPriorityJob', *Time Offset* of 'Random.Exponential(.5)' and *Interrarrival Time* of 'Random.Exponential(1)'.
- Modify the MediumPrioritySource so that it has an *Entity Type* of 'MediumPriorityJob', *Time Offset* of 'Random.Exponential(2)', and *Interrarrival Time* of 'Random.Exponential(2)'.
- Modify the HighPrioritySource so that it has an *Entity Type* of 'HighPriorityJob', *Time Offset* of 'Random.Exponential(3)', and *Interrarrival Time* of 'Random.Exponential(3)'.

Assigning Processing Time

- Within the Processes window, create a New Process and name it 'AssignProcessingTime'.

- Place an Assign step in the process and set the *State Variable Name* to 'ModelEntity.ProcessingTime', *New Value* is 'Random.Triangular(.8,1.5,3)' and *Units* is 'Minutes'.
- Enter this process name in the all three Sources' *Created Entities* Add-On Process Trigger property.
- Set all of the Servers' *Processing Time* properties to 'ModelEntity.ProcessingTime'. Also, change their *Input Buffer* properties to '0'.

Dispatch Routing Logic

- Dispatch needs to route jobs based on their priorities and only to servers with open processing stations. Change the *Ranking Rule* to 'Largest Value First', the *Entity Destination Type* to 'Select From List', the *Node List Name* to 'ServerInputNodes' and the *Blocked Destination Rule* to 'Select Available Only'.
- Within the Dispatch node, double click on the *Entered* property of the Add-On Process Triggers to create the Dispatch_Entered process.

Interruption Process Logic

- Go to the Processes window and within the Dispatch_Entered process, follow the below steps.
- Place a Decide step to determine if any Server is available. Change the *Decide Type* to 'Condition Based', and *Expression* to 'Server1.Capacity.Remaining==1||Server2.Capacity.Remaining==1||Server3.Capacity.Remaining==1||ModelEntity.Priority==1.5'. If a Server is available, the Server*.Capacity.Remaining will return a value of 1 and the entity can be processed as usual. Or, if the entity that entered the node has a Priority of 1.5, that means it's a LowPriority Entity and should interrupt anything.
- From the False exit, add an Interrupt step and change the *Process Name* to 'Server1.OnEnteredProcessing', the *Selection Rule* to 'Smallest Value First' and the *Filter Expression* to 'Candidate.Entity.Priority < Entity.Priority'. Leave the *Value Expression* as the default value 'Candidate.Entity.Priority'.
- In the Interrupt step, Advanced Options, add 2 more processes in Process Names (More): including *Process Name* of 'Server2.OnEnteredProcessing' and *Process Name* of 'Server3.OnEnteredProcessing'. Change the *Save Remaining Time* property to 'ModelEntity.ProcessingTime'.
- On the Interrupted exit, place an Assign step and change the *State Variable Name* to 'ModelEntity.Picture', *New Value* to '1' and make another assignment under Assignments (More) of *State Variable Name* called 'ModelEntity.Priority', and *New Value* of 'ModelEntity.Priority + .5'. This will make sure that if its an entity that has already been interrupted, it keeps its priority the same (and doesn't bump it up to the next level of entity priority).
- Next, place a Release step and change the *Object Type* to 'FromList', *Object List Name* to 'ServerList' and *Release Order* to 'LastSeizedFirst'.
- Lastly, add a Transfer step after the Assign step and change the *From* to 'CurrentStation', *To* to 'Node' and *Node Name* to 'Reroute'.

Embellishments:

In this model all interrupted entities wait on the same path with a FIFO selection method. This is because they are waiting on the same path and only the first entity is being evaluated at the Transfer Node, all the other entities are still waiting to arrive to the node. This means that if a LowPriorityJob is in front of a MediumPriorityJob, MediumPriorityJob would have to wait for the LowPriorityJob to be processed before being considered. To correct this, add another path from ReRoute to Dispatch with *Allow Passing* set to 'False'. Change the *Outbound Link Rule* in the BasicNode to 'By Link Weight'. Set the Link Weight for one Path to 'ModelEntity.Priority == 1.5' and 'ModelEntity.Priority == 2.5' for the other. This will now allow both types of entities to be present at the Node at the same time, correcting the selection logic.

See Also:

[InterruptibleOperator](#) and [InterruptingServerWithMultipleCapacity](#).

InterruptingServerWithMultipleCapacity - SimBit

Problem:

I have a server with capacity of three and I would like a higher priority job to be able to interrupt the processing of one of the lower priority jobs currently being processed, and move the interrupted job back into the input buffer to complete processing later.

Categories:

Decision Logic -- Processing

Key Concepts:

Add-On Process, Assign Step, Before Exiting, Candidate, Capacity.Remaining, Decide Step, Entity, Filter Expression, From Current Station, Interrupt Step, Largest Value First, On Entered, OnEnteredProcessing, Priority, Ranking Rule, Release Step, Save Remaining Time, Selection Rule, Server, Smallest Value First, Source, State Assignments, Transfer Step, Value Expression, Table Row Referencing

Assumptions:

All entities have the same Processing Time distribution and the remaining process time will be saved.

Technical Approach:

When the entity arrives to the Server, it looks at the server and sees if there is any remaining capacity. If not, the entity will interrupt any entity with a lower priority than itself. If the arriving entity has the same or lower priority than the three entities being processed it will enter the Input Buffer and wait for one of the entities to finish processing.

Details for Building the Model:

Simple System Setup

- Drag in a Source, Server, and Sink. Connect them all with Connectors.
- Drag in three Entity Instances and rename them 'LowPriorityJob', 'MediumPriorityJob', and 'HighPriorityJob'.
- Change the color of HighPriorityJob to red and its *Initial Priority* to '3.0', MediumPriorityJob to yellow and its *Initial Priority* to '2.0', and add an additional symbol to LowPriorityJob and change the additional symbol to a light green, but leave its *Initial Priority* '1.0'.
- Set *Interarrival Time* in Source1 to 'Random.Exponential(.6)' minutes.

Creating Multiple Entities from One Source

- Add a Data Table in the Data tab and name it 'JobMix'. Add a Real Property called 'Percentage' and an Entity Object Reference Property called 'EntityType'. Fill in the table so that it looks like:

Percentage	EntityType
70	LowPriorityJob
30	MediumPriorityJob
10	HighPriorityJob
- In Source1, set *Entity Type* to 'JobMix.EntityType'.
- Expand the *Table Row Referencing* -> *Before Creating Entities* properties and set the Table Name to 'JobMix' and Row Number to 'JobMix.Percentage.RandomRow'.

Assigning Process Time to Entities

- In the ModelEntity's Definitions tab, add a Discrete State called 'ProcessingTime'.
- Then, within the model Facility window, in the Source, create a new process in the Created Entity add-on process trigger and add an Assign step. In the step set *State Variable Name* to 'ModelEntity.ProcessingTime' and *New Value* to 'Random.Triangular(1,2,3)'.

Modifying Server1

- Drag Server1's Input Buffer below and resize it so that it is approximately 4 servers long so that you can see the activity.
- Change the Server's *Initial Capacity* to '3'.
- Set the *Ranking Rule* to 'Largest Value First' with a *Ranking Expression* of 'Entity.Priority'.
- Change to *Processing Time* to 'ModelEntity.ProcessingTime'.

Process Logic for Interrupting

- On the Server, add an Add On Process Trigger to Entered to interrupt processing.
- Place a Decide step to determine if the Server is Available. The *Decide Type* is 'ConditionBased' with the *Expression* 'Server1.Capacity.Remaining > 0'. If this is True, interruption is not necessary and no further action is needed.
- If it is False, it means that the Server is full and if the Entity has a higher Priority than any one entity on the Server then we need to interrupt processing. So we place an Interrupt step on the False Branch with the following changes:
- In General Options, change *Process Name* to 'Server1.OnEnteredProcessing', *Selection Rule* to 'Smallest Value First' (This interrupts the entity with the smallest priority), *Value Expression* to 'Candidate.Entity.Priority', and *Filter Expression* to 'Candidate.Entity.Priority < Entity.Priority' (Ensures that the entity the Server is considering interrupting has a lower priority than the one that just entered the Server).
- In Advanced Options, set *Save Remaining Time* to 'ModelEntity.ProcessingTime'. The Server will now use remaining processing time the next time it evaluates ModelEntity.ProcessingTime.
- On the "Interrupted" branch, place an Assign step to change to picture of the interrupted entity. *State Variable Name* is 'ModelEntity.Picture' and *New Value* is '1'. (The "Original" branch is for the higher priority Entity so it will be processed as usual, so therefore no steps are needed.)
- After the Assign, place a Release step releasing Server1. Click on the "..." button and select 'Server1' from the *Object Name* drop down list.
- Then, place a Transfer step to transfer the entity back into the input buffer. Configure the Transfer Step so that *From* is set to 'CurrentStation', *To* to 'Station', and *Station Name* to 'Server1.InputBuffer'.

Enhancement:

Interrupted entities will not necessarily be processed ahead of other waiting entities. To make that happen:

- After entities are interrupted, add an assignment to set 'ModelEntity.Priority' to 'ModelEntity.Priority + 0.1'. This will raise their priority to bring them to the front of the InputBuffer.
- Modify the *Filter Expression* on the Interrupt step to read '(Candidate.Entity.Priority+.2) < Entity.Priority'. This will prevent an interrupted entity from immediately interrupting another entity of the same nominal priority.

See Also:

[InterruptibleOperator](#) and [InterruptingAcrossMultipleServers](#)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

InventoryAndMaterials - SimBit

Problem:

My system consumes materials stored in inventories at different locations. I want my simulation model to show material consumption from site-specific inventories.

Categories:

Supply

Key Concepts:

Bill of Materials, Inventory Element, Material Element, Task Sequence

Assumptions:

The processing at the Servers and Combiner will use Task Sequences to concurrently complete the defined tasks at the object and consume the respective material from the identified inventory. All inventories are located at the object at which the consumption takes place.

Technical Approach:

Two Sources independently feed unique entity types to two Servers respectively, where at the Servers, a Task Sequence is used to facilitate material consumption. Each Server has two tasks, which occur simultaneously as indicated by the *Sequence Number*. Both tasks consume a material which has an associated inventory located at the Server. The entities are then combined via a Combiner where a task sequence is used to facilitate material consumption. The Combiner has one task that consumes one material which has an associated inventory located at the object. Status Labels are placed in the Facility window to show the *QuantityInStock* of each inventory and the aggregate (i.e. total) *QuantityInStock* of the materials, as well as static text to show the initial *QuantityInStock* value for the materials.

Details for Building the Model:

Simple System Setup

- Place two Sources, two Servers, one Combiner, and one Sink in the Facility window.
- Place two ModelEntities in the Facility window.
- Set one Source property *Entity Type* to the first ModelEntity, and the respective property on the other Source to the second ModelEntity.
- Connect the objects via Paths
 - Output@Source1 to Input@Server1; Output@Server1 to ParentInput@Combiner1
 - Output@Source2 to Input@Server2; Output@Server2 to MemberInput@Combiner1
 - Output@Combiner1 to Input@Sink1
- Adjust the size of the ModelEntity which will eventually enter the Combiner at the parent node such that the other entity can fit on the "parent" entity.
- Select the "parent" entity and draw a *BatchMembers* queue (via the Draw Queue icon) on or near the "parent" entity; entering 3D mode (via striking the '3' key) and using the Shift key while moving the queue (with a click-drag motion) will move the queue such that it can be placed on top of the model entity symbol.

Establishing Materials and Inventories

Using Elements, create materials and inventories that are related as indicated by the properties of the elements.

- Create four Material elements and five Inventory elements within in the Definition tab, Element View.
- Set each Material *Location Based Inventory* property to 'True'.
- Set the inventory properties as indicated by the table below.

Inventory	Material Name	Site Object Name	Initial Quantity
Inventory1	Material1	Server1	50
Inventory2	Material1	Server2	50
Inventory3	Material2	Server1	40
Inventory4	Material3	Server2	30
Inventory5	Material4	Combiner1	20

Creating Processing Tasks within the Servers and Combiner

Using Task Sequences, set up a task sequence for each object that requires materials for processing.

- Select Server1 and set property *Process Type* under Process Logic to 'Task Sequence'.
- Select the repeat group symbol (small box in right of property field with "...") in the *Processing Tasks* field to open the task sequence editor.
 - Create two tasks via the Add button.
 - Keep all defaults for the tasks, except change *Material Name* and *Inventory Site Type* in the Materials Requirements section such that the values for one task are 'Material1' and 'ParentObject', respectively and the values for the other task are 'Material2' and 'ParentObject', respectively.
- Select Server2 and repeat the process described for Server1, but instead, values for one task are 'Material1' and 'ParentObject' and the values for the other task are 'Material3' and 'ParentObject'.
- Select Combiner1 and add a task that consumes *Material Name* 'Material4' using *Inventory Site Type* 'ParentObject'.

Creating Status Labels

Using the functions associated with Inventory and Material Elements, create status labels that reflect the *MaterialName*, *InventoryName*, and *QuantityInStock* at each object where there is material stored. In addition, create status labels to show the initial and current aggregate (i.e. total) *QuantityInStock* of each material.

- Near Server1 in the Facility window, create a grid of Status Labels (2 by 3), as shown in the SimBit, and write the following expressions in the respective Status Label Expression property as indicated in the table below, where the first column resolves to the material name of the related inventory, the second column resolves to the inventory that holds the specified material at the specified site, and the third column resolves to the *QuantityInStock* of the specified inventory.

'Inventory1.MaterialName'	'Server1.Inventory(Material1)'	'Inventory1.QuantityInStock'
'Inventory3.MaterialName'	'Server1.Inventory(Material2)'	'Inventory3.QuantityInStock'

- Add a Status Label with static text, like 'Material QIS @ Server1', above the Status Labels created in the previous step for clarity.
- Draw a rectangle (using Drawing ribbon) to place under the Status Labels to visually indicate the relatedness of the Status Labels created in the previous steps.
- Repeat the above steps for Server2 and Combiner1, using the appropriate expressions for each site-material pair.
- Create a grid of Status Labels (4 by 2), as shown in the SimBit, and write the following expressions in the respective Status Label *Expression* property as indicated in the table below, where the first column resolves to material name and the second column resolves to the current aggregate (i.e. total) *QuantityInStock* of the specified material.

'Material1'	'Material1.QuantityInStock'
'Material2'	'Material2.QuantityInStock'
'Material3'	'Material3.QuantityInStock'
'Material4'	'Material4.QuantityInStock'

- Add a Status Label with static text, like 'Material Aggregate QIS', above the Status Labels created in the previous step for clarity.
- Draw a rectangle (using Drawing ribbon) to place under the Status Labels to visually indicate the relatedness of the Status Labels created in the previous steps.
- Create a grid of Status Labels (4 by 2), as shown in the SimBit, and write the following expressions in the respective Status Label *Expression* property as indicated in the table below, where the first column resolves to the material name

and the second column resolves to the static text.

'Material1'	'100'
'Material2'	'40'
'Material3'	'30'
'Material4'	'20'

- Add a Status Label with static text, like 'Material Initial QIS', above the Status Labels created in the previous step for clarity.
- Draw a rectangle (using Drawing ribbon) to place under the Status Labels to visually indicate the relatedness of the Status Labels created in the previous steps.

Notes:

This model is constrained by the available materials, which are required for processing. Because this model does not include material replenishment (i.e. production), the initial *QuantityInStock* of the materials and inventories can be used to identify the constraints. Each processing object (i.e. Server or Combiner) has a material constraint (e.g. Server1 is constrained by Material2, which is entirely stored in Inventory3, with 40 units initially available). The constraint of the entire model is the most rigorous material constraints of the processing objects (i.e. Combiner1 is the system constraint, where Combiner1 is constrained by Material4, which is entirely stored in Inventory5, with 20 units initially available).

Embellishments:

This example shows material consumption from location-based inventory. Material and inventory are not restricted to consumption, they can also be produced. Enhance the model by producing material when it's below a specified level.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

InventoryReplenish - SimBit

This SimBit project includes four models that demonstrate the use of inventory replenishing policies for both 'Continuous' and 'Timer' *Review Period*.

Models included in this SimBit:

1. MinMax – Demonstrates how to incorporate a reorder point (minimum value) and order-up-to-level (maximum) replenishment policy. When the inventory falls to the reorder point, then inventory is replenished to the order-up-to-level.
2. OrderUpTo – Demonstrates how to incorporate an order-up-to replenishment policy. When the inventory falls below the order-up-to-level, then inventory is replenished to the order-up-to-level.
3. ReorderPointReorderQty – Demonstrates how to a reorder point and reorder quantity replenishment policy. When the inventory falls to or below the reorder point, then inventory is replenished with the amount specified in the reorder quantity.
4. Demand-Driven MRP – Demonstrates how to incorporate a demand-driven material requirements planning replenishment policy. When the inventory falls to or below the reorder point (the top of the yellow zone), the policy uses the inventory's net flow position and green zone to determine when to reorder and how much to order .

Model 1: MinMax

Problem:

I have a system that has an inventory replenishment policy that reorders inventory once the inventory in stock reaches a specific value, evaluated either continuously or on a timer.

Categories:

Add-On Process Logic, Materials

Key Concepts:

Add-On Process, Inventory Element, Material Element, MaterialOrderDetail, OnReplenishmentOrder, Produce Step, QuantityProduced, Timer Element

Assumptions:

For a Continuous Review Period, inventory is replenished instantaneously.

Technical Approach:

There are two types of *Review Period* within Inventory Element logic. The inventory logic also includes an On-Replenishment Order Process Add-on Process Logic that employs a Produce step.

Details for Building the Model:

Simple System Setup

- Place a Source, Server, and Sink in the Facility window. Connect the Source to the Server and the Server to the Sink with Paths. Rename the Source as BurgerSource, the Server as BurgerServer, and the Sink as BurgerSink. Place a ModelEntity and set the name as Hamburger.
- Set the *Interarrival Time* on BurgerSource to 'Random.Exponential(2)'.
- Set the *Process Type* on the BurgerServer to 'Task Sequence'. Select the three little dots and Add a sequence. Adjust the *Processing Time* to 'Random.Triangular(1,2,3)'. Go to the Material Requirements dropdown and change the *Material Name* to 'HamburgersMaterial' and the *Inventory Site Type* to 'ParentObject'.
- Place a second Source, second Server, and second Sink in the Facility window. Connect the Source to the Server and the Server to the Sink with Paths. Rename the Source as HotdogSource, the Server as HotdogServer, and the Sink as HotdogSink. Place a second ModelEntity and set the name as Hotdog.
- Set the *Interarrival Time* on HotdogSource to 'Random.Exponential(2)'. Change the *Entity Type* to 'Hotdog'.
- Set the *Process Type* on the HotdogServer to 'Task Sequence'. Select the three little dots and Add a sequence. Adjust the *Processing Time* to 'Random.Triangular(1,2,3)'. Go to the Material Requirements dropdown and change the *Material Name* to 'HotdogsMaterial' and the *Inventory Site Type* to 'ParentObject'.

Creating Elements

- Go to the Definitions window and add two Material Elements, one with the *Name* 'HamburgersMaterial' and the other with the *Name* 'HotdogsMaterial'. Change *Location Based Inventory* for both Material Elements to 'True'. This ensures Inventory Elements can be stored at separate and specific locations within the model.
- Create a Timer Element and change the *Name* to 'BurgerTimer'. Change the *Time Interval* to '10' minutes.
- Create an Inventory Element and change the *Name* to 'HamburgersInventory'. Next, change the *Material Name* to 'HamburgersMaterial', this allows the Inventory Element to store the 'HamburgersMaterial' Material Element in it. Then, change *Site Object Name* to 'BurgerServer', this ensures Simio stores the inventory in a physical location. Adjust the *Initial Quantity* to '8'. Set the *Review Period* to 'Timer', this sets the frequency of the inventory review, which determines whether a replenishment order is required. Set the *Review Timer Name* to 'BurgerTimer' to recall the Timer Element created earlier. Adjust the *Replenishment Policy* to 'Min/Max'. This type of replenishment policy orders up to the Max when the inventory is less than or equal to the Min. Change the *Reorder Point* to '3', when the HamburgerInventory drops to 3 or less, then when the BurgerTimer goes off, a replenishment order will be placed. Set the *Order-Up-To-Level* to '10', this is the level that the HamburgerInventory is replenished to when a replenishment order is placed. Lastly, double-click *On Replenishment Order Process* to create the process that restocks the inventory level. In the Processes window, first place a Delay step with a *Delay Time* of '1' minute to signify the time to restock the Material. Then place a Produce step into your process. Set the *Material Name* to 'HamburgersMaterial', this specifies the material to be produced. Next, adjust the *Inventory Site Type* to 'AssociatedObject', this places the inventory in the BurgerServer. Lastly, change the *Quantity* to 'Token.MaterialOrderDetail.Quantity', this recalls the Max order-up-to value that was defined in the HamburgerInventory and has the process order the corresponding amount.
- Create a second Inventory Element and change the *Name* to 'HotdogsInventory'. Then, change the *Material Name* to 'HotdogsMaterial' and the *Site Object Name* to 'HotdogServer'. Set the *Initial Quantity* to '8'. Set the *Review Period* to 'Continuous', this has the model constantly checking the inventory level. Adjust the *Replenishment Policy* to 'Min/Max' and set the *Reorder Point* to '3' and the *Order-Up-To-Level* to '10'. This ensures as soon as the HotdogsInventory reaches 3, it will immediately replenish. Lastly, double-click *On Replenishment Order Process* to create the replenishment process. In the Process window, first place a Delay step with a *Delay Time* of '1' minute to signify the time to restock the Material. Then drag a Produce step into the newly created process. Set the *Material Name* to 'HotdogsMaterial', the *Inventory Site Type* to 'AssociatedObject' and the *Quantity* to 'Token.MaterialOrderDetail.Quantity'.

Enhancements (done within this model):

- Create Status Labels to show the current inventory in each Server.
- Create Status Plots to show the inventory level in each Server over time.

Model 2: OrderUpTo

Problem:

I have a system that has an inventory replenishment policy that reorders inventory to a specified level evaluated either continuously or on a timer.

Categories:

Add-On Process Logic, Materials

Key Concepts:

Add-On Process, Inventory Element, Material Element, MaterialOrderDetail, OnReplenishmentOrder, Produce Step, QuantityProduced, Timer Element

Assumptions:

For a Continuous Review Period, inventory is replenished instantaneously.

Technical Approach:

There are two types of *Review Period* within Inventory Element logic. The inventory logic also includes an On-Replenishment Order Process Add-on Process Logic that employs a Produce step.

Details for Building the Model:

Simple System Setup

- Click on the Folder icon in the Navigation window in the upper-right hand corner of your Simio window. Select the MinMax model and press 'Ctrl C' on your keyboard to copy the model and then press 'Ctrl V' on your keyboard to

paste the model. Right click on the new model and rename it 'OrderUpTo'.

- Go into your OrderUpTo model and go to the Definitions window. Select HamburgersInventory and adjust the *Replenishment Policy* to 'Order-Up-To'. Repeat the same step for HotdogsInventory. These steps ensure that the inventory levels are replenished to 10 regardless of what the current inventory level is, and the inventory is evaluated on a 10-minute Timer for HamburgerInventory and continuously for HotdogInventory.

Model 3: ReorderPointReorderQty

Problem:

I have a system that has an inventory replenishment policy that reorders a specific amount of inventory once the current inventory level reaches a certain level.

Categories:

Add-On Process Logic, Materials

Key Concepts:

Add-On Process, Inventory Element, Material Element, MaterialOrderDetail, OnReplenishmentOrder, Produce Step, QuantityProduced, Timer Element

Assumptions:

For a Continuous Review Period, inventory is replenished instantaneously.

Technical Approach:

There are two types of *Review Period* within Inventory Element logic. The inventory logic also includes an On-Replenishment Order Process Add-on Process Logic that employs a Produce step.

Details for Building the Model:

Simple System Setup

- Click on the Folder icon in the Navigation window in the upper-right hand corner of your Simio window. Select the MinMax model and press 'Ctrl C' on your keyboard to copy the model and then press 'Ctrl V' on your keyboard to paste the model. Right click on the new model and rename it 'ReorderPointReorderQty'.
- Go into your OrderUpTo model and go to the Definitions window. Select HamburgersInventory and adjust the *Replenishment Policy* to 'Reorder Point/ Reorder Qty'. Repeat the same step for HotdogsInventory. These steps ensure that the inventory levels are replenished by 10 when the current inventory level reaches 3, and the inventory is evaluated on a 10-minute Timer for HamburgerInventory and continuously for HotdogInventory.

Model 4: Demand-Driven MRP

Problem:

I have a system that has an inventory replenishment policy that reorders a specific amount of inventory once the current inventory level reaches a certain level.

Categories:

Add-On Process Logic, Materials

Key Concepts:

Add-on Process, Inventory Element, Material Element, MaterialOrderDetail, OnReplenishmentOrder, Produce Step, QuantityProduced, Timer Element

Assumptions:

For a Continuous Review Period, inventory is replenished instantaneously.

Technical Approach:

There are two types of *Review Period* within Inventory Element logic. The inventory logic also includes an On-Replenishment Order Process Add-on Process Logic that employs a Produce step.

Details for Building the Model:

Simple System Setup

- Click on the Folder icon in the Navigation window in the upper-right hand corner of your Simio window. Select the

MinMax model and press 'Ctrl C' on your keyboard to copy the model and then press 'Ctrl V' on your keyboard to paste the model. Right click on the new model and rename it 'DemandDrivenMRP'.

- Go into your OrderUpTo model and go to the Definitions window. Select HamburgersInventory and adjust the *Replenishment Policy* to 'Demand-Driven MRP'. Repeat the same step for HotdogsInventory. Set *Red Zone Size* to '3', *Yellow Zone Size* to '5', *Green Zone Size* to '10', and *Qualified Spike Demand* to '0'. The inventory is evaluated on a 10-minute Timer for HamburgersInventory and continuously for HotdogInventory.

Model 5: Demand-Driven MRP with Tables

Problem:

I have a system that has an inventory replenishment policy that reorders a specific amount of inventory once the current inventory level reaches a certain level. Demand-driven MRP methodology with a table-based approach is used for defining materials, inventories and buffer zone sizes.

Categories:

Add-on Process Logic, Materials

Key Concepts:

Add-on Process, Data Tables, Demand-Driven MRP, Inventory Element, Material Element, MaterialOrderDetail, OnReplenishmentOrder, Produce Step, QuantityProduced, Timer Element

Assumptions:

For a Continuous Review Period, inventory is replenished instantaneously.

Technical Approach:

There are two types of *Review Period* within Inventory Element logic. The inventory logic also includes an On-Replenishment Order Process Add-on Process Logic that employs a Produce Step. This model differs from the above model in that the materials and inventories elements are 'auto-created' using tables. Properties for the elements are also specified and referenced via data tables.

Details for Building the Model:

Simple System Setup

- Click on the Folder icon in the Navigation window in the upper-right hand corner of your Simio window. Select the DemandDrivenMRP model and press 'Ctrl C' on your keyboard to copy the model and then press 'Ctrl V' on your keyboard to paste the model. Right click on the new model and rename it 'DemandDrivenMRPwithTables'.
- Go to the Definitions window and delete the Materials (HamburgersMaterial and HotDogsMaterial) elements and Inventory (HamburgersInventory and HotDogsInventory) elements. These will be automatically generated using data tables as described below.

Adding Data Tables

- Click on the Data tab and add 3 tables (using Add Table > Add Data Table). Name them Materials, Inventories and BufferZoneSizes.
- Within the Materials table, add a Material element reference property column named 'MaterialName' and select Set Column As Key.
- Within the properties of this column, change the *Reference Type* to 'Create' and *Auto-set Table Row Reference* to 'True'.
- Select the ... on the *Initial Property Values* property and enter the following:
 - *Property Name* 'LocationBasedInventory' and *Value* 'True'
 - *Property Name* 'LogMaterialUsage' and *Value* 'True'
- Add the rows for the two materials within the column, 'HamburgersMaterial' and 'HotDogsMaterial'.
- Note that because these are *Reference Type* 'Create', the two material elements within the Definitions table will be automatically created.
- Within the Inventories table, add an Inventory element reference property named 'InventoryName' and select Set Column as Key.

- Add a Foreign Key property named 'MaterialName' with the *Table Key* property set to 'Materials.MaterialName' referencing the Materials data table.
- Add an Object type reference property column with the name 'ObjectName'.
- Add a Real property named 'InitialQuantity'.
- Add an Enumeration property named 'ReviewPeriod' with the *Enum Type* 'InventoryReviewPeriod'.
- Add a Timer Element property named 'ReviewTimerName'.
- Add a Replenishment Policy property with the name 'ReplenishmentPolicy' – this column's properties will be edited in a later step.
- Add an Expression property named 'ReorderPointOrCondition' and a Real property named 'ReorderQuantityOrUpToLevel'.
- Finally, add a Process Element reference property named 'OnReplenishmentOrderProcess'.
- Add rows to the table for Inventory Names 'HamburgersInventory' and 'HotDogsInventory'
- For HamburgersInventory row, add associated *MaterialName* 'HamburgersMaterial', 'BurgerServer' for the *ObjectName*, *InitialQuantity* of 8, *ReviewPeriod* 'Timer' with *ReviewTimerName* of 'BurgerTimer'. *ReplenishmentPolicy* is 'Demand-Driven MRP' (use 3 / 10 for *ReorderPointOrCondition* and *ReorderQuantityUpToLevel*, respectively) and *OnReplenishmentOrderProcess* of 'HamburgersInventory_OnReplenishmentOrder'.
- For HotDogsInventory row, add associated *MaterialName* 'HotDogsMaterial', 'HotDogServer' for the *ObjectName*, *InitialQuantity* of 8, *ReviewPeriod* 'Continuous' with no (blank) *ReviewTimerName*. *ReplenishmentPolicy* is 'Demand-Driven MRP' (use 3 / 10 for *ReorderPointOrCondition* and *ReorderQuantityUpToLevel*, respectively) and *OnReplenishmentOrderProcess* of 'HotDogsInventory_OnReplenishmentOrder'.
- Next, click on the *InventoryName* key column and within the properties of the column, change the *Reference Type* to 'Create'. Change the *Auto-set Table Row Reference* property to 'True'.
- Select the ... on the *Initial Property Values* property and enter the following:
 - *Property Name* 'Report Statistics' and *Value* 'True'
 - *Property Name* 'MaterialName' and *Value* 'Inventories.MaterialName'
 - *Property Name* 'SiteObjectName' and *Value* 'Inventories.SiteObjectName'
 - *Property Name* 'InitialQuantity' and *Value* 'Inventories.InitialQuantity'
 - *Property Name* 'ReviewPeriod' and *Value* 'Inventories.ReviewPeriod'
 - *Property Name* 'ReviewTimerName' and *Value* 'Inventories.ReviewTimerName'
 - *Property Name* 'ReplenishmentPolicy' and *Value* 'Inventories.ReplenishmentPolicy'
 - *Property Name* 'OnReplenishmentOrderProcess' and *Value* 'Inventories.OnReplenishmentOrderProcess'
 - *Property Name* 'LogInventoryReviews' and *Value* 'True'
- Finally, select the Replenishment Policy column and under the *Default Values* of Dynamic Properties ... enter the following:
 - *Name* 'RedZoneSize' and *Value* 'BufferZoneSizes[1].RedZoneSize'
 - *Name* 'YellowZoneSize' and *Value* 'BufferZoneSizes[1].YellowZoneSize'
 - *Name* 'GreenZoneSize' and *Value* 'BufferZoneSizes[1].GreenZoneSize'
 - *Name* 'QualifiedSpikeDemand' and *Value* '0'
 - *Name* 'ReorderCondition' and *Value* 'Inventories.ReorderPointOrCondition'
 - *Name* 'ReorderPoint' and *Value* 'Inventories.ReorderPointOrCondition'
 - *Name* 'ReorderQuantity' and *Value* 'Inventories.ReorderQuantityOrUpToLevel'

-
- *Name* 'OrderUpToLevel' and *Value* 'Inventories.ReorderQuantityOrUpToLevel'
 - Within the BufferZoneSizes table, add a foreign key property named 'InventoryName' and set the *Table Key* property to 'Inventories.InventoryName'.
 - Add three Real type properties and name them *RedZoneSize*, *YellowZoneSize* and *GreenZoneSize*.
 - Add one row each to the table for Inventory Names 'HamburgersInventory' and 'HotDogsInventory'. Set the *RedZoneSize* for both to '3', the *YellowZoneSize* for both to '5' and the *GreenZoneSize* for both to '10'.
 - Note that these property values for the inventories are referenced in the dynamic property values shown above for the corresponding inventories. If the buffer zone sizes change over time (as with the DDMRP calculators), the table is specified as a Time-Index table and the values are referenced with BufferZoneSizes.RedZoneSize.TimeIndexedValue, etc. notation.
-

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

KeepingWorkerReserved - SimBit

Problem:

I would like for the same vehicle or worker to perform both transport and processing tasks for a particular entity as it moves through the system.

Categories:

Animation, Entity Characteristics, Decision Logic -- Processing, Worker

Key Concepts:

Connector, Keep Reserved If, Path, Ride On Transporter, Secondary Resources, Server, Sink, Source, Worker

Assumption:

The worker or vehicle can only have a ride capacity of '1' to be reserved, otherwise a runtime warning will be issued.

Technical Approach:

The nurse (worker) will be used to transport entity from Source to first Server and then be used for processing at the Server. Upon completion of processing, the same nurse (worker) will transport the entity to the next Server and again be used for processing and transport to the Sink. The Keep Reserved If property within the TransferNode (for transport tasks) and within the Server (for processing tasks) will be set to 'True' (or value of '1').

Details for Building the Model:

Simple System Setup

- Place a Source, two Servers and a Sink within the Facility window. Connect Source1 to Server1, Server1 to Server2 and Server2 to Sink with Paths. Connect the Sink back to the Source with a Path.
- Within the Source1, change the *Interarrival Time* to 'Random.Exponential(.3)' minutes.
- Connect the input node of each Server to the output node of the same Server with Connectors. This will allow the Worker to travel both to/from the Servers.
- Place a ModelEntity from the Project Library and rename it 'Patient'.
- Place a Worker in the Facility window and rename it 'Nurse'. Change the *Park While Busy* property to 'True'. Specify the *Initial Node (Home)* location as Output@Source1 and the *Idle Action* to 'Go To Home' so that any idle workers will return to the Source. Specify the *Initial Number in System* as '3' so that there are 3 Nurses.

Using the Worker for Transport Tasks

- Within the output nodes of the Source, Server1, and Server2, change the *Ride On Transporter* to 'True' and the *Transporter Name* to 'Nurse'. Within the Source1 and Server1 output nodes, specify the *Keep Reserved If* property as 'True'. This will cause the Nurse, upon dropping off the entity at its destination location, to be 'reserved' so that it may immediately be used then for processing at the server.

Using the Worker for Processing Tasks

- Within the Server1 and Server2, change the *Processing Time* to 'Random.Triangular(.2,.3,.4)' minutes and the *Input Buffer* capacity to '0'.
- Within the Secondary Resources section of properties, specify the *Object Name* as 'Nurse'. Specify the *Keep Reserved If* property as 'True'. This will cause the Nurse, upon completion of the processing task, to be 'reserved' so that it may immediately be used then for transport to the next location.

Enhancement:

In this example, there are status label graphics attached to both the ModelEntity and Worker objects. Use the Entity.ID function to return information specific to the entity (i.e, String.Format("P#{0}", Entity.ID)) or the Entity.Population.Index function for information on the specific Worker (String.Format("Nurse#{0}", Entity.Population.Index)).

Send comments on this topic to [Support](#)

KeepQueueTimeForLast10Entities - SimBit

Problem:

I want keep an updated value for the total queue time for the last ten (10) entities in a Server queue.

Categories:

Add-On Process Logic, Buffering, Custom Statistics, Entity Characteristics

Key Concepts:

Add-On Process, Before Processing, Create Step, Decide Step, Destroy Step, Entered, Insert Step, ModelEntity, Real State Variable, Remove Step, Search Step, Status Label, Status Plot, Storage Element, Storage Queue

Assumption:

The total queue time of entities in the queue is only calculated once there are ten values accumulated. From then, the calculation is done on the ten most recent entities in the queue. If the entity does not wait at all, a value of zero is recorded and is included in the total.

Technical Approach:

When an entity enters the Server, its ModelEntity.TimeInQueue is set to the current simulation time. Once the entity has been allocated the Server capacity, a copy of that entity is created that will be stored in a storage queue so that statistics may be calculated. Once ten copy entities are in the storage, the queue is searched and time in queue time calculated. Each time a new entity enters (after the first 10), the first entity is then removed from the storage and system.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server and a Sink from the Standard Library into the Facility window. Use Paths to connect the Source to the Server and the Server to the Sink.

Adding a New State to ModelEntity

- Within the Navigation window, click on ModelEntity. Within the ModelEntity Definitions window, click on the States panel and add a new Real type State with the *Name* 'TimeInQueue'. This will be referenced as ModelEntity.TimeInQueue in the main model.
- Go back to the Model and within the Facility window, place a ModelEntity from the Project Library. While this object is highlighted, click on the Symbols ribbon and place a Status Label (attached) to the entity. This should automatically say *Attached To* 'DefaultEntity' and then add the *Expression* 'TimeInQueue' This way, when each entity instance moves through the system, you can see its time in queue calculation.

Define a New State and Element for the Model

- Within the Definitions window, Elements panel, add a new Storage type element with the *Name* 'Storage1'. This will be used to 'store' all the 10 entity copies for waiting time calculations.
- Also within the Definitions window, click on the States panel and add a new Real type state with the *Name* 'TotalWait'.

Adding Logic to the Server To Calculate Queue Times

- Within the Server, add a new process to the *Entered* Add-On Process Trigger named 'Server1_Entered'. Within the Processes window, add an Assign step to this process. The *State Variable Name* should be 'ModelEntity.TimeInSystem' and the *New Value* should be 'Run.TimeNow'. This will "mark" the time that the entity entered the queue for the server with the current simulation time which will be re-calculated later.
- Go back to the Facility window and again within the Server, add a process name to *Before Processing* of 'Server1_BeforeProcessing'. Within this process add the following steps:
 - Create step with *Create Type* of 'CopyAssociatedObject' and *Object Instance Name* of 'DefaultEntity'.
 - From the Created exit of the Create step, add an Assign step. The *State Variable Name* should be 'ModelEntity.TimeInQueue' and the *New Value* should be 'Run.TimeNow – ModelEntity.TimeInQueue'. This re-calculates the state to be the actual waiting time.

-
- Next, add an Insert step with the *Queue State Name* of 'Storage1.Queue'. Add a Decide step with the *Decide Type* of 'ConditionBased' and an *Expression* of 'Storage1.Queue.NumberWaiting == 10'. This will allow us to only calculate the total wait time once there are ten entities in the queue.
 - Add a Search step to the process from the True exit of the Decide step. The *Collection Type* is 'QueueState', the *Queue State Name* is 'Storage1.Queue' and the *Search Expression* is 'Candidate.ModelEntity.TimeInQueue'. The sum of this expression for the found items in the queue will be stored in the ReturnValue state of the original executing token (i.e., Token.ReturnValue). The *Limit* in Advanced Options should be set to '10' so that we get the sum of all ten items in the queue.
 - From the Original exit of the Search step, add an Assign step that will assign the *State Variable Name* 'TotalWait' to the *New Value* 'Token.ReturnValue' (as discussed above).
 - Now that the calculation is done, we will remove the first entity in the queue so that when the next entity copy enters, the calculations will be done correctly. This is done by using another Search step to get a pointer to the correct entity. The *Collection Type* is 'QueueState' and the *Queue State Name* is 'Storage1.Queue'. Note that the *Limit* remains the default value of '1', as we want a pointer to the first entity found in that queue.
 - From the Found exit of the Search step, add a Remove step with the *Queue State Name* of 'Storage1.Queue' and then from the Removed exit of the step, place a Destroy step, with *Destroy Type* of 'AssociatedObject' which will dispose of the copied entity/token that is no longer needed.
-

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

LearningCurveWithLookup - SimBit

Problem:

I have a machine that increases its efficiency subject to a learning curve function.

Categories:

Lookup and Rate Tables

Key Concepts:

Lookup Table, Learning Curve, Linear Interpolation, Server, Status Plot

Assumptions:

The Server starts at 20% effective. After 12 hours the Server is 90% effective and after 18 hours the Server is 100% effective. At 100% efficiency, the Server's Processing Time is 10 minutes.

Technical Approach:

The Server's learning curve will be represented by a Lookup Table. The Processing Time property of the Server will reference this Lookup Table.

Details for Building the Model:

Simple System Setup

- Add a Source, Server, and Sink to the Facility Window.

Using a Lookup Table

- In the Data Window, select the Lookup Tables panel and add a Lookup Table with the *Name* property of 'LearningCurve'. Add the following (X, f(X)) pairs in the table: (0, 0.2), (12, 0.9), (18, 1). This lookup will be used within the *Processing Time* of the Server as the efficiency.

Specifying the Server Processing Time

- Update the *Processing Time* of the Server to '10/LearningCurve[Run.TimeNow]'. Run.TimeNow is a function that returns the current simulation time.

Embellishments:

To make this model more specific, try changing the Default entity's Travel Logic to a desired speed, the Source's *Interarrival Time* rate or the *Speed Limits* on the Paths.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

LeveledArrivals - SimBit

Problem:

I would like to model a system where parts of different types arrive in a certain order.

Categories:

Arrival Logic, Data Tables

Key Concepts:

AvailableRowCount, Before Creating Entities, Before Exiting, Condition, Data Table, Dynamic Object Property, Entity Type, Math.If(), Real State Variable, Source, State Assignments, Table Row Referencing

Assumption:

This is a simply Source, Server, Sink model that simply demonstrates controlling the order of arriving entities.

Technical Approach:

The Source reads information from a Data Table to determine which entity type to create for each arrival.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server and a Sink into the Facility window and connect them with Paths.
- Place 4 ModelEntity objects into the Facility window and name them PartA, PartB, PartC and PartD.
 - Ensure that the color of each entity is different to help with model verification. To change the color of an entity, select the entity and then select a color from the Color drop down in the Symbols ribbon and click back onto the entity. It should then change colors.

Create Data Table

- The Data Table will contain the order in which the different entities should arrive.
- Go to the Data window and click the Add Data Table icon in the ribbon to add a new table.
- Add a column that is an Entity property by selecting Entity from the Object Reference drop down in the ribbon. The column can be renamed to Parts.
- Fill the table with any orders you'd like – in this example, they arrive in the order A A A B B B C C D and this pattern repeats.

Add Table Referencing To Source object

- Go to the Definitions window and create a new State variable. From within the Definitions window, click on the States panel along the left side of the interface and then click on Real in the ribbon to create a new Discrete State. Name it 'RowNumber'. Change the *Initial State Value* to '1'.
- From within the Facility window, click on the Source and expand the *Table Row Referencing* category. Next, expand the *Before Creating Entities* category and set the *Table Name* to 'Table1' and the *Row Number* to 'RowNumber' (the new State). This is telling the Source object to set a reference to the Data Table before an entity is created.
- Set the *Entity Type* property of the Source to be Table1.Parts, where Table1 is the name of the Data Table and Parts is the name of the column within the table. This will tell the Source to look in this table to determine which type of entity to create.
- Expand the *State Assignments* property category and open the Repeating Property editor of the *Before Exiting* property (by clicking on the ellipse in the text box). Click Add to create a new Assignment and set *State Variable Name* to 'RowNumber' and *New Value* to 'Math.If(RowNumber == Table1.AvailableRowCount, 1, (RowNumber + 1))' This checks to see if the current value of RowNumber is equal to the total number of rows in the table (i.e. are we at the last row?). If True, it sets the value of RowNumber back to 1 so it reads at the beginning of the table again. If False, RowNumber is incremented by 1.

Embellishments:

Change the pattern of arrivals or try adding additional information into the Data Table and using it in the model.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

LogicBasedOnEntityProperty - SimBit

Problem:

I have entities that need to be sent to certain destinations based on their entity type.

Categories:

Decision Logic -- Paths

Key Concepts:

Numeric Property, Path, Selection Weight

Assumptions:

Each source only produces one type of entity. Source1 produces ModelEntity1 (Green) and Source2 produces ModelEntity2 (Red).

Technical Approach:

An Integer Property will be added to the Default Entity. The Property will be set as unique for each entity instance. The Property will then be referenced in order to make a decision.

Details for Building the Model:

Simple System Setup

- Add two ModelEntity objects from the Project Library and two Sources, a Server, and two Sinks from the Standard Library to the Facility Window.

Adding an Integer Property to the Default Entity

- Highlight ModelEntity in the Navigation window (not the Model) and select the Definitions tab and Properties panel.
- Add a new integer property by clicking on Standard Property > Integer.
- Change the *Name* property to 'EntityType'. You can also change its default value here if you like.

Defining the EntityType

- In the Facility Window of the Model, in the ModelEntity1 and ModelEntity2 instances, you will see the new *EntityType* property you just created.
- Set the *EntityType* Property to '1' and '2', respectively.

Creating the Logic Decision

- To make the decision, you will take advantage of the *Selection Weight* property on the outgoing links from Server.
- For one link, change the *Selection Weight* to the expression 'ModelEntity.EntityType==1'. For the other, change the *Selection Weight* to the expression 'ModelEntity.EntityType==2'.

Discussion:

This procedure of using a Property works well for entities that will not change during the run (e.g. ModelEntity2 always has a value of 2). If "EntityType" is something that could change during the run, you would instead make it a State (Properties cannot change during a run, but States can).

See Also:

LogicBasedOnEntityState.spf

LogicBasedOnEntityState - SimBit

Problem:

I have entities that need to be sent to certain destinations based on their entity type.

Categories:

Decision Logic – Paths, EntityCharacteristics

Key Concepts:

Before Exiting, Path, Real State Variable, State Assignments

Assumptions:

Each source only produces one type of entity. Source 1 produces ModelEntity1 (Green) and Source 2 produces ModelEntity2 (Red).

Technical Approach:

A Discrete State Variable will be added to the ModelEntity. An Add-On Process will be used to assign this State Variable a value based on the entity type. The State will then be referenced in order to make a decision.

Details for Building the Model:

Simple System Setup

- Add two ModelEntity objects from the Project Library and two Sources, a Server, and two Sinks from the Standard Library to the Facility Window.
- Change the name of the ModelEntity objects to ModelEntity1 and ModelEntity2 and make sure that Source1 has an Entity Type of 'ModelEntity1', while Source2 has a an Entity Type of 'ModelEntity2'.

Creating the Discrete State Variable

- Highlight ModelEntity in the Navigation window (not the Model) and select the Definitions tab and States panel.
- Add a Discrete State Variable to the ModelEntity with *Name* 'EntityType'.

Assigning the State Variable

- Within the State Assignments section of the Sources, enter the *Before Exiting* repeating editor and add the *State Variable Name* of 'ModelEntity.EntityType' a *New Value* of '1.0' (for Source1) and *State Variable Name* of 'ModelEntity.EntityType' a *New Value* of '2.0' (for Source2).

Creating the Logic Decision

- To make the decision, you will take advantage of the *Selection Weight* property on the outgoing links from the Server.
- For one link, change the *Selection Weight* as the expression 'ModelEntity.EntityType==1.0'. For the other, use the expression 'ModelEntity.EntityType==2.0'.

Discussion:

This procedure of using a State works well for items that could change during the run (e.g. perhaps EntityType needs to change as it proceeds through processing). If the item will not change during the run, it would be easier and more efficient to have EntityType be a Property instead of a State of the ModelEntity.

See Also:

LogicBasedOnEntityProperty.spf

MassFlowProcessing - SimBit

Problem:

I have 2 stockpiles of material and I want material to flow from the first stockpile to the second at a variable rate. When the flow rate changes, the second stockpile has to wait for the new flow rate to reach it before noticing a change in flow rate.

Categories:

Building New Objects / Hierarchy, Level States

Key Concepts:

Assign Step, Button, Conveyor, Create Step, Delay Step, EndTransfer Step, Event, Expression Property, External View, ExternalNode, Fixed Class Object, Label, Level State Variable, OnInitialized Process, OnRun Initialized Process, Process Triggering Event, Real State Variable, Size.Height, Size.Length, Size.Width, Station Element, Status Label, Status Label, Timer Element, Transfer Step

Assumptions:

All flows go from the left to the right. Increased flow means more flow is leaving the first stockpile, meaning the flow rate becomes more negative (and vice versa - a decrease in flow means less material is leaving and the flow becomes less negative).

Technical Approach:

We will create a new object called a StockPile and represent the amount of material¹ in each stock pile with a Level State Variable. At various events, we will change the outgoing rate of the first stockpile object and pass the new rate to the second stockpile object using an entity to represent the leading edge of the new flow of material.

Details for Building the Model:

Adding a State to ModelEntity

- First we must create the State on the Entity that will carry the new Flow Rate information from one pile to the other. To do this, click on ModelEntity in the Navigation Window and go to the States Panel in the Definitions Tab. Add a Real State and name it 'FlowChange'.

Creating the StockPile Object Definition

- In your Project Home Tab, click 'New Model' to add a Fixed Class Object to your Project Library. Rename this Object 'StockPile' by right-clicking on the new object and choosing 'Rename'.
- In the Elements Panel of the Definitions Tab, add a new Station Element by clicking the button in the Ribbon. *Name* this station 'Pile'.
- Next, add an Expression Property from the Standard Property drop-down list in the Ribbon of the Properties Panel. *Name* this property 'InitialOreQuantity'. Change the *Category Name* to 'Initial Ore Quantity'.
- In the States Panel, add a Level State and *Name* it 'OreQuantity'.

Defining the StockPile Object's External View

- In the External Panel of the Definitions tab, you can (optionally) give your StockPile a new Symbol by clicking on Place Symbol button in the Ribbon. If you have an applicable 3D image you can choose Import Symbol. If not, you can choose to Download a Symbol from Trimble Warehouse. You can find this particular symbol by searching for "Trash Pile" – it will be the first option. You may want to resize your symbol in the symbol editor.
- After placing the symbol, click on the External Node button in the Ribbon and bring in 2 External Nodes. Place one on the left-hand side of the object and one on the right-hand side. Click on the left-hand node. Change its *Node Class Name* to 'BasicNode', its *Input Location Type* to 'Station' and the *Station Name* to 'Pile'. Rename this node 'Input'.
- For the right-hand node, change its *Node Class Name* to 'TransferNode' and change its *Name* to 'Output'. Leave the *Input Location Type* to 'None', since this will be the output node.

Defining the StockPile Object Processes

- In the Processes window, select 'OnRunInitialized' from the Select Process drop-down to create a new

OnRunInitialized Process. Drag in an Assign Step from the list of Common Steps. Set the *State Variable Name* to 'OreQuantity' and right-click on the *New Value*, click on Set Referenced Property and select the Property 'InitialOreQuantity' from the list. This Process will set the initial value for each stockpile's ore level to match the user-defined initial ore quantity property that will later be defined in the Model.

- Next, click Create Process in the Ribbon. Set the *Triggering Event* in the Properties Window to 'Pile.Entered'. This process will then be activated when the pile is entered by an entity. Within the process, place an EndTransfer step to signal that the Station has accepted the exchange. After that, place an Assign step that assigns the *State Variable Name* of 'OreQuantity.Rate' the *Value* of '-ModelEntity.FlowChange'. This allows the receiving StockPile to receive at a rate equal (but opposite) to the rate that is flowing out of the first StockPile. Lastly, place a Destroy step to destroy the entity carrying the information after it has been received.

Creating the Model

- In your 'Model' Object, drag in 2 StockPile objects from the Project Library and connect them with a Conveyor. Set the Conveyor's *Initial Desired Speed* to '.05 Meters per Second'.
- In the Elements Panel in the Definitions Tab, add a Timer Element and name it 'RateChange'. Set its *Time Interval* to 'Random.Exponential(10)' and the *Units* to 'Minutes'. This timer will represent random points in time that the flow rate could change in a system. This could be the arrival of a new truck, another worker is on shift, etc.
- In your Events Panel, add an Event and name it 'Stop'.

Adding Model Processes Window

- In the Process tab, click on the Create Process button in the ribbon and create a process with the *Name* 'TimerRateChange'. Set the *Triggering Event* for this Process to 'RateChange.Event'. First, place an Assign step into the process. Set the *State Variable Name* to 'StockPile1.OreQuantity.Rate' and the *New Value* to 'StockPile1.OreQuantity.Rate + Random.Uniform(-10,0)'. This will Increase the "Flow Out" of the stockpile.
- Next, we create an entity to carry the new Rate down the conveyor to the next stockpile. This entity will act as the leading edge of the new flow of material. To do this, we must place a Create step after the Assign and create a *DefaultEntity Object Instance Name*.
- From the Created exit of the Create step, we need to Assign the new Flow Rate that was created in the first Step so we place an Assign. Set the *State Variable Name* to 'ModelEntity.FlowChange' and its *New Value* to 'StockPile1.OreQuantity.Rate'.
- Finally, place a Transfer Step after the Assign. We will Transfer *From* 'FreeSpace' (where the entity is Created), *To* 'Node', *Node Name* ' Output@StockPile1'.
- We want the almost the exact same process for the Stop Event. Copy the first Process by clicking in the white space of the Process. When the process gets shaded with diagonal lines and the Properties window changes to the Process Properties, it means that it is selected, so press Ctrl+C, then Ctrl+V to copy and paste the entire Process.
- Change the *Name* to 'StopFlow', *Triggering Event* to 'Stop' and in the first Assign step change the *New Value* to '0'.

Animating the Stockpiles

- To represent the animation of the stockpiles, we need to make a new process. In the Processes Window, click the Select Process button in the Ribbon and choose 'OnInitialized' from the drop down list. Place an Assign Step to assign the Height, Length, and Width of the StockPiles according to the size of the OreQuantity State. To do this, set the first *State Variable Name* to 'StockPile1.Size.Height' and the *New Value* to 'StockPile1.OreQuantity/20'.
- Click the '...' button next to Assignments (More) and add 5 additional items. Set the first item's *State Variable Name* to 'StockPile1.Size.Length' and its *New Value* to 'StockPile1.OreQuantity/10'. Similarly, the next item will be 'StockPile1.Size.Width' and its *New Value* will be 'StockPile1.OreQuantity/10'. Fill out the next 3 items with the same *State Variable Names* and *Values* but with StockPile2 instead.
- Add a Delay step and set the *Delay Time* to an appropriate amount. 0.5 min was used in this example.
- Drag the process Endpoint to the input of the Delay to create a loop. This allows the animation to update every time the Delay is executed.

Enhancing Animation

- In the Animation Ribbon in the Facility tab, add a Button and the *Button Text* to 'Stop Flow' and set the *Event Name* to 'Stop'.
- Attach Status Labels to each StockPile by clicking on each StockPile object, then clicking the Status Label button in the Symbols Tab. Set the *Expression* to 'OreQuantity', and add another for 'OreQuantity.Rate'. Add both Status Labels for the other StockPile objects, as well.
- Drag in a ModelEntity Instance and attach a Status Label to it and set this *Expression* to 'FlowChange'.

Embellishments:

If left running, StockPile1's OreQuantity will become extremely negative, which in reality is impossible. To end the run when

StockPile1's OreQuantity reaches 0.0, see the SimBit 'UsingMonitor' for detailed instructions.

¹Material refers to the substance flowing from one stockpile to the other – not to be confused with a Simio Material Element

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

MaterialStorageBasicConcepts - SimBit

Material Storage Basic Concepts

This SimBit project includes three models providing examples of using material storage Elements, Properties, States, Functions, and Process Logic.

1. **FundamentalMaterialStorageConcepts:** Demonstrates using the Storage Area and Storage Bin Elements with the AddStock and RemoveStock steps.
2. **HandlingStockRequirements:** Demonstrates creating stock requirements and fulfilling the requirements with reservations. Rules are created to prioritize what area or bin is selected to complete these reservations.
3. **QuantsTables:** Demonstrates recording the quant information in a bin and how to enhance selection conditions to consider information about the quants.

NOTE: We highly recommend reading through the “Material Storage – Discussion and Examples” Help page and any associated Help pages to learn more about material storage prior to reviewing the models in this SimBit. These pages are helpful for getting a base understanding of the features and functionality of material storage. Help pages can be found via the Support ribbon in Simio.

Problem:

Stock in my system should be stored in a bin in an area. The materials have different volumes, and the bins have different capacities. I want to add and remove stock from the bins in an area.

Categories:

Element, Material Storage

Keywords:

Element, Material, Storage Area, Storage Bin

Assumptions:

Bins will contain initial stock.

Technical Approach:

This model is generated with data and uses a data driven approach. The Storage Area, Storage Bin, and Material Elements are created in Data Tables. Data Tables define each Area's child Areas or child Bins and the initial contents of Bins. Stock will be added and removed from the Bins with Process logic triggered when the run is initialized.

Details for Building the Model:

Data Window: Tables

- Create a new Data Table and name it “StorageAreas”
- In this table add a Storage Element Area Property column, name it “StorageArea”
- Make StorageArea a Key by using the Set Column As Key button
- Change the following properties on the StorageArea column
 - *Reference Type* = 'Create'
 - *Initial Property Values*
 - Item one
 - *Property Name* = 'ChildStorageLocations'
 - *Value* = 'AreaAssignments'

- Item two
- *Property Name* = 'ChildStorageLocations[0].StorageLocationType'
- *Value* = 'AreaAssignments.StorageLocationType'
- Item three
- *Property Name* = 'ChildStorageLocations[0].StorageBinName'
- *Value* = 'AreaAssignments.ChildStorageBin'
- Item four
- *Property Name* = 'ChildStorageLocations[0].StorageAreaName'
- *Value* = 'AreaAssignments.ChildStorageArea'
- *Auto-set Table Row Reference* = 'True'
- Add the following data to the table:
- Area1
- Area2
- Area3
- Create a new Data Table and name it "StorageBins"
- Add the following columns and name them accordingly:
- Storage Bin Element Property – "StorageBin"
- Real Property – "VolumeCapacity"
- Make the following changes to the StorageBin column:
- Set this column as a Key by using the Set Column as Key button.
- *Reference Type* = 'Create'
- *Initial Property Values*
- Item one
- *Property Name* = 'InitialVolumeCapacity'
- *Value* = 'Storage.VolumeCapacity'
- *Auto-set Table Row Reference* = 'True'
- Populate the table with the following information:

StorageBins	VolumeCapacity
StorageBin1	1
StorageBin5	5
StorageBin10	10

- Create a new Data Table and name it "AreaAssignments"
- Add the following columns, name them accordingly, and change the following properties:
- Foreign Key Property – "ParentStorageArea"

- *Table Key* = 'StorageAreas.StorageArea'
- Enumeration Property – "Storage Location"
- *Enum Type* = 'StorageLocationType'
- *Default Value* = 'Area'
- Storage Bin Element Property – "ChildStorageBin"
- *Required Value* = 'False'
- Storage Area Element Property – "ChildStorageArea"
- *Required Value* = 'False'
- Populate the table with the following information:

ParentStorageArea	StorageLocation	ChildStorageBin	ChildStorageArea
Area1	Bin	StorageBin1	
Area1	Bin	StorageBin5	
Area2	Bin	StorageBin10	
Area3	Area		Area1
Area3	Area		Area2

- Create a new Data Table and name it "Materials"
- Add the following columns and name them accordingly:
- Material Element Property – "Materials"
- Real Property – "UnitVolume"
- Color Property – "Color"
- Make the following changes to the Materials column:
- Set this column as a Key by using the Set Column as Key button.
- *Reference Type* = 'Create'
- *Initial Property Values*
- Item one
- *Property Name* = 'VolumePerUnit'
- *Value* = 'Materials.UnitVolume'
- Item two
- *Property Name* = 'DisplayColor'
- *Value* = 'Materials.Color'
- *Auto-set Table Row Reference* = 'True'
- Populate the table with the following information:

Material	UnitVolume	Color
Material1	1	Green

Material2	2	LightCoral
Material3	3	CornflowerBlue

- Create a new Data Table and name it "InitialBinContents"
- Add the following columns, name them accordingly, and change the following properties:
- Foreign Key Property – "StorageBin"
- *Table Key* = 'StorageBins.StorageBin'
- Foreign Key Property – "Material"
- *Table Key* = 'Material.Materials'
- Real Property – "Quantity"
- Populate the table with the following information:

StorageBin	Material	Quantity
StorageBin1	Material1	1
StorageBin5	Material2	1
StorageBin10	Material3	2

Processes Window: Process Logic

- Create a new Process and name it "AddAndRemoveStock". This process will add initial stock into the bins, show why a AddStock and RemoveStock might not succeed, and what it looks like when an AddStock and RemoveStock is successful.
- Search step "InitialBinContents"
- *Collection Type* = "TableRows"
- *Table Name* = 'InitialBinContents'
- *Limit* = 'Infinity'
- *Token Wait Action* = 'WaitUntilNewTokenProcessingCompleted'
- Off the Search step's Found branch, we want to add stock for each row in the table.
- AddStock step "InitialStock"
- *Stock Additions*
- First item
- *Storage Bin Name* = 'StorageBins.StorageBin'
- *Material Name* = 'Materials.Material'
- *Quantity* = 'InitialBinContents.Quantity'
- Off the Search step's Original branch, we want to add and remove stock from different bins to try different scenarios.
- AddStock step "NoSpaceInBin"
- *Stock Additions*
- First item
- *Storage Bin Name* = 'StorageBin1'

- *Material Name* = 'Material3'
- *Quantity* = '1'
- *Stock Quant Selection Condition* = 'False'
- RemoveStock step "InsufficientQty"
- *Stock Removals*
- First item
- *Storage Bin Name* = 'StorageBin10'
- *Material Name* = 'Material3'
- *Quantity* = '3'
- AddStock step "Succeeds"
- *Stock Additions*
- First item
- *Storage Bin Name* = 'StorageBin10'
- *Material Name* = 'Material3'
- *Quantity* = '1'
- *Stock Quant Selection Condition* = 'False'
- RemoveStock step "Succeeds"
- *Stock Removals*
- First item
- *Storage Bin Name* = 'StorageBin10'
- *Material Name* = 'Material3'
- *Quantity* = '3'
- Change the color of the process logic steps by using the *Color* property in the General section of the properties on each step.
- Use the Select Process button in the Process ribbon and select OnRunInitialized. This will create the process which is triggered when the run is starting.
- We want this process to execute the AddAndRemoveStock process.
- Execute step "Execute1"
- *Process Name* = 'AndAndRemoveStock'

Facility Window: Animation

- From the Animation ribbon, use the Stack button to draw a Stack that will show the contents of the bin.
- Change the *Type* to 'StorageBin'.
- Select and copy the stack animation twice, so that three Stack animations exist.
- Change the *Storage Bin* property on each Stack so that they show 'StorageBin1', 'StorageBin5', and 'StorageBin10'.

Discussion:

The AreaAssignments table defines the hierarchy of Areas and Bins. Areas can contain both child Areas and child Bins. In

this model, Area3 contains both Area1 and Area2. Area1 contains StorageBin1 and StorageBin5. Area2 contains StorageBin10.

Turn the Trace on in the Run ribbon and run the model. Try running with a Breakpoint on the Succeeds AddStock step.

Change the view in the Facility from 2D to 3D. This can be done by using the 3 key or from the View ribbon. You will see the Stack animations have different vertical heights that correspond to the stock in each bin. If using the breakpoint, you can see the initial bin quantities. As the process logic continues, you can see how the Stack animation changes based on stock being added or removed.

Note the Process Logic steps in Trace when you run. You can turn on Trace from the Run ribbon. If you have colored the steps, the colors will correspond in the Trace window.

When the NoSpaceInBin AddStock is executed, there is no room in StorageBin1 to add Material3. You can see in Trace, it is specified that the Material was unable to be added due to insufficient capacity available.

Similarly, when InsufficientQty RemoveStock is executed, there are not enough units of the Material in the specified Bin to remove the requested amount.

When adding or removing stock, if the entire quantity specified cannot be added or removed, the action will fail and move to check the next item in the list. When the list has been checked and all additions or removals possible are completed, the original Token will depart from the Original exit and continue on in the process.

Watch in Trace when the AddStock and RemoveStock steps succeed.

The *Stock Quant Selection Condition* property can be used to evaluate what quant the stock could be added to. If there is an existing quant that fulfills the condition, the stock could be added into this quant. If no quant fulfills the condition, a new quant will be created. If a new quant should be created each time stock is added into a bin no matter what, the *Stock Quant Selection Condition* property could be set to 'False'. This will reject any candidate quant considered, so the AddStock step will always create a new quant for the material added.

Notes:

Try experimenting with adding and removing different quantities of the material in the different bins. You can change the quantity on the AddStock and RemoveStock steps in the process logic. Notice when an addition or removal fails or succeeds and why.

Consider updating the Storage Bins and Material in the Data Table. Along with the Volume Capacity, the Storage Bins can have Weight Capacity and Total Capacity defined in the table. Along with the Volume per Unit, Materials can also have Weight per Unit and Capacity per Unit defined in the table.

These capacities are independent of each other. The values can all be unique and are also each considered separately. For example, the Bin's Total Capacity might have room for 1 unit, but that 1 unit might be of a Material that exceeds the Bin's Weight Capacity.

For example, you might update the StorageBins and Materials Tables to look like this:

StorageBins	VolumeCapacity	WeightCapacity	TotalCapacity
StorageBin1	1	500	50
StorageBin5	5	500	500
StorageBin10	10	1000	1000

Material	UnitVolume	UnitWeight	UnitCapacity	Color
Material1	1	100	10	Green
Material2	2	200	20	LightCoral
Material3	3	300	50	CornflowerBlue

Model 2: HandlingStockRequirements

Problem:

Orders arrive with a material and quantity requirement. These orders reserve the bin space or stock and then putaway or remove stock based on these reservations. Reservations should be made prioritizing stock in certain bins. Any unmet requirements can use the safety or overflow stock to complete the order.

Categories:

Element, Material Storage

Keywords:

Element, Material, Putaway, Retrieval, Storage Area, Storage Bin

Assumptions:

Bins have initial stock. Putaway and retrieval orders are provided as an input to the system. The arrival time, material, and quantity of each order is known.

Technical Approach:

This model is generated with data and uses a data driven approach. The Storage Area, Storage Bin, and Material Elements are created in Data Tables. Data Tables define each Area's child Areas or child bins and the Initial contents of Bins. Initial Stock will be added with Process logic triggered when the run is initialized.

Entities represent the orders in the system. These entities are created with the requirements for the order, such as the material and quantity. As a putaway or removal order entity enters the system, process logic will trigger to reserve either the space in bins or quantity of stock, depending on the order type. After reservations are made, the reservations will be used to add or remove the corresponding stock.

Details for Building the Model:**Initial Setup**

- Use the "FundamentalMaterialStorageConcepts" model as a starting point. You may right-click the model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Data Window: Tables

- Update the StorageAreas Table to include a new fourth row for "Area4". This will auto-create a new element.
- Add two new columns to the StorageBins table:
- Expression Property "VolumeCapacityAvailable"
- Expression Property "CurrentStockVolume"
- Populate the table with the following information:

StorageBin	VolumeCapacity	VolumeCapacityAvailable	CurrentStockVolume
StorageBin1	1	StorageBin1.CurrentVolumeCapacityAvailable	StorageBin1.CurrentStock.Volume
StorageBin5	5	StorageBin5.CurrentVolumeCapacityAvailable	StorageBin5.CurrentStock.Volume
StorageBin10	10	StorageBin10.CurrentVolumeCapacityAvailable	StorageBin10.CurrentStock.Volume
SafetyStock	1000	SafetyStock.CurrentVolumeCapacityAvailable	SafetyStock.CurrentStock.Volume
Overflow	1000	Overflow.CurrentVolumeCapacityAvailable	Overflow.CurrentStock.Volume

- Update the AreaAssignments to include the following information:

ParentStorageArea	StorageLocation	ChildStorageBin	ChildStorageArea
Area1	Bin	StorageBin1	
Area1	Bin	StorageBin5	
Area2	Bin	StorageBin10	
Area3	Area		Area1
Area3	Area		Area2

Area4	Bin	SafetyStock
Area4	Bin	Overflow

- Update the InitialBinContents to include the following information:

StorageBin	Material	Quantity
StorageBin1	Material1	1
StorageBin5	Material2	1
StorageBin10	Material3	2
SafetyStock	Material1	5
SafetyStock	Material2	5
SafetyStock	Material3	5
Overflow	Material3	1

- Create a new Data Table and name it "PutawayOrders"
- Add the following columns, name them accordingly, and change the following properties:
- String Property – "ArrivalID"
- Make this column a Key by using the Set Column As Key button.
- Real Property – "ArrivalTime"
- *Unit Type* = 'Time'
- *Default Units* = 'Minutes'
- Populate the table with the following information:

ArrivalID	ArrivalTime
PutawayOrder1	0
PutawayOrder2	2

- Create a new Data Table and name it "PutawayStockRequirements"
- Add the following columns, name them accordingly, and change the following properties:
- Foreign Key Property – "ArrivalID"
- *Table Key* = 'PutawayOrders.ArrivalID'
- Integer Property – "RequirementID"
- Foreign Key Property – "Material"
- *Table Key* = 'Materials.Material'
- Real Property – "Quantity"
- Populate the table with the following information:

ArrivalID	RequirementID	Material	Quantity
-----------	---------------	----------	----------

PutawayOrder1	1	HMaterial1	1
PutawayOrder1	2	Material3	3
PutawayOrder2	5	Material3	3
PutawayOrder2	6	Material2	1

- Create a new Data Table and name it "RemovalOrders"
- Add the following columns, name them accordingly, and change the following properties:
- String Property – "ArrivalID"
- Make this column a Key by using the Set Column As Key button.
- Real Property – "ArrivalTime"
- *Unit Type* = 'Time'
- *Default Units* = 'Minutes'
- Populate the table with the following information:

ArrivalID	ArrivalTime
Order1	Order2
1	3

- Create a new Data Table and name it "RemovalStockRequirements"
- Add the following columns, name them accordingly, and change the following properties:
- Foreign Key Property – "ArrivalID"
- *Table Key* = 'RemovalOrders.ArrivalID'
- Integer Property – "RequirementID"
- Foreign Key Property – "Material"
- *Table Key* = 'Materials.Material'
- Real Property – "Quantity"
- Populate the table with the following information:

ArrivalID	RequirementID	Material	Quantity
Order1	3	Material1	1
Order1	4	Material3	4
Order2	7	Material2	1

Facility Window: Objects

- Add two Sources to the Facility, one on top of the other. Rename the top Source "PutawayOrderSource" and the bottom Source "RemovalOrderSource".
- Add two Sinks to the Facility, approximately 12 meters to the right of the Sources. Place the Sinks on top of each other, like the Sources. Name the top Sink "PutawayOrderSink" and the bottom "RemovalOrderSink".
- Add a Path that connects the output node of PutawayOrderSource to the input node of PutawayOrderSink.

- Add a Path that connects the output node of RemovalOrderSource to the input node of RemovalOrderSink.

Definitions Window: Tokens View

- Add a new Token named "MyToken1".
- Give MyToken1 a Stock Reservation Reference State named "StockReservationReferenceState1".

Processes Window: Process Logic

- Make the following updates to AddAndRemoveStock process
- Rename to "AddStock"
- Remove the following steps:
 - NoSpaceInBin AddStock
 - InsufficientQty RemoveStock
 - Succeeds AddStock
 - Succeeds RemoveStock
- Create a new Process. Name it "PutawayOrderArrival".
- On this process, under Advanced Options section of the properties, set *Token Class Name* to 'MyToken1'.
- Add the following steps to PutawayOrderArrival:
 - ReserveBins "Area3or4"
 - *Selection Rules*
 - First item - this rule specifies the candidate bin considered in Area3 must have volume capacity available greater or equal to than the total volume amount the stock requirement is attempting to putaway.
 - *Starting Storage Area Name* = 'Area3'
 - Selection Condition = 'Candidate.StorageBin.CurrentVolumeCapacityAvailable >= Stock.Requirement.Quantity * Stock.Requirement.Material.VolumePerUnit'
 - Second item – this rule specifies that it is preferred that the candidate storage bins in Area3 already have this type of material in it.
 - *Starting Storage Area Name* = 'Area3'
 - *Sub Rule Type* = 'Sort'
 - *Sort Value Expression* = 'Candidate.StorageBin.CurrentStock.QuantityInStock(Stock.Requirement.Material) > 0'
 - *Sort Order* = 'Descending'
 - Third item – this rule specifies that in Area3 the first eligible Quant of the bin should be selected.
 - *Starting Storage Area Name* = 'Area3'
 - *Rule Type* = 'Quant'
 - Fourth item – this rule specifies that in Area4, the candidate storage bin should be the Overflow bin.
 - *Starting Storage Area Name* = 'Area4'
 - Selection Condition = 'Candidate.StorageBin == Overflow'
 - Fifth item – this rule specifies that in Area4 the first eligible Quant of the bin should be selected.
 - *Starting Storage Area Name* = 'Area4'

- *Rule Type* = 'Quant'
- *Save Stock Reservation Reference* = 'MyToken1.StockReservationReferenceState1'
- On the Reserved branch of Area3or4 place AddStock "AddStock1"
- *Stock Additions*
- First item
- *Stock Addition Type* = 'Reservation'
- *Stock Reservation Reference* = 'MyToken1.StockReservationReferenceState1'
- Create a new Process. Name it "RemovalOrderArrival".
- On this process, under Advanced Options section of the properties, set *Token Class Name* to 'MyToken1'.
- Add the following steps to RemovalOrderArrival:
- ReserveStock "Area3"
- *Selection Rules*
- First item – this rule specifies that in Area3 the first eligible Quant of the bin should be selected.
- *Starting Storage Area Name* = 'Area3'
- *Save Stock Reservation Reference* = 'MyToken1.StockReservationReferenceState1'
- On the Original branch of Area3 ReserveStock, add Decide step "UnreservedQuantity".
- *Condition or Probability* = 'Entity.StockRequirements.UnreservedQuantity > 0'
- On the True branch of UnreservedQuantity Decide, add ReserveStock "Area4".
- *Selection Rules*
- First item – this rule specifies that in Area4 the first eligible Quant of the bin should be selected.
- *Starting Storage Area Name* = 'Area4'
- *Save Stock Reservation Reference* = 'MyToken1.StockReservationReferenceState1'
- On the Reserved branch of Area4 ReserveStock step, add RemoveStock step "RemoveStock1".
- *Stock Removals*
- First item
- *Stock Removal Type* = 'Reservation'
- *Stock Reservation Reference* = 'MyToken1.StockReservationReferenceState1'
- o Take the Area3 ReserveStock step's Reserved branch end and connect it to RemoveStock1 RemoveStock step.

Facility Window: Update Object Properties

- On PutawayOrderSource change the following properties:
- Entity Arrival Logic
- *Arrival Mode* = 'Arrival Table'
- *Arrival Time Property* = 'PutawayOrders.ArrivalTime'
- Add-On Process Triggers
- *Created Entity* = 'PutawayOrderArrival'

- Advanced Options
- *Stock Requirements* = 'PutawayStockRequirements'
- Right-click this property to set the reference to the Data Table.
- Open the Stock Requirements repeat group and set the following properties:
- *ID Number* = 'PutawayStockRequirements.RequirementID'
- *Material Name* = 'PutawayStockRequirements.Material'
- *Quantity* = 'PutawayStockRequirements.Quantity'
- On RemovalOrderSource change the following properties:
- Entity Arrival Logic
- *Arrival Mode* = 'Arrival Table'
- *Arrival Time Property* = 'RemovalOrders.ArrivalTime'
- Add-On Process Triggers
- *Created Entity* = 'RemovalOrderArrival'
- Advanced Options
- *Stock Requirements* = 'RemovalStockRequirements'
- Right-click this property to set the reference to the Data Table.
- Open the *Stock Requirements* repeat group and set the following properties:
- *ID Number* = 'RemovalStockRequirements.RequirementID'
- *Material Name* = 'RemovalStockRequirements.Material'
- *Quantity* = 'RemovalStockRequirements.Quantity'

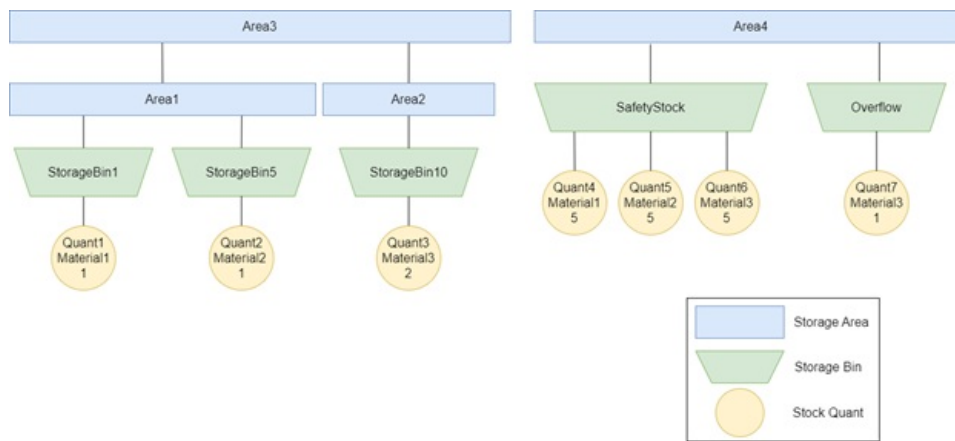
Facility Window: Animation

- From the Animation ribbon, add a Status Table to the Facility. Change its *Table Name* to 'StorageBins' to see information about the Storage Bins.
- Add another Stacked Bar. This Stacked Bar will be for Area4. The Stacked Bar's *Type* should be changed to 'StorageArea' and the *Storage Area* should be specified as 'Area4'.

Discussion:

The Stock Area and Bin hierarchy and current Stock Quants after model initialization is depicted below. The number in the quants are the current QuantityInStock value.

Image 1: Stock after initial contents is added.



When an entity enters the system, it receives requirements specified in the *Stock Requirements* repeat group on the Source. An entity could have one or many requirements. When a ReserveBins or ReserveStock step is executed by an entity with Stock Requirements, each of the entities' requirements will be checked one by one to create reservations.

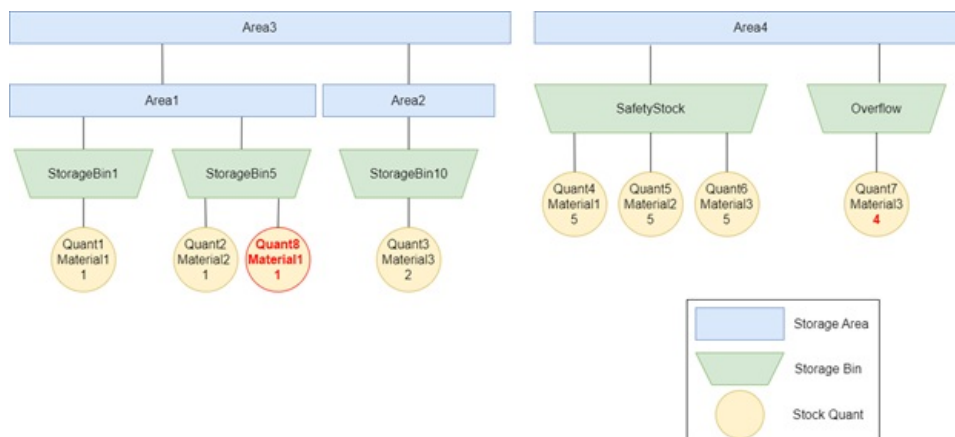
PutawayOrder1, has two requirements. At the ReserveBins step, the first requirement is considered using the defined Selection Rules. When Selection Rules are evaluated, they are evaluated in the order listed. Rules with the same *Starting Storage Area Name* value are considered together and so multiple rules can be used in parallel to achieve the desired behavior.

This first requirement is assessed with the first rule and looks for bins that have available volume capacity that is greater or equal to the volume capacity of the material in the requirement. This excludes StorageBin1 as it does not have any available capacity. Out of the remaining candidates in Area3, the next rule will then sort the bins so that the priority is candidates that already have the same type of material in stock. Of the remaining candidates, neither has any of the same material as this first requirement, so the first available bin in the list is selected: StorageBin5.

The second requirement of the same PutawayOrder1 entity is now assessed at the ReserveBins step. When assessing the first rule for Area3, it is found that none of the candidate locations in this Area would meet the condition of having enough space for the entire quantity of the material. So, the Selections Rules move on to the next Storage Area, Area4. In the fourth rule, it is specified that the Candidate Bin in Area4 should be the Overflow Bin, so the Overflow Bin is selected for this requirement.

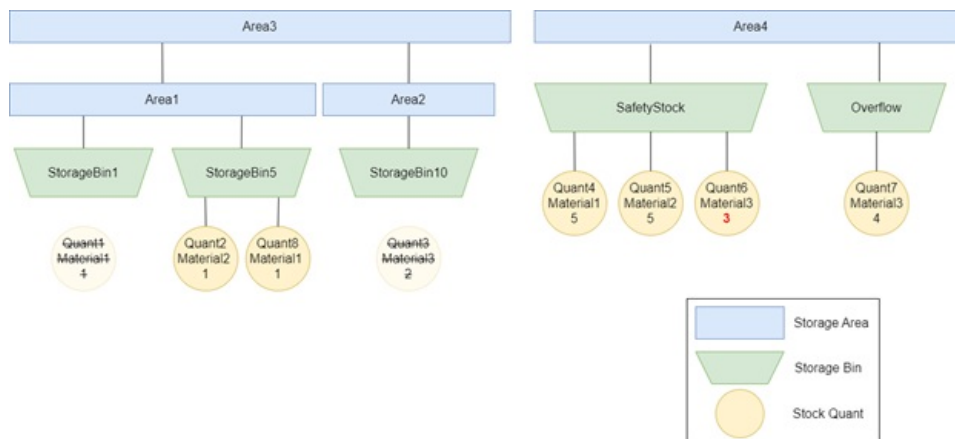
After each Reservation is made, a new token will exit on the Reserved branch of the ReserveBins step. This Token has the Reservation saved to its State Variable from the *Save Stock Reservation Reference* property on the ReserveBins step. This Reservation Reference will be used in the AddStock step.

Image 2: Stock after PutawayOrder1 has been completed.



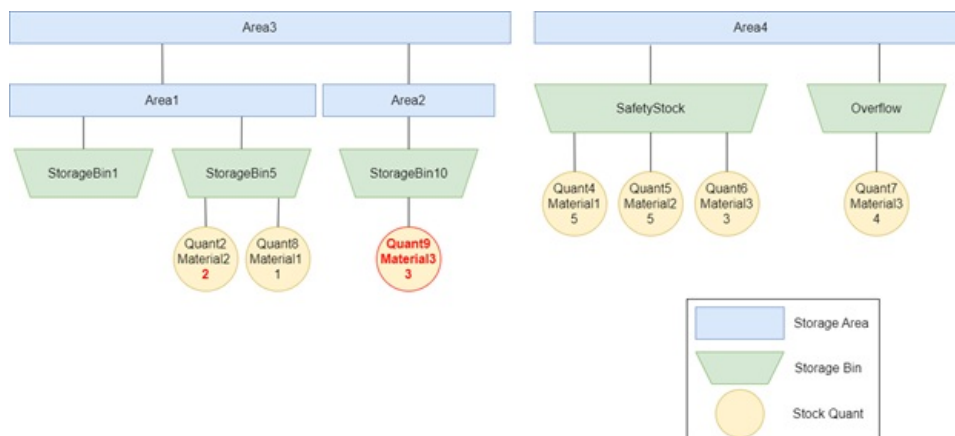
For the removal Order1, it follows a similar path. The entity with the requirements executes a ReserveStock step. The Area3 ReserveStock step's rule looks for a quant in Area3. For the first requirement, it can find the material and quantity in Area3. For the second requirement, it is able to find part of the requirement's quantity so it reserves only that partial quantity. There is no rule in this Selection Rules group that has a condition where all the required material must be found in one location, like the first rule in the PutawayOrderArrival's Area3or4 ReserveBins step. Reservations are able to reserve a portion of requirement by default. There is still outstanding quantity for the second requirement, so the Area4 ReserveStock step is executed which is able to fill the remaining quantity. This is an example of when one requirement might produce multiple reservations. Requirement can create anywhere from 0 to N requirements.

Image 3: Stock after Order1 has been completed.



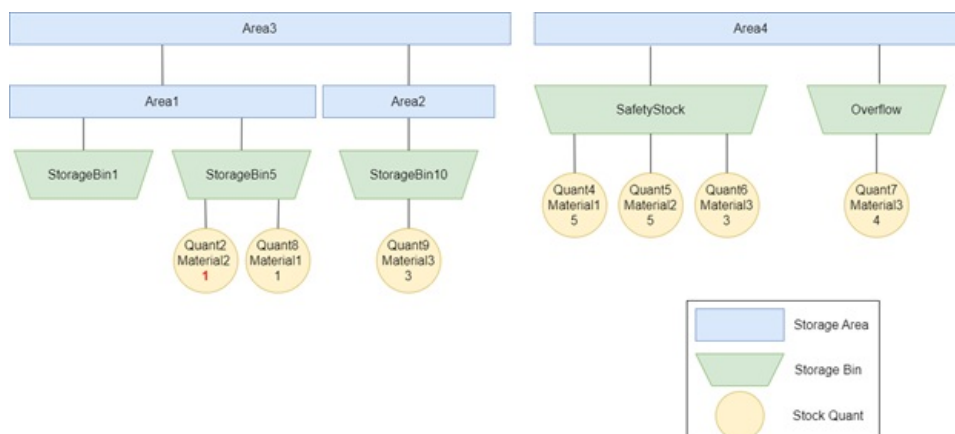
For PutawayOrder2, check how the ReserveBins step is evaluated in Trace compared to previously. Now that there is space in StorageBin10 for Material3, it is selected for the first requirement. For the second requirement, 1 unit of Material2 is added to StorageBin5. There is space in StorageBin5 and it is preferred since StorageBin5 already contains other quantities of Material2.

Image 4: Stock after PutawayOrder2 is completed.



For the last removal Order2, 1 unit of Material is removed from StorageBin5.

Image 5: Stock after Order2 is completed.



Consider how changing the Selection Rules would affect the system. Experiment with different rules to see their impact. Notice how requirements might only be able to be partially filled. In the case of RemovalOrderArrival process, the ReserveStock steps were separated to demonstrate how you can check if a Requirement Quantity has been fulfilled and if not, how further logic can be enacted to handle that. The Area4 Selection Rule could have been added to the same Selection Rule list with Area3 ReserveStock step. Check how Trace changes by moving the Area4 rule as a second item in Area3 ReserveStock.

Notes:

Notice that in the Selection Rules repeat group the *Rule Type* default is different for ReserveBins versus ReserveStock. ReserveBins *Rule Type* default is 'Bins'. ReserveStock *Rule Type* default is 'Quant'.

The ReserveStock step requires a Quant rule to work. If no Quant rule exists, no stock will be selected. The ReserveBins step can use a Quant rule if needed, but if no Quant rule is selected, as long as a Bin is selected, a new quant will automatically be created for the stock being added. It is more important to ensure a Bin rule is created for the ReserveBins step.

In these examples, the Token stores important pieces of information that are used later in the same process logic. Reservations and Quant references can be saved to other locations as well. Note, a Reservation is destroyed when it is fulfilled so any references will also be cleared. Similarly, if the entity with the requirements that created the reservations is destroyed, the reservations will also be destroyed. With a quant, when its QuantityInStock and IncomingQuantity are zero, the quant is automatically destroyed. However, anywhere a quant's reference is saved to a State will persist after it is destroyed.

Model 3: QuantsTables

Note: This model uses an Output Table with a Key column. This feature is currently limited to RPS edition only. If you are interested in learning more about different editions, please contact sales@simio.com.

Problem:

Orders arrive with a material and quantity requirement. These orders should reserve the bin space or stock and then putaway or remove stock based on these reservations. When putaway into a bin, attributes of the material group should be tracked, specifically the creation time of this stock. Reservations should be made prioritizing certain stock and removing the oldest quant groups first. Stock added into the bins should be grouped into quants by the time they were added.

Categories:

Element, Material Storage

Keywords:

Element, Material, Putaway, Quant, Retrieval, Storage Bin, Storage Area

Assumptions:

Bins have initial stock. Putaway and retrieval orders are provided as an input to the system. The arrival time, material, and quantity of an order is known. When material is added to a bin, its time is recorded.

Technical Approach:

This model is generated with data and uses a data driven approach. The Storage Area, Storage Bin, and Material Elements are created in Data Tables. Data Tables define each Area's child Areas or child bins and the Initial contents of Bins. Initial Stock will be added with Process logic triggered when the run is initialized.

Entities represent the orders in the system. These entities are created with the requirements from the order, such as the material and quantity. As a putaway or removal order entity enters the system, process logic will trigger to reserve either the space in bins or quantity of stock, depending on the order's requirements. After reservations are made, the reservations will be used to add or remove the corresponding stock.

When stock is added into a bin, information about the quant is recorded in an Output Table. This quant information is considered when assessing selection rules for reservations.

Details for Building the Model:

Initial Setup

- Use the "HandlingStockRequirements" model as a starting point. You may right-click the model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Data Window: Tables

- Use the Add Table drop down to add an Output Table. Name this table "StockQuants".
- Add the following columns to StockQuants:
 - Integer State Variable – "QuantID"
 - Set this column as a Key by using the Set Column As Key button.
 - Foreign Key State – "StorageBin"

- *Table Key* = 'StorageBins.StorageBin'
- *Date Time State Variable* – "CreationTime"

Definitions Window: Tokens

- Select the MyToken1 token. Add a Stock Quant Reference State Variable called "StockQuantReferenceState1".

Processes Window: Process Logic

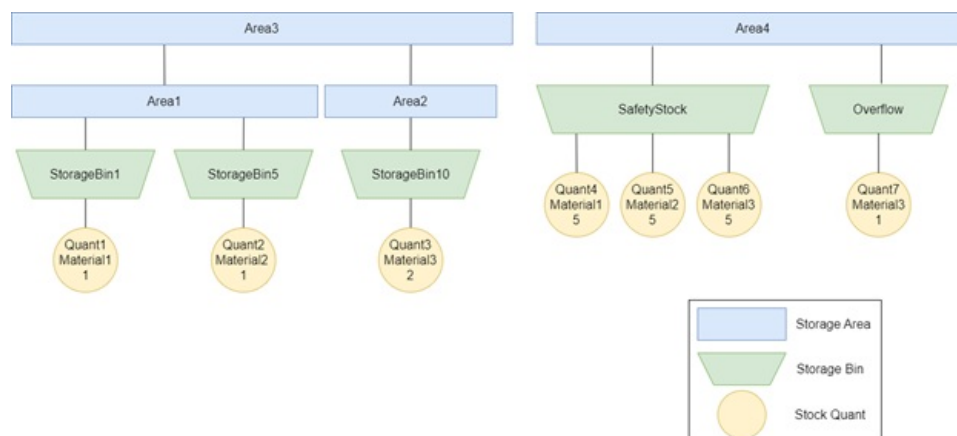
- Make the following changes to the AddStock process:
- Change the *Process Token Class Name* to 'MyToken1'
- On the InitialStock AddStock step, change *Save Stock Quant Reference* to 'MyToken1.StockQuantReferenceState1'
- On the Added branch of InitialStock AddStock step, add AddRow step "StockQuants"
- *Table Name* = 'StockQuants'
- *Key Column Value* = 'MyToken1.StockQuantReferenceState1.IDNumber'
- *Object Type* = 'Token'
- After the AddRow step, add Assign step "StockQuants"
- *State Variable Name* = 'StockQuants.CreationTime'
- *New Value* = 'TimeNow'
- Make the following changes to the PutawayOrderArrival process:
- Area3or4 ReserveBins step
- Selection Rules – third item
- *Selection Condition* =
'StockQuants[StockQuants.QuantID.RowForKey(Candidate.StockQuant.IDNumber)].CreationTime == TimeNow'
- Selection Rules – fifth item
- *Selection Condition* =
'StockQuants[StockQuants.QuantID.RowForKey(Candidate.StockQuant.IDNumber)].CreationTime == TimeNow'
- On the AddStock1 AddStock step, change *Save Stock Quant Reference* to 'MyToken1.StockQuantReferenceState1'
- On the Added branch of the AddStock1 AddStock step, add Decide step "NoRowExists".
- *Condition Or Probability* = '!StockQuants.QuantID.RowForKey(MyToken1.StockQuantReferenceState1.IDNumber) > 0'
- On the True branch of the NoRowExists Decide, place a SetRow step "Storage"
- *Table Name* = 'StorageBins'
- *Row Number* = 'StorageBins.StorageBin.RowForKey(MyToken1.StockQuantReferenceState1.StorageBin)'
- *Object Type* = 'Token'
- After the Storage SetRow, place an AddRow step "StockQuants".
- *Table Name* = 'StockQuants'
- *Key Column Value* = 'MyToken1.StockQuantReferenceState1.IDNumber'
- *Object Type* = 'Token'
- After the AddRow step, add Assign step "StockQuants"

- *State Variable Name* = 'StockQuants.CreationTime'
- *New Value* = 'TimeNow'
- Make the following changes to the RemovalOrderArrival process:
- Area3 ReserveStock step
- *Selection Rules* – first item
- *Sub Rule Type* = 'Sort'
- *Sort Value Expression* =
'StockQuants[StockQuants.QuantID.RowForKey(Candidate.StockQuant.IDNumber)].CreationTime'
- RemoveStock1 RemoveStock step
- *Save Stock Quant Reference* = 'MyToken1.StockQuantReferenceState1'
- On the Removed branch of RemoveStock1 RemoveStock step, add a Decide step "StockQuantEmpty"
- *Condition or Probability* = 'MyToken1.StockQuantReferenceState1.QuantityInStock <= 0 && MyToken1.StockQuantReferenceState1.IncomingQuantity <= 0'
- On the True branch of StockQuantEmpty Decide, place SetRow "StockQuants"
- *Table Name* = 'StockQuants'
- *Row Number* = 'StockQuants.QuantID.RowForKey(MyToken1.StockQuantReferenceState1.IDNumber)'
- *Object Type* = 'Token'
- After StockQuants SetRow, place RemoveRows "StockQuants"
- *Table Name* = 'StockQuants'
- *Removal Type* = 'RemoveActiveRow'
- *Object Type* = 'Token'

Discussion:

The Stock Area and Bin hierarchy and current Stock Quants when the model is initialized is depicted below.

Image 6: Stock after initial contents is added.



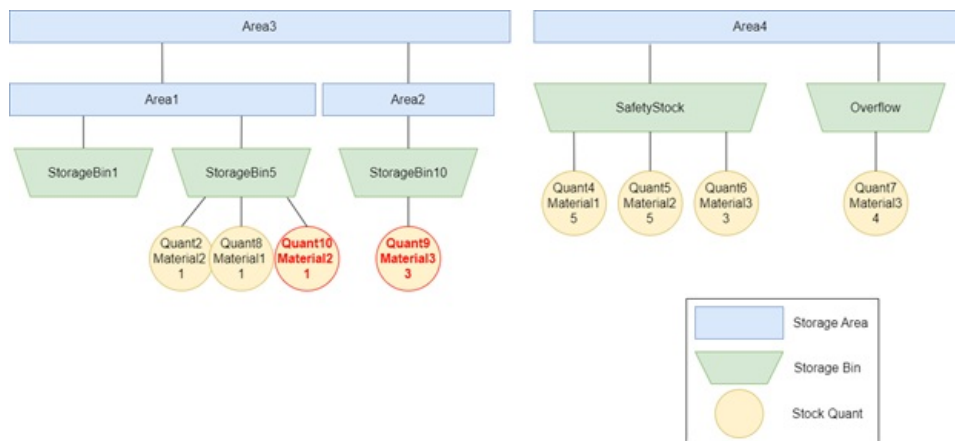
The *Selection Rules* updated in the ReserveBins step changed the Quant rule's *Selection Condition*. A Candidate Quant can only be selected in a bin if the quant's creation time recorded in the StockQuants Table matches the current time. In other words, only stock added at the same creation time can be grouped together in a quant. Else, a new quant will be created.

The *Selection Rules* updated in the ReserveStock step changed the Quant rule to sort the candidate quants by the recorded creation time in the StockQuants Table. This rule's *Sort Value Expression* and *Sort Order* prefer quants that have the smallest creation time, which are the oldest stock quants.

Comparing the previous model HandlingStockRequirements, you will see a couple changes with how the order's stocks were added or removed. For PutawayOrder1, the 3 Units of Material3 are still added to the same Quant7 in the Overflow Bin since both are added at the same simulation time so their creation time values match. For the first PutawayOrder1 and removal Order1, the changes to stock in the system will look identical to Images 2 and 3 in the above section.

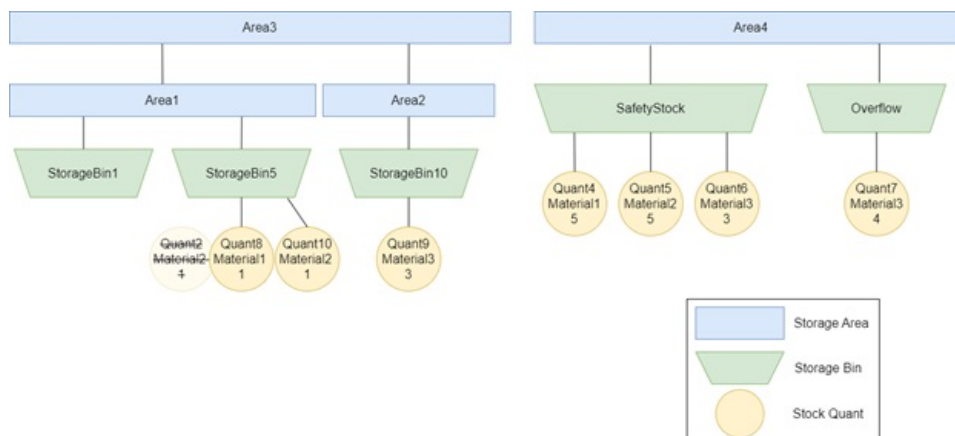
For PutawayOrder2, when Material2 is added to StorageBin5, a new Quant is created since the other Material2 in this Bin has a different Creation Time. Now, there are two different quants of Material2 in StorageBin5.

Image 7: Stock after PutawayOrder2 is completed.



When Order2 goes to remove Material2 from StorageBin5, it is now selecting the oldest quant, Quant2. This removes Quant2 while the recently added Material2 still exists in Quant10.

Image 8: Stock after Order2 is complete.



Using Quant rules when adding stock can create more flexibility in modeling stock that should be grouped together in bins by different attributes. Using Quant rules when removing stock can help choose groups of materials in a bin to remove based on those attributes.

Notes:

When a quant's QuantityInStock and IncomingQuantity are zero, the quant is automatically destroyed. However, anywhere a quant's reference is saved to a State will persist after it is destroyed. This is useful in the Output Table, as it is easy to retain information about the quants even after they have been destroyed.

Consider adding additional columns into the Stock Quant Output Table for your verification of how the model is working. You could choose to record the Material or Quantity in Stock. Note, these are functions you can access through the Quant reference, so they are already available to use in the process logic without saving them off in the Table.

MergingConveyorsControlledByGate - SimBit

This SimBit project (MergingConveyorsControlledByGate) includes three models demonstrating the use of a Resource as a gate for merging conveyors.

1. MergingConveyors_Simple: Use a gate to control entry to the merge point of two conveyors.
2. MergingConveyors_Batching: Use a gate to control entry to the merge point of two conveyors such that Entities from the feeder conveyors enter in batches.
3. MergingConveyors_BatchingWithPriority: Use a gate to control entry to the merge point of two conveyors such that Entities from the feeder conveyors enter in batches, prioritizing the feeder with the most Entities.

Model 1: MergingConveyors Simple

Problem:

A merging conveyor system have a gated merge point that prevents a package from continuing if it is going to collide with packages on the opposing conveyor.

Categories:

Add-On Process Logic, Conveyor Systems

Keywords:

Conveyor, Resource, Add-On Process, Seize Step, Release Step

Technical Approach:

There are two types of Entities that exit from two Sources on separate Conveyors and merge onto single Conveyor. At the merge point, a conveyor gate prevents arriving entities from colliding by stopping them from moving forward on the link. Entities enter the single conveyor on a first in first out basis.

A Resource is created to represent the gate or passage of a single package. An Add-on Process is used at each entry Node to the merging conveyor to seize the gate, thus metering flow to a single entity at a time. At the merge point, another Add-on Process is used to release the gate, allowing the next waiting entity to proceed.

Details for Building the Model:

Simple System Setup

- From the Standard Library place two Source Objects, a Sink Object, and three Basic Nodes in the Facility view. Rename one of the Basic Nodes 'MergePoint'.
- Create Conveyors to connect the following Nodes:
 - Output@Source1 to BasicNode1
 - Output@Source2 to BasicNode2
 - BasicNode1 to 'MergePoint'
 - BasicNode2 to 'MergePoint'
 - 'MergePoint' to Input@Sink1
- Multi-select the previously created conveyors and select the Conveyor image under Path Decorators.
- From the Project Library place two ModelEntity instances. Rename them 'Package1' and 'Package2'.
- Select Package1 and Apply Symbol. Select Box1 image from the dropdown.
- Select Package2 and Apply Symbol. Select Box2 image from the dropdown.

- From the Standard Library place a Resource Object and rename it 'GateControl'.
- Select Source1 and set the *Interarrival Time* property to 'Random.Exponential(0.05)'.
- Select Source2, change *Entity Type* to 'Package2', and set the *Interarrival Time* property to 'Random.Exponential(0.1)'.

Creating Add-On Processes

- Select BasicNode1 and expand the *Add-on Process Triggers* properties. Double-click on *Entered*, which will navigate to the Processes Tab and create a process.
- Rename the process 'SeizeGate' and rename the category 'Gate Processes'.
- Place a Seize Step and open the *Resource Seizes – Repeating Property Editor*. Click Add and change the Object Name to 'GateControl'.
- Select Create Process to create another process.
- Rename the new process 'ReleaseGate' and change the category to 'Gate Processes'.
- Place a Release Step and open the *Resource Releases – Repeating Property Editor*. Click Add and change the Object Name to 'GateControl'.
- Navigate to the Facility Tab.
- Select 'BasicNode2' and expand the *Add-on Process Triggers* properties. Set *Entered* to 'SeizeGate'.
- Select 'MergePoint' and expand the *Add-on Process Triggers* properties. Set *Entered* to 'ReleaseGate'.

Model 2: MergingConveyors Batching

Problem:

I want to enhance the model merging conveyors that prevent package collisions by improving the gate to release batches of 5 entities from each feeding conveyor at a time.

Categories:

Add-On Process Logic, Conveyor Systems

Keywords:

Conveyor, Resource, Add-On Process, Seize Step, Release Step, Decide Step, Assign Step, Integer State, Node Reference State, Owner Type, Owner Object

Technical Approach:

The Sources create 5 Entities at a time and the first Entity to arrive at the entry Node triggers an Add-on Process that allows the Node to seize the gate Resource. The Add-on Processes allow the following 4 Entities arriving at the same Node to pass before releasing the gate. FIFO rule is used to determine the next batch to seize the gate allow the next 5 Entities to pass. A Node Reference State Variable is created on the Model Entity to store the Owner Object of the Seize to be used later in the Release.

Details for Building the Model:

Changes to Simple System Setup

- Multi-select the Sources and change the *Entities Per Arrival* to '5'.
- Navigate to the Definitions Tab and select States.
- Create an Integer State and rename it 'Counter'.
- Using the Navigate pane select ModelEntity, navigate to the Definitions Tab, and select States.
- Add a Node Reference State Variable and rename it 'SeizedOwner'.

Add-on Processes

- Navigate to the Processes Tab and select the 'SeizeGate' process.

- Place a Decide Step before the Seize Step and rename it 'Already Seized?'.
 - Set *Condition Or Probability* to 'ModelEntity.CurrentNode.Node.SeizedResources.LastItem.Is.GateControl'
 - Move the existing Seize Step down the false path of 'Already Seized?'.
 - Under the *Advanced Options* of the Seize Step set *Owner Type* to 'SpecificObject' and *Owner Object* to 'ModelEntity.CurrentNode'
 - Place an Assign Step down the true path of 'Already Seized?' and rename it 'Remember Node'.
 - Set *State Variable Name* to 'ModelEntity.SeizedOwner' and *New Value* to 'ModelEntity.CurrentNode'.
 - Select the 'ReleaseGate' process, place an Assign Step before the existing Release Step, and rename it 'Count'.
 - Set *State Variable Name* to 'Counter' and *New Value* to 'Counter + 1'.
 - Place a Decide Step after 'Count' and before the Release Step. Rename it 'Entire Batch Passed?'.
 - Set *Condition Or Probability* to 'Counter == 5'
 - Place another Assign Step down the True path of 'Entire Batch Passed?' before the existing Release Step and rename it 'Reset Count'.
 - Set *State Variable Name* to 'Counter' and *New Value* to '0'.
 - Under the *Advanced Options* of the Release Step set *Owner Type* to 'SpecificObject' and *Owner Object* to 'ModelEntity.SeizedOwner'.

Model 3: MergingConveyors BatchingWithPriority

Problem:

I want to enhance the model merging conveyors that prevents package collisions and has a gate that release batches of 5 entities from each feeding conveyor at a time by enhancing the gate with a priority rule to release from the feeding conveyor with the most packages.

Categories:

Add-On Process Logic, Conveyor Systems, Data Tables

Keywords:

Conveyor, Resource, Add-On Process, Seize Step, Release Step, Decide Step, Assign Step, Integer State, Model Entity, Node Reference State, Owner Type, Owner Object, Data Tables, Object Reference Property Column, Node Reference Property Column, State Property, Auto-Set Table Row Reference, Functions

Technical Approach:

The Integer State Variables that represent the priority of each feeding conveyor are updated with Add-on Processes on the Conveyors. The entry nodes use the Integer State Variables for a selection condition within the Seize Step. A Data Table is used to reference the priority values to feeding conveyers and the entry nodes.

Details for Building the Model:

Changes to Simple System Setup

- Multi-select Source1 and Package1 then move them to the left making Conveyor1 longer than Conveyor2.
- Navigate to the Definition Tab and select States.
- Create two Integer State Variables and rename them 'Conveyor1Priority' and 'Conveyor2Priority'.
- Select Functions and create a new Function. Rename it 'Top Priority'.
- Set the *Expression* to 'Math.Max(Conveyor1Priority, Conveyor2Priority)'
- Navigate to the Facility Tab and select Animation under the Facility Tools Ribbon.
- Select Status Label and place it near BasicNode1. Set the *Expression* to 'Conveyor1Priority'.

-
- Select Status Label again and place it near BasicNode2. Set the *Expression* to 'Conveyor2Priority'.

Data Table Setup

- Navigate to the Data Tab and select Add Data Table. Rename it 'ConveyorPriority'.
- Add an Object Reference Property column and rename it 'FeedingConveyor'.
- Add a Node Reference Property column and rename it 'NodeOwner'.
- Add a State Property column and rename it 'PriorityState'.
- Under 'FeedingConveyor' add 'Conveyor1' and 'Conveyor2' in first and second row respectively.
- Under 'Node Owner' add 'BasicNode1' and 'BasicNode2' in first and second row respectively.
- Under 'PriorityState' add 'Conveyor1Priority' and 'Conveyor2Priority' in first and second row respectively.
- Select 'FeedingConveyor' and under *Advanced Options* change *Auto-set Table Row Reference* to 'True'.
- Select 'Node Owner' and under *Advanced Options* change *Auto-set Table Row Reference* to 'True'.

Add-On Processes

- Navigate to the Processes Tab and select the 'SeizeGate' process.
- Select the existing Seize Gate step and open the *Resource Seizes – Repeating Property Editor*.
- Under the *Advanced Options* set the *Selection Condition* to 'ConveyorPriority.PriorityState == TopPriority'
- Select Create Process. Rename the new process 'IncreasePriority' and rename the category 'Conveyor Processes'.
- Place an Assign Step and rename it 'AssignPriority'.
- Set *State Variable Name* to 'ConveyorPriority.PriorityState' and *New Value* to 'ConveyorPriority.PriorityState + 1'.
- Select Create Process again. Rename the new process 'DecreasePriority' and change the category to 'Conveyor Processes'.
- Place an Assign Step and rename it 'AssignPriority'.
- Set *State Variable Name* to 'ConveyorPriority.PriorityState' and *New Value* to 'ConveyorPriority.PriorityState - 1'.
- Navigate to the Facility Tab, and multi-select Conveyor1 and Conveyor2.
- Expand the *Add-on Process Triggers* properties and set *Entered* to 'IncreasePriority' and *Exited* to 'DecreasePriority'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

MoveableOperator - SimBit

Problem:

I want to model a system that has two Servers in series and requires a Worker to be present at each Server before processing can occur. Animation should show the Worker moving between the two servers and parking at the appropriate Server for processing.

Categories:

Building New Objects / Hierarchy

Key Concepts:

Add-On Process, Assign Step, Before Exiting, Contents, Decide Step, InputBuffer, On Evaluating Seize Request, On Released, ParkingStation Queue, RideStation Queue, Secondary Resources, Server, State Assignments, Worker

Assumptions:

There is only one Worker in the system. It will work at the first Server until the second Server has 5 or more entities waiting in its queue. The Worker will then work on the second Server until the input buffer is empty and then return to working on the first Server.

Technical Approach:

The Worker is modeled with a standard Worker object. The Worker object travels between two nodes that are located near each Server, which animates the moveable resource. The Worker object is seized by each Server before processing and the model ensures that the object is located at the appropriate node before it begins Processing. Add On Processes on the Worker object will check the contents of the Input Buffer queue to see if the Worker should be working at Server1 or Server2. A state variable at the Model will indicate which Server the Worker should be working on.

Details for Building the Model:

Simple System Setup

- Add a Source, two Servers and a Sink to the Facility Window. Add a ModelEntity object to the Facility Window and change its picture to a box.
- Connect the Source to the first Server, connect the first and second Server together and connect the second Server to the Sink with Paths. Add two Basic Nodes that are connected to each other with a bi-directional path. Place the nodes near each of the Servers. This is the path that the movable resource will take between the Servers.
- Place a Worker object in the Facility window and change its picture to a person by clicking on the Worker object and selecting a new symbol from the Project symbol library in the ribbon.
 - Set the *Park While Busy* property of the Worker to 'True', which will tell the Worker to park at the node while it's processing instead of stay on the link.
 - Set the *Initial Node(Home)* property to 'BasicNode1'.

Adding Logic to the Servers

- Click on Server1 and expand the *Secondary Resources* property category. Under the *Resource for Processing* category, set the *Object Type* property to 'Specific', set the *Object Name* property to 'Worker1'. Set the *Request Move* property to 'To Node' and the *Destination Node* property to 'BasicNode1'. This tells the Server that it must seize Worker1 and it must arrive at BasicNode1 before processing can begin at this Server.
- Repeat the above step for Server2. It must also seize Worker1 and before processing, but set the *Destination Node* property to 'BasicNode2' instead of 'BasicNode1' so that Worker1 arrives at the node closest to Server2.
- Click back on Server1 and expand the *State Assignments* property category. Open the Repeating Property editor for the *Before Exiting* property by clicking on the ellipse that appears in the text box of this property. Click Add and set the *State Variable Name* to 'ModelEntity.Priority' and the *New Value* to '2'. This changes the priority of all entities that leave this Server from the default value of 1 to the new value of 2.

Create New State

- In the Definitions window of the model, go to the States panel.
 - Create a new Real Discrete state by clicking on Real in the ribbon. Name this new state 'WorkerTaskPriority'.

This will keep track of which server the Worker should be working at. Set the *Initial State Value* in the properties window to '1'.

Add On Process Logic for Worker

- Click on the Worker object and expand the Add On Process Triggers property category.
 - Create a new process that is called from the *Released* property by selecting 'Create New' from the drop down of this input box. This new process will check the current Task Priority when Worker is Released from a job.
 - Go to the processes window and in the process called Worker_Released, place a Decide Step which checks the input buffer of Server 2 to see if there are 5 or more entities waiting. Set the *Decide Type* to 'ConditionBased' and the *Expression* to `Server2.InputBuffer.Contents >= 5`. If True, the state WorkerTaskPriority is set to a value of '2' with an Assign Step. If False, another Decide Step checks to see if the input buffer is empty and if so, it assigns the value of '0' to the state WorkerTaskPriority with an Assign Step. This tells the system that the Worker will move to Server2 and work there until the input buffer is empty.
- Back in the Facility window, click on the Worker.
 - Create a new process that is called from the *Evaluating Seize Request* property by selecting 'Create New' from the drop down of this input box. This new process will be executed each time a Server is attempting Allocation of the Worker Object.
 - Go to the processes window and in the process called Worker1_EvaluatingSeizeRequest, place a Decide step that checks to see if the entity that is evaluating its allocation should get the capacity of the Worker. Set the *Decide Type* to 'ConditionBased' and the *Expression* to `Entity.Priority == WorkerTaskPriority`. If True, then the entity will get capacity of the Worker since the Worker object is supposed to stay at that server, based on the value of WorkerTaskPriority. If the priorities do not match, Token.ReturnValue is set to 'False', which means that the allocation attempt of the Worker object was rejected.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

MoveASeizedObject - SimBit

Problem:

I have seized a moveable resource and now I'd like it to move to a couple of locations within the model before I release the capacity of the resource.

Categories:

Worker

Key Concepts:

Decide Step, Delay Step, ID, Is.Worker, ModelEntity, Move Step, Location.Parent, Path, Secondary Resources, Selection Weight, Server, Worker

Technical Approach:

There are two Server objects that both need to seize a Worker object and have the Worker move to the Server before processing can begin. After processing finishes at each Server, the After Processing Add On Process trigger executes a process. This process tells the Worker object to move to a node, delay, then move to another node and delay again. The entire time, the Entity within the Server "owns" the capacity of the Worker. Finally, the Worker is told to move back to the Server where the Entity is still waiting in the Processing station. Once the Worker reaches the Server, the entity releases capacity of both the Server and the Worker and it moves out of the Server.

Details for Building the Model:

Simple System Setup

- Place a Source object, followed by two Server objects that are in parallel, within the Facility window. Rename the Server objects to Room1 and Room2. Connect the Source to the Servers with paths.
- Click on the Source object and change the *Interarrival Time* property to 'Random.Exponential(9)'.
- Place two Sink objects, one near each Server, and connect Room1 to Sink1 with a path and connect Room2 to Sink2 with a path.
- Place two Transfer Nodes into the Facility window, somewhere between Room1 and Room2. Name one TransferNode 'Lab' and the other 'Desk'.
- Place Paths going from the output nodes of the servers to Lab, from Lab to Desk, and from Desk back to the server output nodes.
- Place a Worker object into the Facility window. Set its *Initial Node (Home)* property to 'Desk' and set its *Idle Action* property to 'Go to Home'.
- In the Paths connecting the Output Nodes to Lab, set the *Selection Weight* to 'Is.Worker' to ensure that no Entities travel on these paths (they are for workers only). Entities will go directly into the Sinks.

Logic for Seizing and Moving Worker

- Select Room1 and set its *Processing Time* property to '2' minutes.
- Expand the Secondary Resources category within Room1's properties and under the Resource for Processing category, set the *Object Name* property to 'Worker1'. In this same property category, set the *Request Move* property to 'To Node' and set the *Destination Node* property to 'Output@Room1'. This is where we tell Room1 to seize Worker1 and have it move to Room1, before processing can begin at the server.
- Repeat the above steps for the Room2, but instead of setting *Destination Node* to 'Output@Room1', enter 'Output@Room2'.
- Within the Processes window, create a new process by clicking on 'Create Process' in the ribbon. Name this process 'Server_Processed'.
 - Place a Move Step in this process. Open the *Resource Move Requests* repeat group property window by clicking on the ... (ellipse button). Within the editor window that appears, click the Add button and set *Object Name* to 'Worker1' and *Destination Node* to 'Lab'.
 - Place a Delay Step and set the *Delay Time* to '2' and *Units* to 'Minutes'. This is how long the Worker will wait at the Lab node before moving on.
 - Place another Move Step and within the Resource Move Request – Repeating Property editor window, set *Object Name* to 'Worker1' and *Destination Node* to 'Desk'.

- Place a Delay Step and set the *Delay Time* to '2' and *Units* to 'Minutes'. This is how long the Worker will wait at the Desk node before moving on.
- Place a Decide Step, which will check to see which Room the Worker should return to. Set the *Decide Type* to 'Condition' and the *Expression* to 'ModelEntity.Location.Parent == Room1'. This is checking to see if the Entity that "owns" this Worker right now is currently located in Room1.
- In the True segment leaving the Decide Step, place a Move Step that moves 'Worker1' to 'Output@Room1'. In the False segment leaving the Decide Step, place a Move Step that moves 'Worker1' to 'Output@Room2'.
- Within the Facility window, select Room1 and expand the Add On Process Triggers property category. In the *After Processing* trigger, select the new process you just created. Do the same thing for Room2. The same process can be called from each Server.

Embellishments:

Add additional Move Steps that require the Worker to visit other locations while it is seized by the entity. Or increase the *Interarrival Time* on the Source object and then increase the number dynamic Worker objects there are in the system to '2'.

Discussion:

Because the entity "owns" the Worker object while it is moving through the system, no other object can get capacity of this Worker object. It is "Busy" until the Entity releases the capacity of the Worker. In this example, the Entity releases capacity of the Worker once the Entity leaves the Server.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

MultipleInputArgumentsOnProcesses - SimBit

Problem:

I have a Worker that needs to service 2 different Servers upon a server failure. Rather than having separate add-on processes for each server, I'd like a single process that will seize the Worker when either Server fails.

Categories:

Add-On Process Logic, Worker

Key Concepts:

Add-On Process, Failure, On Failed, On Repaired, Process Input Arguments, Seize Step, Set Referenced Property, Release Step, Server, Worker, Time to Repair, Token, Uptime Between Failures

Assumptions:

A single Worker is used to repair 2 different Servers when they fail. A bi-directional path is used to transfer the Worker between the Servers. The same failure type property values are used with the Servers.

Additional Notes:

Input Arguments on processes allow the user to have single shared process between objects. In this SimBit, the seized worker is directed to the appropriate work location, no matter where the process is called from. While not illustrated here, it's also possible to return a value (or values) in a similar fashion using return values.

Technical Approach:

Two Source-Server-Sink sets of objects connected by paths are placed in the model. A Worker is transferred between the Servers via a bi-directional Path between the Servers. Reliability logic (Uptime Between Failures and Time To Repair) is defined via referenced properties. Add-On processes, along with process input arguments, are used to seize the worker when needed.

Details for Building the Model:

System Setup

- In the Facility window, place a Source, a Server and a Sink object and connect them with Path objects. Repeat this process with a second set of objects below in parallel to the first set. On both Source objects, change *Interarrival Time* to 'Random.Exponential(.5)'.
- Place a BasicNode (BasicNode1) next to Server1, a BasicNode (BasicNode2) next to Server2 and a third BasicNode (HomeNode) between the Source objects. Connect these nodes with a bi-directional path. Place a Worker object in the Facility window and set the following property values: *Initial Node (Home)* to 'HomeNode', *Idle Action* to 'Park At Home'. Click on the HomeNode and in the Appearance ribbon, select Draw Queue and add a ParkingStation.Contents queue. On that same ribbon, turn off the Parking Queue option.
- In the Server1 object, under Reliability Logic, set the *Failure Type* to 'Calendar Time Based' and create a new referenced property for *Uptime Between Failures* called 'UptimeBetweenFailures' and one for *Time To Repair* called 'TimeToRepair'. In Server2, under Reliability Logic, set the *Failure Type* to 'Calendar Time Based' and set a referenced property on *Uptime Between Failures* to 'UptimeBetweenFailures' and on *Time To Repair* to 'TimeToRepair'. Right click on the Model and select Properties. Under Controls, General set *UptimeBetweenFailures* to 'Random.Exponential(5)', *Units* to 'Minutes', *TimeToRepair* to 'Random.Uniform(.25,.5)' and *Units* to 'Minutes'.
- In both of the Server objects, under Add-On Process Triggers, set Failed to 'Server_Failed' and Repaired to 'Server_Repaired'.

Add-On Processes and Input Arguments

- In the Definitions window, under Tokens, add a new token called "MyToken". Add an Object Reference of type *Node* and call it 'Destination'.
- In the Processes window, highlight the Server_Failed process and under Advanced Options, change *Token Class Name* to 'MyToken'. Select Input Arguments to open the repeating property editor. Add an item where *Name* is

'LocalNode' and *State Variable Name* is 'MyToken.Destination'.

- In the Server_Failed process, add a Seize step. Select Resource Seizes to open the repeating property editor. Add an item where Object Name is 'Worker1', Request Move is 'ToNode' and Destination is 'MyToken.Destination'.
- In the Server_Repaired process, add a Release step and select Resource Releases to open the repeating property editor. Add an item where *Object Name* is 'Worker1'.
- In the Facility window, select Server1 and under the Failed Add-On Process Triggers, Input Arguments, set *Local Node* to 'BasicNode1'. Select Server2 and under the Failed Add-On Process Triggers, Input Arguments, set *Local Node* to 'BasicNode2'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

MultipleServerFailures - SimBit

Problem:

I need to model failures at a Server object using more than one type of failure event stream. The Reliability Logic properties on the Server object only allow for a single way to generate failure occurrences.

Categories:

Resources, Failures, Add-On Process Logic

Key Concepts:

Fail, Failure, Math.Remainder, Multiple Failure Types, Processing Count Based Failure, Processing Time Based Failure, Repair, Server

Assumptions:

A processing time based failure occurs at the server approximately every minute and takes 30 seconds to repair. A processing count based failure (preventative maintenance) occurs at the server after every 10 parts are processed and takes a minute to repair.

Technical Approach:

The processing time based failure stream will be modeled using the Reliability Logic properties of the Server object. The processing count based failure stream will be modeled using add-on process logic.

Details for Building the Model:

System Setup

- Place Source, Server and Sink objects in the Facility window. Connect these objects using Connectors.

Modeling the Processing Time Based Failure Stream

- For the Reliability Logic properties of the Server object, specify *Failure Type* as 'Processing Time Based', *Uptime Between Failures* as 'Random.Exponential(1)' minutes, and *Time to Repair* as '.5' minutes.

Modeling the Processing Count Based Failure Stream

- In the Definitions window, add a real State called NumberProcessed.
- In the Facility window, on the Server object, define an Add-On Process Trigger for the After Processing property and call it 'Server1_AfterProcessing'.
- In the Processes window, for the logic of the 'Server1_AfterProcessing Process', add an Assign step and assign the *State Variable Name* 'NumberProcessed' to *New Value* 'NumberProcessed+1'. Next add a Decide step with an *Expression* 'Math.Remainder(NumberProcessed,10) == 0'. On the True exit point of the Decide step, add an Execute step and specify the *Process Name* as 'PrevMaintenance'.
- Still in the Processes window, create a process and name it 'PrevMaintenance'. In this process, add the steps Fail, Delay, and Repair. For the Fail step, specify the *Failure Name* to 'Server1.Failure'. For the Delay step, specify the *Delay Time* to be '1' minute. For the Repair step, specify the *Failure Name* to be 'Server1.Failure'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

MultiServerSystemWithJockeying - SimBit

Problem:

I want to model a service system that consists of a group of parallel servers, where each server has its own waiting line and customers switch between lines if they think they will get served faster.

Categories:

Buffering

Key Concepts:

Buffer Logic, Reneging, Jockeying

Assumptions:

A system for processing customer entities consists of a group of three parallel servers. Each server has its own separate waiting line. When a customer arrives, they will choose the leftmost shortest queue to enter. Servers process their queues in FIFO order, however if a customer at the end of a waiting line can move up by switching lines, then they will (see *Figure 1*).

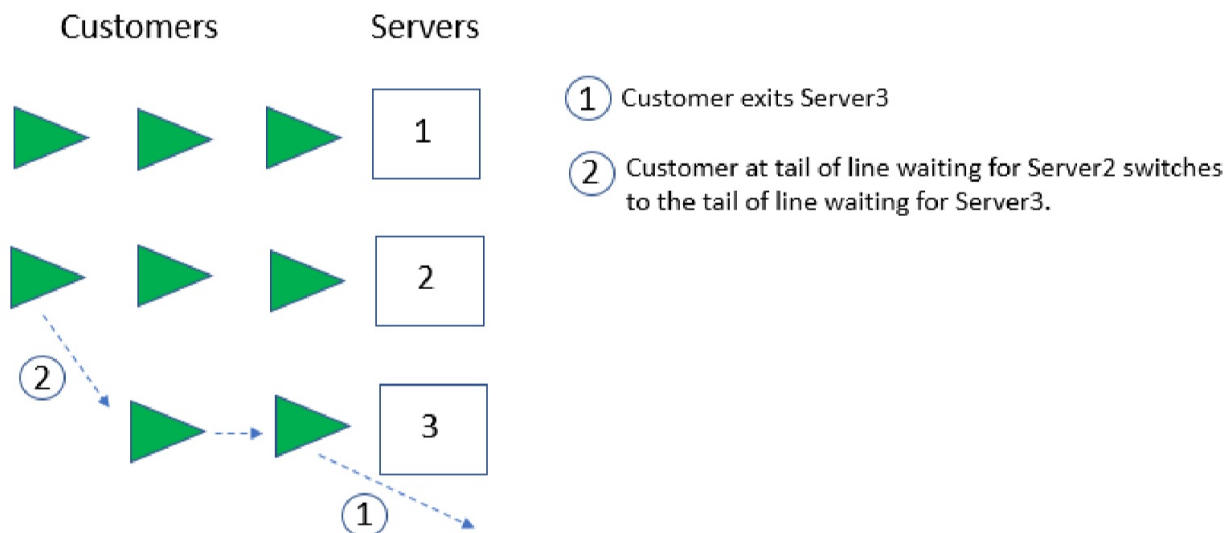


Figure 1 - Parallel Servers with Jockeying Between Waiting Lines

The time between customer arrivals is sampled from an exponential distribution, $EXPO(1)$ minutes. The processing time for each customer is $EXPO(2.8)$ minutes.

The jockeying rule is formalized as follows: If the completion of processing at a server i causes $n_j > n_i + 1$ for some other server j , then the customer from the tail of queue j switches to the tail of queue i . If there are two or more such customers, then the one from the closest, leftmost queue jockeys.

Details for Building the Model:

Facility Window Setup

- Add a Source, three Servers, and Sink to the Facility window. Connect the Source to the Servers and the Servers to the Sink using Connector links.
- Place a ModelEntity object from the Project Library into the Facility window. Name it 'Customer'.

Server Node List Definition

- Go to the Definitions, Lists window. Add a node list and name it 'ServerNodeList'. Specify the nodes in the list as Input@Server1, Input@Server2, and Input@Server3 (in that order).

Source Properties

- Specify the Entity Type as 'Customer' and the Interarrival Time as 'Random.Exponential(1)' minutes.

Source 'Output' Node Properties

- Specify the Entity Destination Type as 'Select From List' and the Node List Name as 'ServerNodeList'. Specify the destination Selection Goal as 'Smallest Value' of Selection Expression 'Candidate.Server.InputBuffer.Contents + Candidate.Server.Processing.Contents'. This will cause a new customer arrival to choose the server that is the leftmost shortest queue.

Custom Function Definitions

- To simplify some expressions, go to the Definitions, Functions window. Add three custom functions named 'NumberAtServer1', 'NumberAtServer2', and 'NumberAtServer3'. The expressions used to return a value for these functions are as follows:

Function Name	Function Expression
NumberAtServer1	Server1.InputBuffer.Contents + Server1.Processing.Contents
NumberAtServer2	Server2.InputBuffer.Contents + Server2.Processing.Contents
NumberAtServer3	Server3.InputBuffer.Contents + Server3.Processing.Contents

Server Properties

- For each server, specify the Processing Time as 'Random.Exponential(2.8)' minutes.
- For each server, go to Buffer Logic -> Input Buffer -> Balking & Reneging Options. Open up the Renege Triggers repeat group and specify event-based renege triggers for each of the server input buffers using the table below as

well as the example model for guidance.

Event-Based Renege Triggers for the Server Input Buffers

Server Name	Renege Triggering Event Name	Renege Condition Description (customer will renege if...)	Renege Node Name
Server1	Output@Server2.Exited	Is last entity in queue and can move up by switching to Server2's queue	Input@Server2
	Output@Server3.Exited	Is last entity in queue and can move up by switching to Server3's queue AND an entity at Server2 is not jockeying to that queue.	Input@Server3
Server2	Output@Server1.Exited	Is last entity in queue and can move up by switching to Server1's queue	Input@Server1
	Output@Server3.Exited	Is last entity in queue and can move up by switching to Server3's queue	Input@Server3
Server3	Output@Server1.Exited	Is last entity in queue and can move up by switching to Server1's queue AND an entity at Server2 is not jockeying to that queue.	Input@Server1
	Output@Server2.Exited	Is last entity in queue and can move up by switching to Server2's queue AND an entity at Server1 is not jockeying to that queue.	Input@Server2

Embellishments:

In this example, we have also used the State Assignments section of properties to change the picture of the entity if renege occurs at a Server. This is done by selecting the On Renege repeat group and specifying the *Assign If* property as 'Reneging From Input Buffer', the *State Variable Name* as 'ModelEntityPicture' and the *New Value* as '1' (for Server1) or whichever entity picture desired. Note that clicking on the ModelEntity, selecting the Add Additional Symbol button multiple times, and then changing the symbol of the entity will allow the graphics to update with such On Renege state assignments.

See also:

SourceWithBalkingIfBlocked.spfx

ServerQueueWithBalkingAndReneging.spfx

ChangingQueuesWhenServerFails.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

NetworkElementShortestPathsLogic - SimBit

Network Element Shortest Paths Logic

This SimBit project includes three models providing examples of using the Network Element's Shortest Paths Logic.

1. **WeightMultiplier:** Demonstrates providing the Network a weighted length when considering the shortest path.
2. **DesiredDirection:** Demonstrates changing the traffic directions on a Bidirectional Path and how the Networks update their shortest paths.
3. **ExclusionCondition:** Demonstrates how to exclude a Link from the shortest path calculations when the Link is a part of the Network.
4. **TiebreakerFewestTurns:** Demonstrates a Network using fewest turns when there are ties for shortest routes.

Model 1: WeightMultiplier

Problem:

Entities should pick the route with the shortest travel time. This should factor in route length and speed limit on the route.

Categories:

Networks, Path

Keywords:

Element, Network, Shortest Path, Weight Multiplier

Assumptions:

Entities will arrive randomly and always choose the path with the smallest travel time. Entities will know their end destination.

Technical Approach:

The Entity will use a specific Network and want to choose the shortest path. That Network Element will have a Weight Multiplier that multiplies the path length and the specified expression to resolve which is the shortest weighted link.

Details for Building the Model:

Facility Window: Objects and Properties

- See the model for approximate placement of all objects.
- Place a Source.
- Place a ModelEntity in the Facility below the Source.
- Place a Sink 16 meters away from the Source. Use the grid to approximate the placement of the object.
- On the Output@Source1 Node change the following properties:
 - Routing Logic
 - *Entity Destination Type* = 'Specific'
 - *Node Name* = 'Input@Sink1'
- Draw a Path, Path1, starting from the Output@Source1. Go up 10 meters and click to place a Vertex for the Link. Go over 16 meters and place another Vertex for the Link. Connect the Path to the Input@Sink1 by going down 10 meters.
- This Path's length should be 36 meters long. You can check this under the Path's *General > Physical Characteristics > Size Properties*.

- Change the Path's properties:
- Travel Logic
- *Speed Limit* = '15'
- *Units* = 'Meters per Minute'
- Draw another Path, Path2, from the Output@Source1 directly to the Input@Sink1
- This Path's length should be 16 meters long. You can check this under the Path's *General > Physical Characteristics > Size Properties*
- Change the Path's properties:
- Travel Logic
- *Speed Limit* = '1'
- *Units* = 'Meters per Minute'
- Draw a third Path, Path3, starting from the Output@Source1. Go down 5 meters and click to place a Vertex for the Link. Go over 16 meters and place another Vertex for the Link. Connect the Path to the Input@Sink1 by going up 5 meters.
- This Path's length should be 26 meters long. You can check this under the Path's *General > Physical Characteristics > Size Properties*.
- Change the Path's properties:
- Travel Logic
- *Speed Limit* = '10'
- *Units* = 'Meters per Minute'

Definitions Window: Elements

- Create a Network Element.
- Change the following properties:
- Basic Logic
- *Links Type* = 'All'
- Shortest Paths Logic
- *Shortest Path Weight Multiplier* = '1/Path.SpeedLimit'

Facility Window: Properties

- On the DefaultEntity Instance, update the following properties:
- Travel Logic
- *Initial Desired Speed* = 'Infinity'
- *Initial Network* = 'Network1'

Discussion:

When considering the Network's shortest path, the *Shortest Paths Weight Multiplier* is applied to the Link's length to determine the link's weighted length. In this example, the weight applied considers the Link's total travel time. The minimum weighted Link will always be chosen. Although the top Path has the longest length, it has the fastest travel speed. Taking that route would ensure the entities got there quickest.

Notes:

The DefaultEntity's *Initial Desired Speed* is set to 'Infinity' so that the Path's *Speed Limit* is the limiting factor and informs which Link has the fastest travel time. If the Entity's speed was slower than the Path's *Speed Limit*, then the entity's speed would be the limiting factor and should be considered in the Network's *Weight Multiplier* instead of the Path's *Speed Factor*.

Model 2: DesiredDirection

Problem:

Entities must share a path but want to travel in opposite directions. The direction of travel allowed through the path must be controlled.

Categories:

Networks, Path Logic

Keywords:

Desired Direction, Element, Network, Shortest Path

Assumptions:

Two different entities will arrive randomly and share a middle path. The Path's direction of traffic will change every two minutes.

Technical Approach:

Process Logic will be used to change the middle Path's Desired Direction so that it will only flow one way at a time. The Networks will consider the Desired Direction of the Path and automatically update its shortest path when the Desired Direction changes.

Details for Building the Model:

Facility Window: Objects and Properties

- See the model for approximate placement of all objects.
- Place two Sources. Place the second Source approximately 3 meters below the first and 18 meters to the left. Rename the Source on the left "RedSource" and the Source on the right "BlueSource". Rotate the BlueSource by holding the ctrl key and clicking and dragging the object's green handles.
- Place a Sink across from the RedSource and above the BlueSource. Name the Sink "RedSink".
- Place a Sink across from the BlueSource and below the RedSource. Name the Sink "BlueSink". Rotate the BlueSink by holding the ctrl key and clicking and dragging the object's green handles.
- Place two ModelEntity Instances, one below each Source. Below the RedSource, name the entity "Red". Below the BlueSource, name the entity "Blue".
- On the RedSource, change the *Entity Type* property to 'Red'.
- On the Output@RedSource, change the Routing Logic's *Entity Destination Type* property to 'Specific' and the *Node Name* property to 'Input@RedSource'.
- On the BlueSource, change the *Entity Type* property to 'Blue'.
- On the Output@BlueSource, change the Routing Logic's *Entity Destination Type* property to 'Specific' and the *Node Name* property to 'Input@BlueSource'.
- Add two BasicNodes. BasicNode1 should be placed approximately 2 meters to the right of and directly in between Output@RedSource and Input@BlueSink. BasicNode2 should be placed 2 meters to the left of and directly in between Input@RedSink and Output@BlueSource. See the model for approximate placement.
- Add the following Paths to connect the Nodes and Objects. It is important to note the Path names for later when the Networks are created. Specified below are the Path names and their starting and ending Nodes respectively:
- Path1: Output@RedSource -> Input@RedSink
- This Path should be drawn by placing a vertex 4 meters up from the Output@RedSource and then placing another vertex 16 meters over.
- Path2: Output@BlueSource -> Input@BlueSink

- This Path should be drawn by placing a vertex 4 meters down from the Output@BlueSource and then placing another vertex 16 meters over.
- Path3: Output@RedSource -> BasicNode1
- Path4: Output@BlueSource -> BasicNode2
- Path5: BasicNode1 -> BasicNode2
- Path6: BasicNode1 -> Input@BlueSink
- Path7: BasicNode2 -> Input@RedSink
- Select only Path5, which is the middle connecting path, and change the following properties:
- Travel Logic
- *Type* = 'Bidirectional'
- *Traffic Direction Rule* = 'Match Desired Direction'
- *Initial Desired Direction* = 'Forward'

Definitions Window: Elements

- Create two Network Elements. Name one "RedNetwork" and the other "BlueNetwork".
- On the RedNetwork, change the following properties:
- Basic Logic
- *MemberLinks* Repeating Property Group
- *Member Link Name* = 'Path1'
- *Member Link Name* = 'Path3'
- *Member Link Name* = 'Path5'
- *Member Link Name* = 'Path7'
- Shortest Paths Logic
- *Shortest Paths Graph Type* = 'LinkDesiredDirections'
- *Auto Update Shortest Paths* = 'True'
- On the BlueNetwork, change the following properties:
- Basic Logic
- *MemberLinks* Repeating Property Group
- *Member Link Name* = 'Path2'
- *Member Link Name* = 'Path4'
- *Member Link Name* = 'Path5'
- *Member Link Name* = 'Path6'
- Shortest Paths Logic
- *Shortest Paths Graph Type* = 'LinkDesiredDirections'
- *Auto Update Shortest Paths* = 'True'

Facility Window: Properties

- On the Red Entity Instance, update the following properties:

- Travel Logic
- *Initial Desired Speed* = '0.5'
- *Initial Travel Mode* = 'Network Only'
- *Initial Network* = 'RedNetwork'

- On the Blue Entity Instance, update the following properties:

- Travel Logic
- *Initial Desired Speed* = '0.5'
- *Initial Travel Mode* = 'Network Only'
- *Initial Network* = 'BlueNetwork'

Processes Window: Process Logic

- Use the Select Process button to show the OnRunInitialized Process.
- Add the following steps to this Process:
- Delay step named "2min"
- *Delay Time* = '2'
- *Units* = 'Minutes'
- Assign step named "ChangeDesiredDirection"
- *State Variable Name* = 'Path5.DesiredDirection'
- *New Value* = 'Math.If(Path5.DesiredDirection == Enum.TrafficDirection.Forward, Enum.TrafficDirection.Reverse, Enum.TrafficDirection.Forward)'
- After the Assign step, loop the end of the Process back to the beginning of the Delay step.

Facility Window: Animation

- In the Facility window, go to the Animation ribbon and add a Status Label next to Path5. Change the Status Label's Expression property to 'Path5.DesiredDirection'

Discussion:

When running the Model, the middle connecting Path, Path5, will only allow traffic to flow in one direction. The Path's traffic must match its Desired Direction State Variable. This value will change every 2 minutes and switch between 'Forward' or 'Reverse'. The entities will try to route to the shortest path possible, but as the Desired Direction of Path5 changes the viable shortest path for the entity changes as well. The Network is set up to automatically detect this change with the *Auto Update Shortest Paths* property. Instead of automatically updating, the *Update Shortest Paths Triggers* could be used instead.

In this Model, the Networks are using the *Shortest Path Graph Type* of 'LinkDesiredDirection'. This setting checks the Desired Direction of the Link before determining if the outbound link is valid. The default value for *Shortest Path Graph Type* is 'LinkTypes'. For 'LinkTypes', the Desired Direction of the Link is not considered. When calculating the shortest path, Links are considered valid based on if they are bidirectional or unidirectional matching the direction of travel.

Notes:

The Desired Direction of a Bidirectional Link is based on how the Link is drawn. The 'Forward' direction matches the direction in which the Link was drawn with traffic going from its originating Node to its ending Node. The 'Reverse' direction matches the direction starting from the Link's ending Node and heading towards the Link's originating Node. Occasionally, it is helpful to change the Link back to unidirectional to see which way the Link was drawn.

The TrafficDirection is an Enumeration in Simio that contains the options for directions of traffic on a Link. 'Forward' corresponds to the value '1' and 'Reverse' corresponds to the value '2'. For more information, check out the "Enumeration"

Help page in the Simio Reference Guide.

In this example, the Links are added to the Networks by adding items into the *Member Links* Repeat Group Property. While in the Facility, you can right-click on a Link and use the 'Add to Network' or 'Remove from Network' options and then see the *Member Links* Repeat Group Property has updated accordingly. Additionally, while in the Facility, the Visibility tab's View Networks option will be able to show these Networks.

Model 3: ExclusionCondition

Problem:

A facility has aisles used by pedestrians and aisles used by vehicles. There is one main aisle that is shared by both pedestrians and vehicles, but they cannot use it at the same time. The main aisle must be dedicated to one group at a time.

Categories:

Networks, Path

Keywords:

Data Driven, Element, Exclusion Condition, Network, Shortest Path

Assumptions:

Entities have a set destination. The pedestrians and vehicles must only use the routes available to them.

Technical Approach:

Two Networks will define the routes available to pedestrians and vehicles. Both Networks will contain the Link for the middle main aisle. An Exclusion Condition on each Network will exclude the middle aisle Link from one Network but allow the Link to be included in the other Network. A variable will change, and an Event is triggered causing the Networks to update and change if they are excluding the middle aisle from their Network.

Details for Building the Model:

Facility Window: Objects and Properties

- See the model for approximate placement of all objects.
- Place a two Sources. Place the second Source approximately 3 meters below the first.
- Place two Sinks across from the two Sources so that the Sources' Output Nodes is approximately 16 meters from the Sinks' Input Nodes.
- Add a Worker Instance and place it near Source1.
- Add a Vehicle Instance and place it near Source2.
- On the Output@Source1 change the following properties:
 - Routing Logic
 - *Entity Destination Type* = 'Specific'
 - *Node Name* = 'Input@Sink1'
- Transport Logic
- *Ride On Transporter* = 'Always'
- Transportation Selection
- *Transporter Name* = 'Worker1'
- On the Output@Source2 change the following properties:
 - Routing Logic
 - *Entity Destination Type* = 'Specific'
 - *Node Name* = 'Input@Sink2'

- Transport Logic
- *Ride On Transporter* = 'Always'
- Transportation Selection
- *Transporter Name* = 'Vehicle1'
- Add two BasicNodes. BasicNode1 should be placed approximately 2 meters to the right of and directly in between Output@Source1 and Output@Source2. BasicNode2 should be placed 2 meters to the left of and directly in between Input@Sink1 and Input@Sink2.
- Add the following Paths to connect the Nodes and Objects. It is important to note the Path names for later when the Networks are created. Specified below are the Path names and their starting and ending Nodes respectively:
- Path1: Output@Source1 -> Input@Sink1
- This Path should be drawn by placing a vertex 4 meters up from the Output@Source1 and then placing another vertex 16 meters over.
- Path2: Output@Source2 -> Input@Sink2
- This Path should be drawn by placing a vertex 4 meters down from the Output@Source2 and then placing another vertex 16 meters over.
- Path3: Output@Source1 -> BasicNode1
- Path4: Output@Source2 -> BasicNode1
- Path5: BasicNode1 -> BasicNode2
- Path6: BasicNode2 -> Input@Sink2
- Path7: BasicNode2 -> Input@Sink1
- Change Path1 and Path2 *Type* property to 'Bidirectional'. Change multiple Paths' properties by multi-selecting the Path objects by holding ctrl + clicking.

Definition Window: States

- Create a String State Variable. Name it "MainAisleTrafficFlow". Set the *Initial State Value* property as 'Pedestrian'.

Definition Window: Events

- Create an Event and name it "ChangeMainAisleFlow".

Definitions Window: Elements

- Create two Network Elements. Name one "PedestrianNetwork" and the other "VehicleNetwork".
- On the PedestrianNetwork, change the following properties:
- Shortest Paths Logic
- *Shortest Paths Exclusion Condition* = 'Is.Path5 && MainAisleTrafficFlow == "Vehicle"'
- *Update Shortest Paths Triggers*
- *Triggering Event Name* = 'ChangeMainAisleFlow'
- On the VehicleNetwork, change the following properties:
- Shortest Paths Logic
- *Shortest Paths Exclusion Condition* = 'Is.Path5 && MainAisleTrafficFlow == "Pedestrian"'
- *Update Shortest Paths Triggers*
- *Triggering Event Name* = 'ChangeMainAisleFlow'

Data Window: Tables

- Create a Table named "NetworkTbl".
- Add a Network Element Property column with the following properties:
 - *Name* = 'Network'
 - *Auto-set Table Row Reference* = 'True'
- Populate the column with PedestrianNetwork and VehicleNetwork references.
- Set the column as a Key.
- In the Schema ribbon, use the drop-down on the Add Table button select 'Add Data Table From Schema' > 'Network'. In the dialog that opens, double click the check box next to the 'Network.MemberLinks' Schema. Click 'OK'. A Table named "TblMemberLinks" should be created with a column called "MemberLinkName".
- Add a Foreign Key Property. Use the Move Left button in the Schema ribbon to move this column to the left. Change the properties on this column:
 - *Name* = 'NetworkKey'
 - *Table Key* = 'NetworkTbl.Network'
- Populate the table with the following information:
 - With the Network Key of 'PedestrianNetwork':
 - Path1
 - Path3
 - Path5
 - Path7
 - With the Network Key of 'VehicleNetwork':
 - Path2
 - Path4
 - Path5
 - Path6

Definitions Window: Elements

- On both Networks, update the *Members Links* Repeating Group Property. Right-click on the Repeat Group property and use 'Set Reference' > 'TblMembersLinks'. If you open the Repeat Group by clicking the button with 3 dots, the *Member Link Name* property should automatically be set to reference 'TblMemberLinks.MemberLinkName'.

Facility Window: Properties

- On the Worker1 Instance, update the following properties:
 - Travel Logic
 - *Initial Travel Mode* = 'Network Only'
 - *Initial Network* = 'PedestrianNetwork'
 - Routing Logic
 - *Initial Node (Home)* = 'Output@Source1'
 - *Idle Action* = 'Go To Home'

- On the Vehicle1 Instance, update the following properties:
- Travel Logic
- *Initial Travel Mode* = 'Network Only'
- *Initial Network* = 'VehicleNetwork'
- Routing Logic
- *Initial Node (Home)* = 'Output@Source2'
- *Idle Action* = 'Go To Home'

Processes Window: Process Logic

- Use the Select Process button to show the OnRunInitialized Process.
- Add the following steps to this Process:
- Delay step named "2min"
- *Delay Time* = '2'
- *Units* = 'Minutes'
- Assign step named "ChangeAisleTraffic"
- *State Variable Name* = 'MainAisleTrafficFlow'
- *New Value* = 'Math.If(MainAisleTrafficFlow == "Vehicle", "Pedestrian","Vehicle")'
- Fire step named "ChangeAisle".
- *Event Name* = 'ChangeMainAisleFlow'
- After the Fire step, take the end of the process and loop it back to the beginning of the process, at the Delay step.

Discussion:

By using the Network's *Shortest Paths Exclusion Condition*, viable paths can be controlled at the Network level. Previously, the Link could have been made unavailable by setting the Traveler Capacity to '0' or by changing the Desired Direction to 'None' if it was a Bidirectional Link. However, those approaches would have made the Path unavailable to any traveler in the entire Model. Now, a Link could be listed as a member of a Network but the *Shortest Paths Exclusion Condition* could render it not a current viable option for the shortest path, thus preventing travel on it. While at the same time another Network could still be using the Link.

As the conditions change during the run, the Network must occasionally update. In this example, the Network should update when the variable in the *Shortest Paths Exclusion Condition* is changed so that the Network has the most up-to-date viable shortest path for each group. The *Update Shortest Path Triggers* has two options for when the *Triggering Event Name* should cause the update. The *Update Type* 'Deferred' was used in this mode as it is the default value and preferred option.

The 'Deferred' *Update Type* option will delay the shortest path calculations until they are required. This means Dijkstra's shortest path calculation will not occur until it is needed by an entity traveling in the system. Even when the shortest path information is required, the calculations are only done for the relevant Node rather than the entire Network. This makes this *Update Type* much more efficient. It could also be helpful if a variable relevant to the Network has updated multiple times, but nothing has needed to use the Network in between those updates. Triggering the Network to do the calculation each time would cause unnecessary computations.

The other *Update Type* is 'Immediate', which means all shortest path calculations happen immediate after the triggering event. This option could become more computationally expensive to constantly update. However, this type may be needed if the exact value of a variable in the event triggering time slice is needed. For example, you may need the value of a Model's State Variable at the time the event is triggered. If the *Update Type* was 'Deferred', this variable's value could change by the time the Network is required to update.

Notes:

In this example, the Network *Member Links* information is stored in Data Tables. This Data Table is fed into the Repeating Group Property. Each row in the Data Table becomes an item in the Repeat Group. Since the Networks have a reference to

a Key, the Key filters down the TblMemberLinks and only uses the related rows. The Network's Key is set via the *Auto-set Table Row Reference* property on the Element Property column in the Data Table.

When the Networks' *Member Links* information is driven by the data table, the Network element itself does not resolve the information from the Data Table during design time. Therefore, while in the Facility, if you right-click on a Link and use 'Add to Network' or 'Remove from Network' you will not see these Network elements available. These Elements only use the information from the Data Table and the information must be changed in the Data Table. Additionally, while in the Facility, the Visibility tab's View Networks option will not show these Networks.

Model 4: FewestTurnsTiebreaker

Problem:

There are multiple shortest path options. With multiple shortest paths, the Network should use a tie breaker rule to select the route with the fewest turns.

Categories:

Networks, Path

Keywords:

Element, Fewest Turns, Network, Shortest Path

Assumptions:

Entities will arrive in a regular cadence and will cycle through the list of destinations.

Technical Approach:

The entity will use a specific Network and want to choose the shortest path. That Network will have multiple shortest paths. The Network should use the *Shortest Paths Tie Breaker Rule* with 'Fewest Turns' to select the shortest route with the least number of turns for the traveler to go to its destination.

Details for Building the Model:

Facility Window: Objects and Properties

- Place a Source. Change the Source's *Interarrival Time* property to '1'. An entity will arrive every minute.
- Place a ModelEntity Instance.
- Create a 4 x 4 grid of Nodes. The Output@Source1 should be the top left corner of the grid. Add BasicNodes spaced out by 3 meters. The order of Nodes placed, position and names of Nodes, is not important for this example, but it is important the Nodes are exactly evenly spaced. In total, 15 BasicNodes should be added.
- Place 8 Sinks. Rotate 4 Sinks by holding the ctrl key and grabbing the green handle on the Sink object. Place the Sinks so their Input Node is approximately 3 meters away from the Nodes on the right and bottom of the grid.
- Draw Paths connecting the Nodes. The order of the Paths placed, position and names of the Paths, are not important for this example. Draw the Path starting from the right Node and ending on the left Node or starting from the top Node and ending on the bottom Node. All Paths should flow pointing to the right or to the bottom. Do not connect the Sink Input Nodes to other Sink Input Nodes. A total of 32 Paths should be placed in the Model.
- Create a Node List containing all the Sinks' Input Nodes. Select all the Input Nodes to the Sink. Use ctrl + mouse click to multi-select the Nodes. Note, the order the Nodes are selected will indicate the order they will be added to the List. In this case, the Nodes were selected in numerical order starting with Sink1. Right-click on one of the selected Nodes. Use the drop-down to 'Add to Node List' > 'Create New Node List...'. A dialog window will pop up to name the new list. Retain the name "NodeList1" and click 'OK'. This Node List can be edited from the Definitions window Lists view.
- Update the Properties on the Output@Source1 Node:
 - Routing Logic
 - *Entity Destination Type* = 'Select From List'
 - *Node List Name* = 'NodeList1'
 - *Selection Goal* = 'Cyclic'

Definitions Window: Elements

- Create a Network Element.
- Change the following properties:
- Basic Logic
- *Links Type* = 'All'
- Shortest Paths Logic
- *Shortest Path Tie Breaker Rule* = 'FewestTurns'

Facility Window: Properties

- On the DefaultEntity Instance, update the following properties:
- Travel Logic
- *Initial Desired Speed* = '0.5'
- *Initial Network* = 'Network1'

Discussion:

When the model is run, the entities will cycle through each Sink as a destination. Since this is a grid with each Link the same length, for some Sink locations there are multiple shortest path options. With the *Shortest Paths Tie Breaker Rule* set to 'FewestTurns', when there are multiple shortest paths, the route with the least number of turns is taken. If there are multiple shortest paths with the same number of turns, the Network will randomly pick one to use. For this rule, a turn is counted at a node if travel from the inbound link to the outbound link is not approximately a straight line. This rule does not consider any Vertices on the Link itself that might be a turn on the Link. This is only considered at a Node when entering a new inbound Link.

To test the behavior without the 'FewestTurn' rule, consider changing the *Shortest Paths Tie Breaker Rule* to 'None' and run the model.

Notes:

The visibility for Queues and Node Labels has been turned off in this example model. This can be changed via the Facility window's Visibility ribbon.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

NeuralNetworks - SimBit

Note: If you expect to see the Neural Network Models view but the option does not appear in the Data tab, please ensure your software license is applied. For further information on Simio edition capabilities or to upgrade your software, please contact sales@simio.com

This SimBit project includes three models providing examples of Simio's Neural Network features.

1. **NeuralNetwork_Baseline:** Provides the setup instructions for the base system and provides a baseline average makespan measurement.
2. **NeuralNetwork_DataCollection:** Demonstrates how to set up a model to automatically record training data for a neural network or other machine learning model.
3. **NeuralNetwork_Training:** Demonstrates how to train a neural network using Simio's neural network trainer and integrate the neural network into a simulation model to make decisions.
4. **NeuralNetwork_OnnxNeuralNetwork:** Demonstrates how to set up a Simio model to use a previously trained neural network model saved in the ONNX file format.

Model 1: NeuralNetwork Baseline

Problem:

I have a system where orders (Entities) are routed at random to one of two lines for processing, and I want to estimate the average order makespan.

Categories:

Data Science

Keywords:

Data Table, RoutingGroup Element

Assumptions:

The modeled system consists of two lines, each with multiple ordinal processing locations. When orders arrive, they are routed either to Line A or Line B and cannot change their line. When the orders complete processing at the final location on their line, they are transferred to a shipping dock where they depart the system.

Technical Approach:

The system is modeled using one Source which creates the orders; one Sink, which destroys the orders; and 7 Servers split between two lines. The order entities are created at random interarrival times by the Source, where they are routed by a RoutingGroup to either Line A or Line B at random.

Details for Building the Model:

Facility Tab, Objects

- Place a ModelEntity instance named 'Order' into the model.
- Place a Source and a Sink about 40 meters apart, as shown in the model. Change the *Interarrival Time* of the Source to 'Random.Exponential(1.8)' minutes.
- Place a Resource named 'ResourceA' with an *Initial Capacity* of '2'.
- Place 7 Servers named A1-A4 and B1-B3 as shown in the model.
- Select Servers A1, A3, and A4. With the Servers selected, change their *Processing Time* to 'Random.Triangular(1, 1.5, 2)' minutes, their *Resource Name* to 'ResourceA', and their Secondary Resources > For Processing > Repeat Group to 'False'.
- Change the *Processing Time* of Server A2 to 'Random.Triangular(4, 4.5, 5)' minutes and change its *Initial Capacity* to '4'.
- Change the *Processing Time* of Servers B1, B2, and B3 to 'Random.Triangular(3, 3.5, 4)' minutes.
- Change the *Initial Capacity* of Server B1 to '2'.
- Connect the two lines of Servers with Paths starting from the Source and leading to the Sink, as shown in the model.

- Add a status label near the sink. Change its *Expression* to 'Order.Population.TimeInSystem.Average'.

Routing Setup

- In the Tables view of the Data tab, create a new table called 'Nodes'
- Create a new Node property column called 'Node'
- Enter 'Input@A1' and 'Input@B1' into the 'Node' column of the 'Nodes' table.
- Change its *Destination Node List Name* to 'Nodes.Node'.
- In the Facility tab, change the *Entity Destination Type* property of the Source's output node to 'Select From List', its *Node List Name* to 'Nodes.Node', and its *Selection Goal* to 'Random'.

Discussion:

This model establishes the current state system and a baseline makespan measurement. The baseline measurement allows us to compare the performance of the system in its current state with the performance after we use a neural network to improve how orders are routed, providing an understanding of the effectiveness of the neural network models. The average makespan for all orders in the current state system is 0.35 hours, although we expect this to decrease after integrating the neural network models later in this example.

Model 2: NeuralNetwork DataCollection

Problem:

I have a system where orders (Entities) are sent either to Line A or Line B. I want to collect data from this system so I can train a neural network model to predict the makespan for either line.

Categories:

Data Science

Keywords:

AssociatedStationLoad, Data Table, Event, Neural Network Element, Neural Network Model, RoutingGroup Element

Assumptions:

The modeled system consists of two lines, each with multiple ordinal processing locations. When orders arrive, they are routed either to Line A or Line B and cannot change their line. When the orders complete processing at the final location on their line, they are transferred to a shipping dock where they depart the system.

Technical Approach:

The system is modeled using one Source, which creates the orders; one Sink, which destroys the orders; and 7 Servers split between two lines. The order entities are created at random interarrival times by the Source, where they are routed by a RoutingGroup to either Line A or Line B at random. Neural network models are added to automate the collection of data from the system. Collecting this data will allow the neural network models to be trained later in this example.

The neural network elements use the RoutingGroup's AssigningDestination event to trigger the input data collection. In this case, the data collected is the load at each Server in the line where the entity is routed. Then, once the entity is done processing, the entity's makespan is recorded and automatically matched to the station load data points collected when the entity was routed. For example, an entity is routed to Line A. When this occurs, the neural network model associated with Line A will record the load at each Server in Line A. Then, after the entity completes processing at Server A4, its makespan will be recorded. The recorded makespan is automatically matched to the station load data points collected when the entity was routed.

Details for Building the Model:

Initial Setup

- Begin with the setup shown in the "Model 1: NeuralNetwork_Baseline" section. You may right-click the NeuralNetwork_Baseline model in the navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Neural Network Setup

- In the Neural Network Models view of the Data tab, create two Feedforward Neural Network Models named 'LineA_MakespanPredictorModel' and 'LineB_MakespanPredictorModel'. Change the Number Input Nodes on 'LineA_MakespanPredictorModel' to '4' and the Number Input Nodes on 'LineB_MakespanPredictorModel' to '3'. Note that for a Feedforward Neural Network Model, these numbers cannot be modified after they are initially assigned. This limitation exists because changing the number of inputs of a neural network model will destabilize the entire network, negating any previous training.
- In the Elements view of the Definitions tab, create two new Neural Network elements named 'LineA_MakespanPredictor' and 'LineB_MakespanPredictor'.

- In 'LineA_MakespanPredictor':
- Change the *Neural Network Model Name* to 'LineA_MakespanPredictorModel'.
- Add 4 new rows to the *Input Value Expressions* repeat group with the values 'Input@A1.AssociatedStationLoad', 'Input@A2.AssociatedStationLoad', 'Input@A3.AssociatedStationLoad', and 'Input@A4.AssociatedStationLoad'.
- Add 1 new row to the *Save Inputs Triggers* repeat group with a *Triggering Event Name* of 'LineSelection.AssigningDestination' and a *Condition* of 'Node.Is.Input@A1'.
- Add 1 new row to the *Save Actual Triggers* repeat group with a *Triggering Event Name* of 'Output@A4.Entered'.
- Set the *Actual Value* expression to 'TimeNow - ModelEntity.TimeCreated'.
- In 'LineB_MakespanPredictor':
- Change the *Neural Network Model Name* to 'LineB_MakespanPredictorModel'.
- Add 3 new rows to the *Input Value Expressions* repeat group with the values 'Input@B1.AssociatedStationLoad', 'Input@B2.AssociatedStationLoad', and 'Input@B3.AssociatedStationLoad'.
- Add 1 new row to the *Save Inputs Triggers* repeat group with a *Triggering Event Name* of 'LineSelection.AssigningDestination' and a *Condition* of 'Node.Is.Input@B1'.
- Add 1 new row to the *Save Actual Triggers* repeat group with a *Triggering Event Name* of 'Output@B3.Entered'.
- Set the *Actual Value* expression to 'TimeNow - ModelEntity.TimeCreated'.

Routing Setup

- In the Tables view of the Data tab, open the 'Nodes' table.
- Create a new Neural Network Element property column called 'NeuralNetworkElement'.
- Enter the following data into the table.

Input@A1	LineA_MakespanPredictor
Input@B1	LineB_MakespanPredictor

- In the Facility tab, change the *Selection Goal* property of the Source's output node to 'Math.If(LineA_MakespanPredictor.IsTrained & & LineB_MakespanPredictor.IsTrained, Nodes.NeuralNetworkElement.PredictedValue(Candidate.Node), Random.Uniform(0, 1))'.

Data Collection

- In the Generate Training Data ribbon in the Neural Network Models view, click the "Select Neural Network Models" dropdown. Ensure that each neural network model is selected.
- Click the "Run" button to begin the data collection process. Simio will automatically run replications of the model until the number of records in each neural network's training data repository equals the neural network's *Maximum Records Limit*. In this case, each neural network has the default *Maximum Records Limit* of '1000'. A neural network model's training records can be viewed by selecting the neural network model, as shown below. In the screenshot, the training data records for the "LineA_MakespanPredictorModel" are visible.

Discussion:

The objective of this model is to collect data so Neural Network Models can be trained. When the Neural Network Model associated with a Neural Network Element is untrained, the "PredictedValue" function will evaluate the Neural Network Element's *Untrained Predicted Value Expression* and return the result. In this case, each Neural Network Element's *Untrained Predicted Value Expression* is left as the default, 'Random.Uniform(0,1)'. When an entity is routed at Output@Source1, it will evaluate the *Untrained Predicted Value Expression* for each line and select the line with the smallest value. With this approach entities will route to a line at random, allowing for random data to be collected before the neural network models are trained. Adding randomness in the data collection stage can be beneficial because the additional variability can increase the variability of the data points captured by the simulation model, which help train a more robust machine learning model.

Data collection is completed automatically by Simio after clicking the Run button in the Generate Training Data ribbon. The neural network inputs are recorded when the Output@Source1 TransferNode selects a destination for the entity, and the entity's actual makespan is collected when they have completed processing at their line. The event 'Output@Source1.RoutingOut.AssigningDestination' references the 'AssigningDestination' event triggered by the 'RoutingOut' RoutingGroup within the 'Output@Source1' TransferNode. The input data for this case is the associated station load at each Server along the selected line, which accounts for entities en route to the station, entities at an input

Keywords:

AssociatedStationLoad, Data Table, Event, Neural Network Element, Neural Network Model, RoutingGroup Element

Assumptions:

The modeled system consists of two lines, each with multiple ordinal processing locations. When orders arrive, they are routed either to Line A or Line B and cannot change their line. When the orders complete processing at the final location on their line, they are transferred to a shipping dock where they depart the system.

Technical Approach:

The system is modeled using one Source, which creates the orders; one Sink, which destroys the orders; and 7 Servers split between two lines. The order entities are created at random interarrival times by the Source, where they are routed by a RoutingGroup to either Line A or Line B. The RoutingGroup uses a trained neural network model for each line to predict the expected makespan if the entity is routed to that line, then routes the entity to the line with the smallest predicted makespan.

Details for Building the Model:**Initial Setup**

- Begin with the setup shown in the “Model 2: NeuralNetwork_DataCollection” section. You may right-click the NeuralNetwork_DataCollection model in the navigation pane and select “Duplicate Model” if you wish to keep this in a separate model as illustrated.

Neural Network Training

- Fast forward the model to collect data for the training the neural network model.
- In the Neural Network Models view of the Data tab, select ‘LineA_MakespanPredictorModel’ and click “Train” in the Neural Network Models ribbon to open the training dialog. Click the “Start Training” button, which will train the neural network model. Click “Save And Close” after the training is complete to save the model.
- Select ‘LineB_MakespanPredictorModel’ and click “Train” in the Neural Network Models ribbon to open the training dialog. Click the “Start Training” button, which will train the neural network model. Click “Save And Close” after the training is complete to save the model.

Discussion:

After clicking the “Start Training” button, Simio will randomly partition the records in the training data repository into training, validation, and testing datasets in accordance with the *Validation Holdout Percentage* and *Test Holdout Percentage* shown in the neural network training dialog. The records from the training dataset are provided to the model directly and used to refine the model’s performance. The validation dataset is used to provide an unbiased estimate of the model’s performance, and to automatically halt training when performance stops improving. This dataset drives the “tuning” of model hyperparameters. The test dataset is used to provide a final unbiased estimate of the model’s performance. The test dataset is important because the process of model “tuning” to perform better on the validation dataset can cause overfitting

After the records in the training data repository are separated, the neural network model is initialized according to the Xavier initialization method defined in the paper “Understanding the difficulty of training deep feedforward neural networks” by Xavier Glorot and Yoshua Bengio¹. With this approach, weights are initialized according to a uniform distribution between the range $-(1/\sqrt{n})$ and $1/\sqrt{n}$, where n is the number of inputs to the neuron. This method randomly initializes the weights in the neural network such that the mean and the variance of the activations is approximately zero, avoiding vanishing or exploding gradients. Biases are initialized to zero. Each time the training process begins, the weights of the network are randomly generated. Different starting weights can cause neural network models to vary in their level of performance, so the performance of a model may not be the same each time you run the neural network trainer.

Once initialization is complete, the Simio neural network trainer begins the training process, using either Adaptive Moment Estimation (Adam) or Stochastic Gradient Descent with Momentum. Training may stop early depending on the early stopping settings or continue until the number of completed training epochs equals the *Max Epochs* setting. Early stopping is controlled by the *Early Stopping Min Delta* and *Early Stopping Patience* settings. *Early Stopping Min Delta* controls an epoch’s minimum decrease in the validation dataset’s mean squared error for the error reduction to be considered an improvement, and *Early Stopping Patience* indicates the number of consecutive epochs that the mean squared error of the validation dataset can show no improvement before training is automatically stopped. Early stopping rules are typically used to halt training before overfitting occurs.

The performance of the neural network model is affected by its training settings such as the selected *Optimizer Type*. The training settings of the neural network model can be changed from the neural network trainer dialog by manipulating properties such as *Number Hidden Nodes* or *Learning Rate*. Changing the parameters of the training process often results in noticeable differences in the model’s performance. When training a neural network model for a project, users are encouraged to experiment with these options to find a solution with an acceptable level of performance.

Once the neural network training is complete, Simio will no longer evaluate each neural network element’s *Untrained Predicted Value Expression*, instead using the inputs in the *Input Value Expressions* repeat group to obtain a makespan estimate from each line’s neural network model. The Output@Source1 TransferNode will select the line with the lowest

makespan when determining where to route the entity. Because the system is now routing entities using an estimate of makespan, the average makespan in this example decreases noticeably from 0.35 hours to approximately 0.20 hours. When replicating this improvement, please note that the reduction in average makespan might vary based on factors such as the neural network model's training data, the training settings of the neural network model, or the random initialization of the neural network.

Model 4: NeuralNetwork_OnnxNeuralNetwork

Problem:

I have a system where orders (Entities) are sent either to Line A or Line B. I want to import a pre-trained ONNX neural network model to predict the makespan if a particular line is selected to route the orders to the line with the smallest predicted makespan.

Categories:

Data Science

Keywords:

AssociatedStationLoad, Data Table, Event, Neural Network Element, Neural Network Model, RoutingGroup Element

Assumptions:

The modeled system consists of two lines, each with multiple ordinal processing locations. When orders arrive, they are routed either to Line A or Line B and cannot change their line. When the orders complete processing at the final location on their line, they are transferred to a shipping dock where they depart the system.

Technical Approach:

The system is modeled using one Source, which creates the orders; one Sink, which destroys the orders; and 7 Servers split between two lines. The order entities are created at random interarrival times by the Source, where they are routed by a RoutingGroup to either Line A or Line B. The LineSelection routing group uses an ONNX neural network model for each line to predict the expected makespan if the entity is routed to that line, then routes the entity to the line with the smallest predicted makespan.

Details for Building the Model:

Initial Setup

- Begin with the setup shown in the "Model 1: NeuralNetwork_Baseline" section. You may right-click the NeuralNetwork_Baseline model in the navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Routing Group Setup

- In the Tables view of the Data tab, create a new table called "Nodes".
- Create a new Node property column called "Node".
- Create a new Neural Network property column called "NeuralNetworkElement".
- Enter the following data into the table.

Input@A1	LineA_MakespanPredictor
Input@B1	LineB_MakespanPredictor

- In the Elements view of the Definitions tab, create a new RoutingGroup called "LineSelection". Change its *Destination Node List Name* to 'Nodes.Node'.
- In the Facility tab, change the *Entity Destination Type* property of the Source's output node to 'Use Custom Routing Group', its *Routing Group Name* to 'LineSelection', its *Selection Goal* to 'Smallest Value', and its *Selection Expression* to 'Nodes.NeuralNetworkElement.PredictedValue(Candidate.Node)'.

Neural Network Setup

- In the Neural Network Models view of the Data tab, create two ONNX Neural Network Models named "LineA_MakespanPredictorModel" and "LineB_MakespanPredictorModel". When prompted to select ONNX files from a file explorer tab, navigate to your Simio installation folder > Examples and select "NeuralNetwork_LineAModel.onnx" and "NeuralNetwork_LineBModel.onnx".
- In the Elements view of the Definitions tab, create two new Neural Network elements named "LineA_MakespanPredictor" and "LineB_MakespanPredictor".
- In "LineA_MakespanPredictor":

- Change the *Neural Network Model Name* to 'LineA_MakespanPredictorModel'.
- Add 4 new rows to the *Input Value Expressions* repeat group with the values 'Input@A1.AssociatedStationLoad', 'Input@A2.AssociatedStationLoad', 'Input@A3.AssociatedStationLoad', and 'Input@A4.AssociatedStationLoad'.
- Add 1 new row to the *Save Inputs Triggers* repeat group with a *Triggering Event Name* of 'LineSelection.AssigningDestination' and a *Condition* of 'Node.Is.Input@A1'.
- Add 1 new row to the *Save Actual Triggers* repeat group with a *Triggering Event Name* of 'Output@A4.Entered'.
- Set the *Actual Value* expression to 'TimeNow - ModelEntity.TimeCreated'.
- In "LineB_MakespanPredictor":
- Change the *Neural Network Model Name* to 'LineB_MakespanPredictorModel'.
- Add 3 new rows to the *Input Value Expressions* repeat group with the values 'Input@B1.AssociatedStationLoad', 'Input@B2.AssociatedStationLoad', and 'Input@B3.AssociatedStationLoad'.
- Add 1 new row to the *Save Inputs Triggers* repeat group with a *Triggering Event Name* of 'LineSelection.AssigningDestination' and a *Condition* of 'Node.Is.Input@B1'.
- Add 1 new row to the *Save Actual Triggers* repeat group with a *Triggering Event Name* of 'Output@B3.Entered'.
- Set the *Actual Value* expression to 'TimeNow - ModelEntity.TimeCreated'.

Routing Setup

- In the Tables view of the Data tab, open the "Nodes" table.
- Create a new Neural Network property column called "NeuralNetworkElement".
- Enter the following data into the table. This will create a syntax error that will be fixed when the neural network elements are added to the model.

Input@A1	LineA_MakespanPredictor
Input@B1	LineB_MakespanPredictor

- In the Facility tab, change the *Selection Goal* property of the Source's output node to 'Math.If(LineA_MakespacePredictor.IsTrained & & LineB_MakespanPredictor.IsTrained, Nodes.NeuralNetworkElement.PredictedValue(Candidate.Node), Random.Uniform(0, 1))'.

Discussion:

Note that the average makespan from a single replication of this approach, 0.19 hours, varies slightly from the average makespan of the approach in the *NeuralNetwork_Training* model, 0.20 hours. This occurs for the same reasons discussed in the "Model 3: NeuralNetworks_Training" section, that a neural network's random initialization and its training settings can affect the overall performance level of the neural network model. Although in this case the model trained in the external program is similar, one advantage of training in an external program is that some packages, such as TensorFlow or PyTorch, have additional training settings to further refine the performance of your neural network model. To see what these additional packages have to offer, we recommend reviewing the documentation for these packages to determine if they are a good fit for your project.

The Open Neural Network Exchange (ONNX) file format is an open-source standard for storing machine learning algorithms. Simio can import and use most fixed-input and single-output machine learning models supported by ONNX. This feature allows for data to be collected in Simio, exported to train a machine learning algorithm in an external program, then for the trained model to be re-imported into a Simio model. An imported ONNX file is saved within the Simio project file, so if the directory of the Simio project file changes, the machine learning model will continue to provide predictions as expected.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

NotifyStep - SimBit

Problem:

When a certain event occurs or when a variable reaches a certain point, I want the simulation to pause while a message is given to the user, asking them to either continue running the simulation or giving them the ability to pause the simulation to investigate the event.

Categories:

Add-On Process Logic, Buffering

Key Concepts:

Contents, CrossingStateChange, Event, InputBuffer, Monitor, Notify Step, On Change Detected Process

Technical Approach:

A message can be displayed to the screen with a Notify Step. This model will place a Notify Step to be executed when a Monitor element fires. This Monitor element fires when the contents of the Server's InputBuffer station exceed 3.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server and a Sink into the Facility window and connect them with Paths.
- Go to the Definitions Window. You should be in the Elements panel (Elements highlighted on the left of the screen).
 - Click on Monitor in the Ribbon to create a new Monitor element.
 - Set the *Monitor Type* property to 'CrossingStateChange'
 - Set the *State Variable Name* to 'Server1.InputBuffer.Contents'
 - Set the *Threshold Value* to '3'.
 - Set the *Crossing Direction* property should be set to 'Positive', which means that this Monitor will fire when the contents of the InputBuffer cross over the value of 3 in the positive direction (i.e. when it increases from 3 to 4).
 - In the drop down of the *On Change Detected Process* property, create a new Process by select "Create New". This new process will be executed when this Monitor fires.
- Go to the Processes window and you should see the new Process that was just created. Place a Notify Step into this process (located under All Steps).
 - Set the *Notification Type* to 'Warning'
 - Set the *Message Heading* to "InputBuffer Monitor" (include the double quotes since this is a String)
 - Set the *Message Content* to "InputBuffer has more than 3 entities" ((include the double quotes since this is a String)

Embellishments:

Change the contents of the String message that is in the *Message Heading* or the *Message Content* properties. First select "Continue running" in the pop up window and then select "Pause the Simulation", to see the behavior of the two choices. Finally, select "Don't show this warning again" at the bottom of the window to turn off future warnings.

Discussion:

The Warning Level is controlled from the Run ribbon under Advanced Options. The user can set the Warning Level to Alert User (this is the default), Write to Trace Only, or Do Nothing. Once the user selects Do Nothing or selects the "Don't show this warning again" check box in the pop up window, they must select Re-enable Disabled Warnings from the Advanced Options tab to get the Warnings to reappear.

ObjectReferenceOnEntity - SimBit

Problem:

I would like to keep track of what Server an entity visited so that this information can be used later in the model. I would like to keep a reference to an object on my entity.

Categories:

Decision Logic -- Paths, Decision Logic -- Processing, Entity Characteristics

Key Concepts:

Active Symbol, Object Reference State, Location.Parent, Selection Weight

Assumptions:

If an Entity is processed at Server1, it must route to Server5 for processing after it finishes at Server4. If an Entity is processed at Server2, it must route to Server6 for processing after it finishes at Server4.

Technical Approach:

An Object Reference State is created on the ModelEntity object, which will be used to hold a reference to which Server the entity visited first (either Server1 or Server2). The Selection Weight properties on the paths the leave Output@Server4 will contain expressions that will route the entity to correct Server, either Server5 or Server6, depending on which Server it visited at the beginning of the model. The logic in these expressions checks the value of the Object Reference State on each entity to determine if it was processed at Server1 or Server2.

Details for Building the Model:

Simple System Setup

- Place a Source (Source1) and Sink (Sink1) at the left and right sides, respectively, of the Facility window.
- Place two Server2 in parallel after the Source and name them Server1 and Server2. Connect Source1 to each new Server with Paths.
- Place two Servers, in series, next. Name them Server3 and Server4. Connect them together with Paths and connect both Server1 and Server2 to Server3 with a Path.
- Place two additional Servers, in parallel, after Server4. Name them Server5 and Server6. Connect Server4 to each new Server with a Path and connect Server5 and Server6 to the Sink with Paths.
- Place a ModelEntity object from the Project Library, into the Facility window. Select the entity in the Facility window and click on 'Add Additional Symbol' icon in the Ribbon. Change the color of the second symbol to Red. First, ensure that the second symbol is displayed by selecting the entity in the Facility window and checking the Active Symbol icon in the Ribbon. If (2 of 2) is displayed, you are viewing the second symbol. Click on the Color drop down in the Ribbon, find Red, and then click back onto the Entity object to change its color. (if you are viewing 1 of 2, or the first symbol, simply select the second symbol from the Active Symbol drop down)

Creating the Object Reference State on the ModelEntity

- Click onto ModelEntity in the Navigation window in the upper right side of the interface. Once ModelEntity is selected, you are now viewing the object definition for this object. Go to the Definitions window of the ModelEntity object.
- Go to the States Panel by selecting States on the left panel.
- Click on the Object Reference icon in the ribbon to create a new Object Reference State. Rename this new state, FirstServer.

Adding Logic to the Model

- Back in the main Model, click on Server1 and expand the State Assignments property category. Add a new assignment in the On Entering property. Click Add to create a new Assignment once the Repeating Property Editor window is open. Set the State Variable Name to 'ModelEntity.FirstServer'. Set the New Value property to 'ModelEntity.Location.Parent'.
 - The function ModelEntity.Location.Parent will return a reference to the object where the entity is current located (it's ParentObject) and at this moment, the ParentObject is Server1, so it will assign a reference of

Server1 to the new state named FirstServer on each entity that passed through.

- Click on Server2 and add the exact same assignment to this Server. (Exact same State Variable Name and same expression for New Value). However, before you exit the Repeat Group editor, add a second Assignment. Set the State Variable Name to 'ModelEntity.Picture' and the New Value to '1'.
 - This will change the color of all entities leaving Server2 to Red. This will help with visually confirming that these entities will be routed to Server6.
- Select the Path that leads from Server4 to Server5. Set the Selection Weight property to 'ModelEntity.FirstServer == Server1'. This tells Simio that only entities that have their FirstServer state set to 'Server1' will be able to take this Path.
- Similarly, select the Path that leads from Server4 to Server6. Set the Selection Weight property to 'ModelEntity.FirstServer == Server2'. This tells Simio that only entities that have their FirstServer state set to 'Server2' will be able to take this Path.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

OneQueueForMultipleServers - SimBit

Problem:

I have multiple servers and I would like one queue that feeds into these two servers.

Categories:

Buffering

Key Concepts:

Allow Passing, Blocked Destination Rule, Input Buffer, Lists, NodeList, Path, Select Available Only, Selection Goal, TransferNode

Assumptions:

If there are entities waiting and one of the servers becomes available, the first entity in the queue will move to the available server for processing.

Technical Approach:

There are two servers, but the same concept can be applied if there were more than two servers. The *InputBuffer* property on each server is set to '0'. This will ensure that entities do not wait at each server. A Transfer Node is placed in front of the two servers. The Transfer Node is configured to only send entities on paths that are available, which ensures that the entities will wait here if there are no servers available. The Path that leads to the Transfer Node has its *Allow Passing* property set to 'False' so that the entities wait one behind the other on this link.

Details for Building the Model:

Simple System Setup

- Place a Source object and a Sink object in the Facility window. In between, place two Servers that are in parallel to each other. Place a Transfer Node in front of these two Servers.
- Connect the Source to the Transfer Node with a Path. Set the path's *Allow Passing* property to 'False'.
- Connect the Transfer Node to the input node of each server with paths.
- Connect the output nodes of each server to the Sink.

Create a Node List

- Go to the Definitions window and open the Lists panel. Create a new Node List by clicking on the Node icon in the ribbon. In the node list, first list Input@Server1, followed by Input@Server2.

Ensuring One Queue

- Click on the Transfer Node. Set the *Entity Destination Type* property to 'Select From List'.
- Set the *Node List Name* property to the name of your new Node List.
- The *Selection Goal* property is set to 'Random' in this example.
- The *Blocked Destination Rule* is set to 'Select Available Only'. This ensures that the entity does not leave this Transfer Node and travel to its destination if there are entities at each server.

Eliminate Input Buffers

- Eliminate the Input Buffer at each server so that the entities cannot wait in this location for server capacity. Set the *Input Buffer* property on each server to 0.

Embellishment:

Experiment with more servers and see the behavior change when you change the *Blocked Destination Rule* property on the Transfer Node.

OverflowWIP - SimBit

Problem:

I have parts that are processed by two machines, always in FIFO order. I have a small WIP area in front of the machines – parts always go here to wait if space is available. If the small WIP area is full, parts go to an overflow area, but are still pulled in FIFO order regardless of location.

Categories:

Buffering

Key Concepts:

Capacity.Allocated, Lists, ModelEntity, ObjectList, Path, Ranking Expression, Ranking Rule, Resource, Secondary Resources, Selection Weight, Smallest Value First, TimeCreated

Technical Approach:

Represent the WIP areas as Servers with the specified capacity, but no processing time. Don't leave the Server until the processing machine is ready to process it. Use resource default behavior to select FIFO from all waiting parts (regardless of location).

Details for Building the Model:

Simple System Setup

- Add a Source, 4 Servers, and a Sink to the Facility Window. Two of the Servers will be the MainWIP and OverflowWIP areas, while the other two Servers will be ServerA and ServerB.

Determining the WIP Area to go to

- In the *Selection Weight* property for the path to MainWIP, use the 'MainWIP.Capacity.Allocated < MainWIP.Capacity'. This will evaluate the number of busy "spots" in this Server as compared to the capacity.
- In the *Selection Weight* property for the path to OverflowWIP, use the 'MainWIP.Capacity.Allocated == MainWIP.Capacity'.

Defining the Machines

- Place resource with *Name* 'ResourceA' and 'ResourceB' representing each machine.
- Open the Definitions tab, Lists panel and add an Object list with Name 'Machines'.
- Add 'ResourceA' and 'ResourceB' to the list.

Details for Machine Selection

- Within each WIP area (MainWIP and OverflowWIP), within the Secondary Resources / Other Resource Seizes section of properties, enter the *After Processing* repeating editor and change the *Object Type* to 'FromList', and the *Object List Name* to 'Machines'. The *Selection Goal* should remain as the default 'PreferredOrder' to select the first available machine from the list. These are seized just before the WIP resource is released.
- Within each Server (ServerA and ServerB), within the Secondary Resources / Other Resource Releases section of properties, enter the *After Processing* repeating editor and change the *Object Type* to 'FromList', and the *Object List Name* to 'Machine'.

Details for Moving to the Selected Machine

- Using paths, connect each of the WIP type Servers to each of the processing type Servers, ServerA and ServerB.
- In the *Selection Weight* property for each path to MachineA, use the expression 'ResourceA.Capacity.UnitsOwned'.
- In the *Selection Weight* property for each path to MachineB, use the expression 'ResourceB.Capacity.UnitsOwned'.
- The combination of the above will result in a weight of 1 for the correct path and a weight of 0 for the other path.

Animation Embellishment:

To make it easier to see the FIFO behavior, we used different entity symbols based on creation time. Look in the *Current Symbol* property for PartA and to see the expression used to use a different symbol each hour.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

PathPlannerElement - SimBit

Path Planner Element

This SimBit project includes five models providing examples of using the Path Planner Element.

1. **OneRoute**: Demonstrates entities using a Path Planner to reserve a route.
2. **MultipleRoutes_NetworkLinkType**: Demonstrates entities using a Path Planner to reserve the shortest route with multiple routes available.
3. **MultipleRoutes_NetworkLinkCurrentDirections_Replan**: Demonstrates entities using a Path Planner to reserve the shortest route based on current link traffic flow with multiple routes available.
4. **MultipleRoutes_NetworkLinkCurrentDirections_NoReplan**: Demonstrates entities using a Path Planner to reserve the shortest route based on current link traffic with multiple routes available, but only replanning the path if there is no current planned path.
5. **Spur**: Demonstrates entities that must go to a destination with only one path in and out.

Model 1: OneRoute

Problem:

Categories:

Keywords:

Assumptions:

Technical Approach:

Details for Building the Model:

Facility Window: Objects and Properties

- See the model for approximate placement of all objects.
- Place a Source. Approximately 15 meters to the left and 6 meters down, place a second Source. Rotate the second Source 180 degrees by holding the ctrl key and clicking and dragging the green handles on the object.
- Place 2 ModelEntity instances in the Facility next to the Sources.
- Color the second ModelEntity instance by using the Color button in the Symbols ribbon.
- Place a Sink approximately 15 meters to the right from the first Source. Place a second Sink approximately 15 meters to the left from the second Source. Rotate the second Sink 180 degrees by holding the ctrl key and clicking and dragging the green handles on the object.
- Place a BasicNode approximately 3 meters to the right and halfway between Source1 and Sink2 Nodes. Place another BasicNode approximately 3 meters to the left and halfway between Sink1 and Source2 Nodes.
- Create Paths to connect the following Nodes:
 - Output@Source1 to BasicNode1
 - BasicNode2 to Input@Sink1
 - Output@Source2 to BasicNode2
 - BasicNode1 to Input@Sink2

- On all the Paths, change the *Allow Passing* property to 'False'.
- Create a TimePath that connects BasicNode1 and BasicNode2. Change the following properties:
- *Type* = 'Bidirectional'
- *Travel Time* = '0.3'

Definitions Window: Elements

- Create a PathPlanner Element.

Facility Window: Properties

- On Source1's Output Node, update the following properties:
- Routing Logic
- *Entity Destination* = 'Specific'
- *Node Name* = 'Input@Sink1'
- *Path Planner Name* = 'PathPlanner1'
- On Source2's Output Node, update the following properties:
- Routing Logic
- *Entity Destination* = 'Specific'
- *Node Name* = 'Input@Sink2'
- *Path Planner Name* = 'PathPlanner1'

Facility Window: Animation

- From the Animation ribbon, use the Queue button to draw a Queue. Click in the Facility to place the first point. Move your mouse and click again to add another vertex. Right-click to stop drawing the queue.
- Change the *Queue State* property to 'PathPlanner1.RequestQueue'.

With a specific Node assigned, entities will attempt to plan a path to get to their destination. The Path Planner plans at a Node and will check for a route from the current location to the destination. Entities may have to wait until a path is available for a planned route to be reserved. In this model, the bidirectional link's traffic direction will change, making it unavailable at times to each entity type. Entity requests will queue with the Path Planner's Route Request Queue. The Path Planner is used to oversee the ranking of route requests over all entities routing since the entities are sharing the same system of links and nodes.

With the Path Planner, the whole route, from the entity's current location to its destination, must be available before the entity progresses. The entity will not only travel part of the way, as doing so may block other routes from being available to other entities.

While running, it may appear that the Route Requests in the queue are being skipped. Batches of entities will travel down the bidirectional path. The Route Requests are being checked according to the Path Planner's *Request Ranking Rule*. The ranking rule is static, and the value is only evaluated one time as soon as the entity enters the request queue. The route checking will always start with the beginning of the queue, but if the first request is not currently feasible because links are not available, the next request will be checked. If the subsequent request is not feasible, the next in the queue is checked, and so on. When a request is feasible, the route is approved, and the entity will complete its planned path. So, it may appear that entities are skipping ahead because the bidirectional link's traffic flow is matching its desired direction, thus making the link available for that entity's route request.

Notes:

Try removing the Path Planner Element from the Nodes to see how the model would act without it.

Consider changing the Path Planner's *Request Ranking Rule* and see how the request ranking changes in the Path Planner's Request Queue.

Model 2: MultipleRoutes_NetworkLinkType

Problem:

Entities must travel in opposing traffic directions. With multiple paths available, entities should choose the absolute shortest route regardless of availability.

Categories:

Element, Path

Keywords:

Element, Path Planner, Network

Assumptions:

Multiple routes are available for an entity to travel to their destination. The entities should choose the shortest path based on all links available.

Technical Approach:

A Path Planner Element will reserve the routes and queue route requests to organize when entities can use the links. The entity's Network properties will determine the shortest path to take.

Details for Building the Model:

Initial Setup

- Use the "OneRoute" model as a starting point. You may right-click the model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Definitions Window: Elements

- Add a Network Element. Change the *Links Type* to 'All'.

Facility Window: Objects and Properties

- Add two BasicNodes, approximately 3 meters above the other two BasicNodes.
- Create Paths between the following Nodes:
 - BasicNode1 to BasicNode3
 - BasicNode3 to BasicNode4
 - BasicNode4 to BasicNode2
- On the newly created Paths, change the following properties:
 - *Type* = 'Bidirectional'
 - *Drawn To Scale* = 'False'
 - *Logical Length* = '50'
- On the Path between BasicNode3 and BasicNode4, change the *Logical Length* to '100'.
- On Source1 and Source2, change the *Maximum Arrivals* to '1'.
- On the DefaultEntity and ModelEntity1 instances, change the *Initial Network* to 'Network1'.

Discussion:

When finding the entity's shortest path, the entity's Network properties determine which links are viable. In this case, the Network is using the *Shortest Paths Graph Type* property 'LinkTypes' which specifies that any link can be chosen for the shortest path if the path is drawn so that the entity can travel in that direction. In other words, the overall shortest route, regardless of availability will be chosen. The bidirectional link in the middle, connected to BasicNode1 and BasicNode2, remains the shortest route over all for travel in either direction. This is still a viable path for the red entity, even though the link is not initially available because its traffic direction opposes the red entity's required direction. Like the first model, the entities will wait, and their requests will queue in the Path Planner's Request Queue until all the links in their route become

available.

Notes:

Remember to assign the entity's Network as the custom Network so the custom Network's properties can be used.

This model limits the number of entities created so the behaviors are easily observable.

Model 3: MultipleRoutes_NetworkLinkCurrentDirections_Replan

Problem:

Entities must travel in opposing traffic directions. With multiple paths available, entities should choose the shortest route based on current traffic direction. Entities should replan their path if the current traffic direction changes and the shortest route has changed.

Categories:

Element, Path

Keywords:

Element, Path Planner, Network

Assumptions:

Multiple routes are available for an entity to travel to their destination. The entities should choose the shortest path available based on current path traffic directions. Entities will replan their path at a Node.

Technical Approach:

A Path Planner Element will reserve the routes and queue route requests to organize when entities can use the links. The Network referenced by the entity will determine the shortest path to take.

Details for Building the Model:

Initial Setup

- Use the "MultipleRoutes_NetworkLinkType" model as a starting point. You may right-click the model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Facility Window: Properties

- On BasicNode3 and BasicNode4, change *Path Planner Name* property to 'PathPlanner1'.

Definitions Window: Elements

- Change the following Network1 properties:
- *Shortest Paths Graph Type* = 'LinkCurrentDirections'
- *Auto Update Shortest Paths* = 'True'

Discussion:

When finding the entity's shortest path, the entity's Network properties determine the viable links that can be chosen. In this case, the Network is using the *Shortest Paths Graph Type* property 'LinkCurrentDirections' which specifies that the link's *Current Direction* needs to match the entity's direction of travel to consider it for the shortest path. Since the green entity plans its route first and changes the middle shortest bidirectional path to match its traffic direction, the red entity Network planning does not consider the middle bidirectional path as a feasible link. The red entity's Network finds the only other feasible route and sends the entity toward the longer path.

On the longer path, the Path Planner Element was placed on the Basic Nodes. When the red entity reaches these Nodes, the Path Planner replans the path to the entity's destination. Since the green entity progressed past that shorter middle bidirectional link, that link's Current Direction can now match the red entity's traffic direction. Backtracking to use the middle link would be a shorter route than continuing with the previously planned route, so the red entity will turn around and follow the new planned path.

When an entity reaches a Node with a Path Planner, the path will be replanned unless the *Path Planner's Path Plan If* condition resolves to 'False'. When a replan occurs, the current route will be unreserved. The entity is not guaranteed to reserve a better or the same route as it might lose out to a higher ranked request in the queue and will have to wait.

A Path Planner Element can only be used with a Transfer step and therefore is commonly used at a Node. A new path cannot be planned while an entity is traversing a link.

Notes:

Remember to assign the entity's Network as the custom Network so the custom Network's properties can be used.

This model limits the number of entities created so the behaviors are easily observable.

Model 4: MultipleRoutes_NetworkLinkCurrentDirections_NoReplan

Problem:

Entities must travel in opposing traffic directions. With multiple paths available, entities should choose the shortest route based on current traffic direction. Entities should not replan their path if they already have a planned path.

Categories:

Element, Path

Keywords:

Element, Path Planner, Network

Assumptions:

Multiple routes are available for an entity to travel to their destination. The entities should choose the shortest path available based on current path traffic directions. Entities should not deviate from their planned path once it has been reserved.

Technical Approach:

A Path Planner Element will reserve the routes and queue route requests to organize when entities can use the links. The entity's Network properties will determine the shortest path to take. The Path Planner will use a condition to reject plan requests if the entity already has already reserved a planned path.

Details for Building the Model:

Initial Setup

- Use the "MultipleRoutes_NetworkLinkCurrentDirections_Replan" model as a starting point. You may right-click the model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

Definitions Window: Elements

- Change the Path Planner's *Plan Path If* property to 'Entity.PlannedPath.Links.NumberItems == 0'.

Discussion:

The only difference between this model and "MultipleRoutes_NetworkLinkCurrentDirections_Replan" is the Path Planner's *Plan Path If* expression. Previously, the expression was always 'True', so any time the Path Planner Element was called, it would accept the request and replan the path. Note, even if the Path Planner is called to replan a route, the same original route could still be chosen, so no difference may be visible perceptible.

In this model, the new *Plan Path If* condition will look at the Entity's list of links in its Planned Path. If there is any number of links in this list, it means the entity is actively routing with a Planned Path. This is an expression that could be used to ignore a Path Planner at a Node if the entity should continue with its previously assigned route. If the NumberItems is zero, then the entity does not have an active Planned Path and the Path Planner would accept the plan request.

When run, when the red entity gets to the second Path Planner at BasicNode4, the Path Planner's *Plan Path If* condition returns false. Even though the state of the links in the Network and the shortest available path has changed, the Path Planner request is rejected, and the entity continues traversing its previously assigned path.

Notes:

Remember to assign the entity's Network as the custom Network so the custom Network's properties can be used.

This model limits the number of entities created so the behaviors are easily observable.

Model 5: Spur

Problem:

Entities need to travel to a location where there is only one link to route in and out on. Entities should not get caught in a deadlock on this spur.

Categories:

Element, Path

Keywords:

Element, Path Planner, Network

Assumptions:

Two entities will arrive close together and attempt to travel to a node with only one connected link.

Technical Approach:

A Path Planner Element will plan the path to the spur point. At the spur point, the entities must turn around on the same path and plan a path to the Sink.

Details for Building the Model:

Facility Window: Objects and Properties

- Add a Source and update these properties:
- *Interarrival Time* = 'Random.Exponential(.05)'
- *Maximum Arrivals* = '2'
- Place a Sink approximately 12 meters away.
- Place a BasicNode halfway between the Output@Source1 and the Input@Sink1.
- Place a TransferNode approximately 5 meters below BasicNode1.
- Create Paths between the following Nodes:
- Output@Source1 to BasicNode1
- BasicNode1 to TransferNode1
- BasicNode1 to Input@Sink1
- On the Path between BasicNode1 and TransferNode1, change the *Type* property to 'Bidirectional'.

Definitions Window: Elements

- Add a Path Planner Element.

Facility Window: Properties

- On Output@Source1 change the following properties:
- Routing Logic
- *Entity Destination Type* = 'Specific'
- *Node Name* = 'TransferNode1'
- *Path Planner Name* = 'PathPlanner1'
- On TransferNode1 change the following properties:
- Routing Logic
- *Entity Destination Type* = 'Specific'
- *Node Name* = 'Input@Sink1'
- *Path Planner Name* = 'PathPlanner1'

Discussion:

Having a spur turn around point might be common when modeling vehicle behavior. Transporters may need to enter on one link and turn around to leave on the same link. This could be seen in warehousing systems where a vehicle is dropping off an entity at a rack and there is only one path to access the rack.

In this case, the first planned path is only accounting for route to the spur end point at TransferNode1. When both entities plan their path, their direction of travel on this spur path will match so the route appears available to both entities. It is not till the entities reach TransferNode1 that their new planned path is unable to be completed because of the traffic flow on the spur path.

The entity must turn around on the same link to enter and exit. When there is more than one entity on this link, the second entity is in the way of the first entity turning around since the bidirectional path can only have traffic flow in one direction at a time. Both entities are unable to leave the path to turn around at the same time.

One option to address this behavior is to change the *Traveler Capacity* of the link to '1'. This will limit the number of entities that can be on the link at one time. With only one entity on the link, the entity can turn around at the end point without running into another entity.

Instead of changing the capacity of the link, custom logic could be built so that entities that reach the dead-end point will be forced to exit the link. This could be achieved with logic that Parks the entity or sends the entity to another Station. Keep in mind that entities can physically exist in 3 places: free-space, links, and Stations. You can prevent a spur deadlock by giving the entity another location to move to, that is not the same link. With this approach, additional logic will be needed to decide when to move the entity back onto the link.

Notes:

This model limits the number of entities created so that certain behavior can be easily seen.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

PathSelectionRule - SimBit

Problem:

I have a bidirectional path that takes entities from two different inputs to two different outputs, with entities coming from both sides of the path. I want to be able to direct how the entities flow onto the path, but I'm not sure how the *Traffic Direction Rule* property on a Path works.

Categories:

Decision Logic -- Paths

Key Concepts:

Allow Passing, Bidirectional Path, Entity Destination Type, Maximum Arrivals, Path, Prefer Desired Direction, Source, Time Offset, Traffic Direction Rule

Assumptions:

There is no passing on the bidirectional path. Entities move onto the link from both node entrances. The systems shown are identical except for the *Traffic Direction Rule* property on the bi-directional Path.

Technical Approach:

A model is developed with a system containing two sources, two sinks and a bidirectional link. This first group of objects is then copied to a second system. To evaluate the difference between the systems, the model will be deterministic, meaning no random distributions will be used. The only difference will be the *Traffic Direction Rule* property on the bidirectional path.

Details for Building the Model:

Simple System Setup

- Place a Source (Source1) and Sink (Sink1) at the left side of the Facility window. Then, place a second Source (Source2) and Sink (Sink2) on the right side.
- Within Source1, change the *Interarrival Time* to '.2' and the under the Stopping Conditions category, change *Maximum Arrivals* to '5'. Within Source2, change the *Time Offset* to '0.1', the *Interarrival Time* to .2 and the *Maximum Arrivals* to '5'.

Setting the Destinations

- Click on the output node of Source1 (Output@Source1) and change the *Entity Destination Type* to 'Specific' and the *Node Name* to 'Input@Sink2'.
- Do the same for Source2 (Output@Source2) and specify the *Entity Destination Type* as 'Specific' and the *Node Name* as 'Input@Sink1'.

Adding Nodes, Paths and a Bidirectional Path

- Place a BasicNode1 to the right of Source1/Sink1. Place a second BasicNode2 to the left of Source2/Sink2.
- Using Paths, connect Source1 to BasicNode1 and Source2 to BasicNode2. Then connect BasicNode1 to Sink1 and BasicNode2 to Sink2. For the Paths connecting the Sources to the BasicNodes, set Allow Passing to 'False'.
- Add a path connecting BasicNode1 and BasicNode2. Change the path's Type to 'Bidirectional', Allow Passing to 'False' and Speed Limit to '.1'.

Copying System1 to System2

- Now that we have a single system completed, we will copy it to make an identical one. This can be done by moving the mouse to the top left corner above the Source1 and Ctrl-Left Click and drag to highlight all the objects in system1 (including both Sources, Sinks, BasicNodes and all paths).
- Ctrl-C will copy the system and position the mouse below system1 and use Ctrl-V to paste the copy. All objects will be renamed with unique names.
- Edit the bidirectional path for System2 and change the Traffic Direction Rule to 'Prefer Desired Direction'. You may need to expand the properties under Type to see this property.
- Change the entity destination Node Name properties in the Source Output Nodes to 'Input@Sink1_1' and

'Input@Sink2_1'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

PeriodicStatistics - SimBit

This SimBit project includes three models providing use cases of and illustrations of using the Periodic Statistic Element.

1. PeriodicStats_TIS24HourCycle: Collect an hourly periodic statistic for entity time in system
2. PeriodicStats_ResourceState3ShiftDay: Collect periodic statistics on resource state over three-shift days.
3. PeriodicStats_ServerUtilization: Collect periodic statistics on daily average Scheduled Utilization over a week.

Model 1: PeriodicStats TIS24HourCycle

Problem:

I want to capture entity time in system as an hourly periodic statistic on a 24-hour cycle for my system.

Categories:

Periodic Statistics

Keywords:

Statistics, Tally Statistic, Periodic Tally Statistic, Results, Periodic Statistic

Assumptions:

Repeating, daily pattern of 1-hour periods; average of same hour on different days is meaningful. Model is run for 48 hours, therefore, there will be two observations of each period.

Technical Approach:

Using a Periodic Statistic Element in conjunction with an already-existing Tally Statistic Element (Sink.TimeInSystem) to capture the average entity time in system (TIS) per hour of a day.

Details for Building the Model:

Run Parameters

In the Run ribbon > (Run Setup Group), set *Ending Type* = Specific '48' Hours

Facility Window, Objects

Place a Source, Server, and Sink in the Facility window, each about 3.5 meters apart, as shown in the model

Creating Elements

In the Definitions window, Elements tab, create a Periodic Statistic with the name 'PeriodicStat_TimeInSystem'

- Set *Element Type* = 'TallyStatistic'
- Set *Tally Statistic Name* = 'Sink1.TimeInSystem'
- Open the repeating property editor for *Period Lengths* and create one row where *Number of Periods* = '24' and *Period Length* = '1' Hours
- Set *Period Name Prefix* = 'Hour'

Facility Window, Labels

As shown in the model, create labels to display:

- The current hour (period): 'PeriodicStat_TimeInSystem.PeriodNumber'
- The current period average time in system: 'Math.Round(PeriodicStat_TimeInSystem.Average, 4)'
- The average time in system across all periods of this number:

'Math.Round(PeriodicStat_TimeInSystem.OverallAverage(PeriodicStat_TimeInSystem.PeriodNumber), 4)'

- The total average time in system (across the entire run): 'Math.Round(Sink1.TimeInSystem.Average, 4)'
- The current simulation hour: 'Math.Round(TimeNow)'

Discussion:

Notice that the Current Period Average TIS and the Total Average TIS for the simulation run have the same value for the first hour of the simulation run, this is because for that period of time, the metrics are being tabulated with the same information; however, this is a special case that only occurs at the start of the run. Thereafter, it is expected that the metrics will report different values, otherwise it is a coincidence. This value, the same for the first hour of the simulation run of this model, for Current Period TIS and Total Average TIS will be expected to be observed one more time.

Because the Periodic Tally Statistic is repeating, Simio is going to synthesize across periods with the same number: day 1, hour 1 average will be averaged with day 2, hour 1 average, et cetera. This metric, shown in this model as Average TIS Across All Hour [n] Periods, is updated at the end of a period observation, hence it returns 'NaN' until the first observation of the specified period has ended, which in this model occurs for the first time for period 1, and subsequently the following, after 24 hours have passed.

The value of the first hour of the simulation run, both for Current Period Average TIS and Total Average TIS, is expected once more as the value of Average TIS Across All Hour 1 Periods (24 hour of simulation run), until the close of this period. When it is the second observation of Hour 1 (now day 2), the Average TIS Across All Hour 1 Periods is simply the value of the one, and only, observed period, until the period finishes, then the Average TIS Across All Hour 1 Periods is the average of the first and second observation averages. This average of averages is updated at the conclusion of the second observation of hour 1, but is not shown as modeled in the Facility window, but can be ascertained in the Results window > Pivot Grid.

Note:

The Periodic Statistic Element could have been defined in a subclassed Server, and the subclassed Server used instead.

Model 2: PeriodicStats ResourceState3ShiftDay

Problem:

I want to capture a periodic statistic of server resource states over an 8-hour period (3-shift day) and a 24-hour cycle.

Categories:

Periodic Statistics

Keywords:

Statistics, Output Statistic, Periodic Output Statistic, Results, Periodic Statistic, Resource State

Assumptions:

Repeating, 3-shift (8 hours) daily pattern; average (change) of same shift between subsequent days is meaningful. Model is run for 48 hours, therefore, there will be two observations of each period.

As defined by Simio, for a Server, the index of resource states are as follows: Starved (0); Processing (1); Off Shift (4).

Note:

This model has view shortcuts: Overview (o), Description (d), System (s), Metrics (m), and Breakpoints (b).

This model uses breakpoints to highlight changes in metrics. To run the model without breakpoints, set the model property *PauseModel* = 'False'.

Technical Approach:

Using a Periodic Statistic Element and Output Statistic Element pair for each Server's resource state to report average time in the specified resource state per shift.

Details for Building the Model:

Run Parameters

- In the Run ribbon > (Run Setup Group), set *Ending Type* = Specific '48' Hours
- In the Run ribbon > (Animation Speeds Group), set *Speed Factor* = '50'

Facility Window, Basic Set-Up

- Place a Source, Server, Server, and Sink in the Facility window, each about 3 meters apart, as shown in the model
- Place the DefaultEntity instance in the model
- Set *Maximum Number in System Limit* = 'Infinity'; because our system has two serial Servers, each with 8 hours of off shift time, we need to increase the soft limit of entities allowed in our system to forgo the warning message about too many entities
- Set the *Capacity Type* = 'WorkSchedule' and the *Initial Work Schedule* = 'StandardWeek' for both Server1 and Server2

Updating StandardWeek Work Schedule

- In the Data window, Work Schedules tab, Pattern Based window, Day Patterns section, expand the StandardDay by clicking on the plus sign
- Set two, and only two, Work Periods such that we have a shift that works from midnight to 8am (08:00) and another that works from 8am (08:00) to 4pm (16:00)
- 00:00 to 08:00, *Value* = '1'
- 08:00 to 16:00, *Value* = '1'
- Because there is no row for times 16:00 to 00:00, there will be no work, the Servers will be off shift

Creating Elements

In the Definitions window, Elements tab, an Output Statistic Element and a Periodic Statistic Element pair are created for each Server, for each resource state of interest; in this model: starved (0), processing (1), off shift (4).

Server1 statistic elements:

- Create an Output Statistic with the name 'OutputStat_Serv1_Starved'
- Set *Unit Type* = 'Time'
- Set *Expression* = 'Server1.ResourceState.TotalTime(0)'
- *Data Source* = 'Server 1'
- *Category* = 'ResourceState'
- *Data Item* = 'TimeStarved'
- Create two more Output Statistics with the names 'OutputStat_Serv1_Procassing' and 'OutputStat_Serv1_OffShift', substituting the appropriate index for the *Expression*, 1 and 4 respectively and the *Data Item* to 'TimeProcessing' and 'TimeOffShift', respectively
- Create a Periodic Statistic with the name 'OutputStat_Serv1_Starved'
- Maintain *Element Type* = 'OutputStatistic'
- Set *Output Statistic Name* = 'OutputStat_Serv1_Starved'
- Set *Anchor Date Type* = 'Specific'
- Set *Anchor Date Time* = '[any day] 00:00:00' (or '12:00:00 AM'); because the repeating pattern occurs daily, the anchor day is irrelevant, only the time matters in this case
- Open the repeating property editor for *Period Lengths* and create one row where *Number of Periods* = '3' and *Period Length* = '8' Hours
- Set *Period Name Prefix* = 'Shift'
- Create two more Periodic Statistics with the names 'PeriodicStat_Serv1_Processing' and 'PeriodicStat_Serv1_OffShift', substituting the appropriate *Output Statistic Name*, 'OutputStat_Serv1_Procassing' and 'OutputStat_Serv1_OffShift' respectively

Server2 statistic elements:

- Repeat the above steps for Server2, making sure to use the appropriate object, element, and index references

Facility Window, Animation

Server1, Shift1 panel:

- Create a (unattached) Status Pie to be used for the Server 1, Shift 1 panel
- In the Status Pie Appearance ribbon, set the *Title* = 'Server 1, Shift 1'
- Open the *Expression* repeating property editor and create three rows
- Row 1
- Set *Expression* = 'PeriodicStat_Serv1_Starved.OverallAverageValueChange(1)'
- Set *Label* = 'Srv1 Shift1 Starved'
- Row 2
- Set *Expression* = 'PeriodicStat_Serv1_Processing.OverallAverageValueChange(1)'
- Set *Label* = 'Srv1 Shift1 Processing'
- Row 3
- Set *Expression* = 'PeriodicStat_Serv1_OffShift.OverallAverageValueChange(1)'
- Set *Label* = 'Srv1 Shift1 Off Shift'
- Create a Floor Label to be used for the Server 1, Shift 1 panel
- Display the same information as in the Status Pie, but in text form; the same core expression will be used to display Server 1, Shift 1 average time idle, processing, and off shift as the Status Pie, with the addition of the Math.Round function to round to 4 decimal places
- For Server 1, Shift 1 Starved, the expression should be '{Math.Round(PeriodicStat_Serv1_Starved.OverallAverageValueChange(1),4)}'
- Repeat for Server 1, Shift 1 Processing and Off Shift

Repeat for Server 1, Shift 2 and Shift 3 and Server 2, Shifts 1-3, making appropriate substitutions for the Periodic Statistic Element and period number

Create an overall Status Pie for Server 1 using a similar approach as above, where the *Expression*, as shown here for total time starved (index 0) does not include the Periodic Statistic Element, 'Server1.ResourceState.TotalTime(0)'

Repeat the immediate above and create an overall Status Pie for Server 2, making appropriate substitutions for object references

Create a Status Label with the *Expression* = 'PeriodicStat_Serv1_Processing.PeriodName' to show the current period (note: any Periodic Statistic element could have been used as they all have the same pattern and anchor)

Additional Functionality

To create breakpoints in the model, and to have the ability to turn them off with a model property, create two Sources, define another instance of ModelEntity, and create a Boolean property. The Sources will be timing mechanisms which will pause the model at 8-hour and just-before-8-hour increments.

- In the Definitions window, Properties tab, create a Boolean property named 'PauseModel', where *Default Value* = 'True'; *Category Name* = 'Execution Type'; *Category Expanded* = 'True'
- In Facility window, place two Sources and another instance of Model Entity (ModelEntity1)
- Name the Sources 'breakpointControl1' and 'breakpointControl2'
- For both Sources, set *Entity Type* = 'ModelEntity1'

- For breakpointControl1, set *Time Offset* = '7.95' Hours; *Interarrival Time* = '7.95' Hours; *Maximum Arrivals* = 'PauseModel*Infinity'
- For breakpointControl1, set *Time Offset* = '8' Hours; *Interarrival Time* = '8' Hours; *Maximum Arrivals* = 'PauseModel*Infinity'
- Right-click on output@breakpointControl1 and select "Breakpoint" from the menu to set a breakpoint on the output node of the Source; then, select the node and set *Entity Destination Type* = 'None (Destroy Entity)'
- Repeat the immediate above for output@breakpointControl2 and breakpointControl2

Discussion:

A Periodic Output Statistic calculates the average and total change among similar periods, `Server.ResourceState.TotalTime([index])` is a good candidate for a Periodic Output Statistic because the expression is cumulative over the run.

As a repeating Periodic Output Statistic, Simio is going to synthesize across periods with the same number: the average and total change from day 1, shift 1 to day 2, shift 1, et cetera. This metric, shown in this model as `Srv[n] Shift [n] [Starved, Processing, Off Shift]`, is updated at the end of a period observation, hence it returns 'NaN' until the first observation of the specified period has ended, which in this model occurs for the first time for shift 1, and subsequently the following, after 8 hours have passed.

The breakpoints at intervals of 7.75 hours and 8 hours prime the model to highlight the value changes of the metrics. For example, when Fast Forward is pressed for the first time, the model is running the first shift, not until after this shift is completed is their an value for the `[PeriodicStatistic].OverallAverageValueChange([period number])`, it is not until this time that the shift Status Pie is updated.

Note:

Status Pies may retain previous values from a prior run before the next run, they will update during the next run, after the previous data has been overwritten. Because the `[PeriodicStatistic].OverallAverageValueChange([period number])` is not updated until after the first observation of the specified period, the shift Status Pies update after each period observation, every second Fast Forward press when *PauseModel* = 'True'.

The Output and Periodic Statistic Element pair could have been defined in a subclassed Server for a cleaner implementation, where the subclassed Server is used instead.

Model 3: PeriodicStats ServerUtilization

Problem:

I want to capture a daily periodic statistic of a Server's Scheduled Utilization, repeating weekly.

Categories:

Periodic Statistics

Keywords:

Statistics, Output Statistic, Periodic Output Statistic, Results, Periodic Statistic, Utilization

Assumptions:

Weekly pattern of days; average Scheduled Utilization for a resource is meaningful. Model is run for two weeks, therefore, there will be two observations of each period.

Technical Approach:

Using Periodic Output Statistic and Output Statistic Element pair to collect the cumulative Scheduled Utilization of a Server and report its daily average (change) Scheduled Utilization in a weekly pattern.

Note:

This model uses breakpoints to highlight changes in metrics. To run the model without breakpoints, set the model property *PauseModel* = 'False'.

Details for Building the Model:

Run Parameters

- In the Run ribbon > (Run Setup Group), set *Ending Type* = Specific '2' Weeks

Facility Window, Basic Set-Up

- Place a Source, Server, and Sink in the Facility window, each about 3.5 meters apart, as shown in the model
- Place the DefaultEntity instance in the model

Creating Elements

- In the Definitions window, Elements tab, create an Output Statistic named 'OutputStat_Serv1_CumScheduledUtilization'; where the cumulative utilization where be recorded
- Set *Expression* = '(Server1.Capacity.ScheduledUtilization * TimeNow) / (PeriodicStat_Serv1_DailyScheduledUtilization.PeriodEndTime - PeriodicStat_Serv1_DailyScheduledUtilization.PeriodStartTime)'
- Where the denominator is the period length, in this case it is a constant and could be replaced with that constant term (i.e. 24); this implementation fails in robustness compared to the as-implemented
- Set *Data Source* = 'Server 1'
- Set *Category* = 'Capacity'
- Set *Data Item* = 'ScheduledUtilization'
- Leaving this field blank would default to a value of OutputValueChange in the Pivot Grid; however, the expression used in this Output Statistic is designed with the Periodic Output Statistic in mind, taking the value difference from period end to start of the cumulative expression, thereby resulting in the (average and total) ScheduledUtilization for the period, across period observations; where average (change) is meaningful for this context
- Set *Report Statistics* (General group) = 'False'
- The run sum of each day's ScheduledUtilization is likely erroneous, therefore it will not be reported (much like for each day, the average ScheduledUtilization is likely more meaningful than the total (across multiple period observations) for a day
- Create a Periodic Statistic named 'PeriodicStat_Serv1_DailyScheduledUtilization'
- Maintain *Element Type* = 'OutputStatistic'
- Set *Output Statistic Name* = 'OutputStat_Serv1_CumScheduledUtilization'
- Set *Anchor Date Type* = 'Specific'
- Set *Anchor Date Time* = '[any Sunday] 00:00:00' (or '12:00:00 AM'); because the repeating pattern occurs weekly, the specific Sunday does not matter, but for this model it is desirable for Day 1 of the pattern to be Sunday midnight (i.e. start of Sunday)
- Open the repeating property editor for *Period Lengths* and create one row where *Number of Periods* = '7' and *Period Length* = '24' Hours
- Set *Period Name Prefix* = 'Day'

Facility Window, Animation

- Add a Floor Label to show the daily average Scheduled Utilization of Server 1 for each day, where the expression for day 1 is '{Math.Round(PeriodicStat_Serv1_DailyScheduledUtilization.OverallAverageValueChange(1),4)}'
- Add a Floor Label to show the cumulative Scheduled Utilization for Server 1, where the expression is nearly the same as the Output Statistic Element expression: 'Math.Round((Server1.Capacity.ScheduledUtilization * TimeNow) / (PeriodicStat_Serv1_DailyScheduledUtilization.PeriodEndTime - PeriodicStat_Serv1_DailyScheduledUtilization.PeriodStartTime),4)'

Organizing Pivot Grid

- To possibly increase ease of use and reading, swap (by drag-and-drop) the position of the Statistic and Time Period columns in the Pivot Grid, thereby grouping all of the average daily ScheduledUtilizations

Additional Functionality

To create breakpoints in the model, and to have the ability to turn them off with a model property, create one Sources, define another instance of ModelEntity, and create a Boolean property. The Source will be the timing mechanism which will pause the model at 24-hour increments.

- In the Definitions window, Properties tab, create a Boolean property named 'PauseModel', where *Default Value* = 'True'; *Category Name* = 'Execution Type'; *Category Expanded* = 'True'
- In Facility window, place one Source and another instance of Model Entity (ModelEntity1)
- Name the Source 'breakpointControl1'
- Set *Entity Type* = 'ModelEntity1'
- For breakpointControl1, set *Time Offset* = '24' Hours; *Interarrival Time* = '24' Hours; *Maximum Arrivals* = 'PauseModel*Infinity'
- Right-click on output@breakpointControl1 and select "Breakpoint" from the menu to set a breakpoint on the output node of the Source; then, select the node and set *Entity Destination Type* = 'None (Destroy Entity)'

Discussion:

See Help for assistance with time-weighted Periodic Statistics.

Note:

The Periodic Statistic Element could have been defined in a subclassed Server, and the subclassed Server used instead.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

PickUpDropOffFlow - SimBit

Problem:

I would like to model flow being carried on a moving entity. The moving entity's Container would be filled with flow. It would then travel to a new destination and then empty the flow out of its Container.

Categories:

Add-On Process Logic, Flow Library

Key Concepts:

Assign Step, Condition, Contents, Custom Object, Decide Step, Event, Execute Step, Fire Step, FlowNode, ModelEntity, Notify Step, Remove Step, Transfer Step, Wait Step

Assumptions:

The model described below will support two entities arriving from the Source object. If this number is altered, the tank capacity and volume will need to be adjusted to support this.

Technical Approach:

There is a Tank that initially contains 600 cubic meters of flow. There is a second Tank that is initially empty but can hold up to 600 cubic meters. The ModelEntity object has a Container element that will hold 300 cubic meters of flow. The traveling entity arrives at a Node and flow is transferring from the first Tank into the Container on the entity. Flow is transferred until the entity's Container is full. At this point, the transfer of flow is stopped and the entity travels to the next Node. At this point, flow is transferred from the entity's Container into the second Tank. When the entity's Container is empty, the entity leaves the Node and travels to the Sink.

Details for Building the Model:

System Setup

- Drag out two instances of ModelEntity, so there is an instance of DefaultEntity and ModelEntity1 in the Facility window. DefaultEntity will be the traveling entity and ModelEntity1 will be the flow.
- Place a Source (Source1) and Sink (Sink1) at the left and right sides, respectively, of the Facility window.
 - Set the *Maximum Arrivals* property on the Source to '2'.
- Place two TransferNodes in between the Source and the Sink and connect each objects with a Path.
 - Click onto the path that connects the Source to the first TransferNode. Set the *Initial Traveler Capacity* to '1' so that only one traveling entity will be on this link at the same time. This is required so that the second entity does not arrive at the first TransferNode and request a transfer of flow while the first entity is still accepting a transfer of flow.
- Place two Tank objects (from the Flow Library). Place one Tank near the first TransferNode and the other Tank near the second TransferNode.
 - For Tank1, set the *Initial Volume Capacity* property to '600 cubic meters'. Add a row to the Initial Contents repeat group property and set the *Entity Type* property to 'ModelEntity1' and the *Quantity* to '600', which indicates that this Tank will initially have 600 cubic meters of ModelEntity1 inside of it.
 - For Tank2, set the *Initial Volume Capacity* property to '600 cubic meters'.
- In the Run Ribbon, set the *Speed Factor* to at least 300 for a more appropriate animation speed.

Adding a Container to the ModelEntity object

- Click on the ModelEntity object in the Navigation window (right side of interface) to enter into this object's definition.
- Go to the ModelEntity's Definitions window and click onto "Container" in the Elements ribbon.
 - Set the Container's *Initial Volume Capacity* to '300' cubic meters.
 - Find the *On Full Process* and *On Empty Process* properties of the Container and select "Create New" for each property so that you have created two new processes that will be triggered when this Container is Full and Empty.

- From within the ModelEntity's Definitions window, go to the Events panel (find Events along the left hand side of the interface). Create two new Events by clicking on "Event" in the Events ribbon. Name one event 'Full' and the other 'Empty'.
- Go to the ModelEntity's Processes window. Find the two new processes that were created earlier from the Container element.
 - Place a Fire Step in the Container1_OnEmptyProcess. Set the *Event Name* of this step to 'Empty'.
 - Place a Fire Step in the Container1_OnFullProcess. Set the *Event Name* of this step to 'Full'.

Process Logic to Transfer Flow

- Navigate back to the main model by clicking onto Model in the Navigation window. Go to the Model's Processes window. Create 4 new Processes by clicking on Create Process in the Process ribbon.
 - Click onto the first process and set its *Name* property to 'FillShip'.
 - In this process, we'll be transferring flow from Tank1 into the entity's Container. First, make sure there is sufficient flow in Tank1 to transfer. Place a Decide Step that checks the condition "Tank1.FlowContainer.Contents > 0".
 - If False, place a Notify Step that tells the user there isn't sufficient flow.
 - If True, place a Transfer Step to transfer the flow.
 - Set *From* to 'CurrentNode'
 - Set *To* to 'Container'
 - Set *Container Name* to 'ModelEntity.Container1'
 - Set *Flow Regulator Name* to 'Output@Tank1.FlowRegulator'
 - Under Advanced Options, set *Entity Type* to 'SpecificObject' and *Entity Object* to 'Tank1.FlowContainer.Contents.FirstItem.Modelentity'. This is telling Simio that you specifically want to find the entity that is currently in the Contents queue of the FlowContainer inside of Tank1 and transfer that entity into the current entity's Container1
 - Click onto the second process and set its *Name* property to 'TransferFromTank1ToShip'.
 - Place an Execute Step with the *Process Name* property set to 'FillShip'. In the Advanced Options, set the *Token Wait Action* to 'None (Continue)'. This will tell Simio to execute the FillShip process, but this token will not wait for that process to finish, it will continue onto the next step.
 - Place a Wait Step and set the *Event Name* to 'ModelEntity.Full'.
 - Place a Decide Step and set the *Expression* to 'Output@Tank1.FlowRegulator.FlowRequestQueue.NumberWaiting>0', which checks if there is still an entity in the Flow Request Queue of the Tank's output node. An entity would still be in this queue if only a partial amount of the flow was transferred out of the tank. We need to remove the entity from this queue because we are done with this transfer of flow. This is how we "cancel" a flow transfer request. Simio will keep on trying to transfer the rest of the flow volume to this entity, unless you remove it from the Flow Request Queue when you are done. If we had transferred all of the volume that was in the Tank, this step would not be necessary.
 - Click onto the third process and set its *Name* property to 'EmptyShip'.
 - In this process, we'll be transferring flow from the entity's Container into Tank2.
 - Place a Transfer Step to transfer the flow.
 - Set *From* to 'CurrentContainer'
 - Set *To* to 'Container'
 - Set *Container Name* to 'Tank2.FlowContainer'
 - Set *Flow Regulator Name* to 'Input@Tank2.FlowRegulator'
 - Under Advanced Options, set *Entity Type* to 'SpecificObject' and *Entity Object* to 'ModelEntity.Container1.Contents.FirstItem.Modelentity'. This is telling Simio that you specifically want to find the entity that is currently in the Contents queue of the entity's Container and transfer that entity into the FlowContainer of Tank2.
 - Click onto the fourth process and set its *Name* property to 'TransferFromShipToTank2'.
 - Place an Execute Step with the *Process Name* property set to 'EmptyShip'. In the Advanced Options, set the *Token Wait Action* to 'None (Continue)'. This will tell Simio to execute the EmptyShip process, but this token will not wait for that process to finish, it will continue onto the next step.
 - Place a Wait Step and set the *Event Name* to 'ModelEntity.Empty'.
- Navigate back to the Facility window of the model and click onto TransferNode1. In the *Entered Add On Process* trigger property, select "TransferFromTank1ToShip" in the dropdown so that this process is executed when the entity enters this node. Similarly, click onto TransferNode2. In the *Entered Add On Process* trigger property, select "TransferFromShipToTank2" in the dropdown so that this process is executed when the entity enters this node.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ProcessModelWithinModel - SimBit

Problem:

There is a series of simple steps in your facility that are repeated many times throughout the operation. You wish to have a small user-defined "model object" that can be placed multiple times in your model logic.

Categories:

Building New Objects / Hierarchy

Key Concepts:

BasicNode, ContinueProcess, Delay Step, EndTransfer Step, Expression Property, ExternalNode, Object Instance Property, Station, Release Step, Resource, Seize Step, Station Element, Transfer Step

Assumptions:

The similar steps that will be made into an object include seizing a resource, delaying for a given amount of time and releasing the resource. The Processes Window steps of Seize, Delay, and Release will be used. All logic within this model object will be specified in the Processes Window.

Technical Approach:

Within the first model, called SeizeDelayRelease, we will define the processing necessary for this single object, which includes the Seize, Delay and Release steps. This logic will remain unchangeable, however, the end user of the SeizeDelayRelease object will be able to specify a single resource name (for seizing / releasing), as well as the delay time.

Details for Building the Model:

Defining the SeizeDelayRelease Process Logic

- Open a new model and rename the Model to SeizeDelayRelease by selecting MySimioProject in the Navigation window, selecting the Models panel, highlighting Model and clicking to rename.
- Open the Processes window of SeizeDelayRelease. Create a process with *Name* 'Process1'.
- Place the Seize, Delay and Release steps into Process1.
- For any properties that the end user will be able to specify, highlight the property, right click and select Set Referenced Property, Create New Referenced Property and specify a property name. In this example, within the Delay step, set the *Delay Time* property to 'DelayTime'. You will see a green arrow to the left of DelayTime, indicating that the value will come from the parent object, SeizeDelayRelease. Do the same for the *Object Name* for Seize and Release steps to specify a 'Resource Name' in the parent object.

Defining the SeizeDelayRelease External View

- Click on the Definitions tab and select the External panel, where you will specify what the user will see in a Facility window when they place this model. Place an ellipse, and two external nodes (one for entry, one for exit) in the view.
- Change the entry node's *Node Class Name* to 'BasicNode'. The logic of this model is only in the Processes window, thus the *Input Location Type* is 'Station', with the *Station* of 'EntryStation'.
- Change the exit node's *Node Class Name* to 'TransferNode'. *Input Location Type* is 'None', as this is a transfer out.

Utilizing the Entry and Exit Node within the Process Logic

- Return to the Processes Window and change the change the Triggering Event property to EntryStation.Entered (recall that the entry node's *Input Location Type* was Station called EntryStation).
- Add an EndTransfer step at the start of the process, before the Seize step. This will stop the transfer into the model so that other logic may begin.
- Place a Transfer step after the Release step. Change the *From* property to 'CurrentStation', since we entered this process via station (EntryStation). The *To* property should be 'ParentExternalNode', with the *External Node Name* of 'Node2' (name of exit node in External Window).

Using the SeizeDelayRelease Model in Another Model

- Open a new model and within its Facility Window, place a Source and Sink from the Standard Library and a SeizeDelayRelease model from the Project Library and connect them with Paths. Place a Resource and change the

Name to 'Resource1'.

- Highlight SeizeDelayRelease and specify the *ResourceName* as 'Resource1' and the *DelayTime* as '3.1'.
- When you run this new model, the logic behind SeizeDelayRelease then includes all process logic specified within that model. You may place SeizeDelayRelease as many times in your new model as you like with varying *ResourceName* and *DelayTime* properties.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RandomValueFromTable - SimBit

Problem:

I would like to use real data for my system, including time between arrivals and processing times, instead of distribution data. Recent research¹ suggests that sampling directly from real data may be more accurate than using a fitted distribution.

Categories:

Arrival Logic, Data Tables, Entity Characteristics

Key Concepts:

Before Creating Entities, Data Table, Foreign Key, Object Reference Property, RandomRow, RandomValue, Table Foreign Key Property, Table Row Referencing

Technical Approach:

This model illustrates use of two different types of data. The top subsystem illustrates using data where you have a standalone set of collected values, for example historical processing times. The bottom subsystem illustrates using data where you have a set of collected values with identifying or categorizing information, for example historical processing times by entity type.

Create a table for each set of arrival and/or processing data that will be read during the simulation run. For data that is entity type specific, a foreign key column within a table is utilized to classify the data. In this example, the first Source/Server/Sink combination will read data for both source arrivals, as well as server processing times from two separate tables. In the second Source/Server/Sink combination, two entity types will be generated that have processing time information located within a relational table.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink in the Facility window. Connect these with Paths.
- Place a second Source, Server, and Sink combination in the Facility window and connect as well.
- Place 3 ModelEntity objects from the Project Library into the Facility Window and change the *Name* property values to 'Green', 'Red' and 'Blue' - change their picture colors to match.

Defining the Data Tables with Product Information

- Click on the Data tab of the model and select the Tables panel. Add a DataTable with the *Name* 'Source1Table' and add a 'Real' type property named 'ArrivalTimes_1'. Copy and paste in any 'real' data you may have for time between arrivals (in this example, we generated 50 data points from StatFit2).
- Add a DataTable with the *Name* 'Server1Table' and add a 'Real' type property named 'ProcessingTimes_1'. Copy and paste in any 'real' data you may have for processing times (in this example, we generated 100 data points from StatFit2).
- Add a third Data Table with the *Name* 'PartTypes'. Add two column properties; including an Entity Object Reference property named 'WhichType' and an Integer type property named 'HowMuch'. Add 'Red' and 'Green' to the Entity Object Reference column and have corresponding 'HowMuch' property values of '40' and '60'. We will use the RandomRow function to generate the entity type from Source2. Click on the column 'WhichType' and select the Set Column as Key button. We will use this as a key for the next table.
- Finally, add a Data Table with the *Name* 'Server2Table' that includes a Foreign Key column named 'PartType' as well as a real type column named 'ProcessingTimes_2'. For the 'PartType' column, set the column property named *Table Key* to 'PartTypes.WhichType', which references the column noted in previous step. In this table, you will add associated times for both 'Blue' and 'Red' type parts. The RandomValue function will be used to randomly pick a value from the associated data for the given part type calling the table.

Generating Random Values for Source1 and Server1

- Within Source1, change the *Entity Type* to 'Green' and the *Interarrival Time* to 'Source1Table.ArrivalTimes_1.RandomValue'. This will randomly select a value from the Source1Table (ArrivalTimes_1 property) as the interarrival time each time an entity is generated.
- Within Server1, change the *Processing Time* property to 'Server1Table.ProcessingTimes_1.RandomValue'. This

function will pull a random value from the Server1Table's ProcessingTimes_1 column each time an entity is processed.

Generating Random Entity Types and then Random Values from Relational Table

- Within Source2, open the *Table Row Referencing->Before Creating Entities* section of properties and specify the *Table Name* as 'PartTypes' and the *Row Number* as 'PartTypes.HowMuch.RandomRow'.
- Change the *Interarrival Time* to 'Random.Exponential(6)'. The *Entity Type* can then be specified as 'PartTypes.WhichType', which will be based on the part type generated (40% Blue and 60% Red based on HowMuch column property values).
- Within Server2, change the *Processing Time* property to 'Server2Table.ProcessingTimes_2.RandomValue'. This function will pull a random value from the associated part type's ProcessingTimes_2 column entries each time an entity is processed.

Footnote:

1) Nelson, WSC2013, <http://www.simio.com/resources/presentations/2013-Winter-Simulation-Conference/pdfs/WSC2013-Titan-Keynote-Nelson-Simulation-Curmudgeon.pdf>

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RecordDistanceTraveled - SimBit

Problem:

I want to record the distance traveled by a vehicle per hour.

Categories:

Custom Statistics, Vehicles, Custom Object

Key Concepts:

Subclass, Real State Variable, Discrete State, TallyStatistic, Movement, Distance, Periodic Statistic, Hourly Statistic

Technical Approach:

The Vehicle object will be sub-classed to make a customized vehicle in order to add a state and statistic on top of the standard vehicle behavior. A process will be added to the vehicle to update the state and record the periodic statistic.

Details for Building the Model:

Creating a Sub-Classed Object MyVehicle

- Create a new vehicle class called MyVehicle that is sub-classed from Vehicle by right-clicking on Vehicle in the Standard Library and choosing Subclass.

Adding the State and Statistic to MyVehicle

- Select MyVehicle in the Navigation window and click on the Definitions tab, States panel to add a discrete state with *Name* 'PreviousDistanceTraveled' and *Unit Type* of 'Length'. This will be used to record the distance traveled at the beginning of each period (each hour).
- Also within the Definitions window, select the Elements panel and add a Tally Statistic with *Name* 'DistanceTraveledPerHour' and *Unit Type* of 'Length'. To make this statistic appear beside the existing distance traveled statistic, under the Results Classification properties, use *Data Source* of '[Object]', *Category* of 'Travel' and *Data Item* 'DistanceTraveledPerHour'.
- Also within the Definitions window Elements panel add a Timer with *Name* 'HourlyTally' and *Time Offset* of '1'. This will be used to trigger a process at the end of the first hour and each subsequent hour.
- Within the Processes window add a new process with *Name* 'RecordAndResetDistanceTraveled' and *Triggering Event Name* of 'HourlyTally.Event'. This process will be triggered at the end of each hour so we can calculate and record the travel during that hour and then reset our internal state. Add two steps to this process.
 - Add a Tally Step with *Tally Statistic Name* of 'DistanceTraveledPerHour' and *Value* of 'TotalDistanceTraveled - PreviousDistanceTraveled'.
 - Add an Assign Step to assign the *State Variable Name* 'PreviousDistanceTraveled' to the *New Value* of 'TotalDistanceTraveled'.

Building the Model

- Select Model from the Navigation window and within the Facility window, place a Source and Sink and connect them with two paths, one from Source1 to Sink1 and the other from Sink1 to Source1.
- Within the Source, change the *Interarrival Time* to be 'Random.Exponential(10)'.
- Change the Source's Output TransferNode property *Ride On Transporter* from 'False' to 'True'. Then change the *Transporter Name* property to 'MyVehicle1'.
- Place one of the subclassed vehicle objects by selecting MyVehicle from the Project Library on the left. Specify the *Initial Desired Speed* to be '0.5' and the *Initial Node (Home)* to be 'Input@Sink1'.

Embellishment:

Add a floor label that will display the distance traveled using the Expression 'MyVehicle1[1].TotalDistanceTraveled'. You can also display the travel distance within the current hour by calculating it from the expression 'MyVehicle1[1].TotalDistanceTraveled - MyVehicle1[1].PreviousDistanceTraveled'. An attached label could be made to the vehicle in a similar fashion but omitting the 'MyVehicle1[1].' term in each expression.

Advanced Notes:

The TotalDistanceTraveled function is very efficient because it is based on a level which is only updated on demand. But if this function (or any level) is used in certain types of animation and status displays like a floor label or plot, the frequent updating required may cause noticeable speed degradation.

The changes made to the subclassed vehicle in this SimBit are all “safe” additions, no built-in logic was overridden. This means that you can confidently update such an object when new changes are introduced by Simio.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

ReferenceBatchedEntity - SimBit

Problem:

I would like to make a decision based on the state (priority) of a batched entity. The Vehicle that the parent entity should select is determined by the priority state of the member entity.

Categories:

Combining and Separating Entities, Entity Characteristics, Vehicles

Key Concepts:

Allow Passing, BatchMembers Queue, Before Exiting, Bidirectional Path, Combiner, Condition Based, Current Symbol Index, Object Reference State, Priority, Ride on Transporter, Selection Rule, State Assignments, Vehicle

Technical Approach:

There are three Sources which create entities that will enter a Combiner. One Source creates what will become the parent entity and the other two Sources create member entities. One of the member Sources assigns a priority of '2' to the member entity and the other assigns a priority of '3' to its entities. After entities are combined, they ride on a Vehicle. There are two Vehicles to select from and one has a priority value of '2' and the other has a priority value of '3'. The Parent entity must have knowledge of the priority state on the member entity because the value of the member entity's priority must equal the value of the Vehicle's priority. The parent entity has an entity reference state which stores a reference to the batched member entity so that it can use a function to determine the priority of the member entity.

Details for Building the Model:

Simple System Setup

- Place three Source objects into the Facility window. Name them Source_Parent, Source_MemberA and Source_MemberB.
 - Set the *Interarrival Time* of Source_Parent to 'Random.Exponential(.5)'
 - Set the *Interarrival Time* of Source_MemberA to 'Random.Exponential(.75)'' and set the *Time OffSet* to '.2'
 - Set the *Interarrival Time* of Source_MemberB to 'Random.Exponential(.75)'
- Place a Combiner object and a Sink object. Connect the Combiner object to the Sink object with a Path. Set the *Type* property of this path to 'Bi-Directional' and the *Allow Passing* property to 'False'. This will allow Vehicles to travel back and forth on the path and not travel on top of each other.
- Connect Source_Parent to the Parent Input Node of the Combiner with a Path. Connect both Source_MemberA and Source_MemberB to the Member Input Node of the Combiner with Paths.
- Place two Vehicle objects into Facility window. Name one VehicleA and keep its color the default blue. Name the other VehicleB and change its color to red by selecting red from the Color drop down in the ribbon while that Vehicle object is selected in the Facility window.

Create a Transporter List

- Select VehicleA and right click. Select 'Add to Transporter List' and then select 'Create New Transporter List'. Name this list, 'TransporterList1'.
- Select VehicleB and right click. Select 'Add to Transporter List' and select 'TransporterList1'.
- To view this List that was just created with a right click shortcut, go to the Definitions tab and then to the Lists Panel. Lists can also be created in this window.

Create Additional Entity Symbols

- Place a ModelEntity into the Facility window. Select the entity and give it a total of three symbols by clicking on the [Add Additional Symbol](#) icon in the ribbon twice so that the Active Symbol drop down now has a symbol for index 0, 1 and 2.
 - Select index 1 from the Active Symbol drop down and select a new color (i.e. light blue) and click onto the entity object in the Facility window to change its color. Similarly, select index 2 from the Active Symbol drop down and select a different color (i.e. pink) and click onto the entity object to change its color. Index 0 should still display a green entity object.

- Select the DefaultEntity object and set its *Current Symbol Index* property to 'ModelEntity.Priority – 1'. This will control which symbol is displayed.
- To animate the batched entity, select the DefaultEntity object in the Facility window and select BatchMembers from the Draw Queue drop down in the Symbols ribbon. Draw the queue near the DefaultEntity object.

Set Properties and States on Entity and Vehicles

- Select VehicleA and set its *Initial Priority* property to '2'. Set its *Initial Home* property to 'Output@Combiner1' and its *Idle Action* to 'Park At Home'.
- Select VehicleB and set its *Initial Priority* property to '3'. Set its *Initial Home* property to 'Output@Combiner1' and its *Idle Action* to 'Park At Home'.
- Select Source_MemberA and open the *Before Exiting* repeat group property under the State Assignments category. Add a State Assignment where the *State Variable Name* is set to 'ModelEntity.Priority' and the *New Value* is set to '2'.
- Select Source_MemberB and open the *Before Exiting* repeat group property under the State Assignments category. Add a State Assignment where the *State Variable Name* is set to 'ModelEntity.Priority' and the *New Value* is set to '3'.

Create Entity Reference State on ModelEntity

- Go to the object definition of ModelEntity by selecting it in the Navigation window in the upper right corner of the interface. Go to the Definitions tab of the ModelEntity and click into the States panel by selecting States on the left side. Create an Entity Reference State by selecting 'Entity' from the Object Reference drop down menu in the ribbon. Rename this new state, 'MyBatchedEntity'. This state can later be referenced using 'ModelEntity.MyBatchedEntity'.

Add Logic to Model

- Navigate back to the main model by selecting Model in the Navigation window. Select the Combiner object and open the *Before Exiting* repeat group property editor, found under the State Assignments category.
 - Add a new State Assignment where the *State Variable Name* is 'ModelEntity.MyBatchedEntity' and the *New Value* is 'ModelEntity.BatchMembers.FirstItem'. The BatchMembers.FirstItem function returns a reference to the first item located in the BatchMembers queue of this entity. In this case, there is only one entity in the BatchMember queue. The entity reference state, MyBatchedEntity now contains a reference to the batched entity.
- Select the Output@Combiner1 node and set the *Ride On Transporter* property to 'True'.
 - Set the *Transporter Type* property to 'From List' and the *Transporter List Name* property to 'TransporterList1'.
 - Set the *Selection Condition* property to 'ModelEntity.MyBatchedEntity.Priority == Candidate.Transporter.Priority'. This will tell the model to only select a Vehicle that has a same Priority as the batched entity.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RegeneratingCombiner - SimBit

Problem:

You have a palletizer or similar machine but you do not know the exact timing of when to have pallets arrive.

Categories:

Combining and Separating Entities

Key Concepts:

BatchMembers Queue, Combiner, On Event Arrival Mode, ParentInputBuffer, Source

Assumptions:

We want to have a supply of three pallets waiting so the process will rarely be delayed by lack of pallets.

Technical Approach:

Use the standard features of a Source object to create one new pallet each time one is consumed. Use the *Initial Number Entities* property on the Source when 'On Event' is used to generate initial entities (must be at least 1, we used 3).

Details for Building the Model:

Simple System Setup

- Place two ModelEntity objects from the Project Library and two Sources, one Combiner, and one Sink from the Standard Library in the Facility window of a new model. Connect Source1 to the ParentInput@Combiner1 with a Conveyor. Connect Source2 to the MemberInput@Combiner1 with a path. Connect Combiner1 to the Sink with a Path.
- Change the *Name* of the one of the ModelEntity instances to 'Pallet' and the other to 'Part'.

Creating the Pallets

- Within Source1, change the *Entity Type* to 'Pallet'. Specify *Arrival Mode* of 'On Event'. For *Event Name*, choose 'Output@Combiner1.Exited'.
- On Source1, specify the number of *Initial Number Entities* as '3'. This will generate pallets to start the simulation.

Creating the Parts and Combining Them onto Pallets

- Within Source2, change the *Entity Type* to 'Part'. Set the *Time Offset* to '.5' minutes so that the parts arrive after the initial pallets.
- Within Combiner1, specify the *Batch Quantity* as '7'. This will combine 7 parts onto a single pallet for later processing. Change the *Processing Time* property to Random.Exponential(1) and the *Parent Input Buffer* to '1'. This will cause the other pallets to wait on the conveyor.

Discussion:

When Source1 is initialized at the beginning of the model run, it will create 3 Pallet entities and send them out of the normal Source exit node.

Combiner1 automatically fires an event every time a parent entity exits the Combiner object. Source1 will wait for that event and create a new entity each time it occurs, effectively replacing the pallet that was just consumed.

Embellishment:

We added an Attached Queue to the Pallet entity symbol and associated it with the *Queue State* named 'Pallet.BatchMembers'. This will display the members in any batch associated with the Pallet, or in this case, after the Pallet "picks up" entities in the Combiner, they will appear on the pallet.

RelationalTableInRepeatingProperty - SimBit

Problem:

I have three types of patients that require different processing times, as well as multiple doctors and/or nurses for processing.

Categories:

Data Tables, Worker

Key Concepts:

Before Creating Entities, Data Table, Dynamic Object Property, Expression Property, Foreign Key, Idle Action, Initial Node (Home), Key Column, Numeric Property, ParkingStation Queue, RandomRow, Ranking Rule, Request Move, ResourceState, Server, Smallest Value First, Status Label, Table Row Referencing, Worker

Assumptions:

Each of the three patient types will have a nurse / doctor combination that determines both the name (doctor/nurse) and quantity of each that is needed. More severe patients will have higher priority. There will be one nurse and two doctors available to the patients.

Technical Approach:

Relational tables are used to store information about the patients. A Data Table is created that will store information about each patient type, including the percentage of each that enters the system, and the processing times. A separate related table will include for each patient type the caregiver type (nurse or doctor) and quantity required. Workers will be used for the Doctors and Nurse and will move between the Servers to process the patients.

Details for Building the Model:

Simple System Setup

- Add a Source, two Servers, and a Sink to the Facility window. Also, place three ModelEntity objects from the Project Library into the window.
- Change the *Names* of the ModelEntities to be 'Patient_Mild', 'Patient_Medium' and 'Patient_Severe'. Change the symbol color of each to distinguish them from one another. This is done by highlighting the symbol and selecting the appropriate color from the Symbols ribbon.

Paths and Extra Basic Nodes

- Use Paths and connect Source1 each Server, and both Servers to Sink1.
- Place a BasicNode above each of the TransferNodes at both Servers. These nodes will be the used by the Nurse and Doctors to move between and serve the patients. BasicNode1 should be above Server1 and BasicNode2 should be located above Server2.
- Place an additional BasicNode located in between the Servers, BasicNode3. This will be used as the 'resting' location for the workers when they are not with a patient.
- Use unidirectional Paths to connect BasicNode1 to BasicNode3, BasicNode3 to BasicNode2 and then in the other direction from BasicNode2 to BasicNode3 and BasicNode3 to BasicNode1.

Defining the Workers

- Place two Worker objects in the Facility window and rename them 'Doctor' and 'Nurse'. Change the color of one of the Workers so that you can tell them apart graphically.
- Change the *Park While Busy* property to 'True' so that graphically we can animate the Workers while they are busy.
- Within each, change the *Initial Node (Home)* to 'BasicNode3' where they will all start out at the start of the simulation run. Additionally, change the *Idle Action* to 'Park At Home' so they return to BasicNode3 when they are done with a patient if they are not needed.
- Because we have two doctors in the system, change the *Initial Number in System* (under Dynamic Objects) of the Doctor object to '2'.

Setting up the Data Tables

- In the Data Window, select the Tables panel and add a Data Table named 'Patients' with the following properties and in the following order:
 - (Entity Object Reference with *Name* 'EntityType') Patient_Mild, Patient_Medium, Patient_Severe
 - (Integer with *Name* 'AmountEach') 50,40,10
 - (Expression with *Name* 'ProcessingTime' , *Unit Type* 'Time and *Default Units* 'Minutes') Random.Triangular(2,3,4), Random.Triangular(4,6,8), Random.Triangular(10,15,20)
 - Select the EntityType column and select the Set Column as Key button – this will allow data from this column to be referenced with a Foreign Key column in another table
- Add another Data Table named 'Workers' with the following properties and in the following order:
 - Foreign Key with *Name* 'EntityType' and *Table Key* of 'Patients.EntityType') Patient_Mild, Patient_Medium, Patient_Medium, Patient_Severe, Patient_Severe
 - Object Instance Transporter with *Name* 'Worker Name')Nurse, Nurse, Doctor, Nurse, Doctor
 - Integer with *Name* 'QuantityNeeded') 1,1,1,1,2
 - **** NOTE:** Notice in this table, there is 1 Nurse needed for Patient_Mild, 1 Nurse and 1 Doctor needed for Patient_Medium and 1 Nurse and 2 Doctors needed for Patient_Severe. The entity type may be referenced multiple times within the Foreign Key Column in this table, depending upon the type of caregivers needed

Creating Multiple Entity Types from Source

- In the Facility Window, expand the Table Row Referencing in the Properties Window of the Source object.
- Under the *Before Creating Entities* subcategory, set the *Table Name* to 'Patients' and the *Row Number* to 'Patients.AmountEach.RandomRow'.
- Change the *Entity Type* to 'Patients.EntityType' and *Interarrival Time* to 'Random.Exponential (5)'.

Specifying the Patient Priority

- Within each of the ModelEntity objects, change the priorities of each. The Patient_Mild has an *Initial Priority* of '2', while Patient_Medium is '1' and Patient_Severe is '0'.
- Within each of the Servers, change the *Ranking Rule* to 'Smallest Value First' based on the *Ranking Expression* 'Entity.Priority'.

Adding Processing Time and Assignment Information to the Servers

- Within each Server, change the *Processing Time* to 'Patients.ProcessingTime'.
- Open the Secondary Resources for each of the Servers, under Other Resource Seizes select *On Entering*, open the Repeating Property Editor and Add a property.
 - Change the *Object Name* to 'Workers.WorkerName' the *Request Move* to 'To Node' and *Destination Node* to 'BasicNode1' for Server1 and 'BasicNode2' for Server.
 - Under Advanced Options, change the *Number of Objects* to 'Workers.QuantityNeeded'.
 - Exit the Repeating Property Editor and right click on 'On Entering' and select 'Set Referenced Property' and select 'Workers'.
 - Change *Must Simultaneously Seize* to 'True'.
- Expand the Other Resource Releases under Secondary Resources, open the Repeating Property Editor and Add a property for both Servers for After Processing.
 - Change the *Object Name* to 'Workers.WorkerName' and the *Number of Objects* 'Workers.QuantityNeeded'.
 - Exit the Repeating Property Editor and right click on 'After Processing' and select 'Set Referenced Property' and select 'Workers'.

Animating the Worker Status and Parking Areas

- The status of the doctors and nurses can be animated using a Status Label from the Animation ribbon. Place three status symbols in the Facility Window. Because the doctors and nurse are dynamic objects, they can be referenced individually. Change the *Expression* of the status labels to 'Nurse[1].ResourceState', 'Doctor[1].ResourceState' and 'Doctor[2].ResourceState', respectively.
- Note that the ResourceState function for Workers is '0' when Idle and '1' when Busy.
- Click on BasicNode1 (next to Workstation1) and select Draw Queue > ParkingStation.Contents from the Appearance ribbon. Place three vertices for the queue so that there is a location for each of the doctors and nurse to appear while they are busy. Do the same for BasicNode2.

RemoveFromAllocationQueue - SimBit

Problem:

I have an entity that is waiting for capacity of a resource type object, such as a Server. If the entity has been waiting for 5 minutes, it leaves the queue and exits the system.

Categories:

Decision Logic -- Processing

Key Concepts:

Add-On Process, AllocationQueue, Assign Step, ContinueProcess, EndProcess, Execute Step, On Associated Object Destroyed, On Entered, Real State Variable, Remove Step, SetNode Step, Status Label, Run.TimeNow, Token Actions, Renege, Reneging

Assumption:

In this model, the scenario is a call center where the entities are incoming calls. The callers wait on the line for one of three customer service reps but if they are not serviced within 5 minutes, they hang up. The waiting time for each entity is displayed in an attached status label, rounded to the next lowest integer.

Technical Approach:

The Remove Step is used to remove an entity from the Server's Allocation Queue if it has been waiting longer than 5 minutes. When the entity enters the Server, it executes a new process (Process1) and then continues with its processing. The new process delays for 5 minutes and then attempts to remove the entity from the Allocation Queue, if it is still in the queue. This process has a property called *Token Associated Object Destroyed Action*, which is set to 'EndProcess'. This indicates that if the object that is associated with the token executing the process (the entity) is destroyed, then end the process. So if the entity is still waiting in the queue, it will be removed by this process. But if the entity is no longer waiting and has been destroyed, this process will be ended.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server and two Sinks into the Facility window. Connect the Source to the Server with a path and connect the Server to each Sink with paths.
- Drag a ModelEntity object into the Facility window. This can be renamed to CustomerCall to represent a phone call.
- Click on the Source and rename it to 'Incoming Calls'. Set the *Interarrival Time* to 'Random.Exponential(.7)'
- Click on the Server and rename it to 'CustomerServiceReps'. Set the *ProcessingTime* to 'Random.Exponential(4)'. Set the *Initial Capacity* to '3'.
- Rename one Sink to 'HangUps' and the other to 'ServicedCustomers'

Add A State to ModelEntity

- Click on the ModelEntity from within the Navigation window on the top right side of the interface. Go to the Definitions tab (this is the Definitions window of the ModelEntity).
- Click on the States panel on the left. Click on 'Discrete State' in the ribbon to create a new state variable. Rename it to 'TimeEnteredLine'.

Process Logic

- Create a new Add On Process in the 'Entered' trigger of the Server. It will be named 'Server1_Entered'.
- From within the Processes window, place an Assign Step in the 'Server1_Entered' process. Set the *State Variable Name* to 'ModelEntity.TimeEnteredLine' and *New Value* to 'Run.TimeNow'. This is recording when this entity begins its wait in the Server queue.
- Place a SetNode Step after the Assign Step. Set the *Destination Type* to 'Specific' and the *Node Name* to 'Input@ServicedCustomers'. By default, entities should exit through this Sink.
- Create a new process by clicking on the 'Create Process' icon in the ribbon. Name this 'Process1'. In the properties of this process, set *Token Associated Object Destroyed Action* property to 'EndProcess'
- Back in the 'Server1_Entered' process, place an Execute Step after the SetNode Step. Set the *Process Name* to 'Process1' and the *Token Wait Action* to 'None (Continue)'. This will cause Process1 to be executed, but the token (and

therefore the entity) will continue on with its action while Process1 begins execution.

- Place a Delay Step in Process1. The *Delay Time* should be 5 minutes.
- Place a Remove Step next, that has its *QueueStateName* set to 'CustomerServiceReps.AllocationQueue'.
- In the Removed segment leaving the Remove Step, place a SetNode Step. The *Destination Type* should be set to 'Specific' and the *Node Name* should be 'Input@HangUp'. This will send removed entities (entities still in the queue after the 5 minute delay), to the HangUp sink.
- After the SetNode Step in Process1, place a Transfer Step. Set *From* to 'CurrentStation' and *To* to 'Node' and *Node Name* to 'Output@CustomerServiceReps'. When an entity is removed from a queue, it must be transferred somewhere. Set the *Token Wait Action* property to 'WaitUntilTransferringEvent'.

Status Labels

- Click on the CustomerCall entity in the Facility window and draw an attached Status Label by clicking the Status Label icon in the ribbon (while CustomerCall is selected). Set the *Expression* property to 'Math.Floor((Run.TimeNow – TimeEnteredLine)*60). This will display the waiting time (rounded) of each entity as it travels the system.
- Click on each Sink object and draw a Floor Label by clicking the Floor Label icon in the ribbon (while the Sink is selected). Then, click on the Edit icon in the ribbon to edit the text of this attached Floor Labels. Type 'Total HangUps: {InputBuffer.NumberEntered}' and 'Total Serviced Customers: {InputBuffer.NumberEntered}'.

Embellishments:

Vary the Delay time to see how the system changes or set the Delay time as a property and vary it within an experiment. Experiment with the capacity of the Customer Service Rep server to see the impact on the number of customers who hang up.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RequestRideFromSameTransporter - SimBit

Problem:

I have entities that select a transporter from a list, after being processed at a Server, the entity must be picked up by the same transporter.

Categories:

Entity Characteristics, Worker

Key Concepts:

Object Reference State, Add-On Process, Assign Step, Transporters, ModelEntity, ObjectList, TransporterList, Ride on Transporter, Entity.CurrentTransporter, Functions

Assumption:

There is one unit of each worker type in the system.

Technical Approach:

A Transporter Reference State Variable will be added to the ModelEntity. An Add-On Process will be used to assign this state variable a value based on the Transporter that picks up the entity. The transporter reference state will then be used to request a ride on a particular worker later in the model.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server, and a Sink from the Standard Library. Use two Unidirectional Paths (one in the forward, and one in the reverse, direction) to connect Source and Server1, Input@Server1 and Output@Server1, and Output@Server1 to Sink1. – *The reason we use 2 paths here is to avoid deadlocks when the 2 transporters are traveling in opposite directions.*
- Then connect the Sink to the Source using just one path.

Adding the Transporters and Transporter List

- Place two Worker objects in the Facility and change the *Name* properties to 'WorkerA' and 'WorkerB'.
- While holding the Ctrl key, click on each of the Worker objects. While they are highlighted, right-click in the highlighted area and select Add to Transporter List > Create a New Transporter List and name the list 'WorkerList'. You will notice now that within the Definitions window, List panel, this new Transporter List is available to view/edit and contains our 2 worker objects.
- Also while both Workers are selected, set their *Initial Node* to 'Output@Source1'.

Adding a Transporter Reference State

- Within the Navigation window, highlight the ModelEntity and go to its Definitions window, States panel.
- From the States ribbon, select Object Reference > Transporter. Change the *Name* of the object reference state to 'WhoPickedMeUp'. This will generate a new state on the entity, referenced ModelEntity.WhoPickedMeUp, which can store a reference to a particular instance of a Worker.

Defining the Logic

- Within the Navigation window, go back to the Model and make sure you are in the Facility window.
- Select one of the Workers, expand its Add-On Process Triggers and double-click on the Loaded trigger (make sure you click on the word Loaded), which will automatically create a new process, and take you to the Processes window.
- Add an Assign step to the process and assign the *State Variable Name* 'ModelEntity.WhoPickedMeUp' to the *New Value* 'Entity.CurrentTransporter' – *This function returns a reference to the transporter object where the entity is currently riding.*
- Go back into the Facility window. Click on the other Worker and add this same process to its Loaded Add-on Process Trigger. The same Assign step will now be called by both Workers.
- Set the *Ride on Transporter* Property in Output@Source1 to 'True', *Transporter Type* to 'From List', and *Transporter List*

Name to 'WorkerList'.

- Finally, set the *Ride on Transporter* Property in Output@Server1 to 'True', leave the *Transporter Type* as 'Specific', and set the *Transporter Name* to 'ModelEntity.WhoPickedMeUp'. – You will have to manually type this value in, the *Transporter Name* property will make the state available in the drop down.

Enhancing the Animation

- Set the Speed Factor in the Run Window to be 0.010.
- Drag in an instance of ModelEntity. While it is selected, click on 'Status Label' and draw a box next to the entity. Set its *Expression* to 'WhoPickedMeUp'. – The 'ModelEntity' is not necessary because the label is already attached to the ModelEntity so it knows what object we are referring to.
- To Rotate the Status Label, press Ctrl and drag on of the green dots in the corner of the label.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ResourceSelectionConcepts - SimBit

This SimBit Project includes four models that demonstrate the use of Secondary Resources For Processing. The models increase in complexity and demonstrate methods for selecting Resources based on attributes like cost, efficiency and skill.

Models included in this SimBit:

1. CostBasedResourceSelection
2. EfficiencyBasedResourceSelection
3. SkillsBasedResourceSelection
4. SecondaryResourceShiftChange

Model 1: CostBasedResourceSelection

Problem:

I have three processing lines which each require one Secondary Resource For Processing, and I would like to select the least costly available Resource.

Categories:

Resources

Key Concepts:

Efficiency, Financials, Largest Value, Lists, Object List, Resource, Resource Efficiency, Secondary Resources, Selection Goal, Status Label

Assumptions:

When using Resources and Efficiency features, the actual Processing duration for a Server is defined as *Processing Time* divided by *Resource Efficiency*. This may be one value if there is only one seized Resource, or an Aggregate Efficiency value determined by *Resource Efficiency Rule*, in the event of multiple seized Resources.

Technical Approach:

This Model contains three competing Source -> Server -> Sink processing lines and three Resources stored in an Object List. Each Resource has a different hourly *Usage Cost Rate*. Each Server requires one Secondary Resource For Processing, and always tries to select the least expensive, available Resource using a *Selection Goal*.

Details for Building the Model:

System Setup

- Place one Source (Source1), one Server (Server1) and one Sink (Sink1) at the left, middle, and right sides (respectively) of the Facility window.
- Connect Source1 to Server1, and Server1 to Sink1, using Paths.
- Change *Interarrival Time* of Source1 to 'Random.Exponential(1)' and, under Stopping Conditions, set *Maximum Arrivals* to '50'.
- Place three Resource objects in the Facility window and change their names to 'Joe', 'Mary' and 'Bill'. You can change the Resource names by either double clicking their labels in the Facility window or navigating to the General section of their Properties Window and editing the *Name* Property.
- Click on Joe and, under Financials -> Resource Costs, change the *Usage Cost Rate* to '15'. Make sure its units are 'USD' in the dropdown. Then, repeat this for Mary and Bill but with *Usage Cost Rate* values of '17.50' and '20' respectively.
- Then, in the Definitions Tab, navigate to the Lists View (left hand side of the screen) and create an Object List. In its Properties window change *Name* to 'Resources'. Then, in the bottom half of the split window, populate the list with the three Resources (Joe, Mary, Bill) that you instantiated in the Facility window.
- Change the *Processing Time* of Server1 to '1' and make sure that its *Units* are 'Minutes' in the dropdown.
- Under Secondary Resources set *Repeat Group* to 'False', *Object Type* to 'Select From List', *Object List Name* to 'Resources', *Selection Goal* to 'Smallest Value', and final *Value Expression* to 'Candidate.Resource.ResourceUsageCostRate'.
- Place one ModelEntity object in the Facility window and notice that its name is DefaultEntity (by default).

- Now copy and paste the Source -> Server -> Sink processing line by holding ctrl then click-dragging to highlight all three objects (and their Paths) at once. Copy (ctrl+c) and paste (ctrl+v) shortcut keys work in Simio, or you may use the icons on the Project Home Ribbon. Copy your processing line containing Source1, Server1 and Sink1 and paste it twice in Facility window. You may leave the names as they are: Source1_1, Server1_2, etc. or change them to Source2, Server3, etc. depending on your preference.

Status Labels

- Click on Joe in Facility window, then navigate to the Animation Ribbon and select a Status Label. Then bring your mouse back into Facility window and notice that its appearance has changed to a crosshair. Now you may click-drag to draw a gray status label above Joe in Facility window. Once its drawn, and you select it, you will notice that its *Attached To* property has the value 'Joe'.
- The goal is to write an *Expression* which will dynamically demonstrate where each Resource is throughout the run. On your Status Label type the *Expression* 'ResourceOwners.FirstItem.CurrentStation'.
- Upon running the Model, you'll encounter a Runtime Warning stating "Unable to evaluate status expression for graphic." The relevant "Possible cause" is that which reads "An attempt was made to access the member of an object or element reference, but the reference is set to the value 'Nothing'."
- One option would be to check the box for "Don't show this warning again", however, it is always recommended to try and resolve Model logic, rather than suppress warnings.
- A solution is to rewrite your Status Label *Expression* as 'Math.If(ResourceOwners.FirstItem == Nothing, " ", ResourceOwners.FirstItem.Entity.CurrentStation)'. Now, you will notice that the Runtime Warning does not occur, because the Status Label simply displays a blank space " " when Joe is idle.
- Repeat the steps to create Attached Status Labels for Mary and Bill with the same 'Math.If...' *Expression* discussed above.

Model 2: EfficiencyBasedResourceSelection

Problem:

I have three processing lines which each require one Secondary Resource For Processing, and I would like to adjust the Processing Times based on *Resource Efficiency*.

Categories:

Data Tables, Resources

Key Concepts:

Data Table, Efficiency, Largest Value, Resource, Resource Efficiency, Secondary Resources, Selection Goal, Server, Status Label, Status Plot

Assumptions:

Each Server has the same *Processing Time* of '1' Hour, by definition, but the actual duration of Processing varies with the Efficiency of the Secondary Resource used For Processing.

Technical Approach:

This Model contains three competing Source -> Server -> Sink processing lines and three Resources stored in a Data Table along with their Efficiency values. Each Server requires one Secondary Resource For Processing and always tries to select the most efficient Resource available using a *Selection Goal*.

Details for Building the Model:

Some of the instructions for this Model refer to the CostBasedResourceSelection Model. If you did not work through the CostBased Model, please refer to its "Details for Building the Model" documentation section as needed.

System Setup

- Start by copying the three processing lines you created in CostBasedResourceSelection and paste them in the empty Facility window of EfficiencyBasedResourceSelection. Also copy and paste Joe, Mary and Bill along with their Attached Status Labels.
- Instead of storing the Resources in a List, as was done in CostBasedResourceSelection, they will now be stored in a Data Table. Navigate to the Data tab and, on the Schema Ribbon click the Add Table dropdown and select 'Add Data Table'. In the Table Properties Window change *Name* to 'SkillsTable'.
- On the Schema Ribbon, select the Object Reference Property dropdown and choose 'Object'. This will create an Object Property column in your SkillsTable.
- Now, in the Property Window for the column you just created, change *Name* to 'Resources' and change *Object Type*

to 'Resource'.

- Next, on the Schema Ribbon, select 'Real' from the Property dropdown to create another column. Name this column 'Efficiency' and enter the values '0.5', '0.75' and '1' respectively for the rows storing Joe, Mary and Bill.
- Next configure the Servers to use the information stored in SkillsTable. Go back to Facility window and select Server1. Some of the Properties copied over from the CostBased Model will persist, but not all of them. Change *Object List Name* to 'SkillsTable.Resources', *Selection Goal* to 'Largest Value', *Value Expression* to 'SkillsTable.Efficiency' and in the Advanced Options dropdown, *Resource Efficiency* to 'SkillsTable.Efficiency'.
- You may have noticed that the *Object Type* property is still 'Select From List', which is as desired, because in this case we are using the Table column as a List. There is more information about Table columns as lists in the "Data Tables" Help topic.
- Make the same updates to Server2 but set *Selection Goal* to 'Random' and notice that *Value Expression* no longer appears.
- Make the same updates (made to Server1) to Server3 but make sure *Selection Goal* is set to 'Smallest Value' and set *Value Expression* to 'SkillsTable.Efficiency'.
- Finally, place one instance of ModelEntity in the Facility window.

Status Labels and Status Plot

- Select (click) Server1 in Facility window, then navigate to the Animation Ribbon and select a Status Label. Then bring your mouse back into Facility window and notice that its appearance has changed to a crosshair. Now you can click-drag to draw a gray status label near the top corner of Server1 in Facility window. Once its drawn, and you select it, you should notice that it has an *Attached To* property with the value 'Server1'.
- We want to display the *Aggregate Efficiency* of each Server. *Aggregate Efficiency* will display the *Resource Efficiency*, at a given Server, based on one or multiple Secondary Resources For Processing — subject to the *Resource Efficiency Rule* in the Server Process Logic, under *Other Processing Options*. You'll notice that a numerical index (0-5) is passed to *Aggregate Efficiency*. The values (0-5) respectively correspond to the six values of the *Resource Efficiency Rule* property: 'None', 'Average', 'Count', 'Maximum', 'Minimum' or 'Sum'. Since we are only using one Secondary Resource For Processing, in this Model, the value is somewhat arbitrary as long as it is not 0 (None) or 2 (Count).
- For the Status Label attached to Server1, type the *Expression* 'Math.If(Processing.Contents.FirstItem == Nothing, " ", Processing.Contents.FirstItem.Entity.SeizedResources.AggregateEfficiency(1))'. Refer to the CostBasedResourceSelection documentation section "Details for Building the Model" -> "Status Labels" for a discussion about the need for 'Math.If...'.
- Navigate to the Animation Ribbon and select a Status Plot. Click-and-drag to draw it in Facility window. Then, open the Repeating Property Editor by selecting the '...' icon within the *Additional Expressions* property.
- Click the Add button three times to create three [Empty] Items in the Repeating Property Editor. Select the first [Empty] Item, and type the *Expression* 'Server1.Processing.Contents.AverageTimeWaiting * 60', and *Label* it 'S1_Avg_Processing'. Repeat these steps for Server2, and Server3 in the other two [Empty] Items. Now when you Run the Model, you'll notice that the Status Plot populates with the average processing time at each server, based on the Entities that have completed processing.

Model 3: SkillsBasedResourceSelection

Problem:

I have two processing lines, which each require two Secondary Resources. Some Workers are only qualified to work at one Server. I would like to use the average of the seized Resource Efficiencies to determine the overall effect on Processing Time.

Categories:

Data Tables, Resources

Key Concepts:

Data Table, Efficiency, Largest Value, Resource Efficiency, Secondary Resources, Selection Condition, Selection Goal, Status Label, Worker

Assumptions:

The Worker 'Joe' is not qualified to work at Server1 and the Worker 'Tom' is not qualified to work at Server2 per their Efficiency values of '0' as defined in the SkillsTable Data Table.

Technical Approach:

This Model contains two Source -> Server -> Sink processing lines and four Workers with different values of Efficiency, at each Server, defined in a Data Table. An Efficiency value of '0' indicates that a Worker is not qualified at a given Server. Each Server requires two Secondary Resources For Processing, and always tries to select the most efficient Resource

available using a *Selection Goal*. A *Selection Condition* guarantees that a Worker must also be qualified to work at a given Server.

Details for Building the Model:

System Setup

- From the Project Home Ribbon, select the Actions dropdown and click the Source, Server, Sink Add-In.
- Hold down ctrl and click-drag to highlight the entire Source -> Server -> Sink processing line, that was created by the Add-In, and copy/paste it to the right in Facility window.
- Drag four Worker objects into Facility window and name them 'Joe', 'Tom', 'Bill' and 'Mary'.
- Place a BasicNode near the Workers and name it 'Home'.
- Use ctrl+click to multi-select Joe, Tom, Bill and Mary. Then edit the following properties: set *Park While Busy* to 'True', *Initial Node (Home)* to 'Home' and *Idle Action* to 'Park At Home'.
- Navigate to the Data Tab and use the Add Table dropdown, on the Schema Ribbon, to Add Data Table. In the Table Properties Window change *Name* to 'SkillsTable'.
- On the Schema Ribbon select Object from the Object Reference Property dropdown. You will now see a column of type Object Property. In its Properties Window, change *Name* to 'Workers', *Object Type* to 'Worker' and *Auto-set Table Row Reference* to 'True'.
- Then populate the Workers column with Joe, Mary, Bill and Tom.
- Next create two Real Property columns, using the Schema Ribbon Property dropdown, and selecting Real (twice). Name the first new column 'EfficiencyAtServer1', and the second new column 'EfficiencyAtServer2'.
- Respectively for Joe, Mary, Bill and Tom, populate the 'EfficiencyAtServer1' column with the values 0, 0.5, 1 and 2. Then, in the same order, populate the 'EfficiencyAtServer2' column with the values 1.5, 0.5, 1.5 and 0.
- Now we will configure the Servers up to use the information in Skills Table. Multi-select Server1 and Server2 by holding ctrl then clicking them both. Set *Processing Time* to '1' (Minutes) and *Resource Efficiency Rule* (in Other Processing Options dropdown) to 'Average'.
- With both Servers still multi-selected: In the Secondary Resources section, under For Processing, set *Repeat Group* to 'False', *Object Type* to 'Select From List', *Object List Name* to 'SkillsTable.Workers', *Selection Goal* to 'Largest Value' and, under Advanced Options, *Number of Objects* to '2'.
- Then, only for Server1, set *Value Expression* (still in Secondary Resources -> For Processing) to 'SkillsTable.EfficiencyAtServer1', *Request Move* to 'To Node', *Destination Node* to 'Input@Server1', and under Advanced Options, *Selection Condition* to 'SkillsTable.EfficiencyAtServer1 > 0', and *Resource Efficiency* to 'SkillsTable.EfficiencyAtServer1'. The *Selection Condition* ensures that a Worker is qualified to work at a Server before it is seized.
- Then, only for Server2 make the same adjustments but reference the 'SkillsTable.EfficiencyAtServer2' column, and 'Input@Server2' Node, in the proper Expressions.
- Drag one ModelEntity into the Facility window.

Drawing Queues

- Select the BasicNode named 'Home' that you placed in Facility window. Then, in the (Node Tools) Appearance Ribbon, click the Draw Queue dropdown and select ParkingStation.Contents. Drag your cursor back into Facility window and notice that it now appears as crosshairs.
- Left click somewhere near the 'Home' Node to make the first Queue Vertex, then right click somewhere else near the 'Home' Node to make another Vertex. Next, on the (Queue Tools) Appearance Ribbon, select the Oriented Point Alignment option.
- Now use the Add Vertex button, on the Appearance Ribbon, to add two more Vertices and place them wherever you'd like the Workers to wait when Idle. The position of the green arrows, extending from the Vertices, may be adjusted to control the direction the Workers will face while they occupy the Queue.
- Follow the same steps to draw two more Oriented Point ParkingStation.Contents Queues *Attached To* Input@Server1 and Input@Server2.

Status Labels and Dynamic Labels

- I would like to create a Dynamic Label (one that travels with the Agent as it moves in the Animation) showing the Efficiency for a given Worker, at a given Server. Multi-select Joe, Tom, Bill and Mary using ctrl+click. In the General section, of their Properties Window, we will use a Math.If statement in *Dynamic Label Text* to evaluate (logically) "if the Worker is at Input@Server1, show its EfficiencyAtServer1, if the Worker is at Input@Server2, show its EfficiencyAtServer1, otherwise show nothing".
- The Expression is as follows: 'Math.If(Worker.CurrentNode == Input@Server1, String.FromReal(SkillsTable.EfficiencyAtServer1), Worker.CurrentNode == Input@Server2, String.FromReal(SkillsTable.EfficiencyAtServer2), " ")'.

- Then you may append (at the beginning) each Worker's *Dynamic Label Text* with their name. So the *Dynamic Label Text* for Joe should be `"Joe" + Math.If(Worker.CurrentNode...)`.
- Finally, make a Status Label attached to Server1, and another attached to Server2 with the following *Expression* `'Math.If(Processing.Contents.FirstItem == Nothing, " ", Processing.Contents.FirstItem.Entity.SeizedResources.AggregateEfficiency(1))'`. See the "Status Labels and Status Plot" section, in *EfficiencyBasedResourceSelection* documentation above, for a discussion about this Expression.

Model 4: SecondaryResourceShiftChange

Problem:

I have some day shift Resources who are replaced by night shift Resources in the middle of the day. I would like the Server to continue working during the shift change and re-calculate remaining Processing Time based on the Efficiency of night shift Resources.

Categories:

Add-On Process Logic, Building New Objects / Hierarchy, Data Tables, Resources, Schedules / Changeovers

Key Concepts:

Add-On Process, Data Table, Efficiency, Lists, ObjectList, Off Shift, On Shift, Output Table, MyResource, Resource Efficiency, Schedules, Secondary Resources, Selection Goal, SubClass, WorkSchedule

Assumptions:

This SimBit uses an Output Table which is a disabled feature for Design Editions. Users with these Editions will be able to view the Output Table but not create their own or edit it.

Technical Approach:

This Model contains one Source -> Server -> Sink processing line and five instances of a Resource Subclass Object called MyResource whose definition contains a Real Property called Efficiency. The value for Efficiency is then assigned at each instance of MyResource, in the General Property Group. Each instance of MyResource follows a WorkSchedule and at 2PM a shift change occurs. The *Off Shift Rule* 'Switch Resources If Possible' allows the Server to continue processing while re-calculating the remaining *Processing Time* based on the Efficiency values of the incoming MyResources.

Details for Building the Model:

Custom Resource Object

- Right-click the Resource Object, in Standard Library, and select Subclass. Notice that the Navigation Window (top right) now shows MyResource along with ModelEntity and the Models you have created.
- Select MyResource in the Navigation Window and navigate to the Definitions Tab. In the Properties View, define a Real Property using the Standard Property dropdown (Properties Ribbon) and selecting Real.
- Change *Name* of the Real Property to 'Efficiency' and set its *Default Value* to '1.0'.
- We will use this custom Resource Object in our Model and be able to define an Efficiency for each instance of MyResource.

System Setup

- Place a Source, Server and Sink in Facility window and connect them with Paths.
- Place five MyResource objects in Facility window and name them 'Joe', 'Mary', 'Bill', 'Sue' and 'Tom'. Notice that MyResource is not located in the Standard Library (to the left of Facility window), but in the [Project Library] which is just below it. In the General section of their Properties Window, set *Efficiency* to '0.75', '1.0' (default), '1.0' (default), '1.5' and '2.0' respectively.
- In the Definitions Tab -> Lists View create an Object List to store the five MyResources just instantiated. Change the *Name* property of your List to 'Resources'.
- Now configure Server1. Set *Processing Time* to '1' (Hours), and *Resource Efficiency Rule* (under Other Processing Options) to 'Average'. Under Secondary Resources -> For Processing, set *Repeat Group* to 'False', *Object Type* to 'Select From List', *Object List Name* to 'Resources', *Selection Goal* to 'Largest Value', *Value Expression* to 'Candidate.MyResource.Efficiency' and *Off Shift Rule* to 'Switch Resources If Possible'. Under Advanced Options, set *Number of Objects* to '2' and *Resource Efficiency* to 'Candidate.MyResource.Efficiency'.
- Place one ModelEntity in Facility window.

Work Schedules

- Navigate to the Data Tab, Work Schedules View.
- Rename the 'StandardWeek' Work Schedule to 'Day' and change its description to something meaningful like 'The

day shift schedule'. Create another Work Schedule by typing 'Night' in the next row and change its description to something similar.

- In the bottom half of the split Window rename the 'StandardDay' Day Pattern to 'DayShift' and give it a Description like 'Standard 6-2 work day'. Then create another Day Pattern called 'NightShift' and give it a Description like 'Standard 2-10 work day'.
- In 'DayShift' set *Start Time* to '6:00 AM', *Duration* to '8 hours' and notice that *End Time* automatically updates to '2:00 PM'. Make sure that *Value* is set to '1'. This corresponds to the Capacity (while On-Shift) of any Object following this WorkSchedule.
- In 'NightShift' set *Start Time* to '2:00 PM', *Duration* to '8 hours' and notice that *End Time* automatically updates to '10:00 PM'. Make sure that *Value* is set to '1'.
- Finally, update the 'Day' Work Schedule to use 'DayShift' Pattern on each day *Monday* through *Friday*. Make the same change to 'Night' but using the 'NightShift' Pattern.
- Go back to Facility window and ctrl+click to multi-select Joe, Mary, Bill, Sue and Tom. In the Properties Window update *Capacity Type* to 'WorkSchedule'. Next, for Joe, Mary and Bill set *Work Schedule* to 'Day'.
- Do the same for Sue and Tom but set *Work Schedule* to 'Night'.

Output Table (disabled in Design Editions)

- In the Data Tab Schema Ribbon use the Add Table dropdown to Add Output Table. Change its *Name* property to 'Results'.
- On the Schema Ribbon, use the State dropdown to add an Integer State column. Change its *Name* to 'EntityID'.
- Again, using the State dropdown (Schema Ribbon), add two DateTime State columns. Name the first one 'StartProcessing' and the second one 'EndProcessing'.
- Finally, add a Real State column and name it 'Duration'. Make sure its *Unit Type* is set to 'Time'.

Process Logic

- In Facility window select the Input@Server1 Node. In the Add-On Process Triggers section, double click on *Entered* and notice that an Add-On Process called 'Input_Server1_Entered' is created.
- Follow the same steps, at Output@Server1 to create an 'Output_Server1_Entered' Add-On Process.
- Next go to the Processes Tab and click 'All Steps' in the bottom left to reveal more Steps. Drag an AddRow Step into the 'Input_Server1_Entered' Process. Click on the AddRow Step and change *Table Name* to 'Results'.
- Click on the Add Row Step you just made, and press F2 to give it a meaningful name like 'AddOneResultsRow'.
- If you simply click in the blank space of the Process you will see its Properties appear in the Properties Window and you may give it a meaningful *Description* if you'd like.
- Next drag an Assign Step into the 'Output_Server1_Entered' Process. Use F2 to give it a meaningful name like 'Results.Duration'.
- Within the 'Results.Duration' Assign Step set *State Variable Name* to 'Results.Duration' and *New Value* to 'Results.EndProcessing – Results.StartProcessing'.
- Click the three dots in *Assignments (More)* to open the Repeating Property Editor. Use the Add button to create an empty Item. Within it, set *State Variable Name* to 'Results.EntityID' and *New Value* to 'ModelEntity.ID'.
- These Processes will populate the Results Output Table throughout the run, and their functionality will be clearer once you can see the Results.

Status Labels

- In Facility window click Server1 and navigate to the Animation Ribbon. Click Status Label and draw it in Facility window near Server1. Make sure that its *Attached To* property is 'Server1'.
- Set its *Expression* to 'Math.If(Processing.Contents.FirstItem == Nothing, " ", Processing.Contents.FirstItem.Entity.SeizedResources.AggregateEfficiency(1))'. There is more discussion about this Expression in the previous Models' "Status Labels" documentation sections.
- Lastly draw one more Status Label near the first one, this time not attached to anything. Type the *Expression* 'DateTime.ToString(TimeNow, "h:mm tt")'. Observe how this *Expression* dynamically displays the Simulation Time throughout the run.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ResourceStatesWhenOffShift - SimBit

Problem:

I have a system with two Servers that each seize a secondary Resource before processing begins. Each Server behaves differently if it is processing an entity when the seized secondary Resource goes Off Shift. Server1 will finish processing any entities that are currently being processed but will not begin working on another entity until the secondary Resource is back On Shift. Server2 will immediately end the processing of any entities when the secondary Resource goes Off Shift.

Categories:

Add-On Process Logic, Resources, Schedules / Changeovers

Key Concepts:

Add-On Process, Interrupt Step, On Off Shift, OnEnteredProcessing, Real State Variable, Resource, Resource, ResourceState, Save Remaining Time, Schedules, Status Pie

Assumptions:

When Server2 immediately ends the processing of any entities when the secondary Resource goes Off Shift, the interrupted entity will remember how much time it has left for processing and get sent back to the InputBuffer of Server2 to queue up so it can finish its processing time when the secondary Resource comes back On Shift.

Technical Approach:

Server1 will use the default behavior when any seized secondary Resource goes Off Shift, which is to continue processing any entities that are in the middle of processing during the shift change. The Resource will be put into the Resource State of "OffShiftBusy" during the time when it is finishing the processing of these entities. Server2 will not follow the default behavior, but instead when the secondary Resource goes Off Shift, it will Interrupt any processing that is currently occurring on Server2. Capacity of Server2 will be immediately released and the entity will be transferred back to Server2.InputBuffer with a Transfer Step.

Details for Building the Model:

Simple System Setup

- Place a Source and a Sink into the Facility window. Place two Servers between that are in parallel to each other. Connect the Source to both Servers with Paths and connect both Servers to the Sink with Paths.
- Place two Resource objects into the Facility window.

Create Work Schedules:

- Go to the Data window and go to the Schedules panel by clicking Schedules in the left hand side of the Data window.
- Add a new schedule by clicking on the Work Schedules button and entering the schedule *Name* 'WorkSchedule1'.
- Set the Start Date (i.e. 6/14/2010) and the number of *Days* to '1'.
- Create a Day Pattern by clicking on the Day Pattern tab. Enter the *Name* 'DayPattern1'.
- Click on the '+' sign to expand the Work Periods.
- Create a schedule that has a *Value* of '1' (on shift) in these time slots: 8:00-8:10am, 8:15-8:25am, 8:30-8:40am, and 8:45-8:55am. Leave everything else empty, which indicates Off Shift times. *The schedule of the run starting time day is adjusted depending on the schedule's number of days. As the schedule's number of days is one day, every day will have the same schedule independently of the fact that the run date is prior to the schedule date. For example a run date starting June 12th vs. a schedule start date on June 14th.*
- Back in the Facility window, click onto each Resource object and change the Capacity Type property of each to 'WorkSchedule' and select the new Schedule for the Work Schedule property. The same schedule can be used for both Resource objects.
- Change the Run Time parameters by clicking on the Run Tab in the Ribbon. Set the Starting Time to 8:00am and set the Ending Type to 1 Hour.

Adding a New State to ModelEntity

- Create a new State on the ModelEntity. This will keep tracking of the processing time and it is needed because during the Interrupt Step, we save off the remaining processing time.

- Click on the ModelEntity within the Navigation Window in the top right. When this is selected, go into the Definitions window. You are now in the Definitions window of the ModelEntity (not the main model).
- Click on the States panel to the left and click on *Discrete State* in the Ribbon to add a new State. Name this ProcessingTime. Set the *Initial State Value* to '.4'
- Go back to the main model by clicking on Model in the Navigation window.
- Go to the Facility window and click on Server1. Set the *Processing Time* property to 'ModelEntity.ProcessingTime'. Do the same for Server2.

Add Process Logic to main Model:

- From the Facility window of the main model, click on Server1. Create two new processes in the *Processing* and *After Processing* Add On Process triggers.
- Do the same for Server2 (click on the Server2 and create two new Add On Processes for *Processing* and *After Processing*)
- Go to the Processes window and place a Seize Step in Server1_Processing process and place a Release Step in the Server1_AfterProcessing process. These steps will seize the secondary Resource before Server1 begins processing and it will release it after it has finished processing. In the Seize Step, set Object Type to 'Specific' and select 'Resource1' from the dropdown of Object Name. Leave everything else as the defaults.
- Do the same for Server2 – add a Seize Step to the Processing process and a Release Step to the Processed process. But instead of seizing and releasing Resource1 – use Resource2.
- Server1 and Resource1 will follow the default behavior when Resource1 goes Off Shift. But Server2 and Resource2 will immediately end processing. So we need to add process logic when Resource2 goes Off Shift. From within the Facility window, select Resource2. Create a new process in the *Off Shift* Add On Process trigger.
- From within the Processes window, place an Interrupt Step in this new process. Set the *Process Name* to 'Server2.OnEnteredProcessing'. Set the *Interrupted Process Action* to 'EndProcess'. And set *Save RemainingTime* to 'ModelEntity.ProcessingTime'.
- In the Interrupted segment of the Interrupt Step, place a Release Step. In the Release Step, set *Object Type* to 'Specific' and select 'Resource2' from the dropdown of *Object Name*. Leave everything else as the defaults.
- After the Release Step, place a Transfer Step to move any entity currently being processed into the InputBuffer. Set *From* to 'CurrentStation' and *To* to 'Station' and *Station Name* to 'Server2.InputBuffer'.

Adding ResourceState Pie Charts to Facility Window:

- In the Facility window, select Resource1 and then click on Status Pie within the Ribbon. Draw the Pie Chart into the Facility window. This will create a Status pie chart that is attached to Resource1. In the properties window of the Status Pie, set *DataType* to 'ListState' and *ListState* to 'ResourceState'.
- Do the same for Resource2. When the model runs, notice the different between the Resource States between the two Resources. Resource1 will have some time where it is 'OffShiftBusy' because it will finish working on an entity, but Resource2 will never have this State because it will always interrupt processing when it goes Off Shift so it will never be OffShiftBusy.

Embellishments:

Change the Work Schedule durations or the Processing times to see changes to flow and balance of the system and therefore to the time spent in each ResourceState.

See Also:

See [Interrupt Step](#) and [List States](#) help pages.

Referencing Tables in Other Objects - SimBit

Referencing Tables in Other Objects

This SimBit project includes two models providing examples of using other object's table references.

1. **ModelUsesObjectDataTableReference:** Demonstrates a Model referencing a Data Table that lives in the ModelEntity definition for routing.
2. **ModelUsesObjectOutputTableReference:** Demonstrates a Model referencing an Output Table that lives in the ModelEntity definition for routing.

Model:

Problem:

Entities can visit five stations in any order and must visit all five. Entities should record which stations they have been to, so they are not routed there again.

Categories:

Data Tables

Keywords:

Data Table, Routing

Assumptions:

Entities can visit any of the five stations in any order and must visit all five.

Technical Approach:

Since each actualization, a member in the entity population, must keep track of its own information, it is easier to keep that information on the actualization itself, rather than in the Model. The ModelEntity definition will have a Data Table with a State Column that will change values when the entity has been to the corresponding location. Routing conditions will check the Entity's Data Table when considering a Candidate Node as a destination. A process will Search to check if all Stations have been visited and will send the entity to the Sink.

Details for Building the Model:

Facility Window: Objects and Properties

- See the model for approximate placement of all objects.
- Place a Source.
- Change the *Interarrival Time* to 'Random.Exponential(2)'.
- Place a Sink approximately 15 meters to the right from the first Source.
- Place five Servers in a circle below the Source and Sink. Name the Servers the following:
 - Station1
 - Station2
 - Station3
 - Station4
 - Station5
- Create a Node List containing all five of the Server Input Nodes. One way to do this is by selecting the Nodes in the Facility, right-clicking on the Node, and using the 'Add to Node List' option in the menu.

ModelEntity Definition

- Navigate to the ModelEntity Definition in the Navigation pane. Go to the Data tab, Tables view.
- Add a Data Table and name it "Stations Table". Create the following columns:
- String Property called "Stations"
- Boolean State Variable called "Done"
- Populate the Stations Table with the following information in each *Stations* row:
- Station1
- Station2
- Station3
- Station4
- Station5
- Return to the Model when done.

Processes Window

- Create a Process and name it "1_Server_AfterProcessing".
- Add an Assign step and name it "StationDone". Change the following properties for this step:
- *State Variable* = 'ModelEntity.StationsTable[ModelEntity.StationsTable.Stations.RowForKey(Server.Name)].Done'
- *New Value* = 'True'
- Create a Process and name it "2_EnteringServerOutputNode".
- Add a Search step and name it "StationsNotDone". Change the following properties for this step:
- Basic Logic
- *Collection Type* = 'TableRows'
- *Table Name* = 'ModelEntity.StationsTable'
- *Match Condition* = 'ModelEntity.StationsTable.Done == False'
- Advanced Options
- *Save Number Found* = 'Token.ReturnValue'
- On the Original branch of the Search step continue the process with a Decide step and name it "AllStationsFinished". Change the *Condition Or Probability* to 'Token.ReturnValue == 0'.
- On the True branch of the Decide step, add a SetNode step and name it "Sink". Change the *Node Name* property to 'Input@Sink1'.
- After the SetNode step, add a Transfer step and name it "FreeSpace". Change the *From* property to 'CurrentNode' and the *To* property to 'FreeSpace'.

Facility Window: Properties

- Change the properties on the five Servers to the following:
- Process Logic
- *Processing Logic* = 'Random.Triangular(0.5,1,3)'
- Add-On Process Triggers

- *After Processing* = '1_Server_AfterProcessing'
- Change the properties on the Output Node of each of the five Servers and the Output Node at the Source to the following:
- Routing Logic
- *Entity Destination Type* = 'Select From List'
- *Node List Name* = 'NodeList1'
- *Selection Goal* = 'Smallest Value'
- *Selection Condition* =
'ModelEntity.StationsTable[ModelEntity.StationsTable.Stations.RowForKey(Candidate.Server.Name)].Done == False'
- On the Output Nodes of each of the five Servers, change the Add-On Process Triggers *Entered* property to '2_EnteringServerOutputNode'.

Facility Window: Animation

- Place an instance of the ModelEntity.
- Use the Attached Animation ribbon to select Status Table. Draw a Status Table near the entity instance.
- Change the Status Table's *Table Name* property to 'ModelEntity.StationsTable'.

Discussion:

Data Tables in other objects can be referenced in the same ways as Data Tables in the main model. The only difference is the instance or realization the table belongs to must be specified. For example, a reference with a specific instance might look like this, 'CustomServer1.DataTable'. If the Token running the process has a reference to the object CustomServer1, then a generic expression, such as 'CustomServer.TableName', could be used and the expression will know to use the Token's reference to CustomServer1 when evaluated.

In this model, the ModelEntity realizations are a known reference where the Data Tables references are evaluated in the logic. The Table References are written with the generic 'ModelEntity.StationsTable' as it will know the exact entity enacting the logic and replace that information in for 'ModelEntity'.

For dynamic objects, realizations can each have unique tables. As we see in this example, the ModelEntity definition is set up with the schema for the Stations Table. As the model runs, assignments are made to the ModelEntity's Data Table to change the State Variable column value. This assignment is unique per each entity realization's Table, else each entity would be changing the same table and could not track their own information.

Similarly, instances of a fixed object will have their own tables. For example, imagine two instances of CustomServer which is a fixed object with a Data Table. CustomServer1 and CustomServer2 will have their own table information unique to their instance. Row references to tables can also be per instance. So, an entity might have a reference to CustomServer1.TableName as row 1, but also a reference to CustomServer2.TableName as row 5. This is still pointing to the same Data Table schema defined in CustomServer, but different rows per instance.

In this case, the benefit of using a Data Table inside the ModelEntity is the table can easily organize the unique information. Instead of using a Data Table, the entity could have State Variables which collect the information about the locations visited. But then each State Variable name, or if a vector, the index, would need to be specified when making the different State Assignments. Similarly, this information could have been kept at the top-level. However, since each entity's Stations visited will be unique, it would be difficult to track in the Model. Especially since there can be any number of ModelEntity realizations in the system at a time.

Notes:

When considering if it is good practice to have Data Table information inside an object definition or in the top-level, consider how the Data Table will be used and if the Data Table needs references to other pieces of information in the top-model. In this case, a String Property Column was used to store the names of the Stations in the ModelEntity's table because the ModelEntity definition has no knowledge of the Station instances in the main model. This could not be able to be an Object Reference Property column type in the ModelEntity. If the Data Table must reference other resources, such as Secondary Resources for processing, or Elements, such as Materials for a Task, it is likely better to keep the Data Table at the top-level model, as those other objects and elements cannot be referenced in a different definition.

A good practice is to keep a Data Table in the top-level and use Key Relationships to narrow down sections of a table for different objects in a model. For example, consider custom Server instances that will use a Data Table for their Task Sequence information. Each custom Server instance could have a Key that points to a Foreign Key that narrows down the

Task Sequence Table.

Model 2: ModelUsesObjectOutputTableReference

Problem:

Entities will have to keep track of their previous locations. Their destinations might not be known ahead of time, so the entities should be able to create a unique set of data based on what is available in the model.

Categories:

Data Tables

Keywords:

Output Table, Routing

Assumptions:

Entities can visit any of the five stations in any order and must visit all five.

Technical Approach:

Each ModelEntity realization will use an Output Table to keep track of the stations an entity has visited. When routing, the ModelEntity's Output Table will check with a RowForKey function to quickly find if a row already exists with that Server's name to decide if a candidate Node is viable. When deciding if the entity is done at all Stations, the ModelEntity will compare its Output Table's Available Row Count with the number of rows from the Model's Data Table that is acting as a Node List.

Details for Building the Model:

Initial Setup

- Use the "ModelUsesObjectDataTableReference" model as a starting point. You may right-click the model in the Navigation pane and select "Duplicate Model" if you wish to keep this in a separate model as illustrated.

ModelEntity Definition

- Navigate to the ModelEntity Definition in the Navigation pane. Go to the Data tab, Tables view.
- Add an Output Table and name it "OutputStationsTable". Create the following columns:
- String State Variable called "Stations Completed"
- Set this column as a Key.
- DateTime State Variable called "Time"
- Return to the Model.

Data Window: Tables

- Add a Data Table and name it "LocationsTable".
- Add a Node Property column, name it NodeLocations, and populate it with the following rows:
- Input@Station1
- Input@Station2
- Input@Station3
- Input@Station4
- Input@Station5
- Add an Output Table and name it "Verification Output Table". Add the following columns:
- String State Variable named "EntityName"
- String State Variable named "Station"

- DateTime State Variable named "Time"

Processes Window

- Update the 1_Server_AfterProcessing process. Delete the existing Assign step.
- Add an AddRow step, name it "MEOutputStationsTable", and change the following properties:
 - *Table Name* = 'ModelEntity.OutputStationsTable'
 - *Key Column Value* = 'Server.Name'
- After the AddRow, add an Assign step, name it "MEOutputTableTime", and change the following properties:
 - *State Variable Name* = 'ModelEntity.OutputStationsTable.Time'
 - *New Value* = 'TimeNow'
- Update the 2_EnteringServerOutputNode process. Delete the Search step.
- Change the Decide step's *Condition Or Probability* to 'ModelEntity.OutputStationsTable.AvailableRowCount == LocationsTable.AvailableRowCount'.
- Create a new process and name it "3_BeforeMEDestroyed".
- Add a Search step, name it "MEOutputTable", and change the following properties:
 - *Collection Type* = 'TableRows'
 - *Table Name* = 'ModelEntity.OutputStationsTable'
 - *Limit* = 'Infinity'
- On the Found branch of the Search step, add an AddRow step, name it "VerificationOutputTable" and change *Table Name* to 'VerificationOutputTable'.
- Continuing on the Found branch, after the AddRow step, add an Assign step, name it "VerificationInfo", and update the following properties:
 - *Assignments (More)*
 - *State Variable Name* = 'VerificationOutputTable.EntityName'
 - *New Value* = 'Entity.Name'
- Item 2
 - *State Variable Name* = 'VerificationOutputTable.Station'
 - *New Value* = 'ModelEntity.OutputStationsTable.StationsCompleted'
- Item 3
 - *State Variable Name* = 'VerificationOutputTable.Time'
 - *New Value* = 'ModelEntity.OutputStationsTable.Time'

Facility Window: Properties

- On the Output Node of the five Servers and the Output Node at the Source, update the following properties:
 - *Node List Name* = 'LocationsTable.NodeLocations'
 - *Selection Condition* = 'ModelEntity.OutputStationsTable.StationsCompleted.RowForKey(Candidate.Server.Name) == 0'
- On Sink1, change the Add-On Process Triggers Entered property to '3_BeforeMEDestroyed'.

Facility Window: Animation

- Remove the attached Status Table animation from the ModelEntity.

Discussion:

Table functions such as AvailableRowCount and RowForKey work with Table References that are referencing a Table in another Object. These functions are helpful as there is information in the top-level model, such as the Table of Nodes, that we can use to compare to.

When routing, the Routing Logic *Selection Condition* is able to check if the ModelEntity has an existing row in its Output Table with the Candidate Node's Server Name. When the Table's RowForKey function does not find a row in the Table, it returns a '0'. If we are looking for the Candidate Server's name and the RowForKey returns '0', we know the entity can still route to this destination.

When checking if all the Stations have been visited, this time, each row does not need to be Searched. If the Entity has visited all the destinations, the number of Rows in the ModelEntity's Output Table should be equal to the number of rows from the model's Data Table that was used as a routing Node list.

Notes:

Tables in other objects can use Object Reference State Variable columns and have object references passed in from the top-model with State Assignments. In the Output Table in the ModelEntity definition, the Stations State Variable column could have instead been an Object Reference and the Station Server reference could have been passed in. However, only certain types of State Variable columns can be a Key, and since the RowForKey function was needed to easily find rows, instead this was changed to a String, and the object's string name was used instead.

Status Table animation did not work with Output Tables prior to Simio version 17.262. To verify the ModelEntities are capturing information in their OutputTables as expected, a process is set up to Search and write out their information into an Output Table on the top-level model. Once an entity is destroyed, its information is destroyed too so its Output Table information would no longer be accessible. In the model, the logic is set up to check this ModelEntity's Output Table information at the Sink and write it to the Model's Verification Output Table. This logic could be changed so that the ModelEntity's Output Table information is written out to the Verification OutputTable at the same time it is added to the ModelEntity's Output Table in the 2_EnteringServerOutputNode process.

This model was updated in Simio version 17.263 to include a Status Table on the DefaultEntity to show the Output Table belonging to that entity actualization. Once the entity actualization is destroyed, that Output Table belonging to that entity no longer exists. The approach above to write to a different Output Table in the top-level model can be used to store the information and retain it after the entity is destroyed.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ResourcesWithWorkSchedules - SimBit

Problem:

There are two employees that work at a ticket counter. The employees follow two different work schedules.

Categories:

Schedules / Changeovers

Key Concepts:

Current Symbol Index, ObjectList, Resource, Schedules, Secondary Resources

Assumptions:

The work schedule is the same each day – it follows a one day cycle.

Technical Approach:

A server represents the ticket counter and a resource object is used to represent an employee. The capacity of each resource is determined by a work schedule.

Details for Building the Model:

Simple System Setup

- Place a Source, Server, Sink and two Resource objects into the Facility Window. Change the *Name* of the resources to 'Employee1' and 'Employee2'.
- If you'd like to change the symbol representing the entity, place a ModelEntity object from the Project Library into the Facility Window and select a new symbol from the symbol tab.

Specifying Information about the Employees

- Click on the Definitions tab, select the Lists panel and create a new Object List with *Name* 'ObjectList1'. Click on the text boxes of the list to get a drop down that will show all the available objects from the model. Put both 'Employee1' and 'Employee2' objects in the list.
- In the Data tab, Schedules panel, add two new schedules by clicking on the Work Schedules button on the Schedule tab and entering the schedule *Name* 'Schedule1' and 'Schedule2'.
- Set the number of *Days* in each schedule to '1'. This indicates that this one day schedule repeats itself for each day the simulation is run.
- Create a two Day Patterns by clicking on the Day Patterns tab. *Name* the two patterns 'DayPattern1' and 'DayPattern2'.
- Click on the '+' sign to expand the Work Periods
- Create a pattern that has a *Value* of '1' (on shift) in these time slots: 8:00 AM - 10:00 AM, and 12:00 PM -2:00 PM for 'DayPattern1'. For 'DayPattern2', specify a *Value* of '1' from 8:00 AM - 12:00 PM for. Leave everything else empty, which indicates Off Shift times.
- Back within the Work Schedules tab, select 'DayPattern1' for 'Schedule1' and 'DayPattern2' for 'Schedule2'.
- Select each Resource and change the *Capacity Type* to 'Work Schedule' and select the appropriate *Work Schedule*, either 'Schedule1' or 'Schedule2'.

Using the Employees at the TicketCounter

- In the Facility Window, within the TicketCounter (Server), open the Secondary Resources and find the Resource for Processing section of properties. Change the *Object Type* to 'FromList' and the *Object List Name* to 'ObjectList1'.

Animating the Employees

- In order to animate the Resource objects to change color when they are busy, click on one of the Employees in the Facility Window. Select Add Additional Symbol from the symbols tab of the Ribbon. With the Active Symbol icon reading 2 of 2, select a new color (i.e. green) from the Color icon in the symbols tab and then click on the center circle of the Resource object. By default, this resource will now change to green when it is busy. (This is controlled by the default expression in the current symbol index property of the resource object).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RotatingVehicleExample - SimBit

Problem:

I want a robot to pick parts from the source, to process them at a server and drop them at the sink.

Categories:

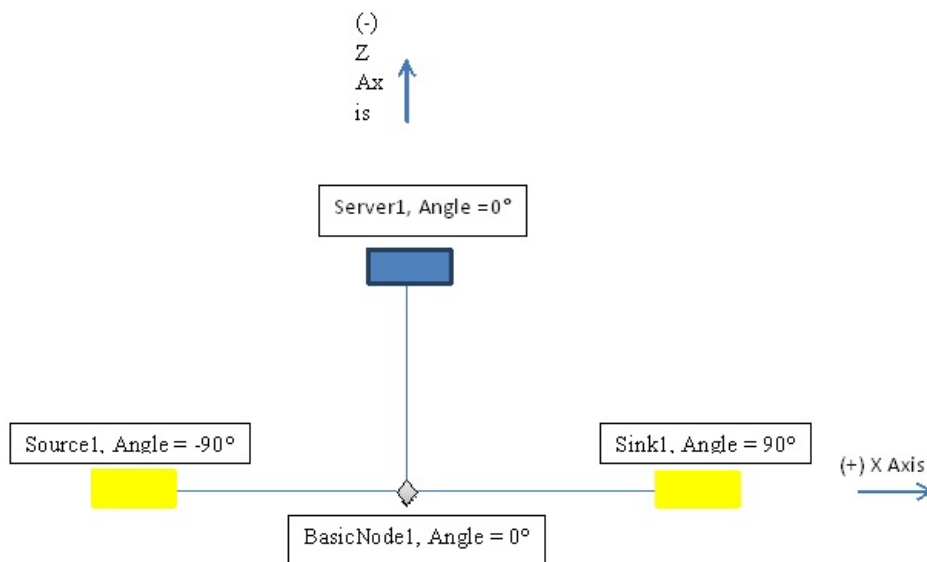
Add-On Process Logic, Vehicles

Key Concepts:

Assign Step, Create Step, Delay Step, Execute Step, Movement.Heading, Movement.X, Movement.Y, Movement.Z, Token Wait Action, Vehicle

Technical Approach:

Add a Vehicle and initialize its coordinates (X, Y, Z) to a basic node placed earlier in the model. Assign new angular values to Movement.Heading and delay the vehicle to rotate the robot.



Simple System Setup

- Add a Source, a Server, a Sink and a Basic Node to the Facility Window.
- Set the Object nodes location to the following values
 - Node_____Location
 - Output@Source1___X=-10, Y=0, Z=0
 - Input@Server1___X=-0, Y=0, Z=-10
 - Output@Server1___X=-0, Y=0, Z=-10
 - Input@Sink1___X=10, Y=0, Z=0
 - BasicNode1___X=0, Y=0, Z=0
- Connect the objects together with Paths. To draw a circular path, drag in an ellipse and use path vertices to draw an arc around it then delete it.

- Set the Source Arrival Mode to 'On Event' and set the Event Name to 'Input@Sink1.Exited'. This means that an entity won't be created until the previously created entity exits the system.
- Set the Initial Number Entities to '1' to create the first entity.

Vehicle Details

- Add a Vehicle to the Facility window with the Name 'Vehicle1' and Initial Desired Speed of '0.2'.
- In the Processes window, select the process OnRunInitilized.
- Add a Create step and set its Object Instance Name to 'Vehicle1'.
- Add an Assign step at the Created branch of the Create Step to initialize the created vehicle location and heading at BasicNode1.

Source Add-On Process

- Within Source1, create an add-on process for Created Entity named 'Source1_CreatedEntity'.
- Within that process in the Processes window, place a Seize step to seize the vehicle.
- Place an Execute step to execute 'Source_Process' to rotate the Vehicle towards the Source object.
- Place a Delay step to model the vehicle loading time of '0.5' minutes.
- Place another Execute step to execute the 'Server_Process' that rotates the Vehicle towards the Server object.
- Change the Token Wait Action of the Execute Step to 'None (Continue)' so that the entity does not wait for the Vehicle to turn before going to the Server.

Server Add-On Process

- Go back to the Facility window and within Server1, create an add-on process at Processed.
- Within the Processes window in this new process, place a Delay step to represent the processing time at Server1, '2' minutes.
- Place an Execute step to execute 'Sink_Process' to rotate the Vehicle towards the Sink object.
- Change the Token Wait Action of the Execute Step to 'None (Continue)' so that the entity does not wait for the Vehicle to turn before going to the Sink.

Sink Add-On Process

- Within the Facility window, highlight Sink1 and create an add-on process at Entered.
- Place a Delay step to model the robot unloading time, '0.5' minutes.
- Place a Release step to release the vehicle.

Rotating the Vehicle

There are three new processes that need to be added to the Processes window, Source_Process, Server_Process and Sink_Process. These processes all use the same logic to rotate the vehicle. The only difference resides in the destination heading at the Decide step and the increments sign (10 or -10) at the Assign step.

- Within each process, place a ConditionBased Decide step to check if the vehicle has reached the specified object. For Source_Process, the Expression is set to 'Vehicle1[1].Movement.Heading == -90'. For Server_Process, the Expression is 'Vehicle1[1].Movement.Heading == 0'. For Sink_Process, the Expression in the Decide step is 'Vehicle1[1].Movement.Heading == 90'.
- Place an Assign step at the False branch of each Decide Step to increment/decrement the vehicle heading by 10. For the Source_Process, the New Value is 'Vehicle1[1].Movement.Heading - 10', while for Server_Process and Sink_Process, the New Value is 'Vehicle1[1].Movement.Heading + 10'.
- Place a Delay step with a Delay Time of '3' and Units of 'Seconds' after the Assign step in each process to adjust the rotation speed. Connect the Delay steps back to the entry points of the Decide steps for each process.

Enhancing the Animation

- In this model, we have placed the objects in a way such as Source1 is at -90°, Server1 is at 0° and Sink1 is at 90°. However, if we don't know the angles we could use the Arctangent to find the object's heading. It is referred to as Math.Atan() in Simio. Also, to find out the distances, we could use the object's Location function.
- In order not to have the robot rotate about its center, we edited the symbol by adding a small polyline to the robot:
 - Click on the project name.
 - Select Symbols.
 - Right click on the robot symbol and select Edit.
 - Draw a small polyline. Leave one third of the robot length between the robot edge and the polyline.

Send comments on this topic to [Support](#)

RoutingWithoutPaths - SimBit

Problem:

My entities travel through a sequence of nodes and there is no travel time in between nodes. Instead of using zero-time TimePaths or Connectors, I'd like to have my entities move directly from node to node without traveling on a link.

Categories:

Data Tables, Decision Logic -- Processing

Key Concepts:

Add-On Process, Data Table, Expression Property, Expression Property, Node Property, On Entered, Real State Variable, Search Step, Transfer Step

Assumption:

The routing information comes from a Simio data table.

Technical Approach:

A [Data Table](#) contains two columns; one which is a node property which contains the routing information and the other is an integer property which indicates the part type. Each entity in the system has a part type and this is used to determine the routing information from the Data Table. A [Transfer Step](#) transfers the entities from Node to Node, eliminating the need for links.

Details for Building the Model:

Simple System Setup

- Place two standard Sources, four Servers and one Sink object into the Facility window.
- Place two ModelEntity objects into the Facility window. Name one ModelEntity "PartA" and the other "PartB". Color PartB red by selecting the entity in the Facility window and selecting the color red from the Color dropdown in the Decoration section of the ribbon.
- Set *Entity Type* in Source1 to 'PartA' and to 'PartB' in Source2.

Create the Data Table

- Create a new Data Table by going to the Data tab and clicking on *Add Data Table* from the Tables section of the Schema ribbon.
- Add a new column to the table by selecting Expression from the Property dropdown in the Columns section of the Schema ribbon. Name this property 'Part Type' by changing the Name property in the properties window.
- Add another column to the table by selecting Node from the Object Reference Property dropdown in the Columns section of the Schema ribbon. Name this property 'MyDestination'.
- Fill in the table with the following information:

<u>MyDestination</u>	<u>PartType</u>
Input@Server1	1
Input@Server3	1
Input@Sink1	1
Input@Server2	2
Input@Server4	2
Input@Sink1	2

Create Custom States and Properties

- Select the ModelEntity object in the Navigation window in the upper right side of the interface. While it is selected, go to the Definitions window. You are now in the Definitions window of the ModelEntity object. You need to add a new State and a new Property to this object.
 - Click on the Properties panel on the left and create a new Expression property by selecting Expression from the Standard Property dropdown in the Add section of the Properties ribbon. Name this new property 'PartType'.

-
- Click on the States panel and add a new State by clicking on Discrete State in the Edit section of the States ribbon. Name this new state 'CurrentIndex' and set its *Initial State Value* property to a value of '1'.

Create Process Logic

- Back in the main Model (select Model in the Navigation window), go to the Processes window and create a new process by clicking on Create Process in the Process section of the Process ribbon.
- Place a [Search](#) step. This step will search the Data table and find the row that matches the criteria specified in the Match Condition property. It will save the row number into a state variable so we can keep track of our current location in the table.
 - Set the *Collection Type* to 'TableRows' and *TableName* to 'Table1'.
 - Set the *MatchCondition* to 'ModelEntity.PartType == Table1.PartType'.
 - Set the *Starting Index* to 'ModelEntity.CurrentIndex'
 - Set the *Save Index Found* to 'ModelEntity.CurrentIndex'
- In the Found segment leaving the Search Step, place an Assign Step. Set the *State Variable Name* to 'ModelEntity.CurrentIndex' and the *New Value* to 'ModelEntity.CurrentIndex + 1'. If a row is found that matches this entity part type, this will increment the CurrentIndex of this entity to keep track of its routing location within the table.
- After the Assign Step, place a Transfer Step. Set *From* to 'CurrentNode' and *To* to 'Node' and *NodeName* to 'Table1.MyDestination'.
- This process is to be called every time an Entity is to move to its next destination. So, this process needs to be set on every Output Node's Entered Add-On Process trigger.

Embellishments:

Experiment with changing the routing within the Table.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ScheduledMaterialArrivals - SimBit

Problem:

I have a data file specifying the times that materials arrive to the system.

Categories:

Data Tables

Key Concepts:

Data Table, Event, Material Element, Numeric Property, Process Triggering Event, Produce Step, Server, Source, Timer Element

Technical Approach:

Create a table that specifies the *Arrival Time* and *Quantity* of each delivery. Create a Timer that triggers a Produce step in a process to introduce Material into the system.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink in the Facility window, link together with Paths.
- Specify that the Source creates 1 arrival with 6 entities per arrival.
 - Change *Entities Per Arrival* to '6'.
 - Change *Maximum Arrivals* to '1'.

Defining the Data Table with Arrival Information

- Open the Data window of the model and select the Tables panel. Use the Add Data Table button to create a table with the *Name* 'MaterialDeliverySchedule' and include the following columns:
 - (Real)'ArrivalTime': 1, 1.5, 0.5
 - (Integer)'Quantity':3, 1, 2

Material Arrivals using Data Table Information

- In the Definitions window, select the Elements panel:
 - Add a Material element *Name* Widget
 - Add a Timer *Name* 'MaterialArrival'. Set the *Interval Type* to 'ArrivalTable'. Set the *Arrival Time Property* to 'MaterialDeliverySchedule.ArrivalTime'.
- In the Processes window:
 - Create a new process *Name* 'MaterialHasArrived'. Set the *Triggering Event Name* to 'MaterialArrival.Event' (the event that is fired by the Timer).
 - Add a Produce step to that process. Set the *Production Type* to 'Material'. Set the *Material Name* to 'Widget'. Set the *Quantity* to 'MaterialDeliverySchedule.Quantity'.

Using the Material Information in the Model

- In the Facility window:
 - Select Server1 and change the *Process Type* to 'Task Sequence'.
 - Select *Processing Tasks* and open the Repeating Property Editor.
 - Change the *Material Name* to 'Widget'.
 - Change the *Quantity* to '1'.

Enhancements:

- Add a Status Label showing the Input Buffer Contents of Server1.
- Add a Status Label showing the number of Widgets in Stock.

Send comments on this topic to [Support](#)

ScheduledMaterialArrivalsWithDeviation - SimBit

Problem:

I have a machine that consumes material and material is scheduled to be available (produced) at specific times. I want to see how variation around the material arrival times affects my system.

Categories:

Arrival Logic, Data Tables

Key Concepts:

Add-On Process, Arrival Time Deviation, Data Table, Material Element, Numeric Property, On Exited, Produce Step, Server, Source, Status Plot, Task Sequence

Assumptions:

We will assume a constant Entity *Interarrival Time* and a constant *Processing Time* so we can see the affects of the material arrival variation.

Technical Approach:

A Server object is used and it consumes 1 unit of Material with a '1' minute *Processing Time*. We set the Material arrival times to be scheduled to arrive just as the material is due to run out. We will use 2 systems: one with no variation around the material arrival time, and one without.

Details for Building the Model:

Simple System Setup

- Place 4 Sources, 2 Servers and 4 Sinks from the Standard Library into the Facility window.
- Connect Source1, Server1, and Sink1 with Paths. Connect Source2 and Sink2 with a Path, as well.
- Place these two groups of objects on the left hand side of the Facility window.
- Now replicate this system with the remaining objects on the right side of the Facility window. Connect Source3, Server2, and Sink3 with paths, as well as Source4 and Sink4.
- Rename Source2 'MaterialArrival1' and Source4 'MaterialArrival2'.

Creating the Material Arrival Table

- In the Tables panel of the Data tab, click Add Data Table. *Name* this table 'MaterialArrivalTable'.
- Under Standard Property, click 'Real' to add a Real Property.
- *Name* this property 'ArrivalTime' and set its *Unit Type* to 'Time' and leave the *Units* 'Hours'.
- We want the Material to arrive every 15 minutes, so we will fill in the table with arrival table with arrivals every 0.25 hours (0.25, 0.5, 0.75, 1, 1.25, etc...) up to 3 hours.

Creating the Material Elements

- Open the Definitions window and select the Elements panel. In the Supply category, click on the Material element twice adding 2 Materials: 'Material1' and 'Material2'.
- Set both of their *Initial Quantities* to '15'.

Process to Produce Material

- The Source-Sink portion of each system is designed to illustrate Material arrivals. So we have to trigger a Produce step somehow.
- In MaterialArrival1, create a process on the *Exited* Add-On Process Trigger.
- In this MaterialArrival1_Exited process, place a Produce Step. Set the *Production Type* to 'Material', the *Material Name* to 'Material1' and the *Quantity* to '15'.
- Do the same thing for MaterialArrival2, create a process in MaterialArrival2's *Exited* Add-On Process Trigger.
- In the MaterialArrival2_Exited process, place a Produce step. Set the *Production Type* to 'Material', the *Material Name*

to 'Material2' and the *Quantity* to '15'.

Defining the Object Properties

- Within the Facility window, edit both Source1 and Source3 (either by pressing 'Ctrl' and selecting both or by editing them separately) so that the *Interarrival Time* is '1' and *Units* are 'Minutes' and that the *Maximum Time* under Stopping Conditions is '3' and *Units* are 'Hours'.
- In Server1, change the *Process Type* to 'Task Sequence' and for the *Processing Tasks* open the Repeating Property Editor by clicking on the button with the three little dots. Add a new task and change the *Name* to 'Consume', the *Processing Time* to '1'. Under the Material Requirements, change the *Material Name* to 'Material1' and the *Quantity* to '1'.
- In Server2, change the *Process Type* to 'Task Sequence' and for the *Processing Tasks* open the Repeating Property Editor by clicking on the button with the three little dots. Add a new task and change the *Name* to 'Consume', the *Processing Time* to '1'. Under the Material Requirements, change the *Material Name* to 'Material2' and the *Quantity* to '1'.
- In both MaterialArrival1 and MaterialArrival2, set the *Arrival Mode* to 'Arrival Table' and the *Arrival Time Property* to 'MaterialArrivalTable.ArrivalTime'.
- In MaterialArrival1, leave the *Arrival Time Deviation* property set to '0.0'. But in MaterialArrival2, set it to 'Random.Uniform(-0.2,0.2)' and *Units* to 'Hours'. This means that the Entity will try to arrive via the time specified in MaterialArrivalTable.ArrivalTime, but the actual arrival time will be anywhere from 0.2 hours early to 0.2 hours late.

Enhancements:

We can add some animation, to show the current remaining number of Materials while we're running this model.

Choose *Status Plot* from Animation ribbon, create a Status Plot, with the *Title* 'Constant Material Arrivals', then set the *X Axis* label to be 'Time' and the *Y Axis* label to 'Quantity'.

Then click on the plot, go to Properties window, add 2 rows in the *Additional Expressions*. One row has the *Expression* set to 'Material1.QuantityAvailable', and the *Label* set to 'Material1'. The other row has the *Expression* set to 'Server1.InputBuffer.Contents', and the *Label* set to 'Buffer Contents'. Then we'll have two animation lines within the plot, one represents the number of Material1 units remaining and the other one represents the number Entities in Server1's Input Buffer.

Repeat this process for the Variable System instead using Server2 and Material2 in the *Expressions*.

These two graphs illustrate how variation upon arrivals can really wreak havoc on the system by occasionally starving the system of material and allowing a queue to form at the Server. With constant arrival times and processing times, it is impossible to recover from these queues and waiting times become significantly longer.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ScheduleUsesAProperty - SimBit

Problem:

I would like to use a Work Schedule where the capacity within the schedule is dependent on a user input property.

Categories:

Schedules / Changeovers

Key concepts:

On Shift, Off Shift, Schedules, Server, WorkSchedule

Technical Approach:

This is a simple Source, Server, Sink model where the Server follows a Work Schedule. The capacity within the Work Schedule is set by the value of a property. Therefore, when the user changes the value of the property for the simulation run, the capacity of the Server uses this value since it is reading this value from within the Work Schedule.

Details for Building the Model:

Simple System Setup

- Place a Source object, a Server object and a Sink object and connect the Source to the Server with a Path and connect the Server to the Sink with a Path.
- Set the *Interarrival Time* of the Source to 'Random.Exponential(.2)' minutes.
- Set the *Capacity Type* property of the Server object to 'WorkSchedule' and set the *Work Schedule* property to 'StandardWeek'.

Defining the Property

- Go to the Properties window by selecting Properties along the left panel from within the Definitions window.
- Select Integer from the Standard Property drop down in the Properties ribbon. Rename this new property, 'ServerCapacity'.

Defining the Schedule

- Go to the Schedules window by selecting Schedule along the left panel from within the Data window.
- Click onto the Day Patterns tab. You'll see the 'StandardDay' day pattern. Click onto the '+' to expand the pattern.
- The *Value* column has a value of '1' by default. This means that when the simulation clock is between the hours listed, the value of '1' is set to the capacity of resource that is following this Day Pattern within the a specified Work Schedule. Replace the '1' with the name of the new property that was created, 'ServerCapacity'. Have the schedule use this new property for both the 8am-12pm timeframe and the 1pm-5pm timeframe. Note that this Day Pattern named 'Standard Day' is then referenced within the 'Standard Week' work schedule.

Setting the Model's ServerCapacity Property

- Before you run the model, set the value of the *ServerCapacity* property by finding it in the Model Properties. Either click into the open space of the Facility window to bring up Model Properties, or right click onto the name of the Model from within the Navigation window and select Model Properties. You'll see the new property, ServerCapacity, and you can set the value and run the model to see how the capacity of the Server is set to the value of this new property.
- You might consider having the model start running at 8:00am, which is when the Work Schedule begins On Shift. This is controlled on the Run ribbon, under Run Setup.

Embellishments:

Create an experiment and create different scenarios where the *ServerCapacity* property has different values and compare the throughput and average time in system of the entities. To create an Experiment, either right click onto the name of the Model in the Navigation Window and select New Experiment or click onto New Experiment from the Project Home ribbon.

SearchStationElement - SimBit

Problem:

I'd like to use a Station element and I'd like to search the entities in a Station and find one that meets my specific criteria.

Categories:

Add-On Process Logic, Combining and Separating Entities, Decision Logic -- Processing

Key concepts:

Active Symbol, Add-On Process, Batch Step, Decide Step, EndTransfer Step, From Current Station, Match Condition, On Created Entity, On Event Arrival Mode, ReturnValue, Search Step, Selection Weight, State Assignments, Station Element, Transfer Step

Technical Approach:

Each time Source1 creates an entity, another entity is created via a Create Step within a process. It is given a new picture (30% - green, 30% - red, 40% - blue). This new entity is then transferred into Station1.

Another Source sends an entity to Server1. Before processing at Server1, the entity searches Station1 to try and find a matching entity (ModelEntity.Picture) and if a match is found, they are batched, processed and they move to the Sink named Batched. If there is not currently a match in the Station, the entity is processed and moves to the Sink named Single.

Details for Building the Model:

Creating Entities for the Station

- Place a Source object and a Sink object and connect them with a Path.
 - In the Source object (Source1), set the *Arrival Mode* property to 'On Event'. Set the *Initial Number Entities* to '10'. Set the *Triggering Event Name* to 'Station1.Exited'.
 - This Source object will start by creating 10 entities, but will then wait to create another entity when an entity exits Station1.
 - Create a new Add-On Process in the *Created Entity* Add-On Process trigger. We will populate this process later when we get to the Processes window.
- Place a ModelEntity from the Project Library into the Facility window.
 - With the DefaultEntity selected, click onto Add Additional Symbol (twice) in the Symbols ribbon so that there are three different symbols now available for this DefaultEntity.
 - Make the second symbol active by selecting 2 of 3 from the Active Symbol drop down in the Symbols ribbon (the entity should still be selected in the Facility window). Click onto Color in the Symbols ribbon and select Red and then click onto the DefaultEntity so that the color turns Red.
 - Make the third symbol active by selecting 3 of 3 from the Active Symbol drop down in the Symbols ribbon (the entity should still be selected in the Facility window). Click onto Color in the Symbols ribbon and select Blue and then click onto the DefaultEntity so that the color turns Blue.
 - You should now have three different symbols for the Default Entity – green, red and blue, where green has a '0' next to it in the Active Symbol drop down, Red has a '1' and Blue has a '2'.

Creating Entities for Server Processing

- Place another Source object, a Server object and two Sink objects into the Facility window. Change the *Name* of one of the Sinks 'Batched' and the other 'Single'.
- Connect Source2 to Server1 with a Path and connect Server1 to each of the Sink objects with a Path.
- In Source2, create a new State Assignment in the Before Existing Assignment property. Use the Discrete distribution to assign a random value to *State Variable Name* of 'ModelEntity.Picture', which is a built-in state on the ModelEntity object (this also controls the color of the DefaultEntity symbol). The *New Value* of the assignment is 'Random.Discrete(0, .3, 1, .6, 2, 1)'. This will assign 30% of the entities green, 30% red and 40% blue.
- In Server1, set the *Initial Capacity* property to '5' and the *Processing Time* property to '.1' minutes. Create a new Add-On Process in the *Processing* Add-On Process Triggers. We will populate this process later when we get to the Processes window.
- Select the path that leads from the Server to the Batched Sink. Set the *Selection Weight* of this path to

'ModelEntity.BatchMembers > 0'. This will force only entities that have another entity in its BatchMembers queue (i.e. batched entities) to follow this path.

- Select the other path which leads from the Server to the Single Sink. Set the *Selection Weight* of this path to 'ModelEntity.BatchMembers == 0'. This will force only entities that do not have an entity in the BatchMembers queue (i.e. unbatched, single entities) to follow this path.

Creating Elements in the Definitions Window

- Within the Definitions tab, select the Elements panel. Click onto Station in the Elements ribbon to create a new Station element.
- Click onto Batch Logic in the Elements ribbon to create a new Batch Logic element. This will be used within process logic to batch entities together.

Process Logic within the Processes Window

- Click onto Create Process in the Processes ribbon to create a new process, named Process1.
 - Find the process property named Triggering Event Name and populate it with the event, 'Station1.Entered'. This indicates that this process will be executed whenever an entity enters Station1.
 - Place an EndTransfer step into this process. This indicates that the transfer has been completed and that the entity is now inside the station.
- Find the Source1_CreatedEntity process. This is the process that is executed each time Source1 creates an entity.
 - Place a Create Step. Set the *Entity Type* property to 'DefaultEntity'.
 - In the Created exit leaving the Create step, place an Assign step. Set the *State Variable Name* property to 'ModelEntity.Picture' and the *New Value* property to 'Random.Discrete(0, .3, 1, .6, 2, 1)'. This will assign 30% of the entities to green, 30% - red, 40% - blue.
 - After the Assign Step, place a Transfer step. Set the *From* property to 'FreeSpace'. Entities are always created in FreeSpace. Set the *To* property to 'Station' and the *Station Name* to 'Station1'.
- Find the Server1_Processing process. This is the process that is executed each time an entity is about to start processing on Server1.
 - Place a Search step. This step will search the Station to try and find an entity with the same color as the entity about to process at the Server. If a matching entity is found, its ModelEntity.ID is saved into the Token.ReturnValue. Set the *Collection Type* to 'QueueState' and the *Queue State Name* to 'Station1.Contents'. Set the *Match Condition* to 'ModelEntity.Picture == Candidate.ModelEntity.Picture'. Set the *Search Expression* to 'ModelEntity.ID'.
 - Place a Decide step out of the Original exit leaving the Search step. Set the *Decide Type* to 'ConditionalBased' and the *Expression* to 'Token.ReturnValue == 0'. This checks to see if the Search Step found a matching entity. If the ReturnValue of the token is anything other than zero, then the Search step found a match and put the ModelEntity.ID of that entity into the Token.ReturnValue.
 - Place a Batch step out of the False exit leaving the Decide step. Set the *Batch Logic Name* to the name of the Batch Element that was created earlier. Set the *Category* to 'Parent'. This entity (which is the original entity at the Server), will become the parent in the batch.
 - Place a Transfer step out of the Found exit of the Search step. Set the *From* property to 'CurrentStation' and the *To* property to 'FreeSpace'. This Step is only executed if a matching entity is found via the Search step. It will transfer that entity out of the Station and into FreeSpace.
 - Place a Batch step after the Transfer step. Set the *Batch Logic Name* to the name of the Batch Element that was created earlier. Set the *Category* to 'Member'. This entity (which is entity that was just transferred out of the Station), will become the member in the batch.

Animating Queues

- Animate the contents of the Station by clicking onto the Animation ribbon and selecting Detached Queue. Draw a queue into the Facility window and set the *Queue State* to 'Station1.Contents'.
- Animate the batched items that are attached to the ModelEntity by selecting the DefaultEntity in the Facility window and selecting Batch Members from the Draw Queue dropdown in the Symbols ribbon. Draw the queue somewhere near the instance of DefaultEntity. This is an attached queue.

SearchTables - SimBit

Problem:

I would like to be able to search through a Data Table and select a row based on certain criteria.

Categories:

Arrival Logic, Data Tables

Key Concepts:

Add-On Process, Assign Step, Collection Type, Current Symbol Index, Data Table, Match Condition, ModelEntity, Node Property, Numeric Property, On Created Entity, Priority, Search Step, SetNode Step, TableRows, TransferNode

Technical Approach:

A Data Table contains an Integer property and a Node Property. Each entity is assigned a value of 1, 2, 3 or 4 to its priority. A token searches the Data Table on behalf of the entity to find the row that matches the value of its priority. Once the appropriate row is found, the destination of the entity is set to the Node that is listed in the Data Table.

Details for Building the Model:

Simple System Setup

- Place a Source and a Sink into the Facility window. Place four Transfer nodes in between the two objects - the nodes should be in parallel. Connect the Source to each of the four nodes and connect each node to the Sink.
- Place a ModelEntity into the Facility window. To help with model verification, we'll have the entity change color depending on its priority (and therefore its destination node).
 - Select the DefaultEntity in the Facility window and click on the Add Additional Symbol icon in the ribbon. You will see that the icon to the right, Active Symbol, now displays 2 of 2. Click on the Color icon, select a color and then click back into the Default Entity in the Facility window. The symbol should change color. You have now changed the color of whatever symbol is active (if it says 2 of 2, then you changed the color of the second symbol that is associated with this Default Entity object).
 - Add two more symbols to this object by repeating the steps above and assign a unique color to each of the four symbols.
- Within the ModelEntity properties, under the Animation section, specify the *Current Symbol Index* property as 'ModelEntity.Priority - 1'. The steps in the process logic below will assign the priority to the entity based on a random discrete distribution and the animation entity picture will then change as priority is assigned.

Create Data Table

- Go to the Data window, and within the Tables panel, click the Add Data Table icon in the ribbon to add a new table.
- Add a column that is an Integer property by selecting Integer from the Standard Property drop down in the ribbon. The column can be renamed to Priority.
- Add a second column to the table – a Node Property, by select Node from the Object Reference drop down in the ribbon. The column can be renamed to Node.
- Fill the table with data – such as:
 - 1 , TransferNode1
 - 2 , TransferNode2
 - 3 , TransferNode3
 - 4 , TransferNode4

Add Process Logic

- From within the Facility window, click on the Source and create a new Add On Process for the trigger called *Created Entity*. (select Create New from the drop down)
- From within the Processes window, you should see the new Process that was created.
 - Place an Assign step in the process. The *StateVariable Name* should be ModelEntity.Priority and the *New Value* should be Random.Discrete(1, .25, 2, .5, 3, .75, 4, 1). Each entity has a 25% chance of being assigned a Priority of 1, 2, 3 or 4.

-
- Place a Search Step after the Assign Step. The *Collection Type* property should be set to 'TableRows' and the *Table Name* property should be the name of your Data Table (Table1). Set the *Match Condition* to be 'ModelEntity.Priority == Table1.Priority'. This will ensure that the entity is set to read the row that matches its priority value.
 - In the Found segment of the Search Step, place a SetNode step. The *Destination Type* property is 'Specific' and the *Node Name* property should be set to 'Table1.Node'.

Embellishments:

Change the matching condition for searching the table and add additional information in the Data Table to be used in the model.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SearchTableUponEnteringObject - SimBit

Problem:

Each object in my model reads data from a table to determine its processing time and the number of resources it must seize before processing. I do not want to hard code the row where each object should look for its data, but instead the entity should search the table whenever it enters the Server to find the appropriate data.

Categories:

Data Tables

Key Concepts:

Add-On Process, Blocked Destination Rule, Cyclic, Data Table, Entity, Entity Destination Type, Expression Property, ID, NodeList, Numeric Property, On Entered, Location.Parent, Real State Variable, Resource, RowNumber, Save Index Found, Search Step, Secondary Resources, Select Available Only, Selection Goal, Server, SetRow Step, String Property, Token, Token State, TransferNode

Technical Approach:

This model contains three Servers in parallel. An entity travels to one of the three Servers, seizes a certain number of Resources, delays for a certain amount of time and then releases the Resources. The number of Resources and the delay duration is different for each Server. This information is stored in a table. To find the appropriate row in the table, the token must search the table and find the row that contains the information for that particular Server. The search uses the ID of the Server object to find the appropriate row in the table.

Details for Building the Model:

Basic Model Setup within the Facility window

- Place a Source followed by a Transfer Node and connect them with a Connector.
- Place three Servers in parallel and another Transfer Node after these Servers. Connect the first TransferNode to all three Servers with Paths and connect the Servers to the second Transfer Node with Paths.
- Place a Sink and connect to the second Transfer Node to the Sink.
- Go to the Definitions window and create a new Node List from the Lists panel on the left. Add Input@Server1, Input@Server2 and Input@Server3 to the Node List.
- Back in the Facility window, set the *EntityDestinationType* property of the first Transfer Node to 'Select From List' and set *Node List Name* to your new List. Set *Selection Goal* to 'Cyclic'.
- Place a standard Resource object and set the *Initial Capacity* to '6'. Name this 'Resources'.

Creating the Table within the Data window

- Create a new Data Table by clicking the 'Add Data Table' icon in the ribbon. Name this table 'ServerData'
- In ServerData, add four columns; An Object property named 'ServerName', an Integer Property column named 'NumberOfResources' and an Expression Property column named 'Hours'.
- Place the following data into the Table:
 - Server1, 3, .1
 - Server2, 2, .05
 - Server3, 1, .02

Adding a New Token in the Definitions window

- Go to the Tokens panel within the Definitions window and click on the 'Add Token' icon in the ribbon.
- Select the name of the new token and then click on the 'Real' icon in the ribbon to add a new Real State to this token. Name this new state, 'RowNumber'. The new token type will be used in the process below. The default token in Simio does not have any custom States and this model will save off a row number within the process into a custom state on the token so we will use this new token type instead of the default Simio token.

Searching the Table in the Processes window

- Create a new Process by clicking 'Create Process' from the ribbon. Name this process 'MyProcess' in the properties window. Click anywhere within the process and open the Advanced Options category in the Process Properties

-
- window. Set the *Token Class Name* to 'MyToken1' , which is the name of the new token you created above.
- Place a Search Step
 - Set the *Collection Type* to 'TableRows'.
 - Set the *Table Name* to 'ServerData'. This tells the Search Step to search through the Table to find the appropriate row.
 - Set the *Match Condition* to 'Entity.Location.Parent == ServerData.ServerName'. The function Entity.Location.Parent returns a reference to the object that this Entity is currently inside of (i.e. the current Server). And we want the Search step to find the row within the ServerName column of the ServerData that matches the current Server.
 - Set *Save Index Found* to 'MyToken1.RowNumber' to save off the row number temporarily.
 - Place a SetRow step in the Original segment leaving the Search step. Set the *Table Name* property to 'ServerData' and the *Row Number* to 'MyToken1.RowNumber'.

Referencing the Table within the Servers

- Return to the Facility window and select Server1. Set the *Processing Time* property to 'ServerData.Hours' to read the processing time from the table.
- Under the Secondary Resources category, find 'Other Resource Seizes' and open the window for Before Processing. Set *Object Name* to 'Resources' and *Units Per Object* to 'ServerData.NumberOfResources'. This tells the Server to seize capacity of Resources before processing begins at this Server.
- Under the Secondary Resources category, find 'Other Resource Releases' and open the window for After Processing. Set *Object Name* to 'Resources' and *Units Per Object* to 'ServerData.NumberOfResources'. This tells the Server to release capacity of Resources after processing is finished at this Server.
- Find the *Entered Add On Process* trigger within this Server and select the new process you named MyProcess from the drop down. This tells the Server to execute this process when an entity enters this Server and this is where the entity will search the table to find the appropriate row.
- Repeat the above steps for the other two Servers.

See Also:

The SimBit named "HierarchyWithTables.spfx" uses this model inside of another model and demonstrates the changes that are needed to this model in order for it to work properly as a submodel inside of another.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SeizingSameResourceFromList - SimBit

Problem:

I have entities that select a resource from a list of resources, but once selected, they must use that same resource throughout their processing.

Categories:

Entity Characteristics

Key Concepts:

Object Reference State, Add-On Process, Assign Step, Resources, Discrete State, ModelEntity, ObjectList, Lists, Resource, Secondary Resources

Assumptions:

There are three resources within the set. Server1 has a capacity of 3, while Server2 has a capacity of 1.

Technical Approach:

An Object Reference State Variable will be added to the ModelEntity. An Add-On Process will be used to assign this state variable a value based on the resource the entity seized from a resource set. The object reference state will then be used to seize that particular resource later in the processing.

Details for Building the Model:

Simple System Setup

- Place a Source, two Servers and a Sink from the Standard Library. Use Paths to connect the Source to Server1, Server1 to Server2, and Server2 to the Sink. Change the *Interarrival Time* of the Source to 'Random.Exponential(3)'.
- Change the *Initial Capacity* of Server1 to '3' because the server can process up to 3 entities at a time. Also change the server's *Processing Time* to 'Random.Triangular(1,2,3)'.
- Change the *Processing Time* of Server2 to 'Random.Triangular(1,2,3)'.

Adding the Resources and Resource List

- Place three Resource objects in the Facility and change the *Name* properties to 'R1', 'R2' and 'R3'.
- Place the mouse to the top left of all three resources, and press the Ctrl key and drag the mouse to encompass all three Resources. While they are all highlighted, right-click in the highlighted area and select Add to Object List > Create a New Object List and name the list 'Resources'. You will notice now that within the Definitions window, List panel, the three resource objects have been added to the Resources list.

Adding an Object Reference State

- Within the Navigation window, highlight the ModelEntity and go to its Definitions window, States panel.
- From the States ribbon, select Object Reference > Object. Change the *Name* of the object reference state to 'WhichResource'. This will generate a new state of the entity, referenced ModelEntity.WhichState, that can store a reference to the resource that is seized in Server1.

Seizing the Resources

- Within the Navigation window, go back to the Model and make sure you are in the Facility window.
- Within Server1, under Secondary Resources, change the *Object Type* to 'From List' and the *Object List Name* to 'Resources'.
- Also within Server1, under the Add-On Process Triggers, double click on *Processing*, which will create a new process named 'Server1_Processing' and will also take you within the Processes window.
- Add an Assign step to the process and assign the *State Variable Name* 'ModelEntity.WhichResource' to the *New Value* 'ModelEntity.SeizedResources.LastItem'. This will store a reference to the last item that the entity seized, which would be one of the 3 resources. We can then use that information within Server2.
- Go back into the Facility window. Within Server2's Secondary Resources, keep the *Object Type* of 'Specific' and add the value 'ModelEntity.WhichResource' as the *Object Name*. This will then use each entity's object reference that was

assigned to determine the resource to seize.

Embellishments:

This model could also easily be built using 3 workers instead of 3 resources. The worker may be selected to move the entity from Server1 to Server2. First of all, you would add 3 worker objects to the model; group select them and right-click to Add to Transporter List (instead of Object List). Change the *Object List Name* in Server1 to the Transporter list name you just created. Finally, if you wanted to have the worker move the entity from one server to the other, you would click on the transfer node of Server1 (Output@Server1), specify *Ride on Transporter* as 'True' and specify the *Transporter Name* as 'ModelEntity.WhichResource.Transporter'.

** Note: If the Object Reference state on the entity was a 'Transporter' object reference, the *Transporter Name* would be specified as 'ModelEntity.WhichResource'; but because the object reference was an Object and Simio is looking for a Transporter Name (as opposed to Object Name), the name must include the '.Transporter' to signify what type of object it is. See help for Element and Object Reference States for more information.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SeizingVehicle - SimBit

Problem:

I would like to model a system that has a vehicle which is used for both transport and non transport activities. The vehicle's capacity is seized by another object for the non-transport activities.

Categories:

Vehicle

Key Concepts:

Add-On Process, Decide Step, Is, Processing, After Processing, Release Step, Ride on Transporter, Seize Step, Vehicle

Technical Approach:

There are two Sources that each produce a different type of entity; PartA and PartB. Both types of entities are processed by Server1. However, PartB type entities require the Vehicle object to be seized before processing at Server1 can occur. A Decide Step before processing checks to see what part type it is and then a Seize step seizes capacity of the Vehicle object and requests a move to the Node near Server1. All entities are then processed by Server2 and the Vehicle is required to transport all entities from Server2 to the Sink. Therefore, the same Vehicle object is transporting entities and also visiting Server1 to help with processing. Helping with Server1 has a higher priority.

Details for Building the Model:

Simple System Setup

- Place two Sources and two ModelEntity objects into the Facility window. Rename the entity objects to PartA and PartB. Set Source1's *Entity Type* property to 'PartA' and Source2's *Entity Type* property to 'PartB'. Change the *Interarrival Time* of Source2 to Random.Exponential(.9) hours.
- Place a Vehicle into the Facility window. Set the *Initial Node(Home)* to 'Output@Server2' and *Idle Action* to 'None'.
- Place two Servers into the Facility window, in series. Connect both Sources to Server1 with Paths and connect Server1 and Server2 with a Path.
- Place a Sink into the Facility window and connect Server2 with the Sink with a Path. Change the *Type* property of the Path to 'Bidirectional'. This is required so the vehicle can travel back and forth on this path.
- In the Output node of Server2, set the *Entity Destination* property to 'Specific' and the *Node Name* to 'Input@Sink1'. Set *Ride On Transporter* to 'True', *Transporter Type* to 'Specific' and *Transporter Name* to 'Vehicle1'.
- Place a TransferNode just above Server1. This will be where the Vehicle moves when it is helping with processing for Server1. Place a Path between Server2 and the new TransferNode. Make this path bi-directional by setting the *Type* property to 'Bidirectional'.

Add Process Logic

- From within the Facility window, click on Server1 and create two new Add On Processes for the triggers *Processing* and *After Processing*. (select Create New from the drop down)
- From within the Processes window, you should see the new Processes that were created.
 - In the Server1_Processing process, place a Decide step. The *Decide Type* is 'Conditional' and the *Expression* is 'Is.PartB'. This checks to see if the entity is a PartB type entity.
 - In the True segment, place a Seize step. Open the Seizes – Repeating Property Editor by pressing the ... button on the right side of the Seizes property. The *Object Type* is 'Specific' and the *Object Name* is 'Vehicle1'. Set the *Request Move* to 'ToNode' and the *Destination Node* to 'TransferNode1'. This will request that Vehicle1 move to TransferNode1 for servicing PartB before it starts processing.
 - In the Server1_AfterProcessing process, place a Decide step. The *Decide Type* is 'Conditional' and the *Expression* is 'Is.PartB'. This checks to see if the entity is a PartB type entity.
 - In the True segment, place a Release Step. Set the *Object Name* to 'Vehicle1' within the Releases – Repeating Property Editor.

SelectEntityTypeFromTable - SimBit

Problem:

I have a data file containing information about my products, product mix and processing parameters.

Categories:

Entity Characteristics

Key Concepts:

Before Creating Entities, Data Table, Data Table, Dynamic Object Property, Expression Property, Numeric Property, RandomRow, Source, Table Row Referencing

Technical Approach:

Create a table that contains the product mix and processing information for each part type possible.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink in the Facility Window.
- Place 3 ModelEntity objects from the Project Library into the Facility Window and change the *Name* property values to 'PartA', 'PartB' and 'PartC'.

Defining the Data Table with Product Information

- Click on the Data tab of the model and select the Tables panel. Add a DataTable with the *Name* 'ProductTable' and include the following columns:
 - 'PartType' – An object reference property of type Entity that will be used in the Source object to decide what is created
 - 'ProductMix' – A standard property of type Real that contains a number to indicate the weight (likelihood) of creating that entity type.
 - 'ProcessTime' – A standard property of type Expression to be used elsewhere in the model.
- There should be 3 rows containing PartType, ProductMix and ProcessTime data: PartA, 50, 2; PartB, 30, 4; and PartC, 20; Random.Uniform(4, 6).

Generating Random Entities using Data Table Information

- Within Source1, expand the Table Row Referencing section of the Properties window.
- Set the *Table Name* to 'ProductTable'. Apply the RandomRow table function to the ProductMix column by setting the value of the *Row Number* to 'ProductTable.ProductMix.RandomRow'. This will select a row based on the weight value in each row (e.g. the first part type (row 1) will be selected $50/(50+30+20)$ or 50% of the time).

Using the Product Information in the Model

- On Source1, specify the *Entity Type* property to be created from 'ProductTable.PartType'.
- On Server1, specify the *Processing Time* property as 'ProductTable.ProcessTime'.

Discussion:

You could add additional columns to provide data for use elsewhere in the model.

There are two forms of referencing table data:

TableName.PropertyName - When you use SetRow as we have done here to set the row for that entity, you do not need to specify the row number.

TableName[RowNumber].PropertyName - If you want to explicitly reference a different row, or make a table reference from an entity that does not have a designated row, you can use this form.

SelectingResourceFromList - SimBit

Problem:

I have a group of 2 Resources called Mechanics. I have three Servers in parallel that all share the group of mechanics for processing and must seize one of the two before it can process the entity.

Categories:

Resources

Key Concepts:

Lists, ObjectList, Resource, Secondary Resources, Server

Assumptions:

There are no restrictions on which mechanic should be seized – either resource can work on any one of the three servers. Each server seizes in Preferred Order, which means if both resources are available, they will always select Resource 1.

Technical Approach:

Place both the Servers and the Resources in the Facility Window before then creating an Object List the Lists panel of the Definitions Window. Use Add-On Processes in each Server to Seize and Release from the List.

Details for Building the Model:

Simple System Setup

- Place a Source, a Sink and three Servers (in parallel) in the Facility Window. Ensure that the Servers each have a capacity of Infinity. The server capacity is not the constraint in this model – the ability to seize a Resource is the constraint.
- Place two Resources in the Facility Window.

Defining the Resource List

- Click on the Definitions tab and select the Lists panel. Since we are creating a List of Resources, which are Objects, we need to click on the Object list button. Change the *Name* of the list to 'Mechanics'.
- Select the first cell in the table and select 'Resource1'. Then do the same for the second row and select 'Resource2'.

Using the Resources in the Server Processing

- Within each of the 3 Servers, enter the Secondary Resources / Resource for Processing section or properties. Change the *Object Type* to 'FromList' and the *Object List Name* to 'Mechanics'.

See Also:

A related SimBit is [SelectingResourcesAsAGroup.spf](#), which would be used if you'd like to seize multiple mechanics together before processing.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SelectingResourcesAsAGroup - SimBit

Problem:

I have a group of 4 Resources called Mechanics. I have a Server that all requires two mechanics for processing and must seize both before it can process the entity.

Categories:

Resources

Key Concepts:

Lists, ObjectList, Resource, Secondary Resources, Server

Assumptions:

There are no restrictions on which two mechanics should be seized. Each server seizes in Preferred Order, which means if all resources are available, they will always select the first two listed in the object list. Other selection rules might result in different usage patterns.

Technical Approach:

Place both the server and the resources in the Facility Window before then creating an Object list in the Lists panel of the Definitions Window. Use Add-On Processes in the Server to Seize and Release from the object list.

Details for Building the Model:

Simple System Setup

- Place a Source, a Sink and one Server in the Facility Window. Ensure that the Server has a capacity of Infinity. The server capacity is not the constraint in this model – the ability to seize a Resource is the constraint.
- Place four Resources in the Facility Window.

Defining the Resource List

- Click on the Definitions tab and select the Lists panel. Since we are creating a List of Resources, which are Objects, we need to click on the Object list button. Change the *Name* of the list to 'Mechanics'.
- Select the first cell in the table and select 'Resource1'. Then do the same for the second row and select 'Resource2', and continue with 'Resource3' and 'Resource4' in the third and fourth rows.

Using the Resources in the Server Processing

- Within Server1, we will use the properties within the Secondary Resources section to seize the resources before processing and release the resources after processing.
- In the Other Resource Seizes section, enter the *Before Processing* repeating editor and change the *Object Type* to 'FromList'. The *Object List Name* should be 'Mechanics' and ensure that the *Number of Objects* is set to '2'. The *Selection Goal* is 'PreferredOrder' so that so that if multiple Resources are available, the Resource that is listed first in the List will be selected.
- Do a similar thing for the Other Resource Releases section, *After Processing* repeating editor – release from an *Object List Name* of 'Mechanics' and the *Number of Objects* of '2'.

Embellishment:

We have added a second symbol to each resource and changed the color of the disk to red to indicate when it is busy.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SelectServerWithShortestLine - SimBit

Problem:

I would like my entities to always go to the Server with the least number of entities waiting in its queue.

Categories:

Decision Logic -- Paths

Key Concepts:

AssociatedStationLoad, Candidate, Entity Destination Type, Lists, Node, NodeList, Selection Goal, SmallestValue, Source, TransferNode

Assumptions:

There is one Source producing entities that are choosing which Server to go to.

Technical Approach:

A Node List of all Input Nodes at the Servers will be created in the Definitions Window. The Routing Logic at the Output Node of the Source will set the *Entity Destination Type* to 'Select From List'. The entities will select the Server with the largest value of remaining input capacity.

Details for Building the Model:

Simple System Setup

- Add a Source, three Servers and three Sinks to the Facility Window. Set a different Processing Time for each of the Servers; for example Server1 is set to '20' minutes, Server2 is set to 'Random.Triangular(1,3,8)' minutes and Server3 is set to '0.1' minutes.
- Use Paths to connect the Source to each Server and each Server to its respective Sink.

Creating a Node List

- Select the Definitions tab and click on the Lists panel to add a Node list and change its *Name* to 'Servers'. Add the three Input Nodes of the Servers to the list. The order of the list will not affect the model.

Adding Logic to the Model

- Select the Source's Output Node and change the *Entity Destination Type* to 'Select From List'. Choose the newly created node list called 'Servers' for the *Node List Name* property.
- The *Selection Goal* should be set to 'Smallest Value' and the *Selection Expression* should be set to 'Candidate.Node.AssociatedStationLoad'. This will ensure that the entity chooses a Server based on the least number of entities waiting or en-route to the Server.

Embellishments:

Try changing the *Interarrival Time* of the Source or the *Processing Times* of the Servers.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SelectSpecificVehicle - SimBit

Problem:

I have two different parts in my system and each part should be carried by a specific vehicle. Therefore, the part will wait for a specific vehicle, even if other vehicles are available.

Categories:

Vehicles

Key Concepts:

DestinationNode, Initial Node (Home), Lists, ModelEntity, NumberRiders, Priority, Ride on Transporter, Server, TransferNode, TransporterList, Vehicle, VisitRequestQueue

Technical Approach:

We will utilize the Priority property on the ModelEntity object and the Priority property on the Vehicle object. We'll assume that the value in the ModelEntity's property must match the value in the Vehicle's property in order for that Vehicle to be selected.

Details for Building the Model:

System Setup

- Place two Source objects and a Server object and connect both Sources to the Server with paths.
- Place a Sink object and connect the Server to the Sink. Also, draw a Path from the Sink back to the Output node of the Server. This path will be utilized by the Vehicles.
- Place two ModelEntity objects from the Project Library and rename one to 'PartA' and the other to 'PartB'. Consider changing the color of one entity by selecting it in the Facility window and selecting a color from the Color icon in the ribbon and then clicking back onto the Entity in the Facility window.
- Assign a value to the *Initial Priority* property on each Part. Click on PartA and set the *Initial Priority* property to a value of '1.0'. Click on PartB and set the *Initial Priority* property to a value of '2.0'.
- Click on Source2 and change the *Entity Type* property to 'PartB'. Ensure that Source1 has its *Entity Type* property set to 'PartA'. This will ensure that each Source will create a different type of entity.
- Place two Vehicle objects, naming one "PartA_Vehicle" and the other "PartB_Vehicle". Consider changing the color of one of the Vehicles.
- Assign a value to the *Initial Priority* property on each Vehicle. Click on PartA_Vehicle and set the *Initial Priority* property to a value of '1.0'. Click on PartB_Vehicle and set the *Initial Priority* property to a value of '2.0'.
- Assign an Initial Node location for each vehicle to tell Simio where to place these vehicles at the beginning of the run. Click on each vehicle and set *Initial Node (Home)* to 'Input@Sink1'.

Vehicle Selection Logic

- We will create a Transporter List. Go to the Definitions window and click on Lists in the left side panel. Click on Transporter in the Ribbon to create a Transporter List.
 - Enter two rows – one for each vehicle you've placed in your model.
- Back in the Facility window, click on the Output Node of Server1. This is where we will tell Simio that each entity entering this node should request a ride on a Vehicle. We will also specify exactly which Vehicle the entity should request from our Transporter List.
- Set *Ride On Transporter* property to 'True'. Set *Transporter Type* to 'From List'. Set *Transporter List Name* to the name of the list you just created. Put this expression into the *Selection Condition* property, 'Candidate.Transporter.Priority == ModelEntity.Priority'. This tells Simio, from the possible list of candidates (transporters), the *Priority* property must equal the *Priority* property on this current ModelEntity.

Discussion:

Consider these other examples to use in the *Selection Condition* property of a Transfer Node, when selecting a Transporter:

Candidate.Transporter.NumberRiders < 2 – To select the Transporter with its current number of entities riding on it as less than 2.

Candidate.Transporter.DestinationNode == Server1 – To select the Transporter that has its Destination Node currently set to Server1.

Candidate.Transporter.VisitRequestQueue == 0 – To select the Transporter that does not currently have any destination visit requests.

Embellishments:

Change the *Initial Ride Capacity* property of each Vehicle (SimBit has *Initial Ride Capacity* set to '3') to see how the system behavior changes.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SeparatorMakesCopies - SimBit

Problem:

I would like to have an entity create three identical copies of itself so there are a total of 4 entities that are exactly the same.

Categories:

Combining and Separating Entities

Key Concepts:

Make Copies, Maximum Arrivals, Separation Mode, Separator

Assumptions:

One Entity type goes into the Separator. The Separator produces three exact copies of the entity. The Source produces 100 entities and therefore the Separator produces 300 copies, which means there is a total of 400 entities at the end of the model. The copies and the originals depart through separate sinks.

Technical Approach:

A Separator is placed into the model and the Separation Mode property is set to 'Make Copies'. Two different Sinks are used in order to collect statistics separately on the original entities and the copied entities.

Details for Building the Model:

Simple System Setup

- Place a Source, Separator and two Sinks in the Facility Window.
- Connect the Source to the Separator and the Separator to both Sinks using Paths such that the parent entity will go to Sink1, while the copies will go to Sink2.

Defining the Source

- Set the *Maximum Arrivals* to '100'. This will turn off the Source once 100 entities have been generated.

Defining the Separator

- Set the *Separation Mode* property to 'Make Copies'. The *Copy Quantity* is set to '3'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SequentialProcessingByBatchSpecifiedInTable

- SimBit

Problem:

I have unlimited inventory and a Server that produces parts from that inventory.

Categories:

Add-On Process Logic, Data Tables

Key Concepts:

Assign Step, AvailableRowCount, Current Symbol Index, Data Table, Decide Step, ModelEntity, Numeric Property, Picture, Real State Variable

Assumptions:

I have a table that tells me what parts I want to produce and in what sequence. When I complete the sequence I will restart.

Technical Approach:

Use a Source, tightly linked to a Server to represent the generic arrivals from inventory. Use a table to supply the production data. Use add-on processes in the Server to control indexing through the table.

Details for Building the Model:

Simple System Setup

- Create a model with Source, Server, and Sink, connected by a Connector and a Path respectively.
- Add a ModelEntity from the Project Library and change the *Name* to 'AnyEntity'.
- Click on AnyEntity and then use the Add Additional Symbol button to add a second entity picture. Click on the color icon to change the color of that first entity (Index 0). The entity at Index 1 will remain the original color.

Setting up the Data Table

- In the Tables panel of the Data Window, select the Add Data Table button to add a table with the *Name* 'DataTable1' with Integer properties for *PartType* and *Quantity*. Set the *PartType* value to match the indexes defined in the AnyEntity Symbol (0 and 1).
- Specify the *Quantity* values required for each *PartType*.

Adding Discrete State Variables to the Model

- Click on the Definitions tab of the model and select the States panel.
- Add a Discrete State Variable with the *Name* 'Index' to track which table row is being processed.
- Add a Discrete State Variable with the *Name* 'Count' to track quantity produced for that row.
- We will use the predefined state Picture on the ModelEntity to track what is being produced.

Process Logic

- Go to the Processes Window and create a Process with the *Name* 'Process1'.
- Place a Decide step with an Assign step connected to the True exit. Place another Assign step then Decide and Assign steps from the False exit. The second Decide step False exit should connect to the first Decide input, as should the last Assign exit.
- Change the *Name* of the first Decide step to 'DecideIsCurrentRowDone'. This will compare the row quantity produced to quantity desired by using the *Decide Type* of 'ConditionBased' and *Expression* of 'Count < DataTable1[Index].Quantity'.
- If the above condition is true, then the True exit Assign step with *Name* 'AssignPictureAndCount' assigns the picture to the row PartType (*State Variable Name* is 'ModelEntity.Picture' and *New Value* is 'DataTable1[Index].PartType') and increments the row production counter (*State Variable Name* is 'Count' and *New Value* is 'Count+1').
- If the above condition is false, then the False exit Assign with *Name* 'AssignRowAndCount' increments the row index

using *State Variable Name* is 'Index and *New Value* is 'Index+1'. It also assigns the row production counter, 'Count' to '0'.

- The next Decide step 'DecideIfEndOfTable' determines if the Index has passed the last specified table row by using 'ConditionBased' evaluation and *Expression* 'Index >=DataTable1.AvailableRowCount' (where the function AvailableRowCount returns the total number of rows in the data table).
- If the above condition is true, then the last Assign step 'AssignRestart' assigns the 'Index' back to '1'.
- After we have decided our new row, both branches go back to the initial Decide.

Connect the Process Logic to the Server

- Go back to the Server properties in the Facility Window. Use the pull down list for the *Entered Add-On Process Triggers* and select 'Process1'.

Discussion:

This could also have been done with a custom object that encompassed both entity creation and processing. In this case you could have actually created the desired entity types rather than using a state to represent entity type.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServerBlockingApproaches - SimBit

This SimBit project includes four models that demonstrate how to block a Server and have an entity wait at the Server until the next Server location is available.

Models included in this SimBit:

1. BlockWithConnectors – Demonstrates how to use Connectors built in logic to block a Server and wait until the next Server is available for an entity to continue.
2. BlockWithProcesses – Demonstrates how to use a set of processes to block a Server and wait until the next Server is available for an entity to continue.
3. BlockWithEntityDestinationType – Demonstrates how to use *Entity Destination Type* to block a Server and wait until the next Server is available for an entity to continue.
4. BlockWithRoutingGroup – Demonstrates how to use Routing Group elements and their *Destination Blocked Condition* to block a Server and wait until the next Server is available for an entity to continue.

Model 1: BlockWithConnectors

Problem:

I have a system that has instantaneous travel between locations, but entities need to wait at their current location until their next location is available.

Categories:

Buffering, Decision Logic -- Paths

Key Concepts:

Connectors, ResourceState

Technical Approach:

Connectors have built in logic that changes the Resource State of a Server to 'Blocked'.

Details for Building the Model:

Simple System Setup

- Place a Source, four Servers, and a Sink in the Facility window. Change the *Name* of the Source to 'Arrested', the *Name* of the four Servers to 'HoldingChamber', 'Booking', 'Interrogation', and 'Court', respectively. Lastly, change the *Name* of the Sink to 'Jail'.
- Link 'Arrested' to 'HoldingChamber', 'HoldingChamber' to 'Booking', 'Booking' to 'Interrogation', 'Interrogation' to 'Court', and 'Court' to 'Jail', all with Connectors.
- Place a ModelEntity in the Facility View and change the *Name* to 'Prisoner'.
- Select the Source 'Arrested'. Change the *Interarrival Time* to 'Random.Exponential(1)' and set the *Units* to 'Hours'.
- Select the Server 'HoldingChamber'. Change the *Initial Capacity* to 'Infinity', the *Processing Time* to '0' and the *Output Buffer Capacity* to '0'.
- Select the Server 'Booking'. Change the *Processing Time* to 'Random.Triangular(5,10,15)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Interrogation'. Change the *Processing Time* to 'Random.Exponential(35)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Court'. Change the *Processing Time* to 'Random.Triangular(30, 60, 90)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.

Model 2: BlockWithProcesses

Problem:

I have a system that requires an Entity to stay at its current Server until the next destination is available while also blocking the Server from accepting another Entity. Add-on processes and a Wait step will be used to hold the entities.

Categories:

Key Concepts:

Add-On Process, Assign Step, Decide Step, Resource State, Wait Step

Technical Approach:

Process logic can be used to change the ResourceState of a Server and to force an Entity to wait until the Server is available for use.

Details for Building the Model:

Simple System Setup

- Place a Source, four Servers, and a Sink in the Facility window. Change the *Name* of the Source to 'Arrested', the *Name* of the four Servers to 'HoldingChamber', 'Booking', 'Interrogation', and 'Court', respectively. Lastly, change the *Name* of the Sink to 'Jail'.
- Link 'Arrested' to 'HoldingChamber', 'HoldingChamber' to 'Booking', 'Booking' to 'Interrogation', 'Interrogation' to 'Court', and 'Court' to 'Jail', all with Paths.
- Place a ModelEntity in the Facility View and change the *Name* to 'Prisoner'.
- Select the Source 'Arrested'. Change the *Interarrival Time* to 'Random.Exponential(1)' and set the *Units* to 'Hours'.
- Select the Server 'HoldingChamber'. Change the *Initial Capacity* to 'Infinity' and the *Processing Time* to '0'.
- Select the Server 'Booking'. Change the *Processing Time* to 'Random.Triangular(5,10,15)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Interrogation'. Change the *Processing Time* to 'Random.Exponential(35)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Court'. Change the *Processing Time* to 'Random.Triangular(30, 60, 90)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.

Creating Processes

- Go to the Facility window and select 'HoldingChamber'. Expand the 'Add-On Process Triggers' section of the Properties pane. Double click on *After Processing* to add a new process here. Place a Decide step in this process. Set the *Condition Or Probability* to 'Booking.Capacity.Remaining > 0 && Input@Booking.NumberTravelers.RoutingIn == 0'. This Decide step checks to see if the 'Booking' Server is available. From the 'False' branch, place a Wait step. Change the *Event Name* to 'Output@Booking.Entered', which forces an entity to wait until another entity reaches the output node of 'Booking'.
 - Go to the Definitions window and add an Integer state variable with the *Name* 'Count', with the *Initial Value* of '0'. This will be used in the logic to make sure only one prisoner leaves the HoldingChamber at a time.
 - As the first step in the process, prior to the Decide step, add an Assign step. Assign the *State Variable Name* 'Count' to the *New Value* '0'.
 - From the Wait step, place an Assign step and another Decide step. Within the Assign step, assign the *State Variable Name* 'Count' to the *New Value* 'Count + 1'. Within the Decide step, enter the *Condition or Probability* as 'Count > 1'. Connect the True exit of the step to the first Assign step in the process.
 - Note that when an entity leaves the Booking Server and the Output@BookingEntered event is fired, all waiting entities will leave the Wait step (since the capacity of the HoldingChamber is > 1, multiple entities may reside there). To avoid allowing all to exit, the Count variable is increased with entity exiting the Wait step, but only the first one will continue to the Booking Server.
- Return to the Facility window and select 'Booking'. Expand the 'Add-On Process Triggers' section of the Properties pane. Double click on *After Processing* to add a new process here. Place a Decide step in this process. Set the *Condition Or Probability* to 'Interrogation.Capacity.Remaining > 0 && Input@Interrogation.NumberTravelers.RoutingIn == 0'. This Decide step checks to see if the 'Interrogation' Server is available. Off the False branch, place an Assign step. This Assign step places the 'Booking' Server into a ResourceState of Blocking'. To do this, change the *State Variable Name* to 'Booking.ResourceState' and the *New Value* to '2'. Lastly, add a Wait step off the Assign step. This Wait step forces an Entity to wait until another Entity enters the Output Node of 'Interrogation'. To do this, change the *Event Name* property to 'Output@Interrogation.Entered'.
- Return to the Facility window and select 'Interrogation'. Expand the 'Add-On Process Triggers' section of the Properties pane. Double click on *After Processing* to add a new process here. Place a Decide step in this process. Set the *Condition Or Probability* to 'Court.Capacity.Remaining > 0 && Input@Court.NumberTravelers.RoutingIn == 0'. This Decide step check to see if the 'Court' Server is available. From the False branch, place an Assign step. This Assign step has a *State Variable Name* of 'Interrogation.ResourceState' and a *New Value* of '2'. This Assign step places the 'Interrogation' Server in a 'Blocked' state if the 'Court' Server is unavailable. Finally, add a Wait step after the

Assign step. Change the *Event Name* to 'Output@Booking.Entered'. This makes an Entity wait until another Entity enters the Output Node of the 'Court' Server.

Model 3: BlockWithEntityDestinationType

Problem:

I have a system that requires an Entity to stay at its current Server until the next destination is available while also blocking the Server from accepting another Entity. Entity Destination Type and Selection Lists will be used for blocking.

Categories:

Buffering, Decision Logic -- Paths

Key Concepts:

Lists, Select Available Only, Select From List, TransferNode

Technical Approach:

When using Entity Destination Type 'Select From List', Simio has built in logic that evaluates an Entity route request to determine whether a candidate location in the Destination Node List is 'Blocked'.

Details for Building the Model:

Simple System Setup

- Place a Source, four Servers, and a Sink in the Facility window. Change the *Name* of the Source to 'Arrested', the *Name* of the four Servers to 'HoldingChamber', 'Booking', 'Interrogation', and 'Court', respectively. Lastly, change the *Name* of the Sink to 'Jail'.
- Link 'Arrested' to 'HoldingChamber', 'HoldingChamber' to 'Booking', 'Booking' to 'Interrogation', 'Interrogation' to 'Court', and 'Court' to 'Jail', all with Paths.
- Place a ModelEntity in the Facility View and change the *Name* to 'Prisoner'.
- Select the Source 'Arrested'. Change the *Interarrival Time* to 'Random.Exponential(1)' and set the *Units* to 'Hours'.
- Select the Server 'HoldingChamber'. Change the *Initial Capacity* to 'Infinity', *Processing Time* to '0' and *Output Buffer Capacity* to '0'.
- Select the Server 'Booking'. Change the *Processing Time* to 'Random.Triangular(5,10,15)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Interrogation'. Change the *Processing Time* to 'Random.Exponential(35)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Court'. Change the *Processing Time* to 'Random.Triangular(30, 60, 90)' minutes. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.

Configuring Node List

- Go to the Lists view on the Definitions tab. Create three Node Lists by clicking the 'Node' button three times.
 - NodeList1 – Change *Name* to 'CourtInput' and add 'Input@Court' to the List.
 - NodeList2 – Change *Name* to 'InterrogationInput' and add 'Input@Interrogation' to the List.
 - NodeList3 – Change *Name* to 'BookingInput' and add 'Input@Booking' to the List.
- Return to the Facility window and change the *Entity Destination Type* for the TransferNodes to 'Select From List'. Note that the Block Destination Rule is, by default, 'Select Available Only'. This will assure that the entity will select the destination only if the Server capacity is available (since buffer size is '0').
 - Output@HoldingChamber – Change *Node List Name* to 'BookingInput'.
 - Output@Booking – Change *Node List Name* to 'InterrogationInput'.
 - Output@Interrogation – Change *Node List Name* to 'CourtInput'.

Model 4: BlockWithRoutingGroup

Problem:

I have a system that requires an Entity to stay at its current Server until the next destination is available whilst also blocking the Server from accepting another Entity. Custom Routing Group and Selection Lists will be used for blocking.

Categories:

Buffering, Decision Logic -- Paths

Key Concepts:

Lists, RoutingGroup Element, Select Available Only, TransferNode

Technical Approach:

Routing Groups have built in logic that evaluates an Entity route request to determine whether a candidate location in the Destination Node List is 'Blocked'.

Details for Building the Model:

Simple System Setup

- Place a Source, four Servers, and a Sink in the Facility window. Change the *Name* of the Source to 'Arrested', the *Name* of the four Servers to 'HoldingChamber', 'Booking', 'Interrogation', and 'Court', respectively. Lastly, change the *Name* of the Sink to 'Jail'.
- Link 'Arrested' to 'HoldingChamber', 'HoldingChamber' to 'Booking', 'Booking' to 'Interrogation', 'Interrogation' to 'Court', and 'Court' to 'Jail', all with Paths.
- Place a ModelEntity in the Facility View and change the *Name* to 'Prisoner'.
- Select the Source 'Arrested'. Change the *Interarrival Time* to 'Random.Exponential(1)' and set the *Units* to 'Hours'.
- Select the Server 'HoldingChamber'. Change the *Initial Capacity* to 'Infinity', *Processing Time* to '0' and *Output Buffer Capacity* to '0'.
- Select the Server 'Booking'. Change the *Processing Time* to 'Random.Triangular(5,10,15)'. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Interrogation'. Change the *Processing Time* to 'Random.Exponential(35)'. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.
- Select the Server 'Court'. Change the *Processing Time* to 'Random.Triangular(30, 60, 90)'. Set the *Input Buffer Capacity* and *Output Buffer Capacity* to '0'.

Configuring Node List

- Go to the Lists view on the Definitions tab. Create three Node Lists by clicking the 'Node' button three times.
 - NodeList1 – Change *Name* to 'CourtInput' and add 'Input@Court' to the List.
 - NodeList2 – Change *Name* to 'InterrogationInput' and add 'Input@Interrogation' to the List.
 - NodeList3 – Change *Name* to 'BookingInput' and add 'Input@Booking' to the List.

Creating Routing Group Elements

Under the Advanced Options section of the Routing Group element, there is a property called *Destination Blocked Condition*. This logical condition evaluates an Entity route request to determine whether a candidate location in the *Destination Node List* is blocked. By default this property is set to 'Candidate.Node.AssociatedStation != Nothing && Candidate.Node.AssociatedStationHasSpaceFor (Entity) == False'. This expression checks the candidate destination nodes to see if there is an Entity there and also checks to see if there is space for an Entity, and if one of these is false, then the Server will be blocked.

- Go to the Elements view on the Definitions tab. Create three Routing Groups by clicking the 'Routing Group' button three times.
 - RoutingGroup1 – Change *Name* to 'RoutingGroupCourt' and the *Destination Node List Name* to 'CourtInput'.
 - RoutingGroup2 – Change *Name* to 'RoutingGroupInterrogation' and the *Destination Node List Name* to 'InterrogationInput'.
 - RoutingGroup3 – Change *Name* to 'RoutingGroupBooking' and the *Destination Node List Name* to 'BookingInput'.
- Return to the Facility window and change the Entity Destination Type for the Output Nodes to 'Use Custom Routing Group'.
 - Output@HoldingChamber – Change *Routing Group Name* to 'RoutingGroupBooking'.
 - Output@Booking – Change *Routing Group Name* to 'RoutingGroupInterrogation'.
 - Output@Interrogation – Change *Routing Group Name* to 'RoutingGroupCourt'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServerFollowsCapacitySchedule - SimBit

Problem:

I have a server that follows a schedule with entities entering the system 24 hours a day. Capacity of server changes over the day.

Categories:

Schedules / Changeovers

Key Concepts:

Current Symbol Index, Off Shift, On Shift, Schedules, Server, Work Exceptions, WorkSchedule

Assumptions:

The server works from 8:00 am to 5:00 pm with a capacity of 2, from 5:00 pm to 9:00 pm with a capacity of 1, and a capacity of 0 otherwise. The server also has a work exception and is off shift for an entire day.

Technical Approach:

A Schedule is created for a Server and the Server is set to follow this Schedule in its Process Logic. When the Server is Off Shift it will turn red indicating a Current Capacity of zero.

Details for Building the Model:

Simple System Setup

- Add a Source, Server and Sink to the Facility Window. Update the *Processing Time* of the Server to '0.25' and *Units* to 'Hours'.
- Connect the Source, Server and Sink using Paths.

Creating a Work Schedule

- In the Data Window, select the Schedules panel and add a Work Schedule with *Name* 'Schedule1'.
- Create a Day Pattern by clicking on the Day Pattern tab and enter the *Name* 'DayPattern1'. Enter the number of *Days* as '7'.
- Click on the '+' sign to expand the Work Periods.
- Specify that 'DayPattern1' that has a *Value* of '2' (on shift) from 8:00 am to 5:00 pm and capacity *Value* of '1' from 5:00 pm to 9:00 pm. Leave everything else empty, which indicates Off Shift times.
- Under Work Schedules, select 'DayPattern1' for days 1 through 5. Because Day 6 and Day 7 have no day pattern specified, it is assumed that the capacity value is 0 (off shift) for those days.
- Expand the '+' sign on 'Schedule1'. Under the Work Period Exceptions tab, add an exception on Day 4 that has the capacity *Value* of '0' (off shift) for 24 hours.
- In the Facility Window, change the Process Logic *Capacity Type* property of the Server to 'WorkSchedule'.
- Change the Process Logic *Work Schedule* property of the Server to 'Schedule1'.

Animating the Off Shift Server

- Select the Server and click the Add Additional Symbol in the ribbon. Then select the first of the two Active Symbols and change its color to Red.
- For the Server, change the Appearance *Current Symbol Index* to 'Server1.CurrentCapacity'.

Embellishments:

To enhance the model for specific needs, try shifting the Server's work schedule, the Source's *Interarrival Time* rate, or the Server's *Processing Time* rate.

ServerFollowsDailySchedule - SimBit

Problem:

I have a server that follows a daily schedule with entities entering the system 24 hours a day.

Categories:

Schedules / Changeovers

Key Concepts:

Current Symbol Index, Off Shift, On Shift, Schedules, Server, WorkSchedule

Assumptions:

The server works an 8 hour shift with a 30 minute break. The server's queue has an infinite capacity.

Technical Approach:

A Schedule is created for a Server and the Server is set to follow this Schedule in its Process Logic. When the Server is Off Shift it will turn red indicating a Current Capacity of zero.

Details for Building the Model:

Simple System Setup

- Add a Source, Server and Sink to the Facility Window. Update the *Processing Time* of the Server to 'Random.Triangular (0.1, 0.15, 0.25)' and the *Units* to 'Hours'.

Creating a Daily Work Schedule

- In the Data Window, select the Schedules panel and add a Work Schedule with the Name 'Shift8Hours'.
- Create a Day Pattern by clicking on the Day Pattern tab and enter the *Name* 'DayPattern1'.
- Change the number of *Days* to '7'.
- Click on the '+' sign to expand the Work Periods.
- Enter the *Start Time* and *End Time* from 8:00 am to 12:00 and 12:30 pm to 4:30 pm that has a *Value* of '1'. Leave everything else empty, which indicates Off Shift times.
- Under Work Schedules, select 'DayPattern1' for days 1 through 5. Days 6 and 7 will not have a day pattern specified, thus the capacity value will be 0 (off shift).
- Within the Facility window, change the Server's Process Logic *Capacity Type* property to 'WorkSchedule'.
- Change the Process Logic *Work Schedule* property to 'Shift8Hours'.

Animating the Off Shift Server

- Select the Server and click the Add Additional Symbol in the ribbon. Then select the first of the two Active Symbols and change its color to Red.
- Change the Appearance *Current Symbol Index* to 'Server1.CurrentCapacity'.

Embellishments:

To enhance the model for specific needs, try shifting the Server's work schedule, the Source's *Interarrival Time* rate, or the Server's *Processing Time* rate.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServerFollowsOddSchedule - SimBit

Problem:

I have a server that follows an odd schedule with entities entering the system 24 hours a day.

Categories:

Schedules / Changeovers

Key Concepts:

Current Symbol Index, Off Shift, On Shift, Schedules, Server, WorkSchedule

Assumptions:

The server works for 45 minutes then rests for 15 minutes. The server's queue has an infinite capacity.

Technical Approach:

A Schedule is created for the Server and the Server is set to follow this Schedule in its Process Logic. When the Server is Off Shift it will turn red indicating a Current Capacity of zero.

Details for Building the Model:

Simple System Setup

- Add a Source, Server and Sink to the Facility Window. Update the *Processing Time* of the Server to be 'Random.Triangular (0.1, 0.15, 0.25)' and the *Units* to be 'Hours'.

Creating a Work Schedule

- In the Data Window, select the Schedules panel and add a Work Schedule with *Name* 'Shift45Minutes'.
- Create a Day Pattern by clicking on the Day Pattern tab and specifying the *Name* 'DayPattern1'.
- Click on the '+' sign to expand the Work Periods.
- Enter into 'DayPattern1' the following information: 12:00 am to 12:45 am and from 12:45 am to 1:45 am and from 2:00 am to 2:45 am that has a *Value* of '1'. Leave everything else empty, which indicates Off Shift times.
- Under Work Schedules, select 'DayPattern1' for day 1.
- Within the Facility Window, change the Server's Process Logic *Capacity Type* property to 'WorkSchedule'.
- Change the Process Logic *Work Schedule* property to 'Shift45Minutes'.

Animating the Off Shift Server

- With the Server selected, click the Add Additional Symbol in the ribbon. Then select the first of the two Active Symbols and change its color to Red.
- Change the Appearance *Current Symbol Index* to 'Server1.Capacity'.

Embellishments:

To enhance the model for specific needs, try shifting the Server's work schedule, the Source's *Interarrival Time* rate, or the Server's *Processing Time* rate.

See Also:

To see how to make a more traditional 8 hour work day schedule, see [ServerFollowsDailySchedule.spf](#).

ServerQueueWithBalkingAndReneging - SimBit

Problem:

I want to model a single server queue with balking and reneging (impatient customers).

Categories:

Buffering

Key Concepts:

Buffer Logic, Balking, Reneging

Assumptions:

A system consists of a single server that processes customer entities in FIFO order. The time between customer arrivals is sampled from an exponential distribution, EXPO(5) minutes. The processing time for each customer is EXPO(4.25) minutes.

An arriving customer balks if the current number waiting at the Server is greater than or equal to its queue length tolerance. This tolerance is sampled from a triangular distribution, TRIA(3,5,9). If the customer does decide to enter the queue, then it has a waiting time tolerance sampled from an Erlang distribution, ERLA(15,2) minutes. If that tolerance time elapses and the customer has not yet been served or is not within a maximum tolerable position from the front of the queue (i.e., within a specific 'stay zone'), then the entity abandons the queue and leaves the system. The stay zone for a customer is sampled from a Poisson distribution, POIS(0.75).

Details for Building the Model:

Facility Window Setup

- Add a Source, Server, and Sink to the Facility window. Connect the Source to the Server and the Server to the Sink using Connector links.
- Place a ModelEntity object from the Project Library into the Facility window. Name it 'Customer'.

Source Properties

- Specify the *Entity Type* as 'Customer' and the *Interarrival Time* as 'Random.Exponential(5)' minutes.

Server Properties

- Specify the *Processing Time* as 'Random.Exponential(4.25)' minutes.
- Go to Buffer Logic -> Input Buffer -> Balking & Reneging Options. Specify the *Balk Decision Type* as 'Conditional' and the *Balk Condition Or Probability* as 'Server1.InputBuffer.Contents >= Random.Triangular(3,5,9)'.
- Open the input buffer's *Reneg Triggers* repeat group. Add a new time-based renege trigger. Specify the *Wait Duration* as 'Random.Erlang(15,2)' minutes. Specify the *Reneg Decision Type* as 'Conditional' and the *Reneg Condition Or Probability* as 'Server1.InputBuffer.Contents.IndexOfItem(Entity) > Random.Poisson(0.75)'.

Embellishments:

Instead of a FIFO ranked queue, assume that customers have different priorities and are processed in priority order. Thus, the arrival of a higher priority customer will bump back lower priority customers in the queue.

Once a customer's waiting time tolerance elapses, it will continue waiting only for as long as it's position in the queue remains within its maximum tolerable range from the front. If a higher priority customer subsequently joins the queue and bumps the customer back outside of its 'stay zone', then the customer reneges.

Hint: Use an event-based renege trigger to evaluate all entities in the server's input buffer queue for possible reneging anytime a new entity joins the queue.

See also:

SourceWithBalkingIfBlocked.spfx

ChangingQueuesWhenServerFails.spfx

MultiServerSystemWithJockeying.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServersUsingTaskSequenceWithDataTables_FlowLine

- SimBit

Problem:

I want to model a system where the entities being processed follow a fixed work path, and where the operations performed at the physical processing locations are task sequences that are also the same for each entity. For modeling purposes, it is desired that all operation data for each physical processing location is defined in a set of relational data tables.

Categories:

MultiTask Server, Data Tables

Key Concepts:

Auto-Set Table Row Reference, Data Table, Materials Element, Process Type, Resources, Server, Task Sequence

Assumptions:

The modeled system is a flow line that consists of three server locations visited in a fixed sequence. Entities enter the system at the front of the line and then are processed through the three servers before finishing at a sink.

Technical Approach:

The operation data for each physical processing location is defined in a set of relational data tables using the design shown in *Figure 1*.

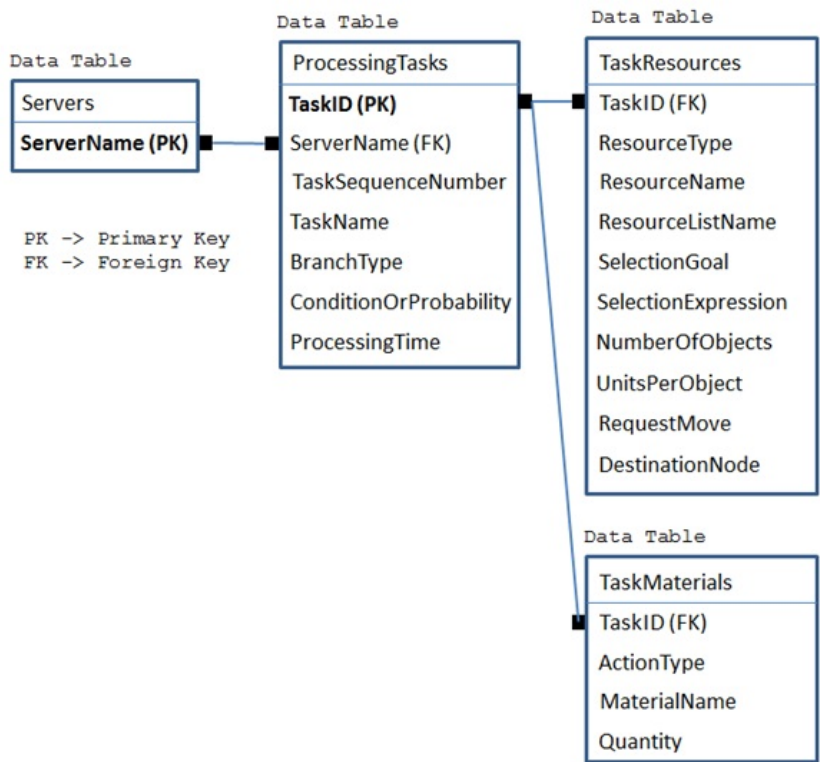


Figure 1 - Relational Data Table Design

Details for Building the Model:

Facility Window Setup

- Add a Source, a Sink and three Servers from the Standard Library to the Facility window. Then connect those objects using Paths from the Standard Library, drawing a path from the Source to the input of Server1, then a path from the output of Server1 to the input of Server2, then a path from the output of Server2 to the input of Server3, and finally a

path from the output of Server3 to the Sink.

- Place a ModelEntity from the Project Library into the window.
- Place as many Resource or Worker objects from the Standard Library into the Facility window as you like, for use as task resources.

Material Elements Setup

- Add as many Material elements to the model as you like, for use as task materials (go to Definitions -> Elements and click on the Material button in the ribbon to add a new Material element).

Data Table Setup

In the Data window, add three data tables to the model using the relational table design shown in Figure 1. The table column property types are as follows:

Servers Data Table

Column Name	Property Type
ServerName (Primary Key)	Object Reference Property Important: Set the <i>Auto-set Table Row Reference</i> option for this column to True, so that the server objects specified in this column are automatically referencing their associated table data.

ProcessingTasks Data Table

Column Name	Property Type
TaskID (Primary Key)	Integer
ServerName	Foreign Key to Servers.ServerName
TaskSequenceNumber	Sequence Number
TaskName	String
BranchType	Enumeration of enum type TaskBranchType
ConditionOrProbability	Expression
ProcessingTime	Expression of unit type Time

TaskResources Data Table

Column Name	Property Type
TaskID	Foreign Key to ProcessingTasks.TaskID
ResourceType	Enumeration of enum type ObjectSeizeType
ResourceName	Object Reference
ResourceListName	Object List Reference
SelectionGoal	Enumeration of enum type SeizeSelectionGoal
SelectionExpression	Expression
NumberOfObjects	Expression
UnitsPerObject	Expression
RequestMove	Enumeration of enum type SeizeRequestVisitType
DestinationNode	Node Object Reference

TaskMaterials Data Table

Column Name	Property Type
TaskID	Foreign Key to ProcessingTasks.TaskID
ActionType	Enumeration of enum type MaterialActionType
MaterialName	Material Element Reference
Quantity	Expression

Mapping the Table Data to the Objects in the Facility Window

Go to the Facility window and do the following:

- **Source object** – Make sure the *Entity Type* property is specified as the model entity type that you placed in the model.
- **For Each Server object** – Specify the *Process Type* as 'Task Sequence'. Then, right-click on the *Processing Tasks* repeat group and set a reference to the 'ProcessingTasks' table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information. Finally, in the *Process Logic*->*Other Task Sequence Options* properties, specify the *Task Resources Referenced Table Name* as the 'TaskResources' table and the *Task Materials Referenced Table Name* as the 'TaskMaterials' table. This indicates that those tables will be used as the referenced data sources for getting task resource and material requirements.
- **For Each Server object** – Open the *Processing Tasks* repeat group. Map the columns from the ProcessingTasks, TaskResources and TaskMaterials tables to the appropriate corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).
- In the Servers table, make sure that the *Auto-set Table Row Reference* property for the *ServerName* column is set to 'True'.

Embellishments:

Experiment with different server operation data in the relational data tables.

See also:

ServersUsingTaskSequencesWithDataTables_JobShop.spfx

ServerUsingTaskSequencesWithWorkers.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServersUsingTaskSequenceWithDataTables_JobShop

- SimBit

Problem:

I want to model a system where the entities being processed have different routings based on type, and where the operations performed at the physical processing locations are task sequences that are also dependent on the job type. For modeling purposes, it is desired that all product mix, job routing, and operation data is defined in a set of relational data tables.

Categories:

MultiTask Server, Data Tables

Key Concepts:

Data Table, Material Element, Process Type, RandomRow, Sequence Table, Server, Source, Table Row Referencing, Task Sequence

Assumptions:

The modeled system has three possible server locations where entities are processed. An entity will enter the system and visit those one or more of the server locations in a sequence that is dependent on the job type. Not all of the servers have to be visited and a server may be visited more than once.

Technical Approach:

The product mix, job routing, and operation data is defined in a set of relational data tables using the design shown in Figure 1.

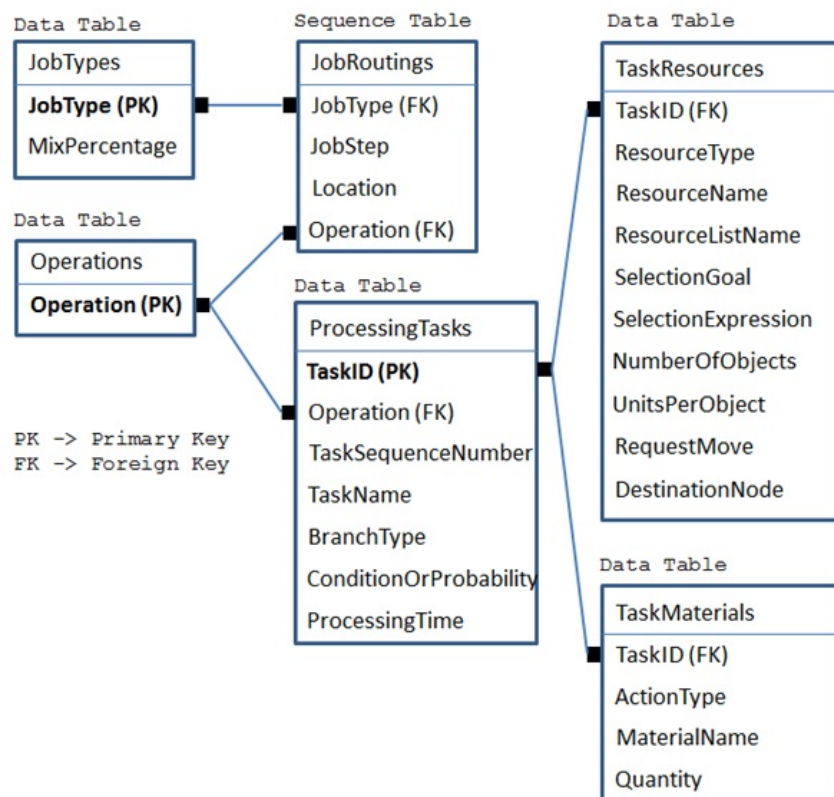


Figure 1 - Relational Data Table Design

Details for Building the Model:

Facility Window Setup

- Add a Source, a Sink and three Servers from the Standard Library to the Facility window. Then connect those objects using Paths from the Standard Library, drawing paths from the Source to the input of each Server, paths to travel from the output of any Server to the input of any other Server, and a path from the output of each Server to the Sink.
- Place as many ModelEntity objects from the Project Library into the window as you like, one for each job type that is processed in the system.
- Place as many Resource or Worker objects from the Standard Library into the Facility window as you like, for use as task resources.

Material Elements Setup

- Add as many Material elements to the model as you like, for use as task materials (go to Definitions -> Elements and click on the Material button in the ribbon to add a new Material element).

Data Table Setup

In the Data window, add one sequence table and four data tables to the model using the relational table design shown in Figure 1. The table column property types are as follows:

JobTypes Data Table

Column Name	Property Type
JobType (Primary Key)	Entity Object Reference
MixPercentage	Integer

JobRoutings Sequence Table

Column Name	Property Type
JobType	Foreign Key to JobTypes.JobType
JobStep	Integer
Location	Sequence Destination (auto added when creating Sequence Table)
Operation	Foreign Key to Operations.Operation

Operations Data Table

Column Name	Property Type
Operation (Primary Key)	String

ProcessingTasks Data Table

Column Name	Property Type
TaskID (Primary Key)	Integer
Operation	Foreign Key to Operations.Operation
TaskSequenceNumber	Sequence Number
TaskName	String
BranchType	Enumeration of enum type TaskBranchType
ConditionOrProbability	Expression
ProcessingTime	Expression of unit type Time

TaskResources Data Table

Column Name	Property Type
TaskID	Foreign Key to ProcessingTasks.TaskID
ResourceType	Enumeration of enum type ObjectSeizeType
ResourceName	Object Reference
ResourceListName	Object List Reference
SelectionGoal	Enumeration of enum type SeizeSelectionGoal
SelectionExpression	Expression
NumberOfObjects	Expression
UnitsPerObject	Expression
RequestMove	Enumeration of enum type SeizeRequestVisitType
DestinationNode	Node Object Reference

TaskMaterials Data Table

Column Name	Property Type
TaskID	Foreign Key to ProcessingTasks.TaskID
ActionType	Enumeration of enum type MaterialActionType
MaterialName	Material Element Reference
Quantity	Expression

Mapping the Table Data to the Objects in the Facility Window

Go to the Facility window and do the following:

- **Source object** – Specify the *Entity Type* property as 'JobTypes.JobType'. Go to the *Table Row Referencing->Before Creating Entities* properties, and specify the *Table Name* as 'JobTypes' and the *Row Number* as 'JobTypes.MixPercentage.RandomRow'. The Source will then randomly create entities of the types specified in the JobTypes table using the specified mix percentages.
- **For Each 'Output' TransferNode object** – Specify the *Entity Destination Type* as 'By Sequence'. Then, when an entity exits the Source or any of the Server objects, it will set its next destination based on the JobRoutings sequence table.
- **For Each Server object** – Specify the *Process Type* as 'Task Sequence'. Then, right-click on the *Processing Tasks* repeat group and set a reference to the 'ProcessingTasks' table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information. Finally, in the *Process Logic->Other Task Sequence Options* properties, specify the *Task Resources Referenced Table Name* as the 'TaskResources' table and the *Task Materials Referenced Table Name* as the 'TaskMaterials' table. This indicates that those tables will be used as the referenced data sources for getting task resource and material requirements. .
- **For Each Server object** – Open the *Processing Tasks* repeat group. Map the columns from the ProcessingTasks, TaskResources, and TaskMaterials tables to the appropriate corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).

Embellishments:

Experiment with different product mix, job routing, and operation data in the relational data tables.

See also:

ServersUsingTaskSequencesWithDataTables_FlowLine.spfx

ServerUsingTaskSequencesWithWorkers.spfx

ServersUsingTaskSequenceWithDataTables_LoopbackBranches

- SimBit

Problem:

I want to model a system where the operations performed at servers are task sequences, and there are conditional or probabilistic loopback branches in the task workflow. For modeling purposes, it is desired that all operation data is defined in a set of relational data tables.

Categories:

Data Tables, MultiTask Server

Key Concepts:

Data Table, Process Type, Server, Task Sequence, Loopback Branches

Assumptions:

The modeled system is a sequential process through two server locations.

The processing task sequence at the first server has a probabilistic loopback branch in the task workflow and is shown in *Figure 1*. After finishing Task3, there is a 30% chance that a 'Rework' result loops the task sequence execution back to Task1.

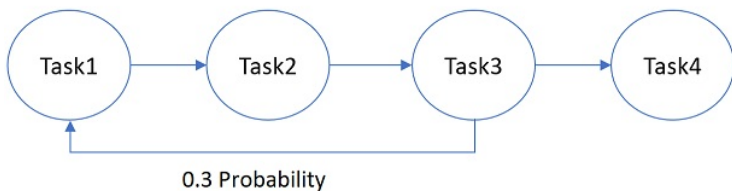


Figure 1 - Task Sequence at Server1

The processing task sequence at the second server has a conditional loopback branch in the task workflow and is shown in *Figure 2*. The processing of each entity at the server repeats a loop of Task1 through Task3 a total of 3 times. The task sequence execution then continues to Task4.

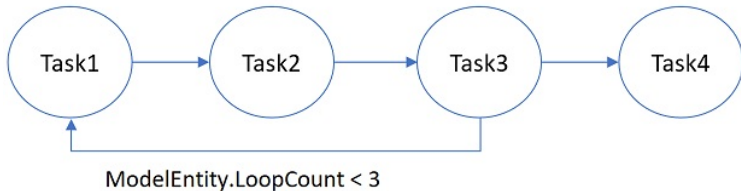


Figure 2 - Task Sequence at Server2

Technical Approach:

The operation data is defined in a set of relational data tables using alternative designs shown in *Figure 3* and *Figure 4*. The first data table design assumes at most one loopback branch from any task in the task workflow. The second data table design allows for possibly multiple loopback branches from any task.

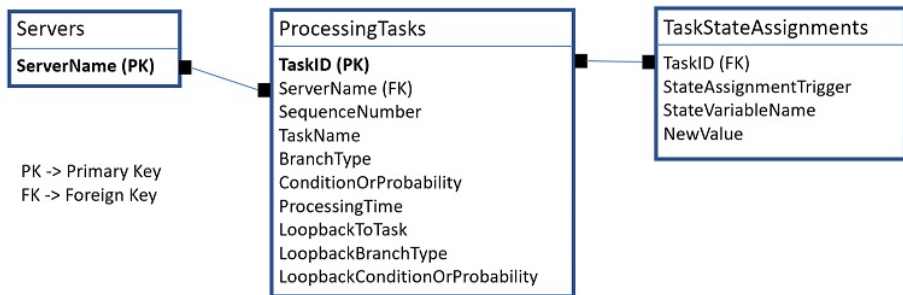


Figure 3 - Relational Data Table Design - At Most One Loopback Branch from Any Task

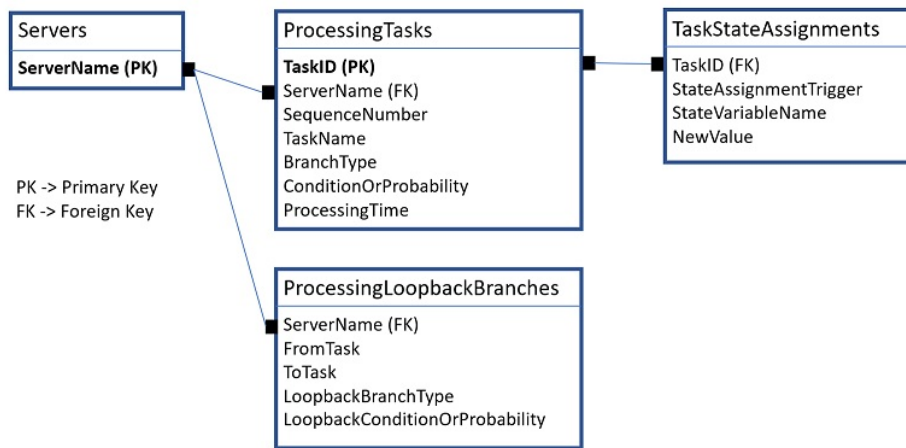


Figure 4 - Relational Data Table Design - Possibly Multiple Loopback Branches from Any Task

Details for Building the Model:

Facility Window Setup

- Add a Source, two Servers, and a Sink from the Standard Library to the Facility window. Use Connector links to connect the objects.
- Place a ModelEntity object from the Project Library into the Facility window. Then, click on ModelEntity in the Navigation window. Go to the Definitions window and click on States. Add a string *CurrentTask* state variable and an integer *LoopCount* state variable to the ModelEntity definition.

Data Table Setup

In the Data window, add data tables to the model using the relational table design shown in either Figure 3 or Figure 4. The table column property types are as follows:

Servers Data Table

Column Name	Property Type
ServerName (Primary Key)	Object Reference Property Important: Set the <i>Auto-set Table Row Reference</i> option for this column to True, so that the server objects specified in this column are automatically referencing their associated table data.

ProcessingTasks Data Table

Column Name	Property Type
TaskID (Primary Key)	Integer
ServerName	Foreign Key to Servers.ServerName
SequenceNumber	Sequence Number
TaskName	String
BranchType	Enumeration of enum type TaskBranchType
ConditionOrProbability	Expression
ProcessingTime	Expression of unit type Time
LoopbackToTask (if using data table design in <i>Figure 3</i>)	Sequence Number
LoopbackBranchType (if using data table design in <i>Figure 3</i>)	Enumeration of enum type LoopbackBranchType
LoopbackConditionOrProbability (if using data table design in <i>Figure 3</i>)	Expression

*ProcessingLoopbackBranches Data Table (if using data table design in *Figure 4*)*

Column Name	Property Type
ServerName	Foreign Key to Servers.ServerName
FromTask	Sequence Number
ToTask	Sequence Number
LoopbackBranchType	Enumeration of enum type LoopbackBranchType
LoopbackConditionOrProbability	Expression

TaskStateAssignments Data Table

Column Name	Property Type
TaskID	Foreign Key to ProcessingTasks.TaskID
StateAssignmentTrigger	List Property Note: Add a string list to the model with choices TaskReady, StartingTask, or FinishedTask and then reference that List Name here. Refer to the Simbit if necessary as a guide.
StateVariableName	State
NewValue	Expression

Once the data tables have been added, enter all required operation data into the tables. Refer to the data tables in the Simbit as a guide.

Mapping the Table Data to the Objects in the Facility Window

Go to the Facility window and do the following:

- **For Each Server object** – Specify the *Process Type* as 'Task Sequence'. Then, right-click on the Processing Tasks repeat group and set a reference to the 'ProcessingTasks' table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information. Then, right-click on the *Loopback Branches* repeat group and set a reference to either the 'ProcessingTasks' table (if using the data table design in Figure 3) of the 'ProcessingLoopbackBranches' table (if using the data table design in Figure 4).
- **For Each Server object** – In the *Process Logic->Other Task Sequence Options* properties, specify the *Task State Assignments Referenced Table Name* as the 'TaskStateAssignments' table. This indicates that table will be used as the referenced data source for getting task state assignments.
- **For Each Server object** – Open the *Processing Tasks* and *Loopback Branches* repeat groups. Map the columns from the ProcessingTasks, ProcessingLoopbackBranches, and TaskStateAssignments tables to the appropriate corresponding properties in the repeat groups (refer to the SimBit if necessary as a guide).
- In the Servers table, make sure that the *Auto-set Table Row Reference* property for the *ServerName* column is set to 'True'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServersUsingTaskSequenceWithDataTables_SequenceDependentSetups - SimBit

Problem:

I want to model a system where the entities being processed have different routings based on type, and where each operation performed at a physical processing location has a possible sequence dependent setup time. For modeling purposes, it is desired that all product mix, job routing, sequence-dependent setup times, and other operation data is defined in a set of relational data tables.

Categories:

MultiTask Server, Data Tables

Key Concepts:

Changeovers, Data Table, Process Type, RandomRow, Sequence Dependent, Sequence Table, Server, Source, Table Row Referencing, Task Sequence

Assumptions:

This SimBit builds upon concepts illustrated by the SimBit 'ServersUsingTaskSequenceWithDataTables_JobShop'. It may be useful to first review that particular example, if you have not already done so, before proceeding with this one.

The modeled system consists of a single work center with two parallel servers. An entity will enter the system, visit one of the two servers in the work center, and then exit. The processing of an entity at a server consists of a possible sequence dependent setup time, depending on product color and style attribute differences between the current job and the previous one, and then a processing time.

When a server becomes available, the next job is selected using a 'Least Setup Time' dispatching rule.

Note that, while the relational table data design in this example is oriented towards modeling a job shop, the presented system has been simplified to a single work center in order to specifically focus on the topic of modeling sequence dependent setups.

The job types processed in the system are as follows:

Job Type	Product Color	Product Style
JobTypeA	Green	StyleA
JobTypeB	Red	StyleB
JobTypeC	Blue	StyleA

Table 1 - Job Types

The setups times (in minutes) at each server to change from one product color to another are as follows:

From\To	Green	Red	Blue	If First Operation
Green	0	2	3	3
Red	1	0	4	2
Blue	3	2	0	3

Table 2 - Product Color Setup Times (in minutes)

The setups times (in minutes) at each server to change from one product style to another are as follows:

From\To	StyleA	StyleB	If First Operation
StyleA	0	3	1
StyleB	2	0	1.5

Table 3 - Product Style Setup Times (in minutes)

Technical Approach:

The product mix, job routing, and operation data is defined in a set of relational data tables using the design shown in Figure 1.

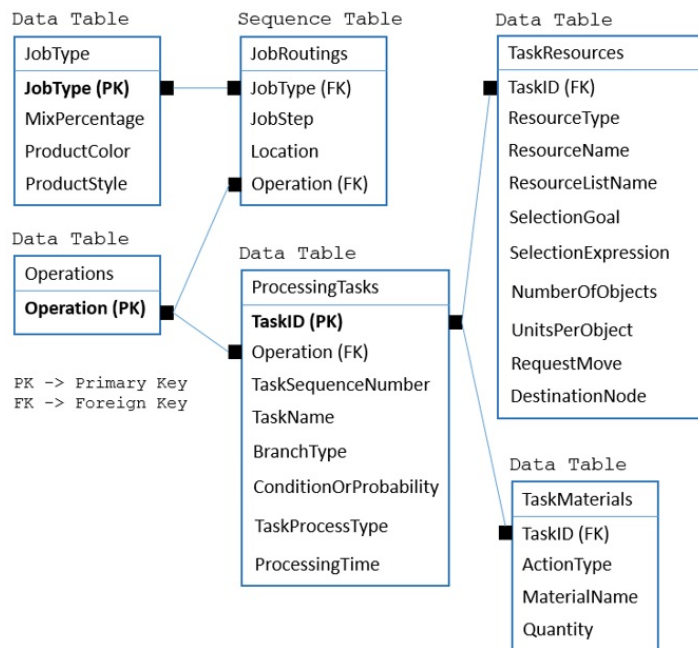


Figure 1 - Relational Data Table Design – Product Mix, Job Routing, & Operation Data

The possible sequence dependent setup times incurred at the servers are defined in a set of relational data tables using the design shown in Figure 2.

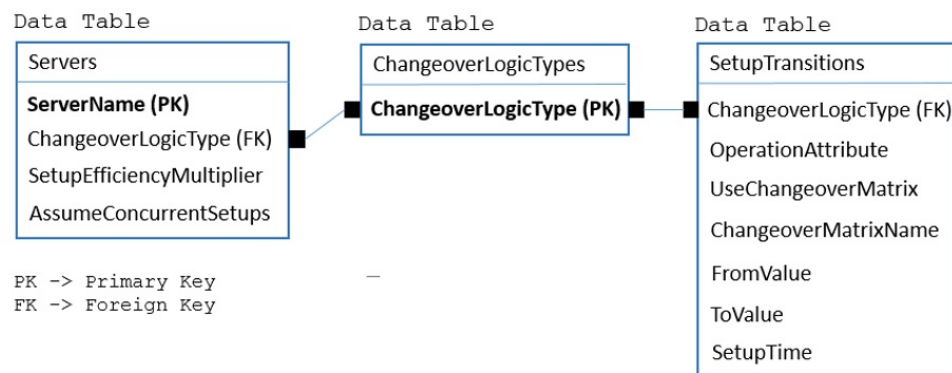


Figure 2 - Relational Data Table Design – Server Changeover Logic

Details for Building the Model:

Facility Window Setup

- Place a Source, a Sink, two Servers, and a stand-alone TransferNode from the Standard Library into the Facility window. Then connect those objects using Connectors and Paths from the Standard Library. Use the SimBit if necessary as a guide.
- Give the TransferNode the Name 'WorkCenter1'.
- Specify the Input Buffer capacity of both servers as '0'.
- Place three ModelEntity objects from the Project Library into the Facility window, naming them 'JobTypeA', 'JobTypeB', and 'JobTypeC'.
- Draw animation of the queue 'Workcenter1.RoutingOut.RouteRequestQueue'. Use the SimBit if necessary as a guide.

Node and String List Setup

- Go to Definitions, Lists and add a node list named 'WorkCenter1ServerInputNodes', a string list named 'ProductStyle', and a string list named 'ProductColor'.
- Add the nodes 'Input@Server1' and 'Input@Server2' to the WorkCenter1ServerInputNodes node list.
- Add the strings 'StyleA' and 'StyleB' to the ProductStyle string list.
- Add the strings 'Green', 'Red', and 'Blue' to the ProductColor string list.

Changeover Matrices Setup

- Go to Data, Changeover Matrices. Add two changeover matrices named 'ProductColorChangeoverTimes' and 'ProductStyleChangeoverTimes'.
- For the ProductColorChangeoverTimes, specify the String List Name as 'ProductColor' and the Time Units as 'Minutes'. Then enter the appropriate setup times in the matrix using Table 2 as a guide.
- For the 'ProductStyleChangeoverTimes', specify the String List Name as 'ProductStyle' and the Time Units as 'Minutes'. Then enter the appropriate setup times in the matrix using Table 3 as a guide.

Changeover Logic Element Setup

- Go to Definitions, Elements and add two Changeover Logic elements named 'ChangeoverLogic1' and 'ChangeoverLogic2'. These two elements will demonstrate two alternative ways to model the same changeover logic.

One approach using setup time data defined in changeover matrices and one approach using setup time data defined completely in data table form.

'WorkCenter1' TransferNode Property Configuration

- The Workcenter1 TransferNode represents the node location where entities wait for one of the two servers in the work center to become available. Specify the *Entity Destination Type* property as 'Select From List' and the *Node List Name* property as 'WorkCenter1ServerInputNodes'. Specify the *Other Routing Out Options* -> *Route Request Dynamic Selection Rule* as 'Standard Dispatching Rule' and the *Dispatching Rule* as 'LeastSetupTime'.

Data Table Setup

In the Data window, add one sequence table and eight data tables to the model using the relational table design shown in Figure 1 and Figure 2. Then enter the appropriate data into the data tables using the SimBit if necessary as a guide. The table column property types are as follows:

JobTypes Data Table

Column Name	Property Type
JobType (Primary Key)	Entity Object Reference
MixPercentage	Expression
ProductColor	List Property of list name 'ProductColor'
ProductStyle	List Property of list name 'ProductStyle'

JobRoutings Sequence Table

Column Name	Property Type
JobType	Foreign Key to JobTypes.JobType
JobStep	Integer
Location	Sequence Destination (auto added when creating Sequence Table)
Operation	Foreign Key to Operations.Operation

Operations Data Table

Column Name	Property Type
Operation (Primary Key)	String

ProcessingTasks Data Table

Column Name	Property Type
TaskID (Primary Key)	Integer
Operation	Foreign Key to Operations.Operation
TaskSequenceNumber	Sequence Number
TaskName	String
BranchType	Enumeration of enum type TaskBranchType
ConditionOrProbability	Expression
Task Process Type	Enumeration of enum type TaskProcessType
ProcessingTime	Expression of unit type Time

TaskResources Data Table

Column Name	Property Type
TaskID	Foreign Key to ProcessingTasks.TaskID
ResourceType	Enumeration of enum type ObjectSeizeType
ResourceName	Object Reference
ResourceListName	Object List Reference
SelectionGoal	Enumeration of enum type SeizeSelectionGoal
SelectionExpression	Expression
NumberOfObjects	Expression
UnitsPerObject	Expression
RequestMove	Enumeration of enum type SeizeRequestVisitType
DestinationNode	Node Object Reference

TaskMaterials Data Table

Column Name	Property Type
TaskID	Foreign Key to ProcessingTasks.TaskID
ActionType	Enumeration of enum type MaterialActionType
MaterialName	Material Element Reference
Quantity	Expression

Servers Data Table

Column Name	Property Type
ServerName (Primary Key)	Object Reference
ChangeoverLogicName	Foreign Key to ChangeoverLogicTypes.ChangeoverLogicType
SetupEfficiencyMultiplier	Expression
AssumeConcurrentSetups	Boolean

Changeover Logic Types Data Table

Column Name	Property Type
ChangeoverLogicType (Primary Key)	Changeover Logic Element Reference

Setup Transitions Data Table

Column Name	Property Type
ChangeoverLogicType	Foreign Key to ChangeoverLogicTypes.ChangeoverLogicType
OperationAttribute	Expression
UseChangeoverMatrix	Boolean
ChangeoverMatrixName	Changeover Matrix Reference
FromValue	Expression
ToValue	Expression
SetupTime	Expression of unit type Time

Mapping the Table Data to the Objects in the Facility Window

Go to the Facility window and do the following:

- Source object – Specify the *Entity Type* property as 'JobTypes.JobType'. Go to the *Table Row Referencing* -> *Before Creating Entities* properties, and specify the *Table Name* as 'JobTypes' and the *Row Number* as 'JobTypes.MixPercentage.RandomRow'. The Source will then randomly create entities of the types specified in the JobTypes table using the specified mix percentages. Change the *Interarrival Time* to 'Random.Exponential(1.5)' minutes.
- For Each 'Output' TransferNode object (the Source and two Servers) – Specify the *Entity Destination Type* as 'By Sequence'. Then, when an entity exits the Source or either of the Server objects, it will set its next destination based on the JobRoutings sequence table.
- For Each Server object – Specify the *Process Type* as 'Task Sequence'. Then, right-click on the *Processing Tasks* repeat group and set a reference to the ProcessingTasks table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information. Finally, in the *Process Logic* -> *Other Task Sequence Options* properties, specify the *Task Resources Referenced Table Name* as 'TaskResources' and the *Task Materials Referenced Table Name* as 'TaskMaterials'. This indicates that those tables will be used as the referenced data sources for getting task resource and material requirements.
- For Each Server object – Open the *Processing Tasks* repeat group. Map the columns from the ProcessingTasks, TaskResources, TaskMaterials, and Servers tables to the appropriate corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).
- For Each Server object – Go to *Advanced Options* -> *Expected Setup Time Expression* and enter the expression 'Servers.ChangeoverLogicType.ExpectedSetupTime(Server, Entity)'. This will be the expression used to estimate expected setup times when selecting the next entity to send to an available server using the 'LeastSetupTime' dispatching rule.

Go to the Definitions -> Elements and do the following:

- For Each Changeover Logic Element - Right-click on the *Setup Transitions* repeat group and set a reference to the SetupTransitions table. This indicates that the SetupTransitions table will be used as the referenced data source for getting that information for the changeover logic. Then open the *Setup Transitions* repeat group. Map the columns from the SetupTransitions table to the appropriate corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).
- For Each Changeover Logic Element – Right-click on the *Setup Efficiency Multiplier* property and set a reference to the table property Servers.SetupEfficiencyMultiplier. Right-click on the *Assume Concurrent Setups If* property and set a reference to the table property Servers.AssumeConcurrentSetups.

Embellishments:

Experiment with different changeover logic, concurrent setup assumptions, and/or setup efficiency multipliers for the servers and note the differences in the total incurred changeover time.

Also note that the SimBit currently has no data within the TaskResources and TaskMaterials tables, but can easily be added by including data within these tables, as the mapping to the Task Sequences is already done.

See also:

ServersUsingTaskSequencesWithDataTables_JobShop.spfx

ServerUsingTaskSequence - SimBit

This SimBit project includes three models that demonstrate the use of the Server's Task Sequence capability.

Models included in this SimBit:

1. **MultiTaskServer_SpecificTime** – Demonstrates the Server's task sequence option of using a Specific Time delay for a task. The time delay is then specified directly within the dialog for each particular task.
2. **MultiTaskServer_ProcessName** – Demonstrates the Server's task sequence option of using a Process Name for a task. The process logic is then included within a specific process within the Processes window.
3. **MultiTaskServer_Submodel** – Demonstrates the Server's task sequence option of using a Submodel to model the process flow for the task. The process logic for the task is then determined by the Facility window entity flow starting at the Submodel Starting Node and continuing until that process flow is complete and the entity is destroyed.

Model 1: MultiTaskServer SpecificTime

Problem:

I want to model multiple tasks within a similar physical location. My entity has a single task first, but then has two tasks that are done in parallel. My tasks are all represented with simple delays.

Categories:

MultiTask Server

Key Concepts:

Active Symbol, Add-On Process, Assign Step, Finished Task, Picture, Process, Process Type, Server, Task Sequence

Assumptions:

Only a single entity is flowing through the system so that the focus can be on the task flow of that single entity. The entity will go through 4 different simple delay tasks, some of which can be done simultaneously.

Technical Approach:

Use the *Process Type* of 'Task Sequence' on the Server object to specify all the tasks, their dependencies and logic.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink from the Standard Library in the Facility window. Connect the Source to the Server and the Server to the Sink with Paths.
- Within the Source, change the *Maximum Arrivals* property to '1' so that only a single entity enters the system.

Adding Tasks to the Server

- Within the Server, change the *Process Type* to 'Task Sequence'. Within the *Processing Tasks* property, open the repeating property editor and specify four different tasks.
- *Sequence Number* set to '10' for the first task with the *Name* of 'Clean'. Within this task, the *Branch Type* is 'Always' (all entities perform this task). The *Process Type* is 'Specific Time' and the *Processing Time* is '1' minute.
- The next two tasks can be done in parallel and can only be started once the first task, 10, is done. Therefore we will set the *Sequence Number* set to '20.1' for the first task with the *Name* of 'Trim1'. Within this task, the *Branch Type* is 'Always'. The *Process Type* is 'Specific Time' and the *Processing Time* is also '1' minute. The other parallel task is *Sequence Number* '20.2', with the *Name* of 'Paint'. The *Branch Type* is 'Always', *Process Type* is 'Specific Time' and *Processing Time* is '5' minutes. The extension from the 20 (20.1 and 20.2) indicates that they can be done in parallel.
- Finally, a task is performed after task 20.1 but can be done prior to 20.2 being complete. Therefore, we will indicate the *Sequence Number* set to '30.1' for the task with the *Name* of 'Trim2'. Within this task, the *Branch Type* is 'Always', the *Process Type* is 'Specific Time' and the *Processing Time* is '1' minute.

Animating the Entity Changes

- Place a ModelEntity from the Project Library into the Facility window. Click on the green symbol and within the

Symbols ribbon, click on Add Additional Symbol to create 5 different symbols (which you will then color with unique colors) for various symbol states.

- When any of the tasks is completed, the animation symbol for the entity (ModelEntity.Picture) will be updated to show that a task has been completed. Within the Processes window, add a process named 'Server1_FinishedTask'. Add an Assign step to this process to assign the *State Variable Name* 'ModelEntity.Picture' to the *New Value* 'ModelEntity.Picture + 1'.
- Re-enter the Server's *Processing Tasks* repeating editor and change the *Finished Task* add-on process to be 'Server1_FinishedTask' for each of the tasks shown. All tasks can use this same process to simply update the entity picture symbol by one each time.

Model 2: MultiTaskServer_ProcessName

Problem:

I want to model multiple more complicated tasks within a similar physical location. My entity has a single task first, but then has two tasks that are done in parallel. The logic for each task is defined within processes in the Processes window.

Categories:

MultiTask Server

Key Concepts:

Active Symbol, Add-On Process, Assign Step, Delay Step, Picture, Process, Process Type, Server, Task Sequence

Assumptions:

Only a single entity is flowing through the system so that the focus can be on the task flow of that single entity. The entity will go through 4 different tasks, some of which can be done simultaneously. The task logic for each task is represented in different processes in the Processes window.

Technical Approach:

Use the *Process Type* of 'Task Sequence' on the Server object to specify all the tasks, their dependencies and logic.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink from the Standard Library in the Facility window. Connect the Source to the Server and the Server to the Sink with Paths.
- Within the Source, change the *Maximum Arrivals* property to '1' so that only a single entity enters the system.

Adding Process Logic for Each Task

- Within the Processes window, use the Create Process to create four unique processes with the *Names* of 'Clean_Process', 'Paint_Process', 'Trim1_Process' and 'Trim2_Process'.
- Within the Clean_Process, add a Delay step with the *Delay Time* of '1' minute. Add an Assign step that will assign the *State Variable Name* of 'ModelEntity.Picture' to the *New Value* of 'ModelEntity.Picture + 1'. The picture assignment will be discussed more below.
- Within the Trim1_Process and Trim2_Process, add the same two steps as above (Delay for 1 minute and Assign entity picture). * NOTE: If these processes for multiple tasks were always going to remain the same, you could alternatively have a single process that has the Delay and Assign and reference that within the Task (steps below).
- Within the Paint_Process, add a Delay step with the *Delay Time* of '5' minutes. Add an Assign step that will assign the *State Variable Name* of 'ModelEntity.Picture' to the *New Value* of 'ModelEntity.Picture + 1'.

Adding Tasks to the Server

- Within the Server, change the *Process Type* to 'Task Sequence'. Within the *Processing Tasks* property, open the repeating property editor and specify four different tasks.
- *Sequence Number* set to '10' for the first task with the *Name* of 'Clean'. Within this task, the *Branch Type* is 'Always' (all entities perform this task). The *Process Type* is 'Process Name' and the *Process Name* refers to the 'Clean_Process' that was created above.
- The next two tasks can be done in parallel and can only be started once the first task, 10, is done. Therefore we will set the *Sequence Number* set to '20.1' for the first task with the *Name* of 'Trim1'. Within this task, the *Branch Type* is 'Always'. The *Process Type* is 'Process Name' and the *Process Name* is 'Trim1_Process'. The other parallel task is *Sequence Number* '20.2', with the *Name* of 'Paint'. The *Branch Type* is 'Always', *Process Type* is 'Process Name' and *Process Name* is 'Paint_Process'. The extension from the 20 (20.1 and 20.2) indicates that these two tasks can be done

in parallel.

- Finally, a task is performed after task 20.1 but can be done prior to 20.2 being complete. Therefore, we will indicate the *Sequence Number* set to '30.1' for the task with the *Name* of 'Trim2'. Within this task, the *Branch Type* is 'Always', the *Process Type* is 'Process Name' and the *Process Name* is 'Trim2_Process'.

Animating the Entity Changes

- Place a ModelEntity from the Project Library into the Facility window. Click on the green symbol and within the Symbols ribbon, click on Add Additional Symbol to create 5 different symbols (which you will then color with unique colors) for various symbol states.
- When any of the tasks is completed, the animation symbol for the entity (ModelEntity.Picture) will be updated to show that a task has been completed. Remember that within the Processes window, we have Assign steps at the end of each process to assign the ModelEntity.Picture to be one more than it previously was.

Model 3: MultiTaskServer Submodel

Problem:

I want to model multiple tasks that may be done in different physical locations while the entity remains at a Server. My entity has a single task first, but then has two tasks that are done in parallel at different locations.

Categories:

MultiTask Server

Key Concepts:

Active Symbol, Add-On Process, Assign Step, Finished Task, Picture, Process Type, Server, Submodel, Task Sequence

Assumptions:

Only a single entity is flowing through the system so that the focus can be on the task flow of that single entity. The entity will go through 4 different tasks, some of which can be done simultaneously. The task logic for the last three tasks is represented within the Facility window.

Technical Approach:

Use the *Process Type* of 'Task Sequence' on the Server object to specify all the tasks, their dependencies and logic.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink from the Standard Library in the Facility window. Connect the Source to the Server and the Server to the Sink with Paths.
- Within the Source, change the *Maximum Arrivals* property to '1' so that only a single entity enters the system.

Submodel Logic for Tasks in Facility Window

- Place three Servers and Sinks in the Facility window (in addition to those placed above). Connect each Server to a Sink with a Path. *Name* the first Server 'Lab1_Test', the second one 'Lab2_Test' and the third 'XRay_Test'. Within both the Lab1_Test and Lab2_Test servers, specify the *Processing Time* of '1' minute. Within the XRay_Test, specify the *Processing Time* of '5' minutes.

Adding Tasks to the Server

- Within the Server, change the *Process Type* to 'Task Sequence'. Within the *Processing Tasks* property, open the repeating property editor and specify four different tasks.
- *Sequence Number* set to '10' for the first task with the *Name* of 'Checkup'. Within this task, the *Branch Type* is 'Always' (all entities perform this task). The *Process Type* is 'Specific Time' and the *Processing Time* is '1' minute.
- The next two tasks can be done in parallel and can only be started once the first task, 10, is done. Therefore we will set the *Sequence Number* set to '20.1' for the first task with the *Name* of 'Lab1'. Within this task, the *Branch Type* is 'Always'. The *Process Type* is 'Submodel', the *Submodel Entity Type* is 'DefaultEntity' and the *Submodel Starting Node* is 'Input@Lab1_Test'. The other parallel task is *Sequence Number* '20.2', with the *Name* of 'XRay'. The *Branch Type* is 'Always', *Process Type* is 'Submodel', *Submodel Entity Type* is 'DefaultEntity' and *Submodel Starting Node* is 'Input@XRay_Test'. The extension from the 20 (20.1 and 20.2) indicates that they can be done in parallel.
- Finally, a task is performed after task 20.1 but can be done prior to 20.2 being complete. Therefore, we will indicate the *Sequence Number* set to '30.1' for the task with the *Name* of 'Lab2'. Within this task, the *Branch Type* is 'Always', the *Process Type* is 'Submodel', *Submodel Entity Type* is 'DefaultEntity' and the *Submodel Starting Node* is

'Input@Lab2_Test'.

Animating the Entity Changes

- Place a ModelEntity from the Project Library into the Facility window. Click on the green symbol and within the Symbols ribbon, click on Add Additional Symbol to create 5 different symbols (which you will then color with unique colors) for various symbol states.
- When any of the tasks is completed, the animation symbol for the entity (ModelEntity.Picture) will be updated to show that a task has been completed. Within the Processes window, add a process named 'Server1_FinishedTask'. Add an Assign step to this process to assign the *State Variable Name* 'ModelEntity.Picture' to the *New Value* 'ModelEntity.Picture + 1'.
- Re-enter the Server's *Processing Tasks* repeating editor and change the *Finished Task* add-on process to be 'Server1_FinishedTask' for each of the tasks shown. All tasks can use this same process to simply update the entity picture symbol by one each time.

Required introduction

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

ServerUsingTaskSequence_AlternativeMethodsForDefiningTaskPrecedence - SimBit

This SimBit project includes three models using the Server object that demonstrate the alternative methods available to define the task precedence dependencies for a task sequence.

Models included in this SimBit:

- **SequenceNumberMethodExample** - Demonstrates how to define the task precedence dependencies for a task sequence by using task sequence numbers.
- **ImmediatePredecessorsMethodExample** - Demonstrates how to define the task precedence dependencies for a task sequence by specifying the immediate predecessors for each task.
- **ImmediateSuccessorsMethodExample** - Demonstrates how to define the task precedence dependencies for a task sequence by specifying the immediate successors for each task.

Figure 1 illustrates the example task sequence that is demonstrated by each of the example models using an alternative task precedence method. First 'Task1' is started. When 'Task1' is finished then both 'Task2' and 'Task3' can start. When 'Task2' is finished, then 'Task4' can start. When 'Task3' is finished, then 'Task5' can start. Finally, 'Task6' can start once both 'Task4' and 'Task5' have finished.

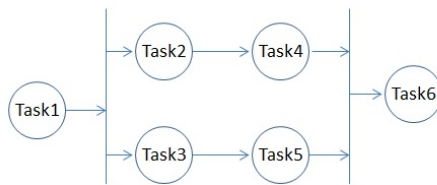


Figure 1 - Example Task Sequence

Model 1: SequenceNumberMethodExample

Problem:

I want to model the example task sequence shown in Figure 1 at a Server object using task sequence numbers to define the task precedence dependencies.

Categories:

MultiTask Server

Key Concepts:

Data Table, Server, Process Type, Task Sequence, Task Precedence Method

Technical Approach:

Table 1 below shows the task data using the 'Sequence Number' task precedence method that will produce the example task sequence shown in Figure 1.

Sequence Number	Task Name
10	Task1
20.1	Task2
20.2	Task3
30.1	Task4
30.2	Task5
40	Task6

Table 1 - Example data using the 'Sequence Number' method to define the task precedence constraints

Details for Building the Model:

Facility Window Setup

- Add a Source, Server, and Sink from the Standard Library to the Facility window. Then connect those objects using Connectors from the Standard Library, drawing a connection from the Source to the input of the Server, then a connection from the output of the Server to the input of the Sink.
- Place a ModelEntity from the Project Library into the window. Make sure the *Entity Type* property for the Source object is specifying that model entity type.

Data Table Setup

In the Data window, add a 'Processing Tasks' data table to the model. The table column property types are as follows:

ProcessingTasks Data Table

Column Name	Property Type
Sequence Number	Sequence Number
Name	String
ProcessingTime	Expression of unit type Time

Mapping the Table Data to the Server in the Facility Window

Go to the Server object in the Facility window and do the following:

- Specify the *Process Type* as 'Task Sequence'. Then, right-click on the *Processing Tasks* repeat group and set a reference to the 'ProcessingTasks' table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information.
- Open the *Processing Tasks* repeat group. Map the columns from the ProcessingTasks table to the appropriate

corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).

Model 2: ImmediatePredecessorsMethodExample

Problem:

I want to model the example task sequence shown in Figure 1 at a Server object, specifying the immediate predecessors for each task to define the task precedence dependencies.

Categories:

MultiTask Server

Key Concepts:

Data Table, Server, Process Type, Task Sequence, Task Precedence Method

Technical Approach:

Table 2 illustrates some task data using the 'Immediate Predecessors' task precedence method that will produce the example task sequence shown in Figure 1.

ID Number	Task Name	Immediate Predecessors
1	Task1	
2	Task2	1
3	Task3	1
4	Task4	2
5	Task5	3
6	Task6	4,5

Table 2 – Example data using the 'Immediate Predecessors' method to define the task precedence constraints

Details for Building the Model:

Facility Window Setup

- Add a Source, Server, and Sink from the Standard Library to the Facility window. Then connect those objects using Connectors from the Standard Library, drawing a connection from the Source to the input of the Server, then a connection from the output of the Server to the input of the Sink.
- Place a ModelEntity from the Project Library into the window. Make sure the *Entity Type* property for the Source object is specifying that model entity type.

Data Table Setup

In the Data window, add a 'Processing Tasks' data table to the model. The table column property types are as follows:

ProcessingTasks Data Table

Column Name	Property Type
ID Number	Integer
Name	String
ProcessingTime	Expression of unit type Time
ImmediatePredecessors	TaskDependency

Mapping the Table Data to the Server in the Facility Window

Go to the Server object in the Facility window and do the following:

- Specify the *Process Type* as 'Task Sequence'. Then, right-click on the *Processing Tasks* repeat group and set a reference to the 'ProcessingTasks' table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information.
- Go to the *Process Logic* -> *Other Task Sequence Options* properties and specify the *Task Precedence Method* as 'Immediate Predecessors Method'.
- Open the *Processing Tasks* repeat group. Map the columns from the ProcessingTasks table to the appropriate corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).

Model 3: ImmediateSuccessorsMethodExample

Problem:

I want to model the example task sequence shown in Figure 1 at a Server object, specifying the immediate successors for each task to define the task precedence dependencies.

Categories:

MultiTask Server

Key Concepts:

Data Table, Server, Process Type, Task Sequence, Task Precedence Method

Technical Approach:

Table 3 illustrates some task data using the 'Immediate Successors' task precedence method that will produce the example task sequence shown in Figure 1.

ID Number	Task Name	Immediate Successors
1	Task1	2,3
2	Task2	4
3	Task3	5
4	Task4	6
5	Task5	6
6	Task6	

Table 3 – Example data using the 'Immediate Successors' method to define the task precedence constraints

Details for Building the Model:

Facility Window Setup

- Add a Source, Server, and Sink from the Standard Library to the Facility window. Then connect those objects using Connectors from the Standard Library, drawing a connection from the Source to the input of the Server, then a connection from the output of the Server to the input of the Sink.
- Place a ModelEntity from the Project Library into the window. Make sure the *Entity Type* property for the Source object is specifying that model entity type.

Data Table Setup

In the Data window, add a 'Processing Tasks' data table to the model. The table column property types are as follows:

ProcessingTasks Data Table

Column Name	Property Type
ID Number	Integer
Name	String
ProcessingTime	Expression of unit type Time
ImmediateSuccessors	TaskDependency

Mapping the Table Data to the Server in the Facility Window

Go to the Server object in the Facility window and do the following:

- Specify the *Process Type* as 'Task Sequence'. Then, right-click on the *Processing Tasks* repeat group and set a reference to the 'ProcessingTasks' table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information.
- Go to the *Process Logic* -> *Other Task Sequence Options* properties and specify the *Task Precedence Method* as 'Immediate Successors Method'.
- Open the *Processing Tasks* repeat group. Map the columns from the ProcessingTasks table to the appropriate corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).

See Also

ServerUsingTaskSequence.spfx

ServersUsingTaskSequencesWithDataTables_FlowLine.spfx

ServersUsingTaskSequencesWithDataTables_JobShop.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServerUsingTaskSequenceWithSequenceDependentSetups - SimBit

Problem:

I have a Server that requires a certain setup time based on which entity type needs to be processed.

Categories:

MultiTask Server

Key Concepts:

Changeovers, List Property, Sequence Dependent, Server, Status Pie, StringList, Time Offset

Assumptions:

Each source only produces one type of entity. The model will complete its run faster if the Server processes all entities of one type before changing over to process entities of another type.

Technical Approach:

A String List is created in the List panel of the Definitions window for the ModelEntity. A List property is defined in the Model Entity's Definitions window, Properties panel. A Server with Task Sequences and a Sequence Dependent Setup type task is used and a Changeover matrix defines the various setup times. A Time Offset will be used to demonstrate the impact of setup times.

Details for Building the Model:

Simple System Setup

- Add two Sources, a Server and two Sinks to the Facility window. Change the *Interarrival Times* of the Sources to '0.1' minutes and the *Maximum Arrival* of each Source to '10'. Also set the *Time Offset* property of one of the Sources to '5' minutes.
- Connect the Sources to the Server and the Server to each Sink with Paths.

Adding a String List and List Property to the ModelEntity

- Click on the ModelEntity (within the Navigation window). Open the Definitions tab and select the Lists panel to add a String List named 'Colors'. Add as many colors as desired.
- Within the Definitions tab, select the Properties panel and add a List Property with the *Name* 'Color'. Set the *List Name* property to 'Colors' and the *Category Name* property under Appearance to 'Properties'. This will give the user the option to specify Color as a property of an entity and the list of Colors will be shown as options.

Adding a String List to the Model

- Click on the Model within the Navigation window to continue modifying the simulation model (instead of the ModelEntity). Go to the Definitions tab and select the Lists panel to add a String List named 'Colors'. Add the same colors, in order, as defined with the ModelEntity string list.

Creating a Changeover Matrix

- In the Data Window, select the Changeovers panel. Create a Changeover Matrix and change the *Name* to 'Cleaning'. Change the *List Name* property to be 'Colors'.
- Fill in the matrix with desired values or follow the example: (Note: these values will be in Minutes):

ChangeOver Matrix Inputs

↓ From \ To →	White	Yellow	Red	Blue	Black
White	0	0.1	0.1	0.1	0.1
Yellow	1	0	0.1	0.1	0.1
Red	1	0.5	0	0.3	0.4
Blue	1	0.5	0.6	0	0.3
Black	1	0.8	0.8	0.5	0

Adding a Changeover Logic Element

- Within the Definitions tab, Elements panel, add a new Changeover Logic element with the *Name* 'ChangeoverLogic1'. Enter the Setup Transitions repeat group and specify the *Operation Attribute* as 'ModelEntity.Color', the *Use Changeover Matrix* as 'True' and the *Changeover Matrix Name* as 'Cleaning' (which was just defined above).

Using the Server with Task Sequences

- The Server will need to have different setup times depending on the sequence of entities in its queue. This is done by changing the *Process Type* property from 'Specific Time' to 'Task Sequence' and entering the Processing Tasks repeat

group. Within the repeat group, add two tasks.

- For the first task, add the *Name* 'Setup' and change the *Process Type* to 'Sequence Dependent Setup'. Set the *Changeover Logic Name* to 'ChangeoverLogic1'.
- For the second task, add the *Name* 'Processing' and specify the *Processing Time* as '1' minute.

Adding an Entity to the Model

- Place two ModelEntity objects from the Project Library into the Facility window. Name one 'RedEntity' and one 'BlueEntity'. Under the Properties section, select the desired color from the Color property drop down list. Then change the color of each entity to match that choice.
- Change the *Entity Type Arrival Logic* property of each Source to match the color of its associated entity.
- Set the *Selection Weight* of the path from the Server to the Sink to 'ModelEntity.Color==List.Colors.[Desired Color]'.
- Add as many Sources and Sinks as desired to the model.

Changing the Server Animation

- Click on Server1 and within the Symbols ribbon, select the Active Symbol button. Select the Setup (7) animation and change the Server color to turquoise (or any color different than the processing color so users can distinguish the Setup and Processing states).

Embellishments:

Try changing the Processing Time of the Server, the Interarrival Times of the Sources, or change values in the Changeover Matrix.

See Also:

[ServersUsingTaskSequenceWithDataTables_SequenceDependentSetups](#)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServerUsingTaskSequenceWithWorkers - SimBit

Problem:

I want to model multiple tasks within a similar physical location. Each task requires a worker to help perform the task. Most tasks are done all the time, while one is done only a percentage of the time.

Categories:

Add-On Process Logic, MultiTask Server, Worker

Key Concepts:

Active Symbol, Add-On Process, Assign Step, BasicNode, Decide Step, InputBuffer, ObjectList, Picture, Process, Process Type, Request Move, Server, Started Task, Status Label, Task Sequence, Worker

Assumptions:

Only a single entity will process at a Server at a time and incoming entities will wait at the Source until a Server is idle before being assigned to one of the two Servers. All tasks are sequential, there are no parallel tasks.

One of the tasks (lab work) is a probabilistic task. We will assume that even if that task is not completed for a given entity that all remaining tasks will continue as planned.

Technical Approach:

Use the *Process Type* of 'Task Sequence' on the Server objects to specify all the tasks, their dependencies, logic and resources required. Tasks are sequential so task numbering will be simple at 10, 20, 30, 40.

Details for Building the Model:

Simple System Setup

- Place a Source, a Server (we will later copy this first one to another) and Sink from the Standard Library in the Facility window. Connect the Source to the Server and then the Server to the Sink with Paths.
- Within the Source, change the *Interarrival Time* property to 'Random.Exponential(6)' to send entities into the system.
- Place two BasicNode objects in the model, both near the bottom of the Facility window below the Server to the left and right. Rename them NursesStation and Lab. Place another Basic Node near the Server and name it 'ExamRoomNode'. This will be where the nurse/doctor visit when performing an entity task. Do not connect the nodes with Paths just yet, we will do that shortly.

Defining Workers

- Place three Worker objects in the Facility window and rename them 'Nurse1', 'Nurse2' and 'Doctor'. Change each of the worker's *Initial Node (Home)* property to 'NursesStation' and *Idle Action* to 'Park At Home'. This will cause them all to be located at the BasicNode NursesStation at the start of the simulation and they will travel back to this area when they are idle.
- Within the Definitions tab, Lists panel, add a Transporter List named 'Nurses' and include Nurse1 and Nurse2.

Adding Tasks to the Server

- Within Server, change the *Process Type* to 'Task Sequence'. Within the Processing Tasks property, open the repeating property editor and specify four different tasks.
- *Sequence Number* set to '10' for the first task with the *Name* of 'Check Vitals'. Within this task, the *Branch Type* is 'Always' (all entities perform this task). The *Process Type* is 'Specific Time' and the *Processing Time* is '3' minutes. Within the Resource Requirements section, we will be requiring a nurse to perform this task. Set the *Object Type* to 'Select From List' and the *Object Name* to 'Nurses' meaning we will select from among the available nurses. Additionally, set the *Request Move* property to 'To Node' and *Destination Node* to 'ExamRoomNode'. This will require the nurse to move to the Server to perform the task (as opposed to staying at its current location).
- Add another processing task with a *Sequence Number* set of '20' and the *Name* of 'Lab Work'. Within this task, the *Branch Type* is 'Probabilistic' with the *Condition or Probability* as '.8', meaning approximately 80% of the entities will

require this task. The *Process Type* is 'Specific Time' and the *Processing Time* is '2' minutes. Within the Resource Requirements section, we will be requiring a nurse to again perform this task. Set the *Object Type* to 'Select From List' and the *Object Name* to 'Nurses' meaning we will select from among the available nurses. Additionally, set the *Request Move* property to 'To Node' and *Destination Node* to 'Lab'. This will require the nurse to move to the Lab to perform the task, even though the entity is still at the Server1 location.

- The third processing task should have a *Sequence Number* set of '30' and the *Name* of 'Nurse Visit'. Within this task, the *Branch Type* is 'Always'. The *Process Type* is 'Specific Time' and the *Processing Time* is '3' minutes. Within the Resource Requirements section, we will be requiring a nurse so specify the same information as the previous two tasks. Set the *Request Move* property to 'To Node' and *Destination Node* to 'ExamRoomNode' so that the nurse returns to Server1 for processing. **IMPORTANT NOTE:** Within the Task Information section of properties, change the *Auto Cancel Trigger* to 'None'. This is very important because this property can allow the cancellation of this task if all predecessor tasks are cancelled if left at the default value (if the probability based lab work task isn't done, the 'None' property value for this field makes sure that the task is not automatically cancelled).
- Finally, the last processing task should have a *Sequence Number* set of '40' and the *Name* of 'Doctor Visit'. Within this task, the *Branch Type* is 'Always'. The *Process Type* is 'Specific Time' and the *Processing Time* is '5' minutes. Within the Resource Requirements section, we will now be requiring the *Object Type* of 'Specific' and *Object Name* of 'Doctor'. Set the *Request Move* property to 'To Node' and *Destination Node* to 'ExamRoomNode' so that the doctor moves to the Server1 for processing.
- Within the main properties of the Server, under the Buffer Capacities section, change the *Input Buffer* to '0'. This will make sure that entities don't accumulate in the Server1 buffer while waiting. We will add some logic to the Source to help with this as well in the below section.
- **NOTE** about Resources Utilized within Processing Tasks – When using Resources within various processing tasks in the Server (Task Sequence based), the resource is 'Reserved', by default, for processing from one task to another. This is done with the two properties *Keep Reserved If* (set to 'True') and *Reservation Timeout* (set to 'Math.Epsilon'). This will ensure that a resource used in one task will not be allocated to another entity in another area when it is to be used for another processing task immediately after (such as in this example, the nurse is used for 3 tasks in a row).

Adding the Second Server and Associated Logic

- Because our Servers will have similar task information, we chose to fill out all the task information first within a single Server, Server1. Now let's copy the Server by highlighting Server1 and using Ctrl-C and Ctrl-V to copy and paste. Change the *Name* of the copied Server to 'Server2'. This way, all 4 tasks within the Processing Tasks are copied and you can simply edit them.
- First, place another BasicNode object near Server2 and name it 'ExamRoomNode2'. This is where the nurses/doctor will travel for processing at this server.
- Edit the Processing Tasks and change the *Destination Node* for the requested move of the workers from 'ExamRoomNode' to 'ExamRoomNode2' for the 10, 30 and 40 sequence number tasks (done at the server). We will still require the lab work to be done at the lab. Change the Lab Work task *Condition or Probability* to be '.7', requiring only 70% of the patients to need lab work from this server.

Connecting the Worker Paths

- Note that we now have 4 BasicNode objects for the workers (Doctor/Nurses) to move between when processing. These workers in this example will not move along any of the same paths as the entities. Note also that 'Bidirectional' type paths, when multiple Workers (and/or Vehicles) are moving can cause potential lockups unless by-pass type paths are used. Therefore, we will avoid that all together and simply draw two unidirectional paths between the various nodes. Connect the NursesStation node to the ExamRoomNode in both directions with Paths. Then, connect the ExamRoomNode to the ExamRoomNode2 with Paths. Finally, connect the ExamRoomNode to the Lab with Paths in both directions.

Selecting Between Multiple Servers from Source

- In this example, we'd like our entities to wait at the Source object if a Server isn't available for processing (no buffer space at Servers). Note that if we simply left the '0' *Input Buffer* on the Servers and no other logic, entities could reside on the Paths connecting the Source and each Server. Therefore, we will first make a List of possible Server options from which to select. Go to the Definitions tab and Lists panel. Add a Node list with the *Name* 'ExamRooms' that includes the 'Input@Server1' and the 'Input@Server2' as the members of the list.
- Within the Facility window, click on the output node of Source1. Change the *Entity Destination Type* to 'Select From List'. The *Node List Name* should be 'ExamRooms'. The *Selection Goal* of 'Preferred Order' will select a node from the list based on the listed order (Server1 first, then Server2). The *Blocked Destination Rule* of 'Select Available Only' will cause the entities to wait at the Source output area if there is no capacity available at the Servers.

Animating the Processing Task Name of the Servers

- Note that in this example, the processing task name is graphically animated for each of the two Servers. It is possible in this particular example for two reasons. First, the Server *Initial Capacity* itself is '1' and does not change. That means that ONLY ONE entity will ever be able to go through the Process Tasks listed within the Server at any given time. Second, the entity processing tasks at the Server are all serial tasks, meaning that only one task is occurring at a time, then the next, and so on. If the entity had multiple tasks occurring simultaneously, signified by the Sequence Number values, the task names would not be able to be displayed as they are.
- When an entity is processing through sequence tasks in the Server, the token(s) associated with that entity have a function 'Token.Task.Name' that carries the task sequence name (i.e., Check Vitals, Lab Work, etc.). Because of the above discussion, each Server will only be going through one task at a time, and therefore we will add two model states to the model. Within the Definitions tab, States panel, add two String type states and name them 'Server1_Task' and 'Server2_Task'. These states will be assigned a value based on the Token.Task.Name function when the task changes.
- In the Processes window, create a new process named 'Task_Assignment'. Within this process, add a Decide step. The *Condition or Probability* property should be 'ModelEntity.Location.Parent == Server1'. This will check the location of the entity (is it at Server1 or Server2 so we know which state variable to update). From the True exit, add an Assign step that assigns the *State Variable Name* 'Server1_Task' to the *New Value* of 'Token.Task.Name' and then from the False exit, add a similar Assign steps that assigns the *State Variable Name* 'Server2_Task' to the *New Value* 'Token.Task.Name'.
- In order to update these variables after every task, enter the Processing Tasks repeating property window for the servers and under the Add-On Process Triggers section, add the *Starting Task* process as 'Task_Assignment' for every task in each server. This is done in the *Starting Task* section (instead of the Finished Task section) since we would like to update the variable in the animation just prior to the task starting and view it during the task. The value will then change when the task changes.
- Optional – The variables will not automatically 'clear' when the entity is done processing. To clear the variables, make a similar process to the above described with Decide and two Assign steps. Assign the state variables to " " (blank, as they are strings but need to be in quotes). On the last task for each Server, reference this new process in the *Finished Task* add-on process trigger.
- Finally, the state variables can be graphically shown in the Facility window by clicking on the Animation ribbon and placing Status Labels with the *Expression* values 'Server1_Task' and 'Server2_Task'.

See Also:

ServersUsingTaskSequencesWithDataTables_FlowLine.spfx

ServersUsingTaskSequencesWithDataTables_JobShop.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ServerWithMaterialConsumption - SimBit

This SimBit Project includes two models, the first illustrates the basic concepts of using materials and BOM and the second model illustrates adding material replenishment to that basic model.

Model 1: ServerWithMaterialConsumption

Problem:

I have a machine that uses some material and produces some material and I'd like to model the usage and production of the material.

Categories:

Materials

Key Concepts:

Bill of Materials, Material Element, QuantityConsumed, QuantityProduced, Task Sequences

Assumptions:

The Source only produces 10 arrivals. One bookshelf is created for every entity that enters the Server and for each bookshelf that is created, 8 nails are consumed and 4 pieces of wood. There are initially 100 units of wood and 400 units of nails.

Technical Approach:

A Server object is used, and it consumes materials from a Bill of Materials. The Server object produces a material (bookshelf). The number of bookshelves produced, and the number of nails and pieces of wood consumed is recorded with an output statistic.

Details for Building the Model:

Creating the Materials and Bill of Materials

- Open the Definitions Window and select the Elements panel. Create three new Material elements by clicking on the Material Icon in the top ribbon menu.
- Change the *Name* of the first material element to be 'Wood'. The *Initial Quantity* should be set to '200'.
- Change the *Name* of the second to be 'Nails'. The *Initial Quantity* should be set to '400'.
- Change the *Name* of the third material element to be 'BookShelf'. The *Initial Quantity* is '0'.
- Create the Bill of Materials by clicking the Bill Of Materials button in the ribbon menu.
- Change the *Name* of the Bill Of Material Element to 'BookshelfBOM'. Add two rows to the *Components* property – one for the *Material Name* 'Nails', which should have a *Quantity* of '8' and the other row for the *Material Name* 'Wood', which should have *Quantity* of '4'.

Simple System Setup

- Place a Source, Server, and Sink from the Standard Library into the Facility window and connect with Connectors.
- Update the Source to set the *Interarrival Time* to 'Random.Exponential(.9)' and the *Maximum Arrivals* property to '50'

Defining the Server

- Within the Facility window, edit the Server object and change the *Process Type* to 'Task Sequence'.
- Open the Processes Tasks repeat group and add two rows.
- For the first row, leave the *Sequence Number* at '10', change the *Name* to 'Setup', and set the *Processing Time* to '0.1' minute. Within the Materials Requirements section set the *Consumption Type* as 'Bill of Materials' and *BOM Name* as 'BookShelfBOM'. The *Quantity* is '1'.
- For the second row set the *Sequence Number* to '20', change the *Name* to 'Process', and set the *Processing Time* to 'Random.Triangular(0.5,0.8,1.2)' minutes. Within the Materials Requirements section set the *Action Type* to 'Produce', leave the *Production Type* as 'Material', and set the *Material Name* as 'BookShelf'.

Model 2: ServerWithMaterialConsumptionAndReplenishment

Problem:

I have a machine that uses some material and produces some material, and I'd like to provide replenishment of the material once depleted. (Enhancement of ServerWithMaterialConsumption).

Categories:

Materials

Key Concepts:

Bill of Materials, Event, Material Element, Monitor, OutputStatistic, Produce Step, QuantityAvailable, QuantityConsumed, QuantityProduced, Task Sequences

Assumptions:

The Source has infinite arrivals. One bookshelf is created for every entity that enters the Server and for each bookshelf that is created, 8 nails and 4 pieces of wood are consumed. There are initially 50 units of wood and 100 units of nails. Each time wood is depleted, we replenish 40 of it; each time nails are depleted, we replenish 80 of them. Due to different replenishment policies, the lead time of replenishment for wood and nails are different.

Technical Approach:

A Server object is used, and it consumes materials from a Bill of Materials and produces a material (bookshelf). We use two Monitor elements to watch Model States that keep track of the numbers of wood and nails remaining in the system. Each monitor fires an event when the number of the material it monitors in the system goes below a certain threshold value, and then this event triggers the replenishment process for this material.

Details for Building the Model:

Updating the Model Described Above

- On Source, reset the *Maximum Arrivals* back to 'Infinity'.
- Open the Definitions Window and select the Elements panel. Change the *Initial Quantity* of Nails to be '50'.
- Still in the Elements panel, click on the Monitor icon in the General ribbon group and name that monitor 'WoodMonitor'. Set the *State Variable Name* to 'Wood.QuantityInStock', the *Monitor Type* to 'CrossingStateChange', the *Crossing Direction* to 'Negative', and the *Initial Threshold Value* to '10'. This monitor will fire an event named 'WoodMonitor.Event' whenever the state variable crosses from 10 to 9.
- Add a second Monitor element which is very similar to the first, name it 'NailsMonitor'. The differences with this monitor are that the *State Variable Name* is 'Nails.QuantityInStock', the *Initial Threshold Value* is set to '20' and the automatically fired event will be 'NailsMonitor.Event'.
- Go to the Processes window. Create two processes named ReorderWood and ReorderNails.
- Under the General category of the first process set the *Name* to 'ReorderNails' and set the *Triggering Event Name* to 'NailsMonitor.Event' (select from the list). Place a Delay step in the process. The *Delay Time* is '5', and the *Units* is set to 'Minutes'. Then place a Produce step in the process. The *Production Type* is 'Material', *Material Name* is 'Nails', and the *Quantity* is '80'. This process will be executed when the remaining quantity of Nails is below 20, and then the process will delay for 5 minutes as the delivery lead time and replenish 80 Nails.
- Name the second process Name to 'ReorderWood' with a *Triggering Event Name* of 'WoodMonitor.Event'. The process itself is very similar to the first process. The differences are the *Delay Time* for Delay step is set to '2' minutes and within the Produce step, the *Material Name* is 'Wood', and the *Quantity* is '40'.

Embellishments:

We can add some animation, to show the current remaining number of materials while we're running this model.

Choose Status Plot from Animation under Facility Tools, create a Status Plot, with the *Title* 'Material Available', then set the *X Axis* label to be 'Time' and the *Y Axis* label to be 'Pieces', and change the time range shown on the *X Axis* to be '30 minutes'.

Then click on the plot, go to Properties Window, add 2 rows in the Additional Expressions. One row has the *Expression* set to 'Nails.QuantityInStock', and the *Label* set to 'Nails'. The other row has the *Expression* set to 'Wood.QuantityInStock', and the *Label* set to 'Wood'. Then we'll have two animation lines within the plot, one represents the number of remaining nails and the other one represents the number of remaining wood.

You may note that this system does briefly run out of Nails on each cycle and that this causes the system to be unbalanced and start building inventory. You might try different settings for the reorder threshold to minimize any process impact.

Send comments on this topic to [Support](#)

ServerWithSequenceDependentSetup - SimBit

Model 1: Represent Setup Time

Problem:

I have a Server that requires a certain setup time based on which Entity type needs to be processed.

Categories:

Changeover Logic

Key Concepts:

Changeovers, List Property, Sequence Dependent, Server, Status Pie, StringList

Assumptions:

Each Source only produces one type of Entity. The model will complete its run faster if the Server processes all Entities of one type before changing over to process Entities of another type.

Technical Approach:

A String List is created in the List panel of the Definitions window for the ModelEntity. A List property is defined in the ModelEntity's Definitions window, Properties panel. A Server with a Sequence Dependent Setup Time is used, and a Changeover Matrix defines the various setup times.

Details for Building the Model:

Simple System Setup:

- Add two Sources, a Server, and three Sinks to the Facility window. Change the *Interarrival Time* of the Sources to '0.1' minutes and the *Maximum Arrivals* of each Source to '5'. Change the *Entities Per Arrival* for both Sources to '2'.

Adding a String List:

- Click on the Definitions tab and select the Lists panel to add a String List named 'Colors'. Add as many colors as desired, in this case 5: White, Yellow, Red, Blue, and Black. Be sure to add this same String List to the Lists within the Definitions tab of the ModelEntity.

Adding a List Property:

- Also, within the Definitions window, select the Properties panel and add a List Property with the *Name* 'Color'.
- Set the *List Name* property under Logic to 'Colors' and the *Category Name* property under Appearance to 'Properties'. This will give the user the option to specify *Color* as a property of an Entity and the List of Colors will be shown as options.

Creating a Changeover Matrix:

- In the Data window, select the Changeovers panel. Create a Changeover Matrix and change the *Name* to 'Cleaning'.
- Change the *List Name* property to 'Colors'.
- Fill in the Changeover Matrix with desired values or follow the example below:

	White	Yellow	Red	Blue	Black
White	0	0.1	0.1	0.1	0.1
Yellow	1	0	0.1	0.1	0.1
Red	1	0.5	0	0.3	0.4
Blue	1	0.5	0.6	0	0.3
Black	1	0.8	0.8	0.5	0

*Note all values are in hours

Adding Changeover Logic Element:

- Click on the Elements view of the Definitions tab and select Changeover Logic. Select the Repeating Property Editor under *Setup Transitions*. Set the *Operation Attribute* to 'ModelEntity.Color' and *Changeover Matrix Name* to 'Cleaning'.

This will have the Changeover Logic element reference the Changeover Matrix created earlier.

Using the Server:

- The Server will need to have different Setup Times depending on the sequence of Entities in its Queue. This is done by changing the *Process Type* to 'Task Sequence'. Open the Repeating Property Editor and add two tasks. The first task, titled 'ChangeOver' has a 'Sequence Dependent Setup' *Process Type* and 'CleaningChangeOverLogic' as the *Changeover Logic Name*. This task references the Changeover Matrix and implements the change times for switching between Entity Color. The second task has a *Processing Time* of '15' minutes.

Adding a String List:

- Place two ModelEntity objects from the Project Library into the Facility window. Name one 'RedEntity' and one 'BlueEntity'. Select the desired color from the *Color* property drop down list. Then change the color of each Entity to match that choice.
- Change the *Entity Type* Arrival Logic property of each Source to match the color of its associated Entity.
- Set the *Selection Weight* of the Path from the Server to the Sink to 'ModelEntity.Color==List.Colors.[Desired Color]'.
- Add as many Sources and Sinks as desired to the model.

Enhancements (done within this model):

- Create a Status Pie to show the Resource State of Server1.
- Create a Floor Label showing the number of Entities created by Color, the Total Setup Time, and the Total Setup Percentage.
- Change the Setup symbol of the Server to a distinct color to easily see when it is setting up.

Model 2: Minimize Setup Time

Problem:

I have a Server that requires a certain setup time based on which Entity type needs to be processed, and I would like to minimize this setup time.

Categories:

Changeover Logic, Servers

Key Concepts:

Changeovers, Dynamic Selection Rule, List Property, Server, Sequence Dependent, Status Pie, String List

Assumptions:

Each Source only produces one type of Entity. The model will complete its run faster since the Server processes all Entities of one type before changing over to process Entities of another type.

Technical Approach:

A String List is created in the List panel of the Definitions window for the ModelEntity. A List property is defined in the ModelEntity's Definitions window, Properties panel. A Server with a Sequence Dependent Setup Time is used, and a Changeover Matrix defines the various setup times.

Details for Building the Model:

Simple System Setup:

- Click on the Folder in the Navigation window, click on RepresentSetupTime and press 'Ctrl + C' to copy and 'Ctrl + V' to paste the model.
- Rename this second model 'MinimizeSetupTime'.

Adjusting the Server to Minimize Setup Time:

- Change the *Dynamic Selection Rule* to 'Standard Dispatching Rule' and the *Dispatching Rule* to 'LeastSetupTime'. Once capacity within the Server becomes available, this type of rule selects the next seize request from its allocation queue using the 'LeastSetupTime' rule which aims to minimize the amount of time the Server spends setting up for Entity processing.
- Under the Server's Advanced options, change the *Expected Setup Time Expression* to 'CleaningChangeOverLogic.ExpectedSetupTime(Server1 ,ModelEntity)' This expression allows the Server to make its selection for the next Entity based on the changeover times stored in the Changeover Matrix.

ServerWithTransferInConstraints - SimBit

Problem:

This small example illustrates usage of the 'Transfer-In Constraints' feature that is available in Advanced Options for the processor-oriented standard library objects.

In this problem, there are three servers arranged in a line. When an entity arrives to a server, if there are already three (3) entities waiting in the input buffer to be processed, then the entity continues to the next server. The exception is the last server in the line, which has no transfer-in constraints and thus accepts all overflow of entities that cannot be processed by the first two servers.

Categories:

Decision Logic -- Paths

Key Concepts:

Server, Queue, Path

Assumptions:

For either 'Server1' or 'Server2' in this model, a maximum of 3 entities may ever be waiting in the input buffer. The last server, 'Server3', has no such constraint and thus can accept all overflow of entities that cannot be processed by the first two servers.

Technical Approach:

Use the Advanced Options section, Transfer-In Constraints functionality available for the Server object.

Details for Building the Model:

Simple System Setup

- Place a Source, three Servers, and a Sink in the Facility window.
- Move the input nodes of the Servers to the top portion of the Servers and the output nodes to the bottom portion below the object name.
- Connect the Source to Server1's input node. Then connect Server1's input node to Server 2's and then to Server 3's node. Logically, the entities will attempt to travel from the Source to Server1. If the entity violates the server's transfer-in constraints, then the entity travels to the second server Server2. If transfer-in constraints for that server are also violated, then the entity continues on to Server3 which will always accept the entity.
- Connect all three Servers output nodes to the Sink.

Logic Details within the Objects

- Within the Source, change the *Interarrival Time* to 'Random.Exponential(.1)'. Within Server1, change the *Processing Time* to 'Random.Triangular(.3,.4,.6)'.
- Within both Server1 and Server2, open the Advanced Options properties and change the *Transfer-In Constraints* to 'Custom Condition'. Then, specify the *Transfer-In Condition* as 'Server.InputBuffer.Contents < 3'. This will cause entities to evaluate the contents of the input buffer for a given server before entering the node at that server. If the condition is true, the entity will enter the node, if not, it will continue.
- Because the arrival rate on the Source is quite high, and the Server1 processing time is longer than the other Servers, you'll see that after a short while, entities will bypass Server1 and continue to Server2. Then once Server2 buffer gets larger, incoming entities continue to Server3.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SetPropertyBasedOnValueOfListProperty - SimBit

Problem:

I have a String List property and the value of a numeric property within the model is set depending on the value of the String List property.

Categories:

Decision Logic -- Paths

Key concepts:

List Property, Math.If, Maximum Arrivals, Path, StringList, TimeOnLink

Technical Approach:

A TimePath connects the Source to the Sink. The Travel Time property on this TimePath is set based on the current value of the List Property named ExitLocation.

If the value of this property is 'Far', the Travel Time is 5 minutes. If the value of this property is 'Near', the Travel Time is 1 minutes.

Details for Building the Model:

Defining the Lists and Properties

- Go to the Lists window by selecting Lists along the left panel from within the Definitions window. Click on String in the ListData ribbon to create a new String List. Name this new list 'ExitLocations'. Add two entries into the list - 'Near' and 'Far'.
- Go to the Properties window by selecting Properties along the left panel from within the Definitions window. Select List from the Standard Property drop down in the Properties ribbon. Change the *Name* of this new property to 'ExitLocation'. Set the *ListName* property of this property to 'ExitLocations'.

Simple System Setup

- Within the Facility window, place a Source object and a Sink object and connect them with a TimePath.
- In the Source object, set the *Interarrival Time* property to '1' minute. Set the *Maximum Arrivals*, under the Stopping Conditions category, to '10'.
- Click onto the TimePath and set the *Travel Time* property to 'Math.If(ExitLocation == List.ExitLocations.Far,5, 1)' This expression checks the value of the *ExitLocation* string list property and if it is set to 'Near', the value of the *Travel Time* property is set to '1'. If the value of the *ExitLocation* string list property is set to 'Far', the value of the *Travel Time* property is set to '5'.

Testing the Model

- Because the Source object is set to only create 10 entities, the model can be tested by seeing how long the model takes to run through 10 entities. The model can also be tested by displaying a status label that returns the average time that an entity spends on the link (i.e. TimePath1.TimeOnLink.Average).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SimpleElevatorObject - SimBit

Problem:

I want to know how to model a simple elevator.

Categories:

Animation, Building New Objects/Hierarchy, Decision Logic – Paths, Vehicles

Key Concepts:

Bind To External Input Node, Bind To External Output Node, Connector, Custom Object, External View, Externally Visible, Idle Action, Initial Node (Home), Location.X, Location.Y, Location.Z, Minimum Dwell Time, ModelEntity, Network Turnaround Method, Object Reference Property, Parking Station Queue, Ride On Transporter, RideStation Queue, Vehicle

Technical Approach:

We build a custom elevator object that consists of 2 transfer nodes and a bi-directional path connecting the two nodes. A standard vehicle object is used as the elevator platform that travels between the two nodes. There are also two external nodes where entities transfer out of the object. The object has two properties for the user to provide inputs; the Minimum Dwell Time of the Elevator and the Capacity of the Elevator.

There are two models that include the Elevator object to demonstrate how it can be used in model logic. The first model uses two elevator objects. The entity must go up to the second floor to travel between the two elevators, and then back down to the first floor to exit the system. The second model uses two elevator objects on top of one another, to model a three story facility. The first elevator carries entities up to the second floor and back to the first floor and the second elevator carries entities to and from the second and third floor.

Enhanced Technical Approach:

To build the Elevator Object:

- Click New Model from the Project Home ribbon to create a new fixed object in this project
- Click onto the new object in the Navigation window and change the name to ElevatorObject. Ensure the ElevatorObject object is selected in the Navigation window and proceed to the next step.

Facility Window:

- Place down two Transfer Nodes. Name them: TransferNode1, TransferNode2
- Drag a standard Vehicle object into the Facility window.
 - Set the *Network Turnaround Method* property to 'Reverse'
 - Set the *Initial Node (Home)* property to 'TransferNode1'
 - Set the *Idle Action* and *OffShiftAction* properties to 'RemainInPlace'
 - Flatten the Vehicle so it looks like a flat elevator platform
- On TransferNode1, set the Entity Destination Type property to 'Specific' and set the Node Name property to 'TransferNode2'.
- Create a new Add On Process in the Entered Add On Process trigger property of TransferNode1 by selecting Create New in the drop down of this property. Do the same for TransferNode2.
- On TransferNode2, set the Entity Destination Type property to 'Specific' and set the Node Name property to 'TransferNode1'.
- On each node, expand the General Properties, and then the Physical Characteristics and Location to find the X,Y,Z Location properties. Set these properties to:
 - TransferNode1 – X: 0, Y: 0, Z: 0
 - TransferNode2 – X: 0, Y: 6, Z: 0
- Draw a Path between the two nodes and set the *Type* property to 'BiDirectional'

Creating the External View of the Elevator Object

- Click onto the TransferNode1 node. Right click and select Bind to New External Input Node. Name this FirstFloorIn1. This creates an External Input Node that sends entities directly to the FirstFloorIn facility node.
- Click onto the TransferNode2 node. Right click and select Bind to New External Input Node. Name this

- SecondFloorIn1. This creates an External Input Node that sends entities directly to the SecondFloorIn facility node.
- Right click onto Vehicle1 and select Externally Visible. This will make this object NOT externally visible in the External View.
- Go to the Definitions Window and into the External View panel. Click onto the External Node in the Ribbon and place an External Node near the TransferNode1 and name it FirstFloorOut. Set the *Node Class Name* property to 'TransferNode'. Place another External Node near the TransferNode2 and name it SecondFloorOut. Set the *Node Class Name* property to 'TransferNode'.

Processes Window:

- In the Add On Process for TransferNode1 Entered process trigger: Place a Decide Step with the Expression property set to 'Is.Modelentity' to check if this is the entity arriving to the node.
 - In the True segment leaving the Decide step, place another Decide step with an *Expression* set to 'Modelentity.PreviousNode != FirstFloorIn' This checks to see if the entity arrived from outside of the elevator or if it came through the elevator from the other node.
 - In the True segment leaving this Decide Step, place a Transfer Step. Set the *From* property to 'CurrentNode', the *To* property to 'ParentExternalNode' and the *ExternalNodeName* to 'FirstFloorOut'. This transfers an entity out of the object if the entity did not just enter into the object.
 - In the False segment leaving this Decide Step, place a SetNode step. Set the *Destination Type* property to 'Specific' and the *Node Name* property set to 'TransferNode2'.
 - After the SetNode step, place a Ride Step. Set the *Transporter Type* property to 'Specific' and the *Transporter Name* to 'Elevator'

Adding Object Reference Properties to the Elevator Object:

- In order to give the user some control over the behavior of the ElevatorObject object, you can create some properties on this object.
 - Click on Vehicle1 and find the *Initial Ride Capacity* property. Right click onto the name of this property and select *Create New Referenced property*. Name this new property 'ElevatorCapacity'.
 - Set the *Minimum Dwell Time Type* of the Vehicle to 'Specific Time'. Right click onto the *Minimum Dwell Time* property and select *Create New Referenced property*. Name this new property 'Minimum Dwell Time'.
 - To see these new properties and change the default values, go to the Definitions window of the ElevatorObject object and go to the Properties Panel. Change the default of the Elevator Capacity to '1' and the default of the Minimum Dwell Time property to 5 seconds.

Create "SingleStory" Model:

- Click New Model to create a new Fixed Object in this project. Name it UsingElevator.
- Place down two Source objects, two Sink objects, and two ElevatorObject objects (from the Project Library).
- Draw paths from one Source to FirstFloorIn on ElevatorObject1, from SecondFloorOut to SecondFloorIn of ElevatorObject2 and then from FirstFloorOut to one of the Sink objects.
- Draw paths from the other Source to FirstFloorIn on ElevatorObject2, from SecondFloorOut to SecondFloorIn of the ElevatorObject1 and then from FirstFloorOut to the other Sink object.

Create "ThreeStories" Model:

- Place an instance of the ElevatorObject object and another directly on top of it. (To move an object up and down in the Facility Window, select the object, hold down the Shift key and drag it up or down)
- Connect the SecondFloorOut node of the first elevator to the FirstFloorIn node of the second elevator with a Connector. Connect the FirstFloorOut node of the second elevator to the SecondFloorIn node of the first elevator with a Connector.
- Place a Source object at the level of the first elevator and connect it to FirstFloorIn with a Path. Add another Source object at the level of the second elevator. Connect this Source to FirstFloorIn of the second elevator and SecondFloorIn of the first elevator. The Source will send half of its entities down to the first floor and half up to the third floor.
- Place three Sinks into the Facility window; one at the first floor level, another at the second floor level and a third at the third floor level. Connect the FirstFloorOut node to the first floor Sink with a Path. Connect the SecondFloorOut node of the first elevator and the FirstFloorOut node of the second elevator to the second floor Sink with a Path. And connect the SecondFloorOut node of the second elevator to the third floor Sink.

SimpleLeastSlackSelectionRule - SimBit

Problem:

I have multiple entities with different due dates and I want to them to be processed with a Least Slack scheduling methodology.

Categories:

Buffering, Lookup and Rate Tables

Key Concepts:

Arrival Mode, Numeric Property, Ranking Expression, Ranking Rule, Rate Tables, Server, Smallest Value First, Time Varying Arrival Rate

Assumptions:

Assume that Processing Time is the same and constant for each type of entity and also assume that entities are allowed to be processed even after their deadline.

Technical Approach:

Entities are created in separate Sources according to an appropriate Time Varying Arrival Rate. The entities then travel to a Server that selects the next job based on a Selection Rule to accept the entity type with the least slack waiting in the queue.

Details for Building the Model:

Simple System Setup

- In the Facility window, place four Sources and four ModelEntity objects to go along with each Source. Also place a Server and a Sink. Connect all the Sources to the Server, as well as the Server to the Sink, with Paths.
- Rename the ModelEntities 'EarlyIn_LateOut', 'EarlyIn_EarlyOut', 'LateIn_LateOut', and 'LateIn_EarlyOut' and rename the Sources that each is generated from in a similar fashion (we used EL, EE, LL, LE). Also, change the color of the entity symbols so that each entity type can be differentiated.
- Extend the Input Buffer for the Server so that all entities in the queue will be visible.

Create Rate Tables

- In each Source set the *Entity Type* to the appropriate Entity. Change *Arrival Mode* to 'Time Varying Arrival Rate' and notice that the *Rate Table* box is now highlighted. This means that a Rate Table has to be created for each type of Entity.
- Go to the Data Window and then to the Rate Tables Panel to create a Rate Table. Create four Rate Tables with names associated with the entity types (i.e., EarlyIn_LateOut_RT, EarlyIn_EarlyOut_RT, etc.).
- The *Interval Size* for each table should be '1' hour, while the *Number of Intervals* for each should be '6'. Note that the last 2 hours in all the tables are 0, indicating no entity arrivals, which will allow the system to empty. The specific rate tables for this example are as follows:

EarlyIn_LateOut

Starting Offset	Ending Offset	Rate (events per hour)
Day 1, 00:00:00	Day 1, 01:00:00	15
Day 1, 01:00:00	Day 1, 02:00:00	7
Day 1, 02:00:00	Day 1, 03:00:00	0
Day 1, 03:00:00	Day 1, 04:00:00	0
Day 1, 04:00:00	Day 1, 05:00:00	0
Day 1, 05:00:00	Day 1, 06:00:00	0

LateIn_LateOut

Starting Offset	Ending Offset	Rate (events per hour)
Day 1, 00:00:00	Day 1, 01:00:00	0
Day 1, 01:00:00	Day 1, 02:00:00	0
Day 1, 02:00:00	Day 1, 03:00:00	6
Day 1, 03:00:00	Day 1, 04:00:00	15
Day 1, 04:00:00	Day 1, 05:00:00	0
Day 1, 05:00:00	Day 1, 06:00:00	0

EarlyIn_EarlyOut

Starting Offset	Ending Offset	Rate (events per hour)
Day 1, 00:00:00	Day 1, 01:00:00	10
Day 1, 01:00:00	Day 1, 02:00:00	10
Day 1, 02:00:00	Day 1, 03:00:00	4
Day 1, 03:00:00	Day 1, 04:00:00	0
Day 1, 04:00:00	Day 1, 05:00:00	0
Day 1, 05:00:00	Day 1, 06:00:00	0

LateIn_EarlyOut

Starting Offset	Ending Offset	Rate (events per hour)
Day 1, 00:00:00	Day 1, 01:00:00	0
Day 1, 01:00:00	Day 1, 02:00:00	0
Day 1, 02:00:00	Day 1, 03:00:00	10
Day 1, 03:00:00	Day 1, 04:00:00	10
Day 1, 04:00:00	Day 1, 05:00:00	0
Day 1, 05:00:00	Day 1, 06:00:00	0

- Once these tables are created, go back to the appropriate Source and select the Rate Table that goes with that Source.

Assigning the Due Date

- The Due Date will be represented as a Property of the ModelEntity, since it will not change values during the simulation run.
- Select ModelEntity in the Navigation Window in the upper right-hand portion of the screen. Go to the Definitions Window and then Properties Panel. Add a Standard Property (type Real) and rename it 'DueDate'.
- To set the DueDate for each entity, go back to the Model and notice in each entity instance, there is a new property called *DueDate*.
- In this example, the deadlines are set as follows: EarlyIn_EarlyOut to '2.5', EarlyIn_LateOut to '7', LateIn_EarlyOut to '6', and LateIn_LateOut to '8'.
 - With the default Run Setup settings, the model starts at midnight (represented numerically as 0), so if the entity is to be "shipped" at 2:30 AM its DueDate is to be set at '2.5'.

Configuring the Server to Select Entities with Least Slack

- The Server in this model is to process entities based off of a Least Slack scheduling methodology. Slack is defined as the Deadline minus the Remaining Process Time. Because this model only has one server, processing time can be ignored.
- Within the Server, change the *Processing Time* to '4' minutes. Then, change the *Ranking Rule* to 'Smallest Value First'. Set the *Ranking Expression* to 'ModelEntity.DueDate'. This means that the Server is going to prioritize the queue according to DueDate, and select the entity with the smallest value, or essentially, the one with the earliest value.

Discussion:

Slack is the "cushion" that an object has between it is potentially done processing and its deadline. In this model, this cushion should have been mathematically described in the *Ranking Expression* as 'ModelEntity.DueDate - (Run.TimeNow + ModelEntity.ProcessingTime)'. The Remaining Processing Time would become a factor if entities had different processing times. This could happen if there were multiple Servers and/or entities followed a different sequence of steps, or if the entities just simply have different processing times.

If the Due Dates were exactly the same for all entities, the Ranking Rule would be the opposite of the one used in this example. It would be set to 'Largest Value First' with a Ranking Expression of 'ModelEntity.ProcessingTime'.

Of course, if both Due Date and Processing Time are not consistent between the entities, the entire expression is necessary.

SimpleTank - SimBit

Problem:

I want to model the level of a fluid tank complete with animation.

Categories:

Add-On Process Logic, Level States

Key Concepts:

Circular Gauge, CrossingStateChange, Floor Label, Level State Variable, Monitor, Rate, Resource, Size.Height, StateStatistic, Status Label

Assumptions:

This tank has a Capacity of 1000 units and fills at a rate of +1000/hr and empties at a rate of -5000/hr with a pause of .1 hours between changes. The height of the tank in the Facility Window has been set to 5 meters.

Technical Approach:

A resource is created along with other drawing elements to represent a tank. Add-On processes are created to mimic the tank filling/emptying logic as well as to animate the level of fluid in the tank.

Details for Building the Model:

Creating the Tank

- Place a resource into the Facility Window.
- Change the symbol to represent the fluid level in the tank – a cylinder was used in this example – and resize it to an appropriate size (the Height was set to 5 in this example).
- Options to improve animation:
 - To represent the shell of the tank, draw a polyline around the outer-edge of the cylinder, leaving an opening so that the fluid level is visible. Change the Object Height to equal the height of the fluid level.
 - To represent the top of the tank, draw a circle and change its Object Height. Place it on top of the other two parts.
 - To show tank action in 2D view, add a dial gauge using the expression TankVolume and range 0 to 1000. In 3D view use "Shift-Drag" to raise it and place it on top of the tank.

Creating the Add-On Process Logic

- Add a Level State Variable called 'TankVolume' in the States panel of the Definitions Window.
- In the Processes Window, create 2 processes called TankFull and TankEmpty to dictate the tank's actions at those points in time.
- For TankEmpty:
 - Add an Assign step and call it StopEmptying. Set the *State Variable Name* to 'TankVolume.Rate' and the *New Value* to '0'.
 - Add a Delay step and call it WaitForSupply. Set the *Delay Time* to '0.1' Hours.
 - Add an Assign step and call it StartFilling. Set the *State Variable Name* to 'TankVolume.Rate' and the *New Value* to '1000'.
- For TankFull:
 - Add an Assign step and call it StopFilling. Set the *State Variable Name* to 'TankVolume.Rate' and the *New Value* to '0'.
 - Add a Delay step and call it Pause. Set the *Delay Time* to '0.1' Hours.
 - Add an Assign step and call it StartEmptying. Set the *State Variable Name* to 'TankVolume.Rate' and the *New Value* to '-5000'.
- In the Definitions Window, Elements panel, add 2 Monitor Elements – MonitorTankFull and MonitorTankEmpty. These monitors trigger an event when its condition is reached.
- For MonitorTankFull:

- Set *Monitor Type* to 'CrossingStateChange'
 - Set *State Variable Name* to 'TankVolume'
 - Set *Threshold Value* to '1000'
 - *Crossing Direction* to 'Positive'
 - *On Change Detected Process* will be 'TankFull'
- For MonitorTankEmpty:
 - Set *Monitor Type* to 'CrossingStateChange'
 - Set *State Variable Name* to 'TankVolume'
 - Set *Threshold Value* to '0'
 - *Crossing Direction* to 'Negative'
 - *On Change Detected Process* will be 'TankEmpty'
- To represent the animation of the tank we need to make a new process.
 - In the Processes Window, click the Select Process button in the ribbon and choose 'OnInitialized' from the drop down list. Drag an Execute step and set Process Name in the Logic to TankEmpty.
 - Add a Delay step called DelayBetweenUpdates and set the *Delay Time* to an appropriate amount. 1 min was used in this example.
 - Add an Assign step. Set the *State Variable Name* to 'Tank.Size.Height'. The *New Value* is going to be a representation of the tank's fluid level. So, when it is full the value should be equal to the height set to the resource (5). In this case it was set to TankVolume/200 because the maximum volume is 1000 and 1000/200 equals the height of the resource – 5.
 - Drag the process Endpoint to the input of the Delay to create a loop. This allows the animation to update every time the Delay is executed.

Embellishments:

We've added a Status Plot to illustrate the TankVolume over time, as well as Status Labels to represent the Tank Volume and Fill Rate. It can be seen that the animated fluid level mirrors the change in TankVolume.

Discussion:

The delay time used in the DelayBetweenUpdates process step may affect run time. A large delay time between animation updates results in a faster run time because the model does not have to update the resource height as frequently. On the other hand, a smaller delay time results in a smoother, more accurate animation at the potential expense of a longer run time.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SingleVehicleUsage - SimBit

Problem:

I have entities that require movement on a vehicle between operations.

Categories:

Vehicles

Key Concepts:

Ride on Transporter, TransferNode, Vehicle

Assumptions:

There is a single vehicle that moves all entities. The vehicle can move up to 2 entities at a time between the objects in the system. All paths are unidirectional.

Technical Approach:

Set the Transfer Nodes on the Source and Server objects to specify that a vehicle is required for movement. With unidirectional paths, the vehicle must have a path to return from the Sink to the Source, even though the entity doesn't travel between those objects.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink in the Facility Window.

Defining the Vehicle and its Characteristics

- Place a Vehicle in the Facility Window.
- Within the Transport Logic section, change the *Initial Ride Capacity* to '2'. This will allow up to 2 entities to be moved on the vehicle at one time. Change the *Load Time* to '1' and the *Unload Time* to 'Random.Uniform(1,3)'. Each entity incurs the load delay when getting on the vehicle and the unload delay when getting off the vehicle.

Modifying the Transfer Nodes

- Click on the Transfer Node (exit) of the Source and change the *Ride On Transporter* property to 'True'. Specify the *Transporter Type* as 'Specific' and *Transporter Name* as 'Vehicle1'.
- Do the same thing as above for Server object's Transfer Node.

Placing the Paths

- Double click on the Path to define the multiple paths between the Source, Server and Sink objects.
- When using vehicles, the Basic Node (entry) on the Server must be connected to the Transfer node (exit) on the Server so the Vehicle can move from one node to the next.
- Click on one of the paths. Note that the *Type* is set to 'Unidirectional'. Since the paths are unidirectional, place a path also from the Sink object back to the Source object. While the entities will exit the system at the Sink, the Vehicle must have a path to return to the Source to pick up additional entities for transfer.

Embellishments:

Try changing the paths to be bidirectional or changing the initial ride capacity of the vehicle.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SingleWorkerCompletesProcessAndMovesToNode - SimBit

Problem:

I want to model a system that has multiple Servers in series and requires a Worker to be present at each Server before processing can occur. The worker completes work across all servers for one entity before another entity can begin processing. Animation should show the Worker moving between the two servers and parking at the appropriate Server for processing. The Server and Worker should change color when working. There is only one Worker in the system.

Categories:

Add-On Process Logic, Worker, Resources

Key Concepts:

Active Symbol, Add-On Process, Before Processing, Initial Node (Home), Move Step, Secondary Resources, Server, Worker

Assumptions:

There is only one Worker in the system. It will work at the first Server, and then proceed to work at the second Server with the same entity that finished at Server1. The Worker will then return to working on the first Server. * NOTE: In this example, the worker moves to the second server at approximately the same speed as the entity – should this be different, additional logic may be required.

Technical Approach:

The Worker is modeled with a standard Worker object. The Worker object travels between two nodes that are located near each Server, which animates the moveable resource. The Worker object is seized by the first Server before processing and the model ensures that the object is located at the appropriate node before it begins Processing. Add On Processes on the first server onExited will request the worker to move to the second server before processing. The second server will release the worker after processing.

Details for Building the Model:

Simple System Setup

- Add a Source, two Servers and a Sink to the Facility window. Change the *Interarrival Time* of the Source to 'Random.Exponential (.53)' minutes.
- Connect the Source to the first Server, connect the first and second Server together and connect the second Server to the Sink with Paths. Add two Basic Nodes that are connected to each other with a bi-directional path. Place the nodes near each of the Servers. This is the path that the movable resource will take between the Servers.
- Place a Worker object in the Facility window. Add a symbol to this worker and change it to green by clicking on the Worker object, adding a new symbol by selecting add additional symbol from the Additional Symbols category in the ribbon. Color the worker green by selecting Color from the decoration ribbon. The worker will already turn green when working as its animation is by default set to 'Worker.Capacity.Allocated > 0'.
 - Set the *Park While Busy* property of the Worker to 'True', which will tell the Worker to park at the node while it's processing instead of stay on the link.
 - Set the *Initial Node(Home)* property to 'BasicNode1'.

Adding Logic to the Servers

- Click on Server1 and expand the *Secondary Resources* property category. Under the *Other Resource Seizes* category, open the repeat group for Before Processing. Add a new repeat group property by selecting *Add*. Select the repeat group and set the *Object Type* property to 'Specific', set the *Object Name* property to 'Worker1'. Set the *Request Move* property to 'To Node' and the *Destination Node* property to 'BasicNode1'. This tells the Server that it must seize Worker1 and it must arrive at BasicNode1 before processing can begin at this Server.
- Repeat the above step for Server2, except instead under Other Resource Releases and After Processing. This will tell Server 2 to release the worker on completion of processing.

Add On Process Logic for Server1

- Click on the Server1 object and expand the *Add On Process Triggers* property category.
 - Create a new process that is called from the *Exited* property by selecting 'Create New' from the drop down of

this input box. This new process will tell the Worker to move to Server 2 once the entity has left Server 1.

- Go to the processes window and in the process called Server1_Exited, place a Move Step. Open the *Resource Move Request* repeat group and add a new property. Set the *Object Type* to specific, the *Object Name* to 'Worker1', and the *Destination Node* to 'BasicNode2'. Close the repeat group.

Adding Animation

- Add a symbol to this worker and change it to green by clicking on the Worker object, adding a new symbol by selecting add additional symbol from the Additional Symbols category in the ribbon. Color the worker green by selecting Color from the decoration ribbon. The worker will already turn green when working as its animation is by default set to 'Worker.Capacity.Allocated > 0'.
- Repeat the same step for Server1 and Server2.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SortingConveyorSystem - SimBit

Problem:

I have a conveyor system that sorts boxes and sends them to the appropriate shipping dock.

Categories:

Conveyor Systems, Sequence Tables

Key Concepts:

By Sequence, Conveyor, Data Table, Dynamic Object Property, Entity Destination Type, Numeric Property, RandomRow, Sequence Table, TransferNode, Table Row Referencing

Assumptions:

There are three different types of boxes and each type of box has its own shipping dock.

Technical Approach:

There is only one Source. It reads a Data Table to determine how many of each type of box to produce. The boxes travel on a conveyor to the Labeling machine (a Server). They leave the Labeling machine and travel on the conveyor system. The system uses Sequence Tables to determine the route of each type of box.

Details for Building the Model:

Simple System Setup

- Place a Source object and a Server object into the Facility window. The Server can be renamed "Labeling Machine". Set its Input Buffer capacity to 0 to force any waiting entities to stay on the conveyor. Place three Transfer Nodes in a line after the Server. And finally, place three Sink objects, one below each of the Transfer Nodes. These Sink objects represent the shipping docks and they can be renamed Box1_Dock, Box2_Dock, Box3_Dock.
- Connect the Source to the Server with a Conveyor. Connect the Server to the first Transfer Node with a Conveyor and connect the other Transfer Nodes together with Conveyors. And finally, connect each Transfer Node to the Sink that was placed below it.
- Drag three instances of ModelEntity into the Facility window. Rename them to Box1, Box2 and Box3. You can also change the symbol and the colors, if desired. This is accomplished by selecting the entity and then choosing a symbol from the Symbol Library found in the Ribbon. The color of the symbol can be changed by selecting a new color from the Color menu in the Ribbon (when the entity is selected) and then clicking on the part of the entity that you'd like changed.

Creating Tables

- Go to the Tables panel within the Data window. Create a new Data Table by clicking on the Add Data Table icon in the Ribbon.
 - Add a new column to this table of type, "Entity". This is done by selecting "Entity" from the Object Reference drop down in the Ribbon. You can rename this property to be "BoxType" from within the Properties window on the right.
 - Add another column to this table of type, "Real". This is done by select "Real" from the Standard Property drop down in the Ribbon. You can rename this property to be "ProductMix".
 - Enter data into this table, such as: Box1 .4 Box2 .3 Box3 .3
- Add Three Sequence Tables by clicking on the Add Sequence Table icon in the Ribbon (3 times). There is one Sequence table for each type of Box. The Sequence Table consists of rows of Nodes and the Entity (Box) will follow this route, going from the node listed first and then proceed row by row.
 - In SequenceTable1, enter the following rows of data: Input@Labeling, TransferNode1, Input@Box1_Dock
 - In SequenceTable2, enter the following rows of data: Input@Labeling, TransferNode1, TransferNode2, Input@Box2_Dock
 - In SequenceTable3, enter the following rows of data: Input@Labeling, TransferNode1, TransferNode2, TransferNode3, Input@Box3_Dock

Using the Data Table

- From within the Facility window, click on the Source and expand the *Table Row Referencing* -> *Before Creating Entities* section in the Properties Window.
- Set the *Table Name* to the name of your Data Table (i.e. 'Table1')
- The *Row Number* is set to 'Table1.ProductMix.RandomRow'. This function tells Simio to use the values in the Product Mix column as a weighted average to determine which row to set at any given time. So in our example, 40% of the time row 1 will be read. 30% of the time, row 2 will be read and 30% of the time, row 3 will be read.
- Set the *Entity Type* property to 'Table1.BoxType'. The Source will create the entity listed in the BoxType column of Table1. Therefore, 40% of the time Box1 types will be created, 30% of the time, Box2 types will be created and 30% of the time, Box3 types will be created.

Using the Sequence Tables

- In order to tell the Entity which Sequence Table to use, click on each Entity instance (Box) and set the InitialSequence property. Box1's property should be set to 'SequenceTable1', Box2 to 'SequenceTable2', and Box3 to 'SequenceTable3'.
- You also need to configure each of the Transfer Nodes in the model to use Sequence Tables to determine the next location for the entity. Therefore, click on each Transfer Node (including the output nodes of the Source and the Server) and set the EntityDestination property to 'BySequence'.

Embellishments:

Experiment by changing the sequences in the Sequence Tables. See how the model behaves when the TransferNodes do not use "BySequence" for their EntityDestination property. Vary the values of the ProductMix column in the Data Table to see different amounts of each box type created.

Instead of using predetermined sequences, other approaches to sort the packages could include:

- Use an add-on process when leaving the Server to set the entity destination to the proper loading dock.
- Add decision logic in add-on processes at Transfer Nodes 1 and 2 to represent a scanner that would "read" the entity and direct it down the appropriate path.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SourceServerSinkApproaches - SimBit

Problem:

Source – Server – Sink (SSS) is a frequently used problem for beginners in simulation because it can be easily compared to queueing theory problems such as M/M/1 and M/M/2. But even that simple model has several possible approaches. We provide this project both to illustrate three approaches and to provide a tool for fast experimentation by beginners.

Categories:

Decision Logic -- Processing

Key Concepts:

Analysis, Capacity.Allocated, Contents, Create Step, Delay Step, Delay Step, Destroy Step, Experiment, Expression Property, Histogram, InputBuffer, OutputStatistic, Pivot Grid, Release Step, Replication, Response, Server, Sink, Status Label, Tally Step, TallyStatistic, Timer Element, Transfer Step, WaitUntilTransferring

Assumptions:

We are assuming a very simple system consisting of a single arrival stream, a single (perhaps multiple capacity) server, and then the entity is disposed. The primary statistics of interest are the time in queue and number in queue.

Unlike most SimBits, our goal here is not to teach you the details of how to build these, but rather to provide the modeling concepts and leave you with a useful tool for beginning analysis.

Technical Approach:

This project file contains three models. We have one model (SSS_Library) built using only the Standard Library. This is the easiest to build, but also runs significantly slower because it contains some overhead of objects and statistics. The other two models are built entirely from processes – one with just tokens (SSS_Process) and one with entities (SSS_Process_with_Entity). While not complex, these are a little more difficult to build and understand because you have to do some of the “plumbing” yourself, but the reward is significantly faster execution.

Using these Objects:

Each object has four properties to adjust their behavior: Time Between Arrivals, Maximum Arrivals, Service Time, and Number of Servers. By adjusting those parameters you can compare simulation results to those of many common queueing problems. By adjusting the number of replications in the experiments, you can see a clear illustration of the value of running sufficient replications.

Running Interactively to watch the provided animation

Select a model in Navigation window in the upper right (each model name begins with “SSS_”). Change to the Facility window for the selected model. Right click on the model name and select Properties to open the Model Properties. You will find the four properties in the Process Logic category. Set them and run to watch the animation.

Running experiments to perform a reliable statistical analysis

From the Navigation window, load the experiment for the selected model. You will see the four Controls (the properties) as well as two Responses (the Key performance Indicators). You can get a quick summary of the results here as well as go to the Pivot Grid for more complete data.

To really appreciate the value of additional replications, select the Response Chart tab in the experiment window, turn on the Histogram and Rotate Plot buttons if not already on, reset the run, then press run and watch how much the early results vary.

Modeling Concepts:

As mentioned above, our goal here is not to teach you the details of how to build these, but rather to provide the modeling concepts. So here are some of the concepts for each approach.

SSS_Library

Build the model by placing the three objects and connecting them with Connectors, not Paths, so they do not impact the timing. We expose the four properties by right-clicking on each property to create a new reference property. Then in the Properties panel of the Definitions tab, we fine tune the “properties” of each property to control things like category, units,

default value, and display name. We also hide the inherited properties that we do not want to clutter our model property window.

SSS_Process

This model is built using the Processes window and the Definitions window. In the processes window, we set up a simple Seize, Delay, Release, Tally. Since only one item is being seized, we can use the Parent Object itself as the constraint. We set up a separate Assign using the OnRunInitialized process to assign that desired quantity of the parent object. Note that this is all done with the Token that is executing the process – we never create any entities.

In the Definitions window we add a Timer to trigger periodic token arrivals to our process. We add a TallyStatistic to collect TimeInSystem and we add two OutputStatistics to collect the final values of the Queue statistics.

SSS_Process_with_Entity

This model is built using the Processes window and the Definitions window similar to above. But instead of using just the tokens, we also create Entities to support better animation. Since we have an entity, we also take advantage of using a Station as our constraint. Transferring into a station is similar to seizing a resource. Destroying the entity after it finishes processing also removes it from the station, the equivalent of a release.

Since the Entity and Station automatically generate their own statistics, we do not need the Tally step, or TallyStatistic and OutputStatistic elements.

Embellishments:

Note that while we have discussed this as a project containing three models, you could also load the spfx file as a library containing three objects.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SourceWithBalkingIfBlocked - SimBit

Problem:

I want to model balking of entity arrivals at a source if there is no immediate space at the downstream server.

Categories:

Buffering

Key Concepts:

Balking, Blocked, Buffer Logic

Assumptions:

There are no buffers in the system, including at the Source, Servers or on Paths, as all paths are represented with connectors, zero-time duration for movement.

Details for Building the Model:

Facility Window Setup

- Add a Source, two Servers, and a Sink to the Facility window. Connect the Source to Server1, Server1 to Server2 and Server2 to the Sink using Connector links.

Source Properties

- Specify the *Interarrival Time* as 'Random.Exponential(2.4)' minutes.
- Under Buffer Logic properties, specify the *Output Buffer Capacity* as '0'. Open the Balking & Reneging Options properties and leave the *Balk Decision Type* as 'Blocked' and the *Balk Node Name* as 'None (Destroy Entity)' so that when an entity is created, if it cannot immediately transfer into Server1 for processing, it will balk.

Server Properties

- Within Server1, specify the *Processing Time* as 'Random.Exponential(2)' minutes.
- Go to Buffer Logic -> Input Buffer and change the *Capacity* to '0'. Also within Buffer Logic properties, under Output Buffer, change the *Capacity* to '0'.
- Within Server2, specify the *Processing Time* as 'Random.Exponential(1.7)' minutes.
- Change both the Input Buffer and Output Buffer *Capacity* properties for Server2 to '0', similar to Server1.

Embellishments:

Send the Balked entity from the Source to a specified *Balk Node Name* for alternative processing.

See also:

ServerQueueWithBalkingAndReneging.spfx

ChangingQueuesWhenServerFails.spfx

MultiServerSystemWithJockeying.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SourceWithCustomEntityTypeProperty - SimBit

Problem:

I would like my Source object to create multiple entity types but instead of reading a table to get the information about which types of entities to create, I would like my user to input this information through a property on the Source object.

Categories:

Arrival Logic, Building New Objects / Hierarchy

Key Concepts:

Create Step, Dynamic Object Property, Expression Property, Override, Process, RandomRow, Repeating Group Property, SetRow Step, Status Label, Subclass

Assumptions:

A new, custom Source object is created, by subclassing the Source from Simio's standard library.

Technical Approach:

We subclass a standard Source object and add a new Repeating Group property that contains a Dynamic Object Instance property (Entity property) and an Expression property. This is the property used to input information into this Source on which type of entity to create and the percentage of which type of entity to create. The OnEntityArrival process is Overridden and modified. A new SetRow step is used to reference the contents of the new RepeatingGroup property. The Create Step is modified to reference the RepeatingGroup's Entity property for its Object Instance Name. This tells the new Source object to set a row in the Repeating Group property (based on the probabilities entered) and then create that particular type of entity.

Details:

Subclass the Standard Source object

- From the standard library in the Facility window, right click on the standard Source object and select "[SubClass](#)". This will place a new, subclassed Source object in your Project Library and you will automatically be navigated into this object. In other words, you'll see the new Source in the [Navigation window](#) (top right corner of interface) and the new Source will be highlighted. This indicates that you are inside of that new Source object – looking at its Facility window, not the main model's Facility window.
- From within this new Source object, go to its Definitions window. Click on the Properties panel on the left to view the Properties window.
 - Create a new [Repeating Group](#) property on this object by clicking the icon in the ribbon named "Repeat Group".
 - Rename this property 'Entity_Types' and set the *Description* in the properties window to 'The type of entity to create and the probability of creating this type.'
 - Set the *Category Name* to 'Arrival Logic'.
 - Ensure that this new property is highlighted in the Properties Window and then select "Entity" from the 'Object Reference' property dropdown of the Ribbon. This is adding a new Dynamic Object Instance type property into this Repeat Group property. This new property should appear under the heading Entity_Types.Properties in the main part of the Properties window, if it were indeed added to the Repeat Group.
 - With this new Dynamic Object Instance property highlighted/selected, change the *Name* to 'Entity_Type' and set the *Description* to 'The type of entity to create'.
 - Set the *Category Name* to 'Arrival Logic'
 - Go back and select/highlight the Entity_Types property and select 'Expression' from the Standard Properties dropdown of the Ribbon. This is adding a new Expression type property into this Repeat Group. This new property should appear under the heading Entity_Types.Properties, alongside the *Entity_Type* property.
 - With this new Expression property highlighted/selected, change the *Name* to 'Probability' and set the

Description to 'The probability of creating this type of entity.'

- Set the *Category Name* to 'Arrival Logic'
- From within this new Source object, go to its Processes window. Find the OnEntityArrival process and select it by clicking anywhere in the process. Click the Override button in the ribbon and see the process change to a White background, which indicates that you can now modify the logic.
 - Place a new SetRow step after the Execute step and before the Create step.
 - Change the *Name* to 'RandomRowInEntity_Types'.
 - Right click on the *Table Name* property and select 'Set Referenced Property' and select the *Entity_Types* Repeat Group property that we just created.
 - Put the expression, 'Entity_Types.Probability.RandomRow' into the expression for *Row Number*, which tells this SetRow step to select a row in the Entity_Types Repeat group, based on the values in the *Probability* property.
 - Ensure the *Object Type* property is set to 'Token' (found in the Advanced Options category), so that we are telling the Token to set a reference to the Repeat Group and not the associated object (which doesn't exist yet).
 - Click on the Create Step and right click on the *Object Instance Name* property. Select 'Set Referenced Property' and select *Entity_Types.Entity_Type*, which tells this Create Step to look there for which type of entity to create

Use the new Source in the main Model

- Navigate back to the main Model by selecting 'Model' in the Navigation window (upper right). From the Facility window, drag a standard Server and a standard Sink from the Standard Library. Find the new Source we created in the Project Library (bottom left, under the standard Library) and drag one instance of the new Source into the Facility window. Connect the three objects with Paths.
- Drag three instance of ModelEntity into the Facility window and rename them to 'PartA', 'PartB' and 'PartC'. Give each entity a unique color by selecting the instance, and selecting color from the Ribbon and then clicking back on the entity to give it that color.
- Click on the instance of the new Source object within the Facility window and find the Entity_Types property in the Arrival Logic category.
 - Click the ellipse at the far right of the new property to open the Repeat Group editor (to begin entering values).
 - Click 'Add' to add a new row into this Repeat Group property. Enter 'PartA' for *Entity_Type* and '.3' for the value of the *Probability* property.
 - Click 'Add' twice more to add a row for creating 'PartB' with a probability of '.4' and 'PartC' with a probability of '.3'.

Embellishments:

Consider adding Status Labels and/or Floor Labels to visually display how many of each type of entity has been created, in order to help debug and validate the behavior of the new Source object. (The expression, 'PartA.NumberCreated' will return the number of this type of entity that was created by the Source.)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SourceWithRateTable - SimBit

Problem:

I have entities for which the arrival rate varies over time.

Categories:

Arrival Logic

Key Concepts:

Arrival Mode, Poisson Distribution, Rate Tables, Source, Time Varying Arrival Rate

Assumptions:

I have an average of 20 entities arrive during every third hour (e.g. 2:00, 5:00, 8:00, 11:00, ...). On other hours I have no arrivals.

Technical Approach:

Specify the arrival data in a Rate Table. Reference that rate table in a Source object.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink in the Facility Window.
- Connect the Source to Server and Server to Sink using Paths.

Defining a Rate Table

- In the Data Window, select the Rate Tables panel and create a new rate table with *Name* 'RateTable1'. Change the *Interval Size* property to '1' and the *Number of Intervals* to '24'.
- Add the data for the average number of arrivals during each period.

Using the Rate Table within a Source

- Change the Source1 *Arrival Mode* property to 'Time Varying Arrival Rate' and *Rate Table* to 'RateTable1'.

Discussion:

Do not expect an exact number of arrivals in any given period. This is random, a non-homogeneous Poisson distribution. Over a sufficient number of trials, it will produce the average number of arrivals specified, but for any given period it may be unlikely to generate the exact number specified.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

StringStates - SimBit

Problem:

I would like to check the priority of part and then use a String State to display to the screen, whether or not the current part is considered a high risk part, a medium risk part or a low risk part.

Categories:

Discrete States, Functions

Key Concepts:

Before Exiting, Condition, Current Symbol Index, Floor Label, Math.If(), ModelEntity, Priority, State Assignments, String State Variable

Assumptions:

There are five different parts produced in this model, each assigned a priority of 1-5, with equal probability. Priority 1 is considered High Risk, Priority 2 is considered Medium Risk and the rest are considered Low Risk.

Technical Approach:

Assign the entity a value of 1-5 in the ModelEntity.Priority state via the Before Exiting State Assignment property of the Source object. Assign a value of "HIGH", "MEDIUM" or "LOW" to a String State, via the On Entering State Assignment property of the Source object, based on the priority of the entity that has just entered this Server. The String State is written to the screen in a Floor Label. The Math.If function is used to check ModelEntity.Priority and then assign a value to the String State.

Details for Building the Model:

Simple System Setup

- Drag a standard Source, a standard Server and a standard Sink into the Facility window. Connect them with Paths.
- Drag a ModelEntity into the Facility window from the Project Library. Rename this to 'Parts'. With the ModelEntity selected in the Facility window, click Add Additional Symbol button in the Ribbon to give this entity a [total of 5 symbols](#). Assign a different color to each symbol.
 - To assign colors to each symbol, select the ModelEntity object and click the Color button in the icon. Select the desired color and then click back onto the ModelEntity object in the Facility window. You should see the color change. You have just changed the color of the "active symbol". To determine which symbol is active, look at the Active Symbol button in the icon. To change which symbol is active, select the appropriate symbol from the Active Symbol dropdown.
 - When the ModelEntity object is selected in the Facility window, find the *Current Symbol Index* property in the properties window. Change the expression to 'ModelEntity.Priority -1'. (The minus 1 is needed because the Symbol index is 0 based but the smallest priority we'll assign is 1)

State Assignments

- Create a new [String State](#) on the Model by going to the Definitions window and clicking on the States panel along the left side of the window.
 - Click the String button in the Ribbon to create a String state. Rename it 'Alert'.
- Back in the model's Facility window, click on the Source object in the Facility window.
 - Find the *Before Exiting* property under the State Assignments category in the properties window and add a row to this repeat group property (by click on the ellipse that appears in this input box).
 - The *State Variable Name* is 'ModelEntity.Priority' and the *New Value* expression should be `Random.Discrete(1,2,2,4,3,.6,4,.8,5,1)`, which assigns the value of 1-5, each with an equal probability of .2
- Click on the Server object in the Facility window.
 - Find the *On Entering* property under the State Assignments category in the properties window and add a row to this repeat group property.

-
- The *State Variable Name* is 'Alert' and the *New Value* expression should be [Math.If](#)((ModelEntity.Priority == 1), "HIGH", (ModelEntity.Priority == 2), "MEDIUM", "LOW") This expression will assign a value of "HIGH" to Alert if the ModelEntity being processed has a priority of 1, it will return "MEDIUM" if the priority is 2 and "LOW" otherwise .

Add a Floor Label

- Add a [floor label](#) in the Facility window that will animate the current value of the Alert state variable. From the Drawing tab of the ribbon, click on the Floor Label button and draw a floor label somewhere in the Facility window.
- With the Floor Label selected, click the Edit button in the Ribbon, and type what you'd like to display. This example uses: The part being processed is {alert} risk. Notice that an expression must be between these brackets {}. The tags that are part of this Floor Label changes the text to the color blue and underlines the text.

Embellishments:

Slowing down the arrival and processing rates might make it easier to see the behavior of the system. In this example, the Source's *Interarrival Time* was changed from the default to 'Random.Exponential(2)' and the *Processing Time* of the Server was changed to '1'. You can also slow down the animation on the Run Tab of the ribbon.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

TableReferenceInRepeatingProperty - SimBit

Problem:

I have three types of entities and want to store their respective information about processing times, which worker to use and which assignments to make within a table.

Categories:

Data Tables, Worker

Key Concepts:

Before Creating Entities, Data Table, Dynamic Object Property, Expression Property, Numeric Property, RandomRow, Ride on Transporter, State Assignments, State Property, Table Row Referencing, Table Transporter Property

Assumption:

There are three types of entities and each one has only one worker that it uses and has only one assignment to make.

Technical Approach:

A Data Table is created that will store information about each entity type. Each row in the table will include information related to a specific entity type, including the percentage of each that enters the system, the worker that is needed to move the entity, the processing time at a given server and finally, the entity state and value to assign when the entity gets to the server.

Details for Building the Model:

Simple System Setup

- Add a Source, Server, and Sink, as well as two Worker objects to the Facility Window. Also, place three ModelEntity objects from the Project Library into the window.
- Change the *Names* of the ModelEntity to be 'Red', 'Blue' and 'Green'. Change the symbol color of each to match the name. This is done by highlighting the symbol and selecting the appropriate Color from the Symbols ribbon. We will be changing the picture of the Red entity, so select the Red entity symbol, click on Add Additional Symbol in the Symbol ribbon and change the picture so that graphically the symbols for 0 and 1 are different (colors or texture).
- Change the color of one of the Workers so that you can tell them apart graphically.
- Use Paths and connect Source1 to Server1, Server1 to Source1 (using unidirectional paths instead of a single bidirectional path) and also Server1 to Sink1.

Setting up the Data Tables

- In the Data Window, select the Tables panel and add a Data Table named 'Table1' with the following properties and in the following order:
 - (Entity Object Reference with *Name* 'EntityType') Blue, Green, Red
 - (Integer with *Name* 'Percentages') 30, 35, 35
 - (Expression with *Name* 'ProcessTime', *Unit Type* 'Time' and *Default Units* 'Minutes') Random.Triangular(1,2,3), 3.4, Random.Uniform(5,6)
 - (State with *Name* 'StateName') ModelEntity.Priority, ModelEntity.Size.Width, ModelEntity.Picture
 - (Expression with *Name* 'StateValue1') 2, 1.4, 1
 - (Transporter Object Reference with *Name* 'Workers') Worker1, Worker2, Worker1

Creating Multiple Entity Types from Source

- In the Facility Window, expand the Table Row Referencing in the Properties Window of the Source object.
- Under the Before Creating Entities subcategory, set the *Table Name* to 'Table1' and the *Row Number* to 'Table1.Percentages.RandomRow'.
- Change the *Entity Type* to 'Table1.EntityType' and set the *Interarrival Time* to 'Random.Exponential(2)'.

Specifying the Worker for Transport

- Within the Source's transfer node, 'Ouput@Source1', change the *Ride on Transporter* property to 'True'.
- Change the *Transporter Name* to 'Table1.Workers'.

Adding Processing Time and Assignment Information to the Server

- Within Server1, change the *Processing Time* to 'Table1.ProcessTime'.
- Expand the State Assignments and click on the button within the *On Entering* property to open the repeating property editor. Use the Add button and add a *State Variable Name* of 'Table1.StateName' and *New Value* of 'Table1.StateValue1'. Close the dialog and you should see the *On Entered* property change to '1 Row'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TallyStatisticsInTable - SimBit

Problem:

I have two different entity types and two exits from the system. I would like to keep entity statistics, by type, for each exit from the system.

Categories:

Custom Statistics

Key Concepts:

Add-On Process, Data Table, Dynamic Object Property, Expression Property, On Created Entity, On Entered, Path, Selection Weight, Table Row Referencing, Table Tally Statistic Element Property, Tally Step, TallyStatistic

Assumptions:

There are two entity types which go through a single server, then through an inspection area where they either pass inspection or fail inspection.

Technical Approach:

A Data Table is used to store information about each entity type as it moves through the system, including the processing time at the Server, the inspection percentages and also the tally statistics names that will be used when the entity either passes or fails inspection.

Details for Building the Model:

Simple System Setup

- Add two Sources, two Servers and two Sinks to the Facility window. Connect both Sources to Server1 with Paths. Connect Server1 to Server2, then Server2 to both Sinks with Paths.
- Change the *Name* of Sink1 to 'Passed_Inspection' and the *Name* of Sink2 to 'Failed_Inspection'.
- Add two ModelEntities from the Project Library to the Facility window. Change the *Name* property (under General category) on one to 'PartA' and the other to 'PartB'. Change the Color of the PartA symbol to red.

Setting Up Tally Statistics

- We will keep statistics on each part type that goes through each exit; therefore 4 tally statistics will be needed. In the Definitions tab, Elements panel, click on the Tally Statistic button and add 4 tallies.
- Change the *Name* properties (under General) to 'PartA_Passed', 'PartA_Failed', 'PartB_Passed' and 'PartB_Failed'.

Adding a Data Table

- Create a new table by clicking on the Data tab, Tables panel and selecting Add Data Table. Within the table, there will be five columns. First, add a Object Reference – Entity, then two Standard Property – Expression fields, then two Element Reference – Tally Statistic columns.
- Change the *Name* of the first column to 'Entity Type'. Add 'PartA' and 'PartB' as the row values in the column.
- Change the *Name* of the second column to 'Process Time' and its associated *Unit Type* to 'Time' and *Default Units* to 'Minutes'. Add 'Random.Triangular(3,4,5)' and 'Random.Triangular(3,5.4,7)' as the processing time values in the column.
- The third column should have the *Name* of 'Inspection Rate'. The rate should be '.9' for PartA and '.84' for PartB.
- Finally, the last two columns, which are tally references, should have *Name* values of 'Passed Tally' and 'Failed Tally'. In the 'Passed Tally' column, enter the 'PartA_Passed' and 'PartB_Passed' tally names. In the 'Failed Tally' column, enter the 'PartA_Failed' and 'PartB_Failed' tally names. These names will be referenced within the Tally step later in this description.

Setting up the Source Objects

- In the Facility Window, change the *Entity Type* property for Source1 to 'PartA'. In order to reference the table data, set the *Table Name* property to 'Table 1' and the *Row Number* to '1'.
- Change the *Entity Type* property for Source2 to 'PartB'. In order to reference the table data, set the *Table Name* property to 'Table 1' and the *Row Number* to '2'.

Modifying the Servers and Paths

- Within Server1, change the *Processing Time* to reference the table with 'Table1.ProcessTime'. Within Server2, change the *Name* to 'Inspection' and the *Processing Time* to 'Random.Triangular(2,2.5,3)'. All part types have the same inspection time distribution.
- On the paths from Inspection to the Passed_Inspection and Failed_Inspection sinks, we will be modifying the *Selection Weight* property to reflect the rate of failure for each part type. For the path going to Passed_Inspection, change the *Selection Weight* to 'Table1.InspectionRate'. For the path going to Failed_Inspection, change the *Selection Weight* to '1-Table1.InspectionRate'.

Using Custom Tally Statistics

- Within the Passed_Inspection Sink, create a new process in the Add-On Process Triggers *Entered* property named 'Passed_Inspection_Entered'. Within the Failed_Inspection object, create a similar new process in the *Entered* property named 'Failed_Inspection_Entered'.
- Within the Processes window, in the Passed_Inspection_Entered process, add a Tally step. Have the *Value Type* of 'Expression', *TallyStatistic Name* property of 'Table1.PassedTally', and *Value* of 'ModelEntity.TimeInSystem'.
- Do the same for the Failed_Inspection_Entered by adding a Tally step, changing the *TallyStatistic Name* to 'Table1.FailedTally' and *Value* to 'ModelEntity.TimeInSystem'.
- The above Tally steps will then point into the correct columns of Table1 to utilize the various Tally Statistics elements for each of the part types, PartA and PartB. These custom statistics can then be seen in the Model statistics, with the Data Source value being the Tally Statistic Name (PartA_Passed, PartA_Failed, etc.).

Embellishments:

The tally statistics that are added may be modified to categorize them in a more meaningful fashion. Within the Tally Statistics elements in the Definitions tab, each tally has a Results Classification area of properties. Using the *Data Source*, *Category* and *Data Item* properties can help to classify the user statistics in the results. For example, you may wish to set them as 'Results', 'PartA' and 'Passed' respectively (and similar for the other Tallies).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TankCleanInPlaceTrigger - SimBit

Problem:

Empty and clean the tank when the inflow entity type changes, but if the tank is already empty, skip cleaning. The time to clean the tank is dependent on which inflow entity type is used.

Categories:

Add-On Process Logic, Flow Library

Key Concepts:

Add-On Process, Assign Step, Changeovers, Clean In-Place Triggers, Decide Step, Delay Step, FlowConnector, FlowSink, FlowSource, LastChangeoverState, List Property, StringList

Technical Approach:

The Inflow entity type changes every 5 minutes by enabling and disabling FlowSource regulators. Using the Tank's Clean-In-Place Triggers, the tank is cleaned on the event when the inflow entity type changes. The required cleaning time is then determined by looking at the specified 'From/To' changeover matrix and getting the time when transitioning from the last changeover state to the current changeover state of the clean in-place operation. When the entity type changes but the tank is empty, the LastChangeoverState variable is set to 0 so the cleaning is skipped.

Details for Building the Model:

Simple System Setup

- Add 3 FlowSources, a Tank, and a FlowSink to the Facility window. Add 3 ModelEntity objects to the Facility window. Connect the FlowSources to the Tank and the Tank to the FlowSink using FlowConnectors.
- Rename the ModelEntity objects 'Flow1', 'Flow2', and 'Flow3'. Select Flow2, click the Color dropdown, select Red, and click Flow2. Select Flow3, click the Color dropdown, select Light Blue, and click Flow3. Set *Entity Type* of FlowSource1 to 'Flow1', *Entity Type* of FlowSource2 to 'Flow2', and *Entity Type* of FlowSource3 to 'Flow3'.

Flow Regulator Logic

- Within the output flow nodes of the FlowSource objects, set *Regulator Initially Enabled* to 'False' for FlowSource2 and FlowSource3. This causes the flow from both FlowSource2 and FlowSource3 to initially be off, while the flow from FlowSource1 will start immediately.
- Within the output flow nodes of FlowSource3 and Tank1, set the *Initial Maximum Flow Rate* to '25' Cubic Meters per Hour.
- Click the Processes window and select OnRunInitialization from the Select Process dropdown. Add the following steps in order - Delay, Assign, Delay, Assign, Delay, and Assign. Click the third Assign step and move the end point to the beginning of the first Delay step.
- Set the *Delay Time* to '5' and *Units* to 'Minutes' for each Delay step.
- For the first Assign, set the *State Variable Name* to 'Output@FlowSource1.FlowRegulator.Enabled' and *New Value* to 'False'. Add a Row to Assignments (More) and set the *State Variable Name* to 'Output@FlowSource2.FlowRegulator.Enabled' and *New Value* to 'True'.
- For the second Assign, set the *State Variable Name* to 'Output@FlowSource2.FlowRegulator.Enabled' and *New Value* to 'False'. Add a Row to Assignments (More) and set the *State Variable Name* to 'Output@FlowSource3.FlowRegulator.Enabled' and *New Value* to 'True'.
- For the third Assign, set the *State Variable Name* to 'Output@FlowSource3.FlowRegulator.Enabled' and *New Value* to 'False'. Add a Row to Assignments (More) and set the *State Variable Name* to 'Output@FlowSource1.FlowRegulator.Enabled' and *New Value* to 'True'.

Clean In-Place Triggers Logic

- Select ModelEntity in the Navigation window. Click the Definitions window then select Lists from the panel. Click the String button. Select StringList1 and rename it to 'FlowTypeList'. Type 'Flow1', 'Flow2', 'Flow3' into the list.
- Click the Property option from the Definitions panel. Click the Standard Property Window and select List. Change the *Name* to 'FlowType'. Set the *List Name* to 'FlowTypeList'.
- Select Model in the Navigation window. Click the Facility window. Click each Model Entity, and set the *Flow Type* to

its Entity's name.

- Click the Definitions window then Lists button. Click the String button. Select StringList1 and type 'Flow1', 'Flow2', 'Flow3' into the list.
- Select the Data window, Changeovers button on the panel. Click the Changeover Matrix button. Set the *String List Name* to 'StringList1' and *Time Units* to 'Minutes' for ChangeoverMatrix1. Fill in the Matrix '0,1,2;4,0,3;5,6,0'.
- Click the Facility window and select the Tank. Add a row to the Clean-In-Place Triggers and set *Triggering Event Name* to 'Tank1.InflowEntityTypeChanged', *Cleaning Time Type* to 'Sequence Dependent', Operation Attribute to 'ModelEntity.FlowType', and *Changeover Matrix* to 'ChangeoverMatrix1'.
- Double click *New Inflow Entering* label of the Add-On Process Triggers to create a new process.
- In the Processes window, add a Decide step and an Assign step (from the True exit of the Decide step) to the new process.
- Within the Decide step, set the *Expression* to 'Tank1.FillStatus==0', and on the Assign step, set the *State Variable Name* to 'Tank1.LastChangeoverState' and *New Value* to '0'. This will check to see if the tank is empty (fillstatus is '0'), the changeover will not be necessary.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TaskSequenceAndWorker - SimBit

This SimBit project includes two models that demonstrate the use of the Server's Task Sequence capability including Workers.

Models included in this SimBit:

1. **TaskSequenceAndWorker_InServer** – Demonstrates the Server's task sequence option including a number of process tasks requiring workers. The data within this model is stored directly within the Server object.
2. **TaskSequenceAndWorker_InTable** – Demonstrates the Server's task sequence option including a number of process tasks requiring workers. The data within this model is stored in a table and referenced from within the Server object. This provides additional flexibility as the data can be read in from an external data source.

Model 1: TaskSequenceAndWorker_InServer

Problem:

I would like to perform a number of concurrent tasks at a Server with a worker, including having the worker move to another location, perform a task then move back to my location. The information about the tasks is specified in the Server.

Categories:

Add-On Process Logic, Decision Logic -- Processing, Decision Logic -- Paths, MultiTask Server, Resources, Worker

Key Concepts:

Add-On Process, Bidirectional Path, DestinationNode, Keep Reserved If, People, Release Step, Request Move, Resource, Seize Step, Server, Task Sequence, TransferNode, Worker

Assumption:

We will be reserving the Worker within the Server's processing tasks for each task and then again for the transport task from one Server to the next. It is assumed that the object ride capacity is '1' when using reserved workers/vehicles.

Technical Approach:

There are multiple tasks that occur within the waiting area of the model, including seizing a room resource, seizing the nurse worker to go and clean the room, then moving the nurse to the patient location. The 'Keep Reserved If' options are set to 'True' so that the same nurse performs the various operations, as well as is reserved for transporting the entity patient to the Exam Server. All tasks that occur within the waiting room are specified using the Task Sequence option for processing in a Server.

Details for Building the Model:

Simple System Setup

- Place a Source, two Servers and a Sink in the Facility window. Change the *Name* of the Source to 'ArrivalArea', the first Server to 'WaitingArea' and the second Server to 'Exam1'. *Name* the Sink 'Door'. Add a ModelEntity to the model and name it 'SickPatient'. Position the Servers and their input/output nodes as shown in the example.
- Place three TransferNode objects in the Facility window, one to the top left of the Exam1 server ('NurseStation'), one directly below that and in line with the WaitingArea and Exam1 servers ('TransferNode1') and one below the Exam1 server ('TransferNode2').
- Connect the ArrivalArea source to the WaitingArea server with a Path. Use Paths to connect the NurseStation node to TransferNode1 and then all other possible paths. Note that using bidirectional paths is an easy way to allow travel in both directions, but keep in mind this can cause potential deadlocking with multiple workers/vehicles and can cause entity logic trouble if the entity destination is 'Continue' and not a specific destination.
- Connect the input/output nodes in the Exam1 server with a Connector or Path.

Defining the Worker (Nurse)

- Place a Worker object in the Facility window with the *Name* 'Nurse'. Change the *Initial Node (Home)* to 'NurseStation' and the *Idle Action* to 'Park At Home'. Note that in this particular model, we only have a single nurse (*Initial Number in System* under Population is default of '1').

Entity Arrivals

- Within the ArrivalArea source, change the *Interarrival Time* to 'Random.Exponential(2.5)' minutes. The connection to the WaitingArea will allow all entities to directly move to that area for processing.

Waiting Area Logic

- Within the WaitingArea server, change the *Initial Capacity* to 10. This will allow 10 entities to be waiting in the area in a graphical queue to perform the various tasks specified.
- Change the *Process Type* to 'Task Sequence' and enter the *Processing Tasks* repeating properties dialog. All tasks will occur sequentially, therefore we will number them as such.
 - For *Sequence Number* '10', the *Name* is 'ReserveRoom' and the *Process Type* is 'Process Name'. The *Process Name* that will include the logic for this process is 'GetRoom'. (see below).
 - For *Sequence Number* '20', the *Name* is 'SendNurseToRoomAndClean', the *Process Type* is 'Specific Time' and *Processing Time* is 'Random.Triangular(.5,1,1.5)'. It is important to note that for this process task, the nurse is required. Thus, the *Object Name* is 'Nurse', the *Request Move* is 'ToNode' and *Destination Node* is 'Input@Exam1'. This task will seize the nurse, move the nurse to the Exam1 location and delay for the processing time specified.
 - For *Sequence Number* '30', the *Name* is 'BringBackNurse', the *Process Type* is 'Specific Time' and *Processing Time* is '0'. The nurse is also required for this task. Thus, the *Object Name* is 'Nurse', the *Request Move* is 'ToNode' and *Destination Node* is 'Output@WaitingArea'. This task will seize the nurse, move the nurse to the WaitingArea location.
 - One key concept with multiple process tasks using resource requirements is that by default, within the Advanced Options section of properties, the *Keep Reserved If* is set to 'True'. Therefore, the resource is reserved automatically when used from one task to another.
- Within the WaitingArea output node, change the *Entity Destination Type* to 'Specific' and the *Node Name* to 'Input@Exam1'. It is important to specify the entity destination when a network of nodes will be passed to that the entity continues to its correct destination. Change the *Ride On Transporter* to 'True' and the *Transporter Name* to 'Nurse'. Note that the *Keep Reserved If* property is blank, meaning we will not reserve the nurse for processing in the exam room, although if the nurse was needed for that processing, this should be set to 'True'.

Exam Logic

- Within the Exam1 server, change the *Processing Time* to 'Random.Triangular(1,2,3)' minutes. Within the Add-On Process Triggers, specify the process name 'LeaveRoom' on the *Exited* property. Next, we will determine the logic for that process, as well as the 'GetRoom' process specified in the WaitingArea process task.

GetRoom and LeaveRoom Processes Logic

- Within the Facility window, place a Resource object with the *Name* 'Room1'. This room object will be used to control the number of entities (patients) that are allowed to have the room reserved/cleaned or be in the room at any given time.
- Within the Processes window, create two processes. The first process has a *Name* of 'GetRoom' and should contain a single Seize step that seizes the *Object Name* 'Room1' resource. This is done, as noted above, within the first task of the WaitingArea, before the nurse worker is moved to the room for cleaning.
- The second process has a *Name* of 'LeaveRoom' and should contain a single Release step that releases the *Object Name* 'Room1' resource. This release is performed once the processing time at the Exam1 server is complete and the entity is exiting the server.

Model 2: TaskSequenceAndWorker_InTable

Problem:

I would like to perform a number of concurrent tasks at a Server with a worker, including having the worker move to another location, perform a task then move back to my location. The information about the tasks should be specified in a table so that it is easily editable and/or imported from an external data file.

Categories:

Add-On Process Logic, Data Tables, Decision Logic -- Processing, Decision Logic -- Paths, MultiTask Server, Resources, Worker

Key Concepts:

Add-On Process, Bidirectional Path, Data Table, DestinationNode, Enumeration Property, Keep Reserved If, Node Property, Object Reference Property, People, Process Element Reference Property, Release Step, Request Move, Resource, Seize Step, Server, Set Referenced Property String Property, Task Sequence, Table Sequence Property, TransferNode, Worker

Assumption:

We will be reserving the Worker within the Server's processing tasks for each task and then again for the transport task from one Server to the next. It is assumed that the object ride capacity is '1' when using reserved workers/vehicles.

Technical Approach:

There are multiple tasks that occur within the waiting area of the model, including seizing a room resource, seizing the nurse worker to go and clean the room, then moving the nurse to the patient location. The 'Keep Reserved If' options are set to 'True' so that the same nurse performs the various operations, as well as is reserved for transporting the entity patient to the Exam Server. All tasks that occur within the waiting room are specified using the Task Sequence option however, all property values are specified directly within a data table.

Details for Building the Model:

Simple System Setup

- Same as Model 1 above.

Defining the Worker (Nurse)

- Same as Model 1 above.

Entity Arrivals

- Same as Model 1 above.

Data Tables and Process Task Properties

- Within the Data window, add a new Data Table to the model. Change the Name of the table to 'ActivityTasks'. Include the following column types/names:
 - Sequence Number property (Standard Property) – Name is 'TaskSequenceNumber'
 - String property (Standard Property) – Name is 'TaskName'
 - Enumeration property (Standard Property) – Name is 'ProcessType' (Enum Type is 'TaskProcessType')
 - Expression property (Standard Property) – Name is 'ProcessingTime' (Unit Type is 'Time' and Default Units are 'Minutes')
 - Process Element property (Element Reference) – Name is 'ProcessName'
 - Object property (Object Reference) – Name is 'ObjectName'
 - Enumeration property (Standard Property) – Name is 'RequestMove' (Enum Type is 'SeizeRequestVisitType')
 - Node property (Object Reference) – Name is 'DestinationNode'
- Each row in the table will then represent a unique processing task. Since there will be three processing tasks, there should be three rows, including the following data:
 - Task Sequence Number '10', Task Name 'ReserveRoom', Process Type 'Process Name', Processing Time '0', Process Name 'GetRoom', Object Name *blank*, Request Move 'None' and Destination Node *blank*
 - Task Sequence Number '20', Task Name 'SendNurseToRoomAndClean', Process Type 'Specific Time', Processing Time 'Random.Triangular(5,1,1.5)', Process Name *blank*, Object Name 'Nurse', Request Move 'ToNode' and Destination Node 'Input@Exam1'
 - Task Sequence Number '30', Task Name 'BringBackNurse', Process Type 'Specific Time', Processing Time '0', Process Name *blank*, Object Name 'Nurse', Request Move 'ToNode' and Destination Node 'Output@WaitingArea'
- IMPORTANT NOTE: When using Table Data for entry into a repeating group of properties, you must have a column for each property that MAY be changed in a given entry / process task – if the property remains as the default value for all entries, it is not necessary to have a column feed-in (i.e., the Selection Goal for the Resource Requirement).

Waiting Area Logic

- Within the WaitingArea server, change the Initial Capacity to 10. This will allow 10 entities to be waiting in the area in a graphical queue to perform the various tasks specified.
- Change the Process Type to 'Task Sequence' and enter the Processing Tasks repeating properties dialog. The task data will come directly from the particular table generated in the above step.
 - Set the Sequence Number to 'ActivityTasks.TaskSequenceNumber'
 - Set the Name to 'ActivityTasks.TaskName'
 - Set the Process Type to 'ActivityTasks.ProcessType'
 - Set the Processing Time to 'ActivityTasks.TaskProcessingTime'
 - Set the Process Name to 'ActivityTasks.ProcessName'

-
- Set the *Object Name* to 'ActivityTasks.ObjectName'
 - Set the *Request Move* to 'ActivityTasks.RequestMove'
 - Set the *Destination Node* to 'ActivityTasks.DestinationNode'
 - Then close the Processing Tasks repeating property editor.
 - Within the *Processing Tasks* property field, type in 'ActivityTasks' which is the name of the table. This is important for the repeating editor to know the table reference where all internal data is stored.
 - Within the WaitingArea output node, change the *Entity Destination Type* to 'Specific' and the *Node Name* to 'Input@Exam1'. It is important to specify the entity destination when a network of nodes will be passed to that the entity continues to its correct destination. Change the *Ride On Transporter* to 'True' and the *Transporter Name* to 'Nurse'. Note that the *Keep Reserved If* property is blank, meaning we will not reserve the nurse for processing in the exam room, although if the nurse was needed for that processing, this should be set to 'True'.

Exam Logic

- Same as Model 1 above.

GetRoom and LeaveRoom Processes Logic

- Same as Model 1 above.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Tracking Tank Contents By Product Type Using Table States

- SimBit

Problem:

I want to keep track of individual quantities of flow volume within a Tank during a simulation run.

Categories:

Add-On Process Logic, Flow Library

Key Concepts:

Assign Step, Delay Step, Flow Connector, Flow Source, Monitor, Tank

Assumptions:

Using State Columns in a Data Table is a Professional/RPS license feature. This model could be made with a non-Professional/RPS license by replacing the State Columns with using States on the Model level that has values assigned to it with an Assign step.

Technical Approach:

A data table comprised of a Key Column of each Entity and Level State Columns for each Tank in the system can keep record of flow quantities by Entity Type and Tank with the use of Add-On Process logic and Monitors.

The RowForKey function is used to resolve which Liquid row in the Tank Contents data table corresponds with which volume in a specific Tank. RowForKey returns the row index of the row for which the specified KeyColumnName has the specified keyValue.

In lieu of using the RowForKey function, an alternative approach where the index of each entity type is at a known index in the table could be considered.

Details for Building the Model:

Simple System Setup

- Go to ModelEntity in the Navigation Pane:
 - Create a String List named 'LiquidTypeName' that has 'RedLiquid' and 'GreenLiquid' as its two strings.
 - Create a List Property named 'LiquidType' whose *List Name* is 'LiquidTypeName'.
- Return to the Model and place two ModelEntity objects in the Facility window. Rename one to 'RedLiquid' and the other to 'GreenLiquid', respectively. Change the Color of RedLiquid to Red.
- Select 'RedLiquid' and change the *Initial Priority* to '2.0' and ensure that *LiquidType* is 'RedLiquid'.
- Select 'GreenLiquid' and set *LiquidType* to 'GreenLiquid'.
- Place two FlowSource objects in the Facility window. Rename one to 'RedLiquidSource' and the other to 'GreenLiquidSource', respectively.
- Select RedLiquidSource and ensure that the *Entity Type* is 'RedLiquid'.
- Select Output@RedLiquidSource and set *Initial Maximum Flow Rate* to '1' and *Regulator Initially Enabled* to 'False'.
- Select GreenLiquidSource and ensure that the *Entity Type* is 'GreenLiquid'.
- Select Output@GreenLiquidSource and set *Initial Maximum Flow Rate* to '1' and *Regulator Initially Enabled* to 'False'.
- Place two Tank objects in the Facility window.
- Select Tank1 and set the *Contents Ranking Rule* to 'Smallest Value First' and *Contents Ranking Expression* to 'ModelEntity.LiquidType'.
- Select Input@Tank1 and set *Initial Maximum Flow Rate* to '1'.
- Select Output@Tank1 and set *Initial Maximum Flow Rate* to '0.3333'.
- Select Input@Tank2 and set *Initial Maximum Flow Rate* to '0.3333'.
- Link the following Nodes together with FlowConnector objects:
 - Output@RedLiquidSource to Input@Tank1.
 - Output@GreenLiquidSource to Input@Tank1.
 - Output@Tank1 to Input@Tank2.

Creating Monitors

- Open the Definitions window and select the Elements view, in the General category, click Monitor twice to add two Monitors to this model.
- Select the first Monitor and rename it 'Tank1FlowMonitor'.

- Select the second Monitor and rename it 'Tank2FlowMonitor'.
- Select Tank1FlowMonitor and open the Repeating Property Editor for *Monitored State Variables (More)* and then Add two Items:
 - *State Variable Name* 'Tank1.FlowContainer.CurrentVolumeFlowIn.Rate'.
 - *State Variable Name* 'Tank1.FlowContainer.CurrentVolumeFlowOut.Rate'.
- Select Tank2FlowMonitor and open the Repeating Property Editor for *Monitored State Variables (More)* and then Add two Items:
 - *State Variable Name* 'Tank2.FlowContainer.CurrentVolumeFlowIn.Rate'.
 - *State Variable Name* 'Tank2.FlowContainer.CurrentVolumeFlowOut.Rate'.

Setting Up the Data Table

- Navigate to the Data tab and select the Tables view and click Add Table.
- Add an Entity Column, rename it 'ProductType', and Set Column As Key:
 - Add a row for RedLiquid.
 - Add a row for GreenLiquid.
- Add two Level State Columns, one for each Tank:
 - Rename one 'Tank1Level'.
 - Rename the other 'Tank2Level'.

Creating Processes

- Create an *OnRunIntialized* Add-On Process:
 - Place an Assign step with two assignments:
 - Set *State Variable Name* to 'Output@RedLiquidSource.FlowRegulator.Enabled' and *New Value* to 'True'.
 - Set *State Variable Name* to 'Output@GreenLiquidSource.FlowRegulator.Enabled' and *New Value* to 'False'.
 - Place a Delay step with a *Delay Time* of '1 Hour'.
 - Place an Assign step with two assignments:
 - Set *State Variable Name* to 'Output@RedLiquidSource.FlowRegulator.Enabled' and *New Value* to 'False'.
 - Set *Set Variable Name* to 'Output@GreenLiquidSource.FlowRegulator.Enabled' and *New Value* to 'True'.
 - Place a Delay step with a *Delay Time* of '1 Hour' and have this Delay step loop back to the first Assign step.
- Navigate to the Definitions tab, select Tank1FlowMonitor, and then double click on *Triggered Process Name* to create an Add-On Process.
 - Place an Assign step with two assignments:
 - Set *State Variable Name* to 'TankContents[TankContents.ProductType.RowForKey(RedLiquid)].Tank1Level.Rate' and *New Value* to 'Math.If(Input@Tank1.FlowRegulator.OutputFlowReceivers.FirstItem.Is.RedLiquid, Tank1.FlowContainer.CurrentVolumeFlowIn.Rate,0.0)-Math.If(Output@Tank1.FlowRegulator.OutputFlowReceivers.FirstItem.Is.RedLiquid, Tank1.FlowContainer.CurrentVolumeFlowOut.Rate,0.0)'.
 - Set *State Variable Name* to 'TankContents[TankContents.ProductType.RowForKey(GreenLiquid)].Tank1Level.Rate' and *New Value* to 'Math.If(Input@Tank1.FlowRegulator.OutputFlowReceivers.FirstItem.Is.GreenLiquid, Tank1.FlowContainer.CurrentVolumeFlowIn.Rate,0.0)-Math.If(Output@Tank1.FlowRegulator.OutputFlowReceivers.FirstItem.Is.GreenLiquid, Tank1.FlowContainer.CurrentVolumeFlowOut.Rate,0.0)'.
- Navigate to the Definitions tab, select Tank2FlowMonitor, and then double click on *Triggered Process Name* to create and Add-On Process.
 - Place an Assign step with two assignments:
 - Set *State Variable Name* to 'TankContents[TankContents.ProductType.RowForKey(RedLiquid)].Tank2Level.Rate' and *New Value* to 'Math.If(Input@Tank2.FlowRegulator.OutputFlowReceivers.FirstItem.Is.RedLiquid, Tank2.FlowContainer.CurrentVolumeFlowIn.Rate,0.0)-Math.If(Output@Tank2.FlowRegulator.OutputFlowReceivers.FirstItem.Is.RedLiquid, Tank2.FlowContainer.CurrentVolumeFlowOut.Rate,0.0)'.
 - Set *State Variable Name* to 'TankContents[TankContents.ProductType.RowForKey(GreenLiquid)].Tank2Level.Rate' and *New Value* to 'Math.If(Input@Tank2.FlowRegulator.OutputFlowReceivers.FirstItem.Is.GreenLiquid, Tank2.FlowContainer.CurrentVolumeFlowIn.Rate,0.0)-Math.If(Output@Tank2.FlowRegulator.OutputFlowReceivers.FirstItem.Is.GreenLiquid, Tank2.FlowContainer.CurrentVolumeFlowOut.Rate,0.0)'.

Enhancements:

Animation can be added to show the current volume in each respective Tank broken down by Entity Type as well as the

total volume in the Tank.

- Add a Floor Label for each Tank, the Floor Label for Tank1 could look something like this:
 - Total Volume In Tank1: {Tank1.FlowContainer.Contents.Volume}
 - Total Red Liquid Volume: {TankContents[TankContents.ProductType.RowForKey(RedLiquid)].Tank1Level}
 - Total Green Liquid Volume: {TankContents[TankContents.ProductType.RowForKey(GreenLiquid)].Tank1Level}

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TransferLine - SimBit

Problem:

You have a transfer line that has a single conveyor with multiple workcenters. All parts on the conveyor move together. None can move unless all are ready.

Categories:

Add-On Process Logic, Conveyor Systems

Key Concepts:

Accumulating, Add-On Process, BasicNode, Conveyor, DesiredSpeed, On Entered, Real State Variable, TransferNode

Assumptions:

Entities will move synchronously and all are aligned at the designated stopping locations (either workcenter or intermediate location) together.

Technical Approach:

Use multiple non-accumulating conveyors. Use nodes with add-on processes to delay for the operations and coordinate the conveyor segments.

Details for Building the Model:

Simple System Setup

- Place a Source and Sink object in the Facility Window.
- Place 3 BasicNodes and 1 TransferNode in the middle of the Source and Sink objects. Connect the Source to the first BasicNode with a Path, as well as the TransferNode to the Sink with a Path.
- Use 3 Conveyors to connect the three BasicNodes together, as this will represent our workcenters that move synchronously.

Defining System States

- Click on the Definitions tab and select the States panel. Add a Discrete State with the *Name* 'NumberStationsWorking' that represents how many stations have operations in progress that would prevent the line from moving.

Using Add-On Process Triggers

- On the two internal nodes representing the workcenters, in the *Entered* trigger for add-on processes, add the process named 'TransferLineNode_Entered'. Open the Processes Window and define the logic for this process.
- Add an Assign step that increases the *State Variable Name* 'NumberStationsWorking' to 'NumberStationsWorking + 1' to indicate that this operation is in-process. In the same Assign step, assign the 'Conveyor*.DesiredSpeed' of each conveyor to 0.
- Use a Delay step for the processing time at that workcenter (if you needed processing resources you would seize before the delay and release after the delay).
- Add another Assign step to decrement 'NumberStationsWorking' to indicate that this operation is complete.
- Use the Decide step to determine if all other operations are done ('NumberStationsWorking == 0') and if all other operations are done, restore the 'Conveyor*.DesiredSpeed' of each conveyor using an Assign step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TransporterListForVehiclesOnDifferentSchedules

- SimBit

Problem:

I require vehicles to transport my entities and there are vehicles that follow different work schedules. I would like to select whichever vehicle is currently On Shift. If more than one vehicle is On Shift, I will select whichever is closest to the entity that requires a ride.

Categories:

Schedules / Changeovers, Vehicles

Key concepts:

Entity Destination Type, Initial Node (Home), On Shift, Ride On Transporter, Schedules, TransporterList, Transporters, WorkSchedule

Technical Approach:

The entity needs a ride on a Transporter to get from Server1 to Sink1. There are two possible Vehicles; one follows a standard day shift and the other is on second shift, which begins at 3:00pm. A Transporter List is used so the entity selects whichever Vehicle is available at the time.

Details for Building the Model:

Creating Vehicle Schedules

- Go to the Schedules window by selecting Schedules along the left panel from within the Definitions window.
- Click on the Day Patterns tab and change the *Name* of the Day Pattern to 'StandardFirstShift'. Keep the contents of the Day Pattern with the default 8am-12pm (1) and 1pm-5pm (1).
- Create a second Day Pattern by starting to type in the next row of the Day Pattern table. *Name* this new Day Pattern 'StandardSecondShift'.
 - Click onto the '+' next to this new Day Pattern to enter the times for this new shift.
 - Create a row where the *Start Time* is '3:00 PM' and the *End Time* is '7:00 PM'. The *Value* should be '1'.
 - Create another row where *Start Time* is '8:00 PM' and the *End Time* is '12:00 AM'. The *Value* should be '1'.
- Click back onto the Work Schedules tab. Change the *Name* of the default Work Schedule listed to 'FirstShiftWeek'. Set the *Days* column to '1' so that this schedule will repeat the same daily schedule for the duration of the simulation. Set the *Day1* column to 'StandardFirstShift'.
- Create a new Work Schedule by creating another row in the table. *Name* the new Work Schedule 'SecondShiftWeek'. Set the *Days* column to '1' and the *Day1* column to 'StandardSecondShift'.

Simple System Setup

- Place a Source, Server and Sink object in the Facility window. Place a TransferNode between the Server and Sink. Connect the Source to the Server with a Path. Connect the Server to the Sink, then the Sink to the TransferNode, and the TransferNode back to the Server with Paths. This should form a circle so that traffic will travel in one direction from the Server to Sink to TransferNode and back to the Server.
- In the Source, set the *Interarrival Time* property to 'Random.Exponential(4)' minutes.
- Set the *Processing Time* of the Server object to 'Random.Triangular(3,4,5)' minutes.
- Place two Vehicle objects and change the *Name* properties to 'FirstShift' and 'SecondShift'.
 - Set the *Initial Node (Home)* property of both Vehicles to the name of the TransferNode that was placed. Set the *Idle Action* and the *OffShift Action* to 'ParkAtHome'.
 - Set the *Initial Desired Speed* of each Vehicle to '.1' Meter per Second.
 - Set the *Capacity Type* of each Vehicle to 'WorkSchedule' and set the FirstShift vehicle's *Work Schedule* property to 'FirstShiftWeek' and the SecondShift vehicle's *Work Schedule* property to 'SecondShiftWeek'.

Defining the Vehicle List

- Go to the Lists window by selecting Lists along the left panel from within the Definitions window.

-
- Click on the Transporter in the ListData ribbon to create a new Transporter List. Add the names of two vehicles that were placed into the model.

Selecting the Vehicle From the List

- Within the Facility window, click onto the Output@Server1 node. Set the *Entity Destination Type* to 'Specific' and the *Node Name* property to 'Input@Sink1'. This will ensure that the entity is dropped off at the Sink and that it will not return back to the Server. Set the *Ride On Transporter* property to 'True'. Set the *Transporter Type* property to 'Select From List' and set the *Transporter List Name* to the name of the Transporter List that was created.

Testing the Model

- Run the model beginning at 1:00pm. At this point in time, the First Shift vehicle is available. At 3pm, the Second Shift vehicle will come On Shift and they are both available until 5pm, which is when the First Shift vehicle is no longer available.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TransportingVaryingSizedEntities

This SimBit project TransportingVaryingSizedEntities includes two models demonstrating a modeling approach for considering an Entity's size for loading on a Transporter with limit capacity.

1. TransportingVaryingSizedEntities: Supplement the Vehicle's logic such that an Entity's size is a constraint on the number of Entities that can ride on the Vehicle.
2. TransportingVaryingSizedEntities_CustomVehicle: Enhance the Vehicle's logic (custom object) such that an Entity's size is a constraint on the number of Entities that can ride on the Vehicle.

Model 1: TransportingVaryingSizedEntities

Problem:

I want to model parts that require different amounts of my transporter's carrying capacity based on their entity type.

Categories:

Add-On Process Logic, Standard Library, Vehicles

Keywords:

Source, Sink, Bi-Directional Path, Vehicle, Decide Step, Assign Step, Integer State Variables, Ride On Transporter, Status Label, Dynamic Label Text

Technical Approach:

Entities and their respective transport size requirements are created and defined at the Source. A Vehicle is used to transport Entities from the Source to the Sink along a Path. A State Variable is used to represent the carrying capacity of the Vehicle. Add-on Processes on the Vehicle update and check the State Variable.

Details for Building the Model:

Simple System Setup

- From the Standard Library place a Source and Sink Object in the Facility view.
- Use a Path to connect the Output@Source to the Input@Sink
- Change Path's *Type* to 'Bi-Directional'.
- From the Project Library place a ModelEntity instance in the Facility view. Change the name to 'Part'.
- While 'Part' is selected, navigate to the Symbol's Ribbon under Object Tools. Click Add Additional Symbol twice.
- Using the Active Symbol dropdown, select image 1, the second in the list. Enlarge the image to be twice its original size and change the color to Red.
- Again, using the Active Symbol dropdown, select image 2, the last in the list. Enlarge the image to be three times its original size and change the color to Blue.
- Using the Navigation Window, select ModelEntity and navigate to the Definition tab, select States.
- Create an Integer State and rename it 'TransportSize'
- Using the Navigation Window, select return to the model (TransportingVaryingSizedEntities) and navigate to the Facility Tab. Select 'Source1' and under the *Entity Arrival Logic* properties set the *Interarrival Time* to 'Random.Exponential(.5)'
- Expand the *State Assignments* properties and open the *Before Exiting – Repeating Property Editor*.
- Click Add to create an assignment. Set *State Variable Name* to 'ModelEntity.TransportSize' and *New Value* to 'Random.Discrete(1, 0.33, 2, 0.66, 3, 1)'.

- Again, click Add to create another assignment. Set *State Variable Name* to 'ModelEntity. Picture' and *New Value* to 'ModelEntity.TransportSize-1'.
- Navigate to the Definitions Tab and select States.
- Create an Integer State and rename it 'Vehicle1CurrentCarryCapacity'
- Set *Initial State Value* to '3'.
- Return to the Facility view and from the Standard Library place a Vehicle instances in the Facility view.
- Under the *Transport Logic* properties set the *Initial Ride Capacity* to 'Vehicle1CurrentCarryCapacity'
- Under the *Travel Logic* properties set the *Initial Desired Speed* to '0.75' Meters per Second
- Select 'Output@Source1' node and under the *Transport Logic* properties set the *Ride On Transport Request* to 'True'.
- Using the dropdown box for the Transporter Name property, select 'Vehicle1'.

Creating Processes

- Select 'Vehicle1' and expand the *Add-on Process Triggers* properties. Double click on *Unloaded* to create an 'Vehicle1_Unloaded' Add-On Process:
- Place an Assign step and rename it 'IncreaseCapacity'.
- Set *State Variable Name* to 'Vehicle1CurrentCarryingCapacity' and *New Value* to 'Vehicle1CurrentCarryingCapacity + ModelEntity.TransportSize'.
- Navigate back to the Facility view, select 'Vehicle1', and double click on *Loaded* to create an 'Vehicle1_Loaded' Add-On Process:
- Place an Assign step and rename it 'ReduceCapacity'.
- Set *State Variable Name* to 'Vehicle1CurrentCarryingCapacity' and *New Value* to 'Vehicle1CurrentCarryingCapacity - ModelEntity.TransportSize'.
- Navigate back to the Facility view, select 'Vehicle1', and under double click on *Evaluating Transport Request* to create an 'Vehicle1_EvaluatingTransportRequest' Add-On Process:
- Place a Decide step and name it 'FIFO'.
- Set *Condition Or Probability* to 'Is.ModelEntity & & ModelEntity.CurrentStation.Contents.IndexOfItem(ModelEntity)==1 || Vehicle.CurrentNode== ModelEntity.CurrentNode'.
- Down the True path of 'FIFO' place a Decide step and rename it 'EnoughSpace?'.
- Set *Condition Or Probability* to 'Vehicle1CurrentCarryingCapacity >= ModelEntity.TransportSize'.
- Down the False path of 'Enough Space?', place an Assign step and rename it 'Reject Request'.
- Set *State Variable Name* to 'Token.ReturnValue' and leave *New Value* set to 'False'.
- Connect the False path of 'FIFO' into 'Reject Request'.

Enhancements

Animation can be added to show the current open carrying capacity of the vehicle and the amount of ride space each entity will consume.

- While in the Facility view, navigate to the Animation Ribbon under Facility Tools and select Status Label.
- Place a label above the path and set the *Expression* to 'Vehicle1CarryingCapacity'.
- Select Status Label again and change the *Expression* to ' "Available Carrying Capacity" '.
- Select 'Part', and expand the *Animation* property settings.

- Set *Dynamic Label Text* to 'ModelEntity.TransportSize'.

Model 2: Transporting Varying Sized Entities Custom Vehicle

Problem:

I want to model parts that require different amounts of my transporter's carrying capacity based on their entity type, using a custom object.

Categories:

Building New Objects / Hierarchy, Custom Object, Vehicles

Keywords:

Subclass, Override, Decide Step, Assign Step, Integer State Variables, Ride On Transporter, Status Label, Dynamic Label Text

Technical Approach:

Instead of using a standard Vehicle object and add-on processes, a custom Vehicle is created and enhanced to incorporate the carrying capacity process logic.

Details for Building the Model:

Creating a Sub-Classed Object MyVehicle

- From the Standard Library, right-click on Vehicle and select Subclass. This creates and navigates to a newly created 'MyVehicle'.
- Navigate to the Definition Tab of MyVehicle and select States. Add an Integer State and rename it 'CurrentCarryingCapacity'.
- Select Properties and find the 'InitialRideCapacity' property.
- Set the *Default Value* to 'MyVehicle.CurrentCarryingCapacity'.
- Navigate to the Processes Tab and find select the 'OnEvaluatingRiderAtPickup' process.
- Click Override on the Process Ribbon under Process Tools.
- Place a Decide step before the 'EvaluatingTransportRequestAddOnProcess' Execute step. Rename it 'Enough Space?' and change the *Color* to 'Yellow'.
- Set the *Condition or Probability* to 'CurrentCarryingCapacity >= ModelEntity.TransportSize'
- Place an Assign Step down the False path of 'Enough Space?'. Rename it 'RejectRequest' and change the *Color* to 'Yellow'.
- Set *State Variable Name* to 'Token.ReturnValue' and leave *New Value* set to 'False'.
- Select 'OnEvaluatingRiderReservation' process and click Override on the Process Ribbon under Process Tools.
- Place a Decide step ahead of the 'IfAcceptTransportRequest' Decide step. Rename it 'FIFO' and change the *Color* to 'Yellow'.
- Set *Condition Or Probability* to 'Is.ModelEntity & & ModelEntity.CurrentStation.Contents.IndexOfItem(ModelEntity)==1 || Vehicle.CurrentNode== ModelEntity.CurrentNode'.
- Connect the False path of 'FIFO' to the 'RejectTransportRequest' Assign Step.
- Select 'OnRiderLoading' process and click Override on the Process Ribbon under Process Tools.
- Place an Assign Step after the 'OnRiderLoadingAssignments' Assign Step. Rename it 'ReduceCarryingCapacity' and change the *Color* to 'Yellow'.
- Set *State Variable Name* to 'CurrentCarryingCapacity' and *New Value* to 'CurrentCarryingCapacity - ModelEntity.TransportSize'.
- Select 'OnRiderUnloading' process and click Override on the Process Ribbon under Process Tools.

-
- Place an Assign Step before the 'IfVehicleKeptReserved' Decide Step. Rename it 'IncreaseCarryingCapacity' and change the *Color* to 'Yellow'.
 - Set *State Variable Name* to 'CurrentCarryingCapacity' and *New Value* to 'CurrentCarryingCapacity + ModelEntity.TransportSize'.

Changes to Base Model

- Navigate to the Project Library by selecting the 'TransportingVaryingSizedEntities' Folder in the Navigation Window.
- Select the 'TransportingVaryingSizedEntities' then create a duplicate model by copying and pasting. Rename the new model 'TransportingVaryingSizedEntities_CustomVehicle'.
- Navigate to 'TransportingVaryingSizedEntities_CustomVehicle' by selecting it in the Navigation window.
- Navigate to the Definitions tab and select States.
- Delete the state named 'Vehicle1CurrentCarryCapacity'.
- Navigate to the Processes Tab and delete all three *Vehicle1 Add-On Processes*.
- Navigate to the Facility Tab and delete 'Vehicle1'.
- From the Project Library, place a 'MyVehicle' object instance next to 'Part'.
- Set *Initial Desired Speed* to by '0.75' Meters per Second.
- Select 'Output@Source1' node and under the *Transport Logic* properties set the *Ride On Transport Request* to 'True'.
- Using the dropdown box for the *Transporter Name* property, select 'MyVehicle1'.

Enhancements

Animation can be added to show the current open carrying capacity of the custom vehicle.

- While in the Facility view, select 'MyVehicle1' and navigate to the Animation Ribbon under Facility Tools.
- Select Status Label and place the label below 'MyVehicle1' and set the *Expression* to 'CurrentCarryingCapacity'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TravelWithSteeringBehavior - SimBit

Problem:

Demonstrate more flexible modeling of entity movement in free space. This uses Travel step to illustrate various options. These features are generally used when you want to show more realistic movement of pedestrians and people. These concepts are often used in agent-based modeling.

Categories:

People, Add-on Process Logic, Decision Logic -- Paths

Key Concepts:

SteeringRule, Pedestrian, People, Agent, Transfer Step

Assumptions:

We are modeling with the Standard Library but would like to use process logic to attain additional realism in modeling people movement.

Technical Approach:

The *Steering Behavior* options on the Travel step only work in Free Space. We will supplement the Standard Library objects with process logic that will Transfer the entity into free space, initiate realistic movement with the Travel step, then Transfer the entity onto its target destination. We will repeat this process with three different configurations to illustrate optional ways of using the Travel Step:

- A - Using a simple path (a line) and specify on the Travel step the width of the travel path (e.g. how far from the line) in which entity can travel.
- B - Follow along a network (but still in free space) and use the defined width of each path to constrain the entity travel.
- C - Enhance option B using a rule to have entities avoid colliding with each other.

Details for Building the Model:

Option A Setup

- Place a Source (SourceA), two BasicNodes (BasicNodeA1 and BasicNodeA2), and a Sink (SinkA) in the Facility window. Connect with Paths.
- Place a ModelEntity object (DefaultEntity) and change its *Initial Network* to 'No Network (Free Space)' so that all movements will be made in Free Space, instead of on the network.
- Change *Interarrival Time* of SourceA to 'Random.Exponential(.03)'.
- In the Processes Window, Create Process and name 'OutputSourceA_Exited'.
- Add a Transfer Step with *From* set to 'CurrentNode' and *To* set to 'FreeSpace'.
- On the OK branch add a Travel step with *Steering Behavior* set to 'Follow Network Path'. This indicates that even though the entity is now in free space (not on the network), we want to use the drawn network paths to guide its movement.
- Also on the Travel step:
 - Set *Starting Node* to 'Output@SourceA' to tell the entity where on the network to start.
 - Set *Ending Node* to 'Input@SinkA' to tell the entity where on the network to end.
 - Set *Path Width* to '4' meters to let the entity travel 2 meters to either side of the line.
- Add a second Transfer Step to the end of the process with *From* set to 'FreeSpace', *To* set to 'Node', and *Node Name* set to 'Input@SinkA'.
- Moving back to the Facility window, click on the output node of the source (Output@SourceA) and in the Add-On Process Triggers Category set *Exited* to 'Output_SourceA_Exited'.

At this point you should be able to run the model. You will see that instead of following the line exactly that entities will stray within the 4 meters surrounding the line. But it will still get its direction from the network almost as though it is really on the network. But it is not on the network (it is in free space) so it will not respect any of the options specified on the paths.

Option B Setup

Execute all of the steps in Option A with the following changes:

- Wherever an 'A' appears in a name make it a 'B'.
- For the three paths, change the path *Width* (under the General category > Physical Characteristics > Size) to '6', '3', and '2' respectively. You can also set the color of the Paths using the Color button on the Drawing ribbon.
- On the Travel step:
 - Leave *Path Width* set to its default of 'Candidate.Link.Size.Width' which lets the entity react to the width of the path it is traveling on.
 - Set *Update Time Interval* to '0.3' to indicate your sensitivity to it straying off the path. You can experiment with this number. A smaller number will provide better animation performance, but it will do so at the cost of execution speed.

Option C Setup

Execute all of the steps in Option B with the following changes:

- Wherever an 'B' appears in a name make it a 'C'.
- On the Travel step:
 - Set *Avoid Collisions* to 'True' to indicate that you want the entities to try to avoid colliding with each other.

You should now be able to run the model and see how the three sets of options compare to each other. You can adjust the various parameters to customize the performance.

Embellishments:

The Steering Rules for the Travel step are implemented in user code. More advanced users can use the example C# code found in the Examples\UserExtensions folder to implement their own, more complex or customized rules. An alternate approach to using the Travel step in processes in the way illustrated above is to instead customize the process logic included in the ModelEntity definition.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TurnaroundMethod - SimBit

Problem:

I would like to explore the different options that are available for an entity turnaround method on a network.

Categories:

Entity Characteristics

Key Concepts:

Deadlock, Network Turnaround Method

Technical Approach:

We set up three options each with a source, three bidirectional paths and 2 transfer nodes.

Details for Building the Model:

Simple System Setup

- For each option, place a Source and two TransferNodes in parallel. Change the *Maximum Arrivals* property on each Source to '1'. This allows you to see the movement of a single entity at a node as it transfers between links.
- Use three bidirectional paths to link each node to the Source and to link the two nodes.
- Place three ModelEntities (Exit_Entity, Rotate_Entity & Reverse Entity) and assign each one to a Source by setting the *Entity Type* in the Source Arrival Logic to the corresponding entity name.
- The Exit_Entity *Network Turnaround Method* will keep its default value, 'Exit & Re-enter'.
- Set the Rotate_Entity *Network Turnaround Method* to 'Rotate In Place'.
- Set the Reverse_Entity *Network Turnaround Method* to 'Reverse'.

Discussion:Exit Entity:

This entity will 'Exit & Re-enter' a node when it is changing directions on a bidirectional link. This is the default setting for entities, workers and vehicles. The entity actually exits the link, and re-enters it as it turns around. This method is the most deadlock avoidance friendly option.

Rotate Entity:

This entity will 'Rotate in Place' at a node when it is changing directions on a bidirectional link. The entity never exits the link, but simply rotates and continues on the same link in the opposite direction. This method could cause deadlocks.

Reverse Entity:

This entity will 'Reverse' its animation symbol at a node when it is changing directions on a bidirectional link. The entity never actually leaves the link when changing directions, it simply goes in a reverse direction (tail first). This allows for modeling such things as elevators.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Understanding Run Performance with Dashboards

- SimBit

Understanding Run Performance with Dashboards

This SimBit project includes three models providing examples of using a Search step with corresponding Dashboards displaying the run time of each model. All three models show the same objective with different approaches and how each approach affects the run time. The base of these models is the same as the final three models of the EfficientSearchStepPractices SimBit.

1. **SearchStep2_MinimizeExpressionDashboard:** Demonstrates using the Search step to minimize an expression by looking through the entire Data Table with Dashboard.
2. **SearchStep2_ForwardSearchDashboard:** Demonstrates using the Search step to look through a Data Table in the forward direction so the whole Data Table is not searched with Dashboard.
3. **SearchStep2_ForwardSearchWithIndexDashboard:** Demonstrates using a Search step to look through a Data Table starting at a specific index and moving forward so a smaller section of the Data Table is searched with Dashboard.

NOTE: Dashboards are only available in RPS, Professional, and Academic editions of Simio.

Model 1: SearchStep2_MinimizeExpressionDashboard

Problem:

Shipments arrive with different traits. An arrival schedule is provided which details when a certain shipment is expected. A data table holds the arrival schedule information and traits of the expected shipment. I want to look through the data table to find when the next purple or orange color shipment is arriving, so I can prepare for that shipment type.

Categories:

Add-On Process Logic, Dashboard, Data Table, Timer Element

Keywords:

Dashboard, Data Table, Event, Output Table, Process Logic

Assumptions:

The shipment will arrive based on the Arrival Table which specifies the shipment's traits.

Technical Approach:

On the shipment entity's arrival, a process will be executed with a Search step that looks through the Arrival Table to find the next shipment that has either a purple or orange color trait. The information on the next shipment of this type will be displayed in the Facility.

Details for Building the Model:

Initial Setup

- Begin with the model "SearchStep2_MinimizeExpression" from the EfficientSearchStepPractices SimBit.

Definitions Window: Elements

- In the Elements View of the Definitions Window, create a Timer called "EveryHourOnTheHour".
- *Time Offset* = '3600 - (DateTime.Second(Run.TimeNow) + (DateTime.Minute(Run.TimeNow) * 60))'
- *Units* = 'Seconds'

Definitions Window: States

- Add DateTime Variable named "PrevDateTime" to the model. Keep all default values.
- Add DateTime Variable named "CurrentDateTime" to the model. Keep all default values.

Data Window: Tables

- In the Schema ribbon, add an Output Table which will store information during model run about the real time spent on every hour of simulation time.
- RunInfo
- One Date Time State Variable column
- *Name* = 'TimeNowDateTime'
- Two Real State Variable columns
- *Name* = 'TimeNow'
- *Name* = 'RealTimeDuration'
- *Unit Type* = 'Time'
- *Units* = 'Seconds'

Processes Window: Process Logic

- Select the OnRunInitiated Process to create a Process that happens when the run initializes.
- Create an Assign step "PreviousDateTime" to assign the system current date time to PrevDateTime :
- *State Variable Name* = 'PrevDateTime'
- *New Value* = 'DateTime.SystemNow'
- Create a new Process "EveryHourOnTheHourProcess" that will run every time the EveryHourOnTheHour Timer goes off.
- *Triggering Event Name* = 'EveryHourOnTheHour.Event'
- *Category* = 'Timers'
- On the "EveryHourOnTheHourProcess" add the following steps:
- Assign step "CurrentDateTime" to store the current date time into CurrentDateTime:
- *State Variable Name* = 'CurrentDateTime'
- *New Value* = 'DateTime.SystemNow'
- After the Assign step, add an Add Row step "RunInfo" to add a new row to the RunInfo Table:
- *Table Name* = 'RunInfo'
- *Object Type* = 'Token'
- After the Add Row step, add an Assign step "OutputTableAssignments"
- *State Variable* = 'RunInfo.TimeNowDateTime'
- *New Value* = 'TimeNow'
- *State Variable* = 'RunInfo.TimeNow'
- *New Value* = 'TimeNow'
- *State Variable* = 'RunInfo.RealTimeDuration'
- *New Value* = 'CurrentDateTime – PrevDateTime'

- *Units* = 'Seconds'
- After the "OutputTableAssignments" Assign step, add another Assign step "PrevDateTime":
- *State Variable* = 'PrevDateTime'
- *New Value* = 'CurrentDateTime'

Results Window: Dashboard Reports

- Use the Export button on the Dashboards ribbon to export the Dashboard "PerformanceOverSimulationTime" from the SimBit and save it in a convenient location. This will be saved as an .xml file.
- Import Dashboard "PerformanceOverSimulationTime.xml" to see the model performance over simulation time.

Facility Window: Run

- In the Unit Settings, change Time Units to Seconds for optimal viewing of the Dashboard.

Discussion:

Run this model with the Fast-Forward option. Note how long in seconds the model takes to run.

Navigate to the "PerformanceOverSimTime" Dashboard via the Dashboard Reports View of the Results Window. The "TotalDuration" bar chart on top shows the total amount of real time that the simulation took to complete. What appears to be a line graph is a Range Filter that displays the length of real time the simulation takes at a given point in the simulation. The ends of the Range Filter can be dragged to display certain hours of simulation time on the entire dashboard. The "Second" bar chart displays the sum of seconds in real time over a given hour of simulation time.

Notice that the "Second" bar chart has a somewhat consistent flat trend. This trend makes sense as the Search step used in this model with the *Search Expression* as 'Minimize Expression' has a consistent number of rows to search through. While it would be expected that the chart would be flat, there is some slight variation may exist which could be due to randomness in the model.

Notes:

During the run, the Output Table collects the real system time the model took to run an hour of the simulation time. If you wish to collect a different interval of simulation time, you can change how frequently the Timer is triggered.

Model 2: SearchStep2_ForwardSearchDashboard

Problem:

Shipments arrive with different traits. An arrival schedule is provided which details when a certain shipment is expected. A data table holds the arrival schedule information and traits of the expected shipment. I want to look through the data table to find when the next purple or orange color shipment is arriving, so I can prepare for that shipment type. I do not want to have to search through the whole table to find this information.

Categories:

Add-On Process Logic, Data Tables

Keywords:

Dashboard, Data Table, Event, Output Table, Process Logic

Assumptions:

The shipment will arrive based on the Arrival Table which specifies the shipment's traits. The Arrival Table will be in chronological order.

Technical Approach:

On the shipment entity's arrival, a process will be executed with a Search step that looks through the Arrival Table to find the next shipment that has either a purple or orange color trait. The Search step will look forwards through the Data Table and will stop when it finds a match and meets its limit of number of items to find.

Details for Building the Model:

Initial Setup

-
- Apply all the changes made to SearchStep2_MinimizeExpressionDashbaord to the SearchStep2_ForwardSearch model.

Discussion:

Run this model with the Fast-Forward option. Note how long in seconds the model takes to run.

Go to the "PerformanceOverSimTime" Dashboard to see the real-time performance of the model during its run. Notice that the "Second" bar chart has a somewhat consistent upward trend. This makes sense as the *Search Expression* is 'Forward' which searches every record from the first until its goal is achieved. As the Arrival Table is sorted by Arrival Time, it takes less time to search early in the simulation, but it must evaluate every row on the way to the goal row as the simulation runs.

Model 3: SearchStep2_ForwardSearchWithIndexDashboards**Problem:**

Shipments arrive with different traits. An arrival schedule is provided which details when a certain shipment is expected. A data table holds the arrival schedule information and traits of the expected shipment. I want to look through the data table to find when the next purple or orange color shipment is arriving so I can prepare for that shipment type. I do not want to have to search through the whole table to find this information.

Categories:

Add-On Process Logic, Data Tables

Keywords:

Dashboard, Data Table, Event, Output Table, Process Logic

Assumptions:

The shipment will arrive based on the Arrival Table which specifies the shipment's traits. The Arrival Table will be in chronological order.

Technical Approach:

On the shipment entity's arrival, a process will be executed with a Search step that looks through the Arrival Table to find the next shipment that has either a purple or orange color trait. The Search step will look forward through the Data Table starting at a specific index and will stop when it finds a match and meets its limit of number of items to find.

Details for Building the Model:

- Apply all the changes made to SearchStep2_MinimizeExpressionDashboard to the SearchStep2_ForwardSearchWithIndex model.

Discussion:

Run this model with the Fast-Forward option. Note how long in seconds the model takes to run.

Go to the "PerformanceOverSimTime" Dashboard to see the real-time performance of the model during its run. Notice that the "Second" bar chart is consistent and low throughout the entire simulation. This makes sense as the *Search Expression* is 'Forward' with a *Starting Index*. With the addition of the index, the forward search is much more efficient as the search starts at the row of the current arrival.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UpdateStateInModelFromObject - SimBit

Problem:

I have a user-created object in my model and want to update a state variable in the main model from within that object. **
NOTE: This is an extension of the SimBit 'ProcessModelWithinModel.spfx'.

Categories:

Building New Objects / Hierarchy

Key Concepts:

BasicNode, ContinueProcess, Delay Step, EndTransfer Step, Expression Property, ExternalNode, Object Instance Property, Release Step, Resource, Seize Step, Station Element, Submodel, Transfer Step, State Property

Assumptions:

The SimBit 'ProcessModelWithinModel.spfx' is used as a baseline for this example. The steps show how to add a state property reference to an object so that a state in the main model can be appropriately updated.

Technical Approach:

A state type property will be utilized within the user-created object. Once the object is placed in a model, the property will then reference the state defined within the model.

Details for Building the Model:

ProcessModelWithinModel.spfx

- Open the SimBit 'ProcessModelWithinModel.spfx' and review the corresponding *.pdf file for logic details.

Defining a State-Type Property

- Within the Navigation bar, click on the SeizeDelayRelease object.
- Go to the Definitions window, Properties panel to add a new property to the object. This is done by selecting Standard Property and then clicking on State within the pull-down list.
- Rename the state 'WhichState' and change the *Category Name* to 'Properties' so that it appears in the same area with the ResourceName and DelayTime properties for the object.

Assigning the State within the SeizeDelayRelease Object

- Click on the Processes window of the SeizeDelayRelease object. After the Seize step, add an Assign step.
- Within the Assign step, right click on the *State Variable Name* property, select 'Set Referenced Property' and click on 'WhichState'. This will cause 'WhichState' to be placed in the property field with a green arrow, indicating a reference from an object property.
- Within the *New Value* field, enter the value 'WhichState + 1'. This will cause each token going through this process to increase the associated state by 1.

Defining the State within the Model

- Within the Navigation window, click on the Model and go to the Definitions tab, States panel.
- Add a new Integer type State and change the *Name* to 'Model_Count'.
- Within the Facility window of the model, add an animated status label of this state by going to the Animation ribbon, selecting and placing a Status Label and entering the *Expression* 'Model_Count'.

Defining Which State to Reference within the Object

- Click on the SeizeDelayRelease object within the Model's Facility Window.
- Notice in the Properties window, under the category Properties, there is a new property called *WhichState*, which you will specify as 'Model_Count'.

Send comments on this topic to [Support](#)

UserDefinedListState - SimBit

Problem:

I would like to create my own, custom List State that will produce time persistent statistics on each value that is assigned to the state.

Categories:

Buffering, Custom Statistics, Discrete States

Key Concepts:

Before Exiting, Buffer Capacity, List State, Math.If(), On Entering, State Assignments, Status Pie, StringList

Technical Approach:

We set up a system that has two servers in series (Server1 and Server2), which are in parallel with two other servers (Server3 and Server4) in series. We would like statistics on when either Server1 or Server2 are busy and statistics on when either Server3 or Server4 is busy. In other words, we'd like to know when the top line of servers is busy and when the bottom line of servers is busy. By creating a custom ListState for Line1 (which includes Server1 and Server2) and a custom ListState for Line2 (which includes Server3 and Server4), we will get statistics such as the average amount of time spent in the Busy state and the number of occurrences that each Line was in the Busy state.

Details for Building the Model:

Simple System Setup

- Place a Source and a Sink object in the Facility window
- In between these two objects, place 4 Server objects. Set the *Input Buffer Capacity* and the *Output Buffer Capacity* to '0' for all 4 servers so that we do not have any buffering in the system. Server1 and Server2 are in series so therefore connect the Source to Server1 with a Path, connect Server1 to Server2 with a Connector and connect Server2 to the Sink with a Path.
- Server3 and Server4 are in series so therefore connect the Source to Server3 with a Path, connect Server3 to Server4 with a Connector and connect Server4 to the Sink with a Path.

Create the two List States:

- From the Definitions window, go to the States panel. Create new List State by clicking on 'List' in the States ribbon. Name this new state 'Line1ListState'. Create another new ListState and name is 'Line2ListState'.
- Leave the States Panel and go to the Lists panel (found on the left side of the interface). Create a new String List by clicking on 'String' in the ribbon menu. Name this new List 'Line1'. Enter the word 'Idle' into the first row of the List and the word 'Busy' into the second row of the List.
- Create another String List by clicking on 'String' in the ribbon menu. Name this new List, 'Line2'. Enter the word 'Idle' into the first row of the List and the word 'Busy' into the second row of the List.
- Go back to the States panel where you created the List States and select Line1ListState. In the properties window of this State, set the *String List Name* property to 'Line1' and set the *Initial State Value* property to 'Idle'.
- Select Line2ListState. In the properties window of this State, set the *String List Name* property to 'Line2' and set the *Initial State Value* property to 'Idle'.

Assign Values to List States:

- Go back to the Facility window where we'll assign values to our new List States. Select the first Server in the top series (Server1) and expand the State Assignments property category in the properties window. Click on the ellipse (the ... button) in the *On Entering* property and when the On Entering Repeating Property Editor window appears, click Add. Set the *State Variable Name* to 'Line1ListState' and the *New Value* to '1'. Assigning a value of 1 to this List State is equivalent to setting it to the list value of 'Busy' (the List is 0-indexed so Idle =0 and Busy = 1).
- Click on the second Server in the top series (Server2) and expand the State Assignments property category in the properties window. Click on the ellipse (the ... button) in the *On Exiting* property and when the On Entering Repeating Property Editor window appears, click Add. Set the *State Variable Name* to 'Line1ListState' and the *New Value* to 'Math.If(Server1.Processing.Contents>0, 1, 0)'. This Math.If statement says that if there is an entity being processed at Server1, then keep the value of the Line1ListState at '1'. If not, then set it to '0', which is Idle.
- Select the first Server in the bottom series (Server3) and expand the State Assignments property category in the

properties window. Click on the ellipse (the ... button) in the *On Entering* property and when the On Entering Repeating Property Editor window appears, click Add. Set the *State Variable Name* to 'Line2ListState' and the *New Value* to '1'.

- Click on the second Server in the bottom series (Server4) and expand the State Assignments property category in the properties window. Click on the ellipse (the ... button) in the *On Exiting* property and when the On Entering Repeating Property Editor window appears, click Add. Set the *State Variable Name* to 'Line2ListState' and the *New Value* to 'Math.If(Server2.Processing.Contents>0, 1, 0)'. This Math.If statement says that if there is an entity being processed at Server2, then keep the value of the Line2ListState at '1'. If not, then set it to '0', which is Idle.
- Place two Status Pie charts from the Animation ribbon into the Facility window. Set the *Data Type* property of the Status Pie to 'List State' and the *List State* property to 'Line1ListState' (and 'Line2ListState' for the second chart).

Embellishments:

Consider adding additional values to each List and then assigning the appropriate index into the ListState to see that a ListState can contain more than just the two values (0 and 1) that this SimBit demonstrated.

Discussion:

The statistics for these new ListStates are displayed in the Results window after the run is complete. For each ListState, there are statistics for Average Time, Total Time, Percent Time and Number of Occurrences for each State Value (i.e. Busy, Idle in our example).

Most of the Simio Standard Library objects contain a built in ListState, called ResourceState, which is automatically updated with values such as "Busy", "Idle", "OffShift", etc.

There are functions available for a ListState, such as `MyListState.AverageTime(stateValue)`, `MyListState.TotalTime(stateValue)`, `MyListState.PercentTime(stateValue)`, `MyListState.NumberOccurrences(stateValue)`

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UsingAddRowandOutputTable - SimBit

Problem:

I want to be able to write entity specific output information to an output data table at the end of the simulation run. **Simio Professional and RPS Editions only**

Categories:

Data Tables

Key Concepts:

Add-On Process, Assign Step, AddRow Step, Data Table, Output Table, Real State Variable, Run.TimeNow, State Assignments, String State Variable, Maximum Arrivals

Assumptions:

All data is added to the output table when the entity is about to leave the system. Both functions and state variable information is used to write the appropriate state information to the output table.

Technical Approach:

An output data table is utilized to write out information regarding an entity to a table at the end of the simulation run. The AddRow step is used to add a new row to the table with each entity leaving the system, while the Assign step will assign values to the various columns in the table for that given entity instance.

Details for Building the Model:

Simple System Setup

- Place a Source, two Servers and a Sink in the Facility window. Use paths to connect the Source to each of the Servers, and each of the Servers to the Sink.
- Within the Source, open the Stopping Conditions section and change the *Maximum Arrivals* property to '20'.
- Within the Run Setup section of the Run ribbon, change the *Ending Type* to 'Unspecified (Infinite)' so the simulation will stop when there are no more entities in the system.

Defining an Entity State

- Within the Navigation window, click on the ModelEntity. Go to the Definitions window, States panel and add a String state variable with the *Name* 'WhichServer'. This will be used to store (with the entity) which of the two servers it utilized. This state can be referenced as ModelEntity.WhichServer.
- Within the Navigation window, click on the Model and place a ModelEntity from the Project Library into the Facility window.

Defining the Output Table

- Still within the Model, open the Data window, select the Tables panel and add an Output Table.
- Still within the Data window, click on the States ribbon to add several states to the output table. First, add a Real state variable and change the *Name* to 'TimeEntered'. This will store the simulation time that the entity entered the system.
- Then, add a String state variable with the *Name* 'EntityName'. This will store the entity name, such as DefaultEntity.94.
- The next column added should be another Real state variable with the *Name* 'TimeLeaving' that will store the simulation time that the entity finished in the system.
- Finally, add another String state variable with the *Name* 'ServerName' that will indicate which of the two Servers through which the entity was processed.

Adding a Row to an Output Table During the Run

- Because output tables are tables with no data at the start of the simulation run, data must be added during the run through the use of object states. To add a row to an output table, the AddRow step is used.
- Within the Navigation window, go back to the Model. In the Facility window, within the input node of the Sink (Input@Sink1), double click directly on the *Entered* add on process trigger so that it automatically creates a process

within the Processes window named 'Input_Sink1_Entered'. This should reference the process from within the input node, as well as place you within the Processes window to begin defining the process.

- Within the process, add an AddRow step with a *Table Name* of 'OutputTable1'. Each time an entity goes through the Sink, a new row will be added to the output table.
- Next, add an Assign step and make the following assignments to the states in the table:
 - *State Variable Name* – 'OutputTable1.TimeEntered', *New Value* – 'ModelEntity.TimeCreated'
 - *State Variable Name* – 'OutputTable1.EntityName', *New Value* – 'ModelEntity.Name'
 - *State Variable Name* – 'OutputTable1.TimeLeaving', *New Value* – 'Run.TimeNow'
 - *State Variable Name* – 'OutputTable1.ServerName', *New Value* – 'ModelEntity.WhichServer'

Assigning the ModelEntity.WhichServer Value

- Within Server1, under the State Assignments section of properties, click on *On Entering* to enter an assignment through the repeating property editor. Assign the *State Variable Name* 'ModelEntity.WhichServer' to the *New Value* of 'Server1.Name'.
- Within Server2, do the same thing, except assign the *State Variable Name* 'ModelEntity.WhichServer' to the *New Value* of 'Server2.Name'.

Running the Model

- You will notice that the output table, while it has column headers, does not include any data. Once the simulation has been run, the output table will be filled with data upon clicking the Stop button.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UsingaMonitor - SimBit

Problem:

I have a simple model with a source, server and sink and would like to know at what point in time during the simulation run the queue for the server exceeds 9 for the first time.

Categories:

Add-On Process Logic, Custom Statistics

Key Concepts:

Add-On Process, Contents, CrossingStateChange, Event, InputBuffer, Monitor, On Entered, Process Triggering Event, Real State Variable, Stopping Condition, TimeNow

Assumptions:

Since the time is the only information desired, the simulation stops running after the queue reaches 9 for the first time.

Technical Approach:

A state variable is updated with the number of entities in the server queue each time a new entity enters the server. A Monitor element is used to track this state variable and trigger an event when the state variable goes over the value of 9. The triggering of this event executes a process that updates another variable with the current simulation time and the simulation stops. The variable with the simulation time is displayed in the Results Window as a State Statistic.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink in the Facility Window. Update the *Processing Time* property of the Server to be 'Random.Triangular(.2,.3,.5)' so that the Server has some queuing.
- Connect the Source and Server, as well as the Server and Sink with Paths.

Creating Discrete State Variables

- Click on the Definitions tab and select the States panel. Create two discrete state variables, one called 'NumberInQueue' and the other called 'TimeWhenOver9'.

Using an Add-On Process Trigger

- Within the Server object, create an Add-On Process by double-clicking on *Entered*. This will take you to the Processes Window. In this process, add an Assign step that assigns the *State Variable Name* 'NumberInQueue' to the *New Value* of 'Server1.InputBuffer.Contents'. (Note: this is a function that returns the number in the inputbuffer queue for this Server).

Creating and Using a Monitor

- Select the Elements Panel within the Definitions tab to add a Monitor element with its *Name* as 'Monitor_Over9'. The *Monitor Type* property should be changed to 'CrossingStateChange'. The *State Variable Name* that it is monitoring should be set to 'NumberInQueue' and the *Threshold Value* is '9'.

Adding a New Process

- Create a new process using the Create Process icon in Processes Window. Set the *Triggering Event* property for the process to the name of the Monitor you just created, followed by .Event, 'Monitor_Over9.Event'.
- Place an Assign step in this new process and have that Assign step update the *State Variable Name* 'TimeWhenOver9' to the current simulation time, using with the function 'Run.TimeNow' as the *New Value*.

Stopping the Simulation Run

- Click on the Definitions Window and select the Elements panel to create a new State Statistic.
- Set the *State Variable Name* to 'TimeWhenOver9' and the *Stopping Condition* to 'TimeWhenOver9 > 0'. This will stop the simulation run once this threshold is met for the first time.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UsingAStorageQueue - SimBit

Problem:

I would like entities to wait in a queue and then I'd like to search the queue for a particular entity and remove it from the queue.

Categories:

Add-On Process Logic, Combining and Separating Entities

Key Concepts:

Add-On Process, Allow Passing, Assign Step, Batch Step, BatchLogic Element, Candidate, Decide Step, Detached Queue, Entities Per Arrival, Insert Step, ModelEntity, Movement.X, Node, NodeList, On Entered, AssociatedStationLoad, Random Symbol, Real State Variable, Remove Step, Route Step, RoutingGroup Element, Search Step, Selection Goal, SetNode Step, SmallestValue, Storage Element, Storage Queue, Time Offset, TimeCreated, UnBatch Step

Technical Approach:

Entities arrive in pairs by the Source creating two entities per arrival and batching them together. When the pair reaches the first node, they are un-batched. One entity is routed to an available Server and the other entity is put into a Storage queue, where it sits until its partner has finished processing. In order to match up the entities again, we search the Storage queue for an entity that has the exact same "TimeCreated" stamp on it, which indicates that these two entities were originally paired together upon creation. Once the correct entity is found in the queue, it is removed and batched with its partner, before traveling to the Sink.

Details for Building the Model:

Simple System Setup

- Place a Source, a Sink and one TransferNode into the Facility window. Place the TransferNode so that it is in between the Source and the Sink. Connect the Source to the TransferNode with a Path and connect the TransferNode to the Sink with a Path.
- Click on the Source, and set the Time Offset property to '.1' and set the Entities Per Arrival to '2'.
- In both Paths, set the *Allow Passing* property to "False" so that we are able to see the entities traveling one by one on the paths.
- Place three Servers into the Facility window. Set the *Initial Capacity* property of each Server to '2'.
- Draw a Path from the TransferNode to the Input Node of each Server. Draw a Path from the Output Node of each Server back to the TransferNode – for a total of 6 new paths.

Animating the Entity as People:

- We animate the entities as different people so it is easier to see the different partners splitting and pairing back together. Drag the ModelEntity object from the Project Library on the bottom left into the Facility window.
- Select the new DefaultEntity within the window. Select a symbol of a person from the Symbols dropdown in the Ribbon. The entity should change from a green triangle to the person that was selected.
- With the DefaultEntity selected, click on the Add Additional Symbol button in the Ribbon. The Active Symbol button in the Ribbon should now show (2 of 2) because there are now two symbols associated with DefaultEntity. You can change the hair color, skin color or clothing color of the person symbol by clicking on Color in the Ribbon and then clicking on the specific part of the person symbol.
- Keep adding additional symbols and altering their colors, in order to provide variety to the entities traveling in the model.
- With the DefaultEntity selected, change the *Random Symbol* property, in the properties window, to 'True'. This will tell Simio to randomly select a symbol from the multiple symbols you've created for DefaultEntity.

Create Elements – Storage, Batch Logic and Routing Group:

- From within the Definitions window, click on the Elements panel and add a Storage element by clicking on the Storage button in the Ribbon. Rename it 'WaitingArea'
- Add a Batch Logic element by clicking on the Batch Logic button in the Ribbon. The default property values are fine.
- Click on the Lists panel within the Definitions window and create a Node list by clicking on the Node button in the

Ribbon. Enter Input@Server1, Input@Server2, Input@Server3 as the nodes within the List. (In order to add a new row to the list, click outside of the list or hit Tab)

- Back in the Elements panel, add a Routing Group element by clicking on the Routing Group button in the Ribbon. In the *Node List Name* property, select the node list that you just created.

Create new State Variables:

- Add a State to the ModelEntity that will tracked when it has finished processing at a Server. Click on the ModelEntity within the Navigation window (top right). Once the ModelEntity is highlighted (you are within the ModelEntity model), click on the Definitions window. Click on the States panel and click on Discrete State in the ribbon, to add a new State to the ModelEntity. Rename it 'Processed'.
- Within the Facility window, within each Server, go to the State Assignments section of properties and add a new assignment *Before Exiting*. Within the repeating property editor, assign the *State Variable Name* 'ModelEntity.Processed' (created in above step) to a *New Value* of '1'. This state for the entity will later be evaluated once it enters the TransferNode1 to determine if the entity is entering or exiting.
- Add a State to the main Model that will help us keep track of which entity will be the Parent and which will be the Member. With the Model selected in the Navigation window, go to the Definitions window and click on the States panel. Add a new Discrete State named 'TrackBatching'. Set the *Initial State Value* property to '1'. Also, add a state named 'WhichOne' which will store the associated token found in the Search step so that we can Remove the correct entity from the Storage queue.

Add Processing Logic:

- From within the Processes window of the main Model, create a new process by clicking on the 'Create Process' button in the Ribbon.
 - Place a Decide Step. Set the *Decide Type* property to 'ConditionBased' and set the *Expression* to 'TrackBatching == 1'. This checks to see if this is the first created entity and if so, it will become the parent.
 - In the True segment leaving the Decide Step, place an Assign Step. This step should set the TrackBatching state to the value of '2'.
 - After this Assign Step, place a Batch Step. Use the BatchLogic element created earlier and set the *Category* property to 'Parent'.
 - In the False segment leaving the Decide Step, place a Batch Step. Use the BatchLogic element created earlier and set the *Category* property to 'Member'.
 - After this step, place an Assign Step. This step should set the TrackBatching state to the value of '1', which indicates that this batch is finished and a new batch will start.
- In the Facility window, click on the Output Node of the Source object and select this new Process from the drop down of the *Entered Add On Process* trigger. This will cause this new process to be triggered whenever an entity enters this output node.
- While in the Facility window, click on the TransferNode and select 'Create New' from the drop down of the *Entered Add On Process* trigger. This will create a new process that will be triggered whenever an entity enters this node.
- From within the Processes window, place a Decide Step within this new process. Set the *Decide Type* property to 'ConditionBased' and set the *Expression* to 'ModelEntity.Processed == 0'. This checks to see if the entity is just arriving to the node and it not yet processed or if the entity is leaving the system and just finished processing at a Server.
 - In the True segment leaving the Decide step, place a Route step. Set the *Routing Group Name* property to the Routing Group element that was created earlier. Set the *Selection Goal* to 'Smallest Value' and the *Selection Expression* to 'Candidate.Node.AssociatedStationLoad'. This step will set the destination node for the entity to one of the three possible Servers, based on the number of entities currently en-route and processing at the various stations (AssociatedStationLoad).
 - After the Route Step, place an UnBatch step (the defaults are fine).
 - In the Member segment leaving the UnBatch Step, place an Insert Step. Set the *Queue State Name* property to 'WaitingArea.Queue', the storage queue created earlier.
 - After the Insert Step, place an Assign Step. Set 'ModelEntity.Movement.X' to '1000'.
 - Back in the False segment of the first Decide Step, place a Search step. Set *Collection Type* to 'QueueState'. Set *Queue State Name* to 'WaitingArea.Queue'. And set *Match Condition* to 'ModelEntity.TimeCreated == Candidate.Entity.TimeCreated'. This searches the Waiting Area queue and finds the entity that was originally created at the same time as this entity. Click on the Advanced Options within the Search step and specify the *Save Index Found* property as 'WhichOne' to save which token in the storage was found in the Search.
 - In the Original segment leaving the Search step, place a Batch step. Use the BatchLogic element created earlier and set the *Category* property to 'Parent'.
 - After this step, place a SetNode step. Set *Destination Type* to 'Specific' and the *Node Name* to 'Input@Exit' (or Input@Sink). This will route entities that have already been processed, out of the system and ensure that they

do not follow a path back to the Servers.

- In the Found segment of the Search step, place a Remove step. Set the *Queue State Name* to 'WaitingArea.Queue'. Under Advanced Options, set the *Removal Type* to 'AtRankIndex' and *Rank Index* to 'WhichOne'. This will remove the partner entity from the queue (after it's been found by the Search step).
- In the Removed segment of the Remove step, place a Batch step. Use the BatchLogic element created earlier and set the *Category* property to 'Member'. This will batch up this removed entity as a member to its parent partner that searched for it.

Animate the Waiting Area Storage Queue:

And finally, animate the Storage queue within the Facility window. Click on the Detached Queue button from within the Animation ribbon. Draw the queue within the Facility window. Set the *Queue State* within the properties window to 'WaitingArea.Queue'.

See Also:

Help topics for [Storage](#), [Batch Logic](#), [Routing Group](#), [Insert Step](#), [Remove Step](#), [Search Step](#), [Batch Step](#), [Route Step](#), [SetNode step](#).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UsingButtonsToAffectSystem - SimBit

Problem:

I want to use external buttons to control the arrival rate of my system and to Stop/Start the conveyor within my system.

Categories:

Arrival Logic

Key Concepts:

Assign Step, Button, Conveyor, Decide Step, DesiredSpeed, Event, Math.Max(), Maximum, Process Triggering Event, Real State Variable

Assumptions:

The initial arrival rate is set to 0.5 (unit/minute), each time the arrival rate is increased/decreased, the rate changes (increases/decreases) 0.05 (unit/minute). The minimum arrival rate should be 0.05 (unit/minute), it means that when the arrival rate is 0.05 (unit/minute), it cannot be decreased further. There is no upper limit for increasing the arrival rate.

The initial speed of the conveyor is set to 0.5 (meters/sec). To start the conveyor again when it's stopped, the resume speed is set to be the speed just before it's stopped. We only control the Stop/Start for Conveyor1 which connects the Server to the Sink.

Technical Approach:

Four events will be created: StopConveyor, StartConveyor, IncreaseRate, DecreaseRate. Those four events will trigger four processes which will implement corresponding actions. Also, we create four buttons by using Animation tool, each one of these buttons is logically connected to an event which has the corresponding function with the Button.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink from the Standard Library into the Facility Window. Connect them with Conveyors. Conveyor2 connects Source and Server, Conveyor1 connects Server and Sink. Set the *Desired Speed* for both the Conveyors to .5.

Creating Discrete State Variables

- Open the Definitions Window and select the States panel, add two Discrete State Variables with Names 'Rate' and 'ResumeSpeed'.
- 'Rate' will track the arrival rate of the Source; its *Initial State Value* is '0.5'.
- 'ResumeSpeed' will save the speed of Conveyor1 before it's stopped, and resume it with the saved speed. Its *Initial State Value* is '0'.
- Set the *Interarrival Time* of the Source to '1/Rate'.

Creating Events

- Open the Definitions Window and select the Events panel.
- Create four events with names: StopConveyor, StartConveyor, IncreaseRate, DecreaseRate.

Generating Processes Corresponding to Events

- Open the Processes Window; add four processes by clicking on the *Create Processes* ribbon.
- Name the first process 'Decrease'. Then go to the Properties Window, set the *Triggering Event* to 'StartConveyor'. Add an Assign step to this process, within this Assign step, set the *State Variable Name* to 'Conveyor1.DesiredSpeed', the *New Value* is 'ResumeSpeed'.
- Name the second process 'Increase'. Then go to the Properties Window, set the *Triggering Event* to 'StopConveyor'. Add two Assign steps to this process. For the first Assign, set the *State Variable Name* to 'ResumeSpeed', the *New Value* is 'Conveyor1.DesiredSpeed'. This step will save the current speed of Conveyor1 to the state variable 'ResumeSpeed'. For the second Assign, set the *State Variable Name* to 'Conveyor1.DesiredSpeed', the *New Value* is '0'. This will stop Conveyor1.

-
- Name the third process 'RateDown'. Then go to the Properties Window, set the *Triggering Event* to 'DecreaseRate'. Add an Assign step to this process. For this Assign, set the *State Variable Name* to 'Rate', the *New Value* is 'Math.Max(.05, Rate-0.05)'. This will ensure that the arrival rate will not be decreased below 0.05 (unit/minute).
 - Name the fourth process 'RateUp'. Then go to the Properties Window, set the *Triggering Event* to 'IncreaseRate'. Add an Assign step to this process; then set the *State Variable Name* to 'Rate', the *New Value* is 'Rate+.05'.

Creating Buttons

- Click on *Animation* ribbon under the *Facility Tools*, choose *Button*, and create four Buttons.
- Put two of them near the Source. For the first Button, input 'Increase Interarrival Rate' for the *Button Text*, set the *Event Name* to 'IncreaseRate'. For the second Button, input 'Decrease Interarrival Rate' for the *Button Text*, set the *Event Name* to 'DecreaseRate'.
- Put the other two Buttons near Conveyor1 (which connects Server and Sink). For the first Button, input 'Stop Conveyor' for the *Button Text*, set the *Event Name* to 'StopConveyor'. For the second Button, input 'Start Conveyor' for the *Button Text*, set the *Event Name* to 'StartConveyor'.

Embellishments:

We can add some animation to show the current Arrival Rate of the Source while we're running this model. Choose *Status Label* from *Animation*, Set the *Expression* to 'Rate'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Using Relational Tables - SimBit

Problem:

I have data that exists in different tables and the tables are linked together with Foreign Keys, as in a relational database. I would like an Entity to reference data from these tables.

Categories:

Data Tables

Key Concepts:

Arrival Table, Expression Property, Foreign Key, Key, ModelEntity, Numeric Property, Picture, Relational Table, Selection Weight, State Assignments, String Property

Assumptions:

To easily witness that the entity is reading the correct information from the tables from within the Facility window, only red Entities will visit the TireService, blue Entities will visit the OilService and green Entities will visit the BrakeService.

Technical Approach:

Three Data Tables are created that are all linked together with Foreign Keys so that an Entity can reference information from all three tables easily. The Source gets entity *Arrival Times* from the Arrivals table and each Server gets its *Processing Time* from the ServiceTimes table. The Entity's color (symbol) is determined from the Picture table.

Details for Building the Model:

Simple System Setup

- Add a Source, a Sink and three Servers to the Facility window. Connect the Source to each of the Servers and each Server to the Sink with Paths. Also, drag a ModelEntity object into the window. Rename the Source 'Entrance', the Sink 'Exit' and the three Servers 'TireService', 'OilService' and 'BrakeService', respectively.

Setting up the Data Tables

- To add Data Tables, click on the Data tab and select the Tables panel.
- Add a Data Table named 'Picture' with the following Properties and in the following order:
 - (Integer) PictureIndex : 0, 1, 2
 - (String) Color: Green, Red, Blue
 - Make this column the Primary Key to this table by clicking the *Set Column as Key* icon in the ribbon.
- Add a Data Table named 'ServiceTimes' with the following Properties and in the following order:
 - (String) TypeOfService : Brakes, Oil, Tires
 - Make this column the Primary Key to this table by clicking the *Set Column as Key* icon in the ribbon.
 - (Expression) ServiceTime: Random.Triangular(1.5, 2, 2.5), 1, Random.Triangular(.5, 1, 1.2)
 - (Foreign Key) CarColor: Green, Blue, Red
 - To add this column, click the *Foreign Key* icon in the ribbon and then rename the property to 'CarColor'. Set the *TableKey* property to 'Picture.Color'.
- Add a Data Table named 'Arrivals' with the following Properties and in the following order:
 - (Real) ArrivalTime : 0, .25, .5, .75, 1
 - Make this column the Primary Key to this table by clicking the *Set Column as Key* icon in the ribbon.
 - (Foreign Key) TypeOfService: Brakes, Tires, Oil, Oil, Tires
 - To add this column, click the *Foreign Key* icon in the ribbon and then rename the property to 'TypeOfService'. Set the *TableKey* property to 'ServiceTimes.TypeOfService'.

Changing Object Properties

- In the Source:
 - Change the Arrival Logic *Arrival Mode* property of the Source to 'Arrival Table' and the *Arrival Time* Property to 'Arrivals. ArrivalTime'.

-
- Change the Arrival Logic *Maximum Arrivals* to '5'.
 - Within the State Assignments section, open the *Before Exiting* repeating property editor and assign the *State Variable Name* to 'ModelEntity.Picture' and the *New Value* to 'Picture.PictureIndex'
 - In each Server:
 - Change the Processing Time property to 'ServiceTimes.ServiceTime'
 - In each Path:
 - Change the Selection Weight to:
 - 'ModelEntity.Picture == 1' for the Path linking Entrance to TireService
 - 'ModelEntity.Picture == 2' for the Path linking Entrance to OilService
 - 'ModelEntity.Picture == 0' for the Path linking Entrance to BrakeService

Adding Additional Symbols

- In the Facility window, with the ModelEntity selected, click the Add Additional Symbol icon in the ribbon twice to add two additional symbols for this object. Keep the color of symbol 0 as green, change symbol 1 to red and symbol 2 to blue. For additional details on this process, see the [Adding Additional Symbols](#) Help page.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Using Relational Tables To Define Node Lists - SimBit

Problem:

I want to model a system where the entities being processed have different routings based on type, and each job step in a routing is performed at a work center area that consists of two or more identical servers. It is desired that all product mix, job routing, lists of possible server input nodes for each job entering a work center area, and operation data is defined in a set of relational data tables.

Categories:

Data Tables, Decision Logic -- Processing, Sequence Tables

Key Concepts:

By Sequence, Data Table, Entity Destination Type, RandomRow, Sequence Table, Server, Source, Table Row Referencing

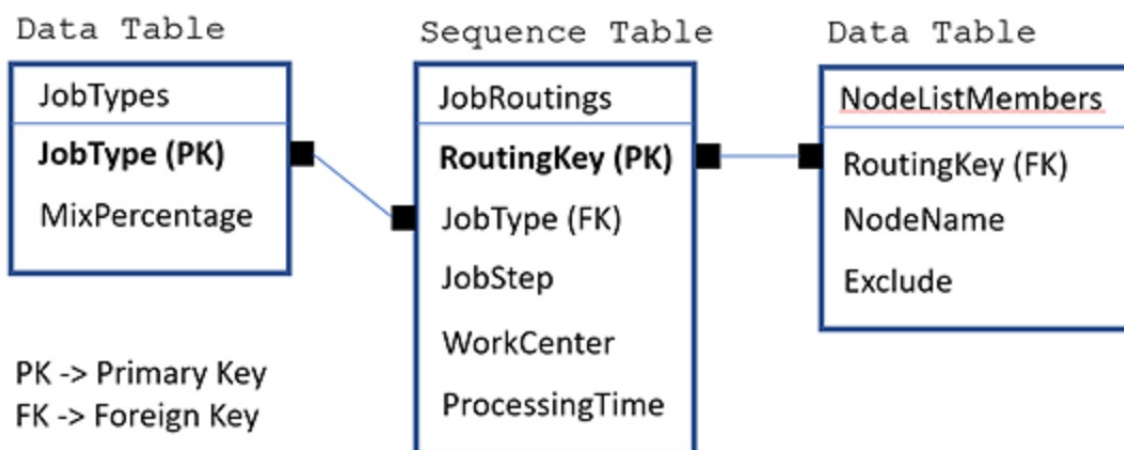
Assumptions:

The modeled system has four possible work center areas where entities are processed. An entity will enter the system and visit one or more of the work centers in a sequence that is dependent on the job type. Each work center area consists of two identical servers, either of which can be used to process a job. Not all of the work centers have to be visited and a work center may be visited more than once.

Technical Approach:

The product mix, job routing, lists of possible server input nodes for each job entering a work center, and operation data is defined in a set of relational data tables using the design shown in *Figure 1*.

Figure 1 - Relational Data Table Design



Details for Building the Model:

Facility Window Setup

- Add a Source, a Sink and 8 Servers (2 per work center area) from the Standard Library to the Facility window. Specify the name of the Sink object as 'Ship'. Then, for each of the four work centers, add a standalone TransferNode. These nodes will represent the work center entry points. Name those nodes 'WorkCenter1', 'WorkCenter2', 'WorkCenter3', and 'WorkCenter4'. Refer to the SimBit if necessary as a guide.
- Place as many ModelEntity objects from the Project Library into the window as you like, one for each job type that is processed in the system.

Data Table Setup

In the Data window, add one sequence table and two data tables to the model using the relational table design shown in Figure 1. The table column property types are as follows:

JobTypes Data Table

Column Name	Property Type
JobType (Primary Key)	Entity Object Reference
MixPercentage	Integer

JobRoutings Sequence Table

Column Name	Property Type
RoutingKey (Primary Key)	String
JobType	Foreign Key to JobTypes.JobType
JobStep	Integer
WorkCenter	Sequence Destination (auto added when creating Sequence Table)
ProcessingTime	Expression of unit type Time

NodeListMembers Data Table

Column Name	Property Type
RoutingKey	Foreign Key to JobRoutings.RoutingKey
NodeName	Node Object Reference
Exclude	Boolean

Mapping the Table Data to the Objects in the Facility Window

Go to the Facility window and do the following:

- **Source object** – Specify the Entity Type property as 'JobTypes.JobType'. Go to the *Table Row Referencing* -> *Before Creating Entities* properties, and specify the *Table Name* as 'JobTypes' and the *Row Number* as 'JobTypes.MixPercentage.RandomRow'. The Source will then randomly create entities of the types specified in the JobTypes table using the specified mix percentages.
- **For Each 'Output' TransferNode object** – Specify the *Entity Destination Type* as 'By Sequence'. Then, when an entity exits the Source or any of the Server objects, it will set its next destination based on the JobRoutings sequence table.
- **For Each Server object** – Specify the *Processing Time* as 'JobRoutings.ProcessingTime'. Go to the *Buffer Logic* properties and set the input and output buffer capacities to zero.
- **For Each Standalone TransferNode Representing a Work Center Entry Point** – Go to the Routing Logic properties and specify the *Entity Destination Type* property as 'Select From List', the *Node List Name* property as 'NodeListMembers.NodeName', and the *Selection Condition* property as 'NodeListMembers.Exclude == False'.

Embellishments:

Experiment with different product mix, job routing, and operation data in the relational data tables. Trying to add more servers to the work center areas, or using the Exclude column in the 'NodeListMembers' data table to remove a server as a choice for a particular job step at a work center.

See also:

ServersUsingTaskSequencesWithDataTables_JobShop.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Using Relational Tables To Define Task Resource Lists

- SimBit

Problem:

I want to model a server where the entity processing is a task sequence, and where all operation data is defined in a set of relational data tables including lists of possible resources that can perform specific tasks.

Categories:

MultiTask Server, Data Tables, Resources

Key Concepts:

Server, Auto-Set Table Row Reference, Data Table, Process Type, Task Sequence, Secondary Resources

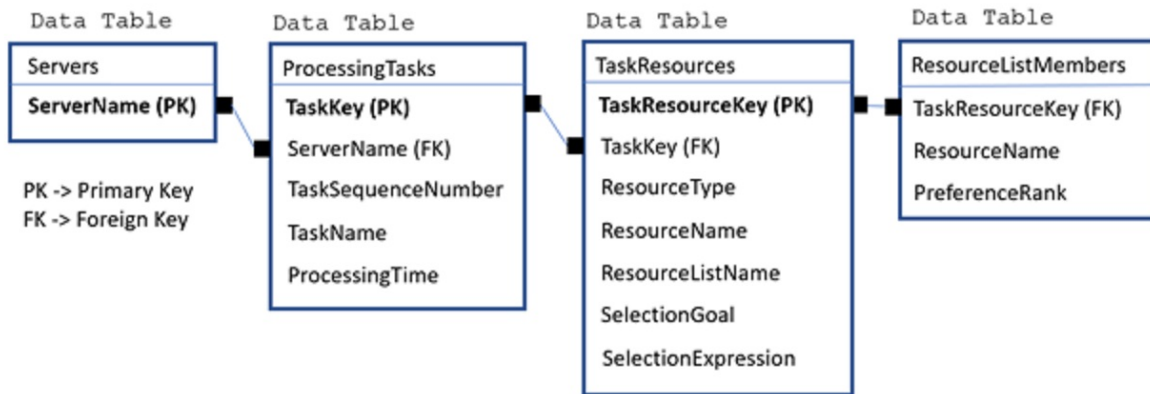
Assumptions:

The modeled system is a flow line that consists of two server locations visited in a fixed sequence. Entities enter the system at the front of the line and then are processed through the two servers before finishing at a sink.

Technical Approach:

The operation data for each server, including lists of possible resources that can perform specific tasks, is defined in a set of relational data tables using the design shown in *Figure 1*.

Figure 1 - Relational Data Table Design



Details for Building the Model:

Facility Window Setup

- Add a Source, a Sink and two Servers from the Standard Library to the Facility window. Then connect those objects using paths from the Standard Library, drawing a path from the Source to the input of Server1, then a path from the output of Server1 to the input of Server2, and finally a path from the output of Server2 to the Sink.
- Place a ModelEntity from the Project Library into the window.
- Place as many Resource or Worker objects from the Standard Library into the Facility window as you like, for use as task resources.

Data Table Setup

In the Data window, add four data tables to the model using the relational table design shown in *Figure 1*. The table column property types are as follows:

Servers Data Table

Column Name	Property Type
ServerName (Primary Key)	Object Reference

ProcessingTasks Data Table

Column Name	Property Type
TaskKey (Primary Key)	String
ServerName	Foreign Key to Servers.ServerName
TaskSequenceNumber	Sequence Number
TaskName	String
ProcessingTime	Expression of unit type Time

TaskResources Data Table

Column Name	Property Type
TaskResourceKey (Primary Key)	String
TaskKey	Foreign Key to ProcessingTasks.TaskKey
ResourceType	Enumeration of enum type ObjectSeizeType
ResourceName	Object Reference
ResourceListName	Object List Reference
SelectionGoal	Enumeration of enum type SeizeSelectionGoal
SelectionExpression	Expression

ResourceListMembers Data Table

Column Name	Property Type
TaskResourceKey	Foreign Key to TaskResources.TaskResourceKey
ResourceName	Object Reference
PreferenceRank	Integer

Mapping the Table Data to the Objects in the Facility Window

Go to the Facility window and do the following:

- **Source object** – Make sure the *Entity Type* property is specified as the model entity type that you placed in the model.
- **For Each Server object** – Specify the *Process Type* as 'Task Sequence'. Then, right-click on the Processing Tasks repeat group and set a reference to the 'ProcessingTasks' table. This indicates that the ProcessingTasks table will be used as the referenced data source for getting processing task information. Finally, in the *Process Logic* -> *Other Task Sequence Options* properties, specify the *Task Resources Referenced Table Name* as the 'TaskResources' table. This indicates that specific table will be used as the referenced data source for getting task resource requirements.
- **For Each Server object** – Open the *Processing Tasks* repeat group. Map the columns from the ProcessingTasks and TaskResources tables to the appropriate corresponding properties in the repeat group (refer to the SimBit if necessary as a guide).
- In the Servers table, make sure that the *Auto-set Table Row Reference* property for the *ServerName* column is set to 'True'.

Embellishments:

Experiment with different server operation data in the relational data tables.

See also:

ServersUsingTaskSequencesWithDataTables_FlowLine.spx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UsingTheStipulateStepForNodeLists - SimBit

This SimBit project includes three models that demonstrates the ways to allocate entities to multiple Servers using a either the Gantt drag and drop node list requirements / routing dependencies or Stipulate steps. Models included in this SimBit:

1. **ServerProcessesFIFO** – Demonstrates allocation of fixed number of entities to one of multiple Servers using a node list and simple Preferred Order rule.
2. **ServerProcessesFromDependenciesInGantt** – Demonstrates forced sequence of entities to one of multiple Servers based on drag and drop method of node list requirements and routing dependencies in the Resource Plan Gantt (RPS version only).
3. **ServerProcessesFromStipulateStep** – Demonstrates forced sequence of entities to one of multiple Servers based on Stipulate steps within the Processes window.

Model 1: ServerSelectedFromNodeList

Problem:

I wish to have a fixed number of entities transferred into one of a group of Servers selected from a node list.

Categories:

Arrival Logic, Data Tables, Decision Logic -- Processing, RPS

Key Concepts:

Arrival Table, NodeList, Server

Assumptions:

Four entities will be created at the start of the simulation and enter into the system based on the order in the table. Entities will select the Server to transfer into based upon a node list and selection rule.

Technical Approach:

A table will be used to create the entities at a specific time in the simulation model. A node list and selection goal will be used to determine the routing.

Details for Building the Model:

Simple System Setup

- Place a Source, three Servers and a Sink object into the Facility window. Connect the Source to all three Servers and all Servers to the Sink with paths and change the *Allow Passing* property on the paths between the Source and Servers to 'False' to graphically see the entities each enter the Server area.
- Place 4 ModelEntity objects from the Project Library and rename them 'Green', 'Blue', 'Red' and 'Yellow'. Change the color of each to match accordingly. Change the *Display Name* for each ModelEntity object to 'Table.OrderOfArrival'. This will show the associated column value for each entity within the Resource Plan Gantt.

Data Table Structure

- Go to the Data tab and add a data table.
- First, add an Object Reference type property of type Entity. Change the *Name* to 'EntityType' and enter the names of the entities in the following order from the entity list – Blue, Green, Red, and Yellow.
- Add a DateTime Standard Property that will be used as the arrival time of the entity. Change the *Name* to 'ArrivalTime' and set the arrival time for each of the 4 rows (entities) to the same start of simulation time value (9/9/2016 12:00:00 AM in SimBit). This will cause all entities to be generated at the same time.
- Add a Standard Property of type Color to the table and select a matching color for each entity type. This column is used within the Resource Plan Gantt.
- Add a fourth column to the table of type String. *Name* the column 'OrderName' and type in an associated name for each entity – in this case BlueOrder, GreenOrder, RedOrder and YellowOrder.

Create the Node List for the Servers

- Go to the Definitions tab, List panel and add a node list with the *Name* 'ServerNodeList'. Add 'Input@Server1',

'Input@Server2' and 'Input@Server3' to the list in that order.

Configure the Facility Window Objects

- In the Facility window, change the following Source properties. Set the *Entity Type* to 'Table1.EntityType'. Change the *Arrival Mode* to 'Arrival Table' and the *Arrival Time Property* to 'Table1.ArrivalTime'.
- Within the output node of the Source, change the *Entity Destination Type* to 'Select From List' and the *Node List Name* to 'ServerNodeList'. Leave the *Selection Goal* as 'Preferred Order' so that the entities select Server1 if possible.
- Within each the Servers, under the Advanced Options section, change *Log Resource Usage* to 'True' so that the usage of the Servers will be tracked for the Resource Gantt.

Running the Model in Planning Tab:

For Simio RPS users, go to the Planning tab and select the Resource Plan view. Press the Create Plan button on the Operational Planning ribbon. Zoom in using the + on the Gantt ribbon as necessary. Note that the order of entity processing is FIFO, or BlueOrder – GreenOrder – RedOrder – YellowOrder, which is the same order in which the entities were created within Table1.

Model 2: ServerSelectedFromDependenciesInGantt

Problem:

I wish to have a fixed number of entities process through a group of Servers based on the order in which I specify in the Simio RPS Edition – Resource Plan Gantt using drag and drop for node list routing dependencies and requirements.

Categories:

Arrival Logic, Data Tables, Decision Logic -- Processing, RPS

Key Concepts:

Arrival Table, NodeList, Node List Routing Dependencies, Node List Requirements, Server

Assumptions:

Four entities will be created at the start of the simulation and enter into the system based on the order in the table. *

IMPORTANT NOTE * - users will only see the order of entity processing interactively through the Facility Model view in the Planning tab and NOT through the Facility window, as drag and drop Node List Routing Dependencies and Node List Requirements are only used for planning mode.

Technical Approach:

A table will be used to create the entities at a specific time in the simulation model. A node list and selection goal will be used to determine the routing although drag and drop will be used to modify such routing in the planning mode.

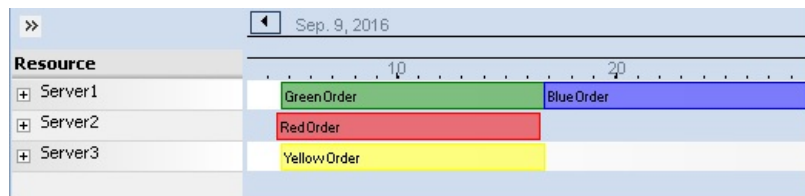
Details for Building the Model:

Building the Base Model

- Follow the same steps as shown in Model 1: ServerSelectedFromNodeList above.

Running the Model in Planning Tab:

For Simio RPS users, go to the Planning tab and select the Resource Plan view. Press the Create Plan button on the Operational Planning ribbon. Zoom in using the + on the Gantt ribbon as necessary. Note that the order of entity processing is FIFO, or BlueOrder – GreenOrder – RedOrder – YellowOrder, which is the same order in which the entities were created within Table1. Now, click on the RedOrder entity and drag and drop it to Server2. Move the YellowOrder entity to Server3 using the same method of drag and drop. Click on the GreenOrder and move it to be before the BlueOrder at Server1. Note the Node List Routing Dependencies and Node List Requirements that are created after these steps are complete.



Stipulations

Node List Routing Dependencies

	NodeList Name	Candidate Entity	Follows Entity	Prior
▶	ServerNodeList	BlueOrder	GreenOrder	
*				

Node List Requirements

	Candidate Entity	Node List Name	Required Node
▶	YellowOrder	ServerNodeList	Input@Server3
	RedOrder	ServerNodeList	Input@Server2
*			

Model 3: ServerSelectedFromStipulateStep

Problem:

I wish to have a fixed number of entities process through a group of Servers based on the order in which I specify using the Stipulate steps in the Processes window for node list requirements and node list routing dependencies.

Categories:

Arrival Logic, Data Tables, Decision Logic -- Processing, RPS

Key Concepts:

Arrival Table, NodeList, OnRun Initialized Process, Node List Routing Dependencies, Node List Requirements, Server, Stipulate Step

Assumptions:

Four entities will be created at the start of the simulation and enter into the system based on the order in the table. Through the use of the Stipulate steps, the Server that is selected from the node list can be controlled.

Technical Approach:

A table will be used to create the entities at a specific time in the simulation model. A node list and selection goal will be used to determine the routing. A string value must be associated with each entity to specify the exact entity information in the Stipulate steps.

Details for Building the Model:

Building the Base Model

- Follow the same steps as shown in Model 1: ServerSelectedFromNodeList above.

Adding the Stipulate Steps

- Within the Processes window, use the Select Process button to select the OnRunInitializedProcess. Within this process, place three (3) Stipulate steps.
- Within the first Stipulate step, leave the *Stipulation Type* to 'NodeListRoutingDependency'. Specify the *Candidate Entity* as "BlueOrder", the *Node List* as 'ServerNodeList' and the *Entity to Follow* as "GreenOrder". Change the *Prior Usage Count* to '1'. This will cause the Blue entity to follow the Green entity.
- Within the second Stipulate step, change the *Stipulation Type* to 'NodeListRequirement'. Specify the *Candidate Entity* as "RedOrder", the *Node List* as 'ServerNodeList' and the *Required Node* as 'Input@Server2'. This will cause the Red entity to always select Server2 from the list.
- Within the third Stipulate step, change the *Stipulation Type* to 'NodeListRequirement'. Specify the *Candidate Entity* as "YellowOrder", the *Node List* as 'ServerNodeList' and the *Required Node* as 'Input@Server3'. This will cause the Yellow entity to always select Server3 from the list.

Running the Model in Planning Tab:

For RPS users, go to the Planning tab and select the Resource Plan view. Press the Create Plan button on the Operational

Planning ribbon. Zoom in using the + on the Gantt ribbon as necessary. Note that the entities are processed as follows - Green then Blue at Server1, Red at Server2 and Yellow at Server3, which is the logic defined within the Stipulate steps.

See Also:

UsingTheStipulateStepForResourceAllocation.spfx

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Using the Stipulate Step For Resource Allocation

- SimBit

This SimBit project includes three models that demonstrate the ways to allocate entities through a Server using either the Gantt drag and drop resource dependencies or Stipulate steps. Models included in this SimBit:

1. **ServerProcessesFIFO** – Demonstrates allocation of fixed number of entities through Server based on standard FIFO rule.
2. **ServerProcessesFromDependenciesInGantt** – Demonstrates forced sequence of entities through Server based on drag and drop method of resource dependencies in the Resource Plan Gantt (RPS version only).
3. **ServerProcessesFromStipulateStep** – Demonstrates forced sequence of entities through Server based on Stipulate steps within the Processes window.

Model 1: ServerProcessesFIFO

Problem:

I wish to have a fixed number of entities process through a Server first in first out (FIFO) based on the order in which they arrive to the system.

Categories:

Arrival Logic, Data Tables, RPS

Key Concepts:

Arrival Table, Server

Assumptions:

Four entities will be created at the start of the simulation and enter into the system based on the order in the table. Entities will flow through the Server in the order in which they are created.

Technical Approach:

A table will be used to create the entities at a specific time in the simulation model.

Details for Building the Model:

Simple System Setup

- Place the Source, Server and Sink objects into the Facility window. Connect the objects with paths and change the *Allow Passing* property on the path between the Source and Server to 'False' to graphically see the entities each enter the Server area.
- Place 4 ModelEntity objects from the Project Library and rename them 'Green', 'Blue', 'Red' and 'Yellow'. Change the color of each to match accordingly.

Data Table Structure

- Go to the Data tab and add a data table.
- First, add an Object Reference type property of type Entity. Change the *Name* to 'EntityType' and enter the names of the entities in the following order from the entity list – Blue, Green, Red, and Yellow.
- Add a DateTime Standard Property that will be used as the arrival time of the entity. Change the *Name* to 'ArrivalTime' and set the arrival time for each of the 4 rows (entities) to the same start of simulation time value (9/9/2016 12:00:00 AM in SimBit). This will cause all entities to be generated at the same time.
- Add a Standard Property of type Color to the table and select a matching color for each entity type. This column is used within the Resource Plan Gantt.

Configure the Facility Window Objects

- In the Facility window, change the following Source properties. Set the *Entity Type* to 'Table1.EntityType'. Change the *Arrival Mode* to 'Arrival Table' and the *Arrival Time Property* to 'Table1.ArrivalTime'.

- Within the Server, under the Advanced Options section, change *Log Resource Usage* to 'True' so that the usage of the Server will be tracked for the Resource Gantt.

Running the Model in Planning Tab:

For RPS users, go to the Planning tab and select the Resource Plan view. Press the Create Plan button on the Operational Planning ribbon. Zoom in using the + on the Gantt ribbon as necessary. Note that the order of entity processing is FIFO, or Blue – Green – Red – Yellow, which is the same order in which the entities were created within Table1.

Model 2: ServerProcessesFromDependenciesInGantt

Problem:

I wish to have a fixed number of entities process through a Server based on the order in which I specify in the RPS Edition – Resource Plan Gantt using drag and drop for resource allocation dependencies.

Categories:

Arrival Logic, Data Tables, RPS

Key Concepts:

Arrival Table, Resource Allocation Dependencies, Server

Assumptions:

Four entities will be created at the start of the simulation and enter into the system based on the order in the table. * IMPORTANT NOTE * - users will only see the order of entity processing interactively through the Facility Model view in the Planning tab and NOT through the Facility window, as drag and drop Resource Allocation Dependencies are only used for planning mode.

Technical Approach:

A table will be used to create the entities at a specific time in the simulation model.

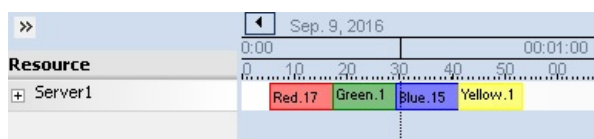
Details for Building the Model:

Building the Base Model

- Follow the same steps as shown in Model 1: ServerProcessesFIFO above.

Running the Model in Planning Tab:

For RPS users, go to the Planning tab and select the Resource Plan view. Press the Create Plan button on the Operational Planning ribbon. Zoom in using the + on the Gantt ribbon as necessary. Note that the order of entity processing is FIFO, or Blue – Green – Red – Yellow, which is the same order in which the entities were created within Table1. Now, click on the Red entity and drag and drop it ahead of the Blue and Green. Move the Blue entity after the Green by using the same method of drag and drop. Note the Resource Allocation Dependencies that are created after these steps are complete.



Resource/List Name	Candidate Entity	Follows Entity	Prior
Server1	Blue.15	Red.17	
Server1	Green.16	Red.17	
Server1	Blue.15	Green.16	
*			

Model 3: ServerProcessesFromStipulateStep

Problem:

I wish to have a fixed number of entities process through a Server based on the order in which I specify using the Stipulate steps in the Processes window for resource allocation dependencies.

Categories:

Arrival Logic, Data Tables, RPS

Key Concepts:

Arrival Table, OnRun Initialized Process, Resource Allocation Dependencies, Server, Stipulate Step

Assumptions:

Four entities will be created at the start of the simulation and enter into the system based on the order in the table. Through the use of the Stipulate steps, the order in which the entities are processed at the Server can be controlled.

Technical Approach:

A table will be used to create the entities at a specific time in the simulation model. A string value must be associated with each entity to specify the exact entity information in the Stipulate steps.

Details for Building the Model:

Building the Base Model

- Follow the same steps as shown in Model 1: ServerProcessesFIFO above.
- Within the Data tab, add another column to the table of type String. *Name* the column 'OrderOfArrival' and type in an associated name for each entity – in this case Order1, Order2, Order3 and Order4. These values will be referenced in the Stipulate steps.
- Within the Facility window, change the *Display Name* for each ModelEntity object to 'Table.OrderOfArrival'. This will show the associated column value for each entity within the Resource Plan Gantt.

Adding the Stipulate Steps

- Within the Processes window, use the Select Process button to select the OnRunInitializedProcess. Within this process, place an Execute step. Enter the *Process Name* as 'Process1'.
- Use the Create Process button to create a new process named 'Process1'. Place three (3) Stipulate steps within this process.
- Within the first Stipulate step, change the *Stipulation Type* to 'ResourceAllocationDependency'. Specify the *Candidate Entity* as ""Order2"", the *Resource or List* as 'Server1' and the *Entity to Follow* as ""Order3"". Change the *Prior Usage Count* to '1'. This will cause the Green entity to follow the Red entity.
- Within the second Stipulate step, change the *Stipulation Type* to 'ResourceAllocationDependency'. Specify the *Candidate Entity* as ""Order1"", the *Resource or List* as 'Server1' and the *Entity to Follow* as ""Order2"". Change the *Prior Usage Count* to '1'. This will cause the Blue entity to follow the Green entity.
- Within the third Stipulate step, change the *Stipulation Type* to 'ResourceAllocationDependency'. Specify the *Candidate Entity* as ""Order4"", the *Resource or List* as 'Server1' and the *Entity to Follow* as ""Order1"". Change the *Prior Usage Count* to '1'. This will cause the Yellow entity to follow the Blue entity.

Running the Model in Planning Tab:

For RPS users, go to the Planning tab and select the Resource Plan view. Press the Create Plan button on the Operational Planning ribbon. Zoom in using the + on the Gantt ribbon as necessary. Note that the order of entity processing is FIFO, or Red – Green – Blue – Yellow, which is the order defined within the Stipulate steps.

See Also:

UsingTheStipulateStepForNodeLists.spfx

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VehicleDoingSimultaneousLoading - SimBit

Problem:

Standard Vehicle/Worker behavior is to sequentially load items to be transported. But in some cases, like a bus with multiple doors, you may want to load several entities concurrently. This model illustrates three entity types waiting to be picked up by a vehicle at the same time. Each entity type has a different load time and the vehicle departs after the last entity finishes loading.

Categories:

Add-On Process Logic, Vehicles

Key Concepts:

Add-On Process, Bidirectional Path, Simultaneous Load, Pickup Step, Search Step, Vehicle, Wait Step

Technical Approach:

The Vehicle event 'Entered Node' contains custom add-on process logic. When the Vehicle enters the TransferNode named 'TransferNode1', a search is performed for all entities that have arrived and are waiting at the RidePickupQueue of the Vehicle. The Vehicle has a ride capacity of 10. The load time is defined in the Pickup step for the different entity types and the vehicle waits until all of the entities that are waiting to ride are picked up. The entities are then transported to the input node of the Sink object and dropped off.

Details for Building the Model:

Facility Window Setup

- Add three Sources, a Transfer Node, a Sink, and a Vehicle from the Standard Library. Connect each Source to the TransferNode using Connectors. Then connect the TransferNode to the Sink with a Bidirectional Path.
- Place three ModelEntity instances in the Facility window next to each Source.
- *Name* the entity instances 'EntityType1', 'EntityType2', and 'EntityType3'.
- In the first Source, change the *Entity Type* to 'EntityType1' then set the other Sources to their corresponding *Entity Types*.
- Set the *Maximum Arrivals* under the Stopping Conditions Properties to '1' on all the Sources.
- In the Properties of the TransferNode that has already been placed, change the *Entity Destination Type* to 'Specific' and the *Node Name* to 'Input@Sink1'. Under the Transport Logic Properties, change *Ride On Transporter* to 'True'. Specify the Vehicle's name for the *Transporter Name*.
- Select the Vehicle object, change the *Initial Ride Capacity* to '10'. Change the *Initial Desired Speed* to '0.1' and Select 'Input@Sink1' as the Vehicle's *Initial Node (Home)*.

Defining the Process Logic for the Vehicle

- Within the Vehicle object, under the Add-on Process Triggers Properties, double right click on the words *Entered Node*. A new process will be automatically created.
- Select the Processes tab to open the Processes window. A new empty process is now available to edit. Place the Decide step and change the *Condition Or Probability* to 'Vehicle.RideStation.Capacity.Remaining > 0 && Vehicle.CurrentNode.RidePickupQueue > 0'.
- Place an Assign step to assign the *State Variable Name* 'LoadingTime.Rate' to have a *New Value* of '1'.
- Place a Search step after the Assign step and change the *Collection Type* to 'QueueState' and the *Queue State Name* to 'Vehicle.CurrentNode.RidePickupQueue'. This will search through the RidePickupQueue at the TransferNode. Change the *Limit* under Advanced Options to the expression 'Vehicle.RideStation.Capacity.Remaining'.
- Place a Pickup step under the Search step so that all the entities that were found are picked up. Change the *Node Name* in the Wait step to 'Vehicle.CurrentNode' and the *Load Time* to 'Math.If(Entity.Is.EntityType1, 8, Entity.Is.EntityType2, 10, 2)'. And change the *Units* to 'Minutes'. This expression defines how long the load time of each Entity Type is. If the Entity is EntityType1, then it's load time is 8 minutes. If it is EntityType2, it's load time is defined as 10 minutes and any other Entity Type (in this case it's EntityType3) will have a load time of 2 minutes. Change the *Transporter Type* to 'SpecificObject' and the *Transporter Object* will automatically be defined as 'Vehicle'.
- A Wait step is placed after the Search step for the original entity that was used for the Search. Change the *Event Name* under the Basic Logic Properties to the expression 'Vehicle.RideStation.Entered'. The *Event Condition* should

be set to 'Vehicle.RideStation.Capacity.Remaining == 0 || (Source1.OutputBuffer.Contents + Source2.OutputBuffer.Contents+Source3.OutputBuffer.Contents) == 0'. This says the Vehicle will wait until the event called 'Vehicle.RideStation.Enter' occurs and the additional Event Condition is met where there aren't any entities in the RideStation or there aren't any entities in the output buffer queues of the Sources.

- Finally, place an Assign step after the Wait step. Assign the *State Variable Name* 'LoadingTime.Rate' a *New Value* of '0'.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

VehicleFinishesAndParksWhenOffShift - SimBit

Problem:

I want a Transporter to follow the same Work Schedule as a Server in my system. If the Server goes Off Shift when the Transporter is the middle of traveling on a link, the Transporter should finish traveling to the end, park and stop working. When the Server goes back On Shift, the Transporter will begin working where it left off.

Categories:

Vehicles

Key Concepts:

Add-On Process, Off Shift, On Off Shift, On On Shift, On Shift, Real State Variable, Ride on Transporter, Schedules, Vehicle, Wait Step, Fire Step

Assumptions:

The Vehicle and Server will follow the same WorkSchedule. If the vehicle goes Off Shift while transporting an entity, it will continue and drop off the entity before parking. If the vehicle goes Off Shift while moving to pick up an entity, it will not perform the pickup until it is On Shift.

Technical Approach:

When a vehicle goes off shift, it will automatically drop off and/or pick up any entities that it is scheduled to drop off / pick up. Because we would like our vehicle to stop at a node when it is off shift, we will use Add-On Process Logic at the node where it may pick up to ensure that the vehicle stops until the off shift period has ended. This will be done with a Decide step condition and Wait / Fire steps.

Details for Building the Model:

Simple System Setup

- Add a Source, a Server and a Sink to the Facility Window.
- Connect the objects together with Paths. Ensure that the Type property of the path from the Server to the Sink is set to 'Bi-directional' so the vehicle can travel back and forth.
- In order to slow down the entity arrival to best view the vehicle's behavior, change the Interarrival Time of the Source object to 'Random.Exponential(4)' minutes.

Adding the Vehicle

- Add a vehicle object from the Standard Library with the Name 'Vehicle1'.
- Select the Output Node of the Server and set the *Ride On Transporter* property to 'True', the *Transporter Type* to 'Specific' and then select your Vehicle for the *Transporter Name* property.
- In order to slow down the animation to best view the vehicle's behavior, change the *Desired Speed* property of the Vehicle to '0.04' (meters per second).
- Change the *Initial Ride Capacity* to 'Infinity' and set the *Initial Node (Home)* property of the Vehicle to 'Output@Server1'.

Creating a Work Schedule

- Go to the Data Window and click on the Schedules panel. Click the Work Schedule icon in the Schedule ribbon to add a new Work Schedule. Change the *Name* to 'Schedule1' and set the *Days* to '1'.
- Add a Day Pattern with the *Name* 'DayPattern1' to include the following entries: 8:00 AM – 8:15 AM, 8:30 AM – 8:45 AM and 9:00 AM – 12:00 PM. All of these rows should have the *Value* of '1' indicating a capacity of one or on shift. The times when there aren't any values are assumed to be off shift.
- Under Work Schedules, select 'DayPattern1' for day 1.

Adding the Schedule to the Server and Vehicle

- In the Facility window, select the Server object and set the *Capacity Type* property to 'WorkSchedule'. Select the work

schedule that was just created for the *Work Schedule* property.

- Do the same for the Vehicle object. Under the Process Logic section of properties, change the *Capacity Type* to 'WorkSchedule' and add the new work schedule that was created.

Adding Vehicle Off Shift Logic

- Within the Definitions window, States panel, add a new Real state variable with the *Name* 'OffShift'. We will assign this variable to '1' when the vehicle goes off shift and back to '0' when on shift.
- Within the Definitions window, Events panel, add a new event with the *Name* 'Event1'. This will be used within the Wait/Fire steps placed shortly.
- Within the Facility window, highlight the Vehicle and add *On Shift* and *Off Shift* add-on processes, named 'Vehicle1_OnShift' and 'Vehicle1_OffShift', respectively.
- Within the Processes window, 'Vehicle1_OffShift' process, add an Assign step and specify the *State Variable Name* 'OffShift' to the *New Value* of '1'.
- Within the Processes window, 'Vehicle1_OnShift' process, add an Assign step and specify the *State Variable Name* 'OffShift' to the *New Value* of '0'.
- Within the Facility window, highlight the Vehicle and add *Entered Node* add-on processes, named 'Vehicle1_EnteredNode'. Within this process, we will evaluate if the vehicle is off shift and if so, will wait at the node.
- Within the Vehicle1_EnteredNode process, place a Decide step. Change the *Decide Type* to 'ConditionBased' and enter the *Expression* 'OffShift==1&&Is.Vehicle&&Vehicle.NumberRiders==0'. From the True exit of the Decide, place a Wait step, with the *Event Name* of 'Event1'. Looking at the condition specified, we are checking to see if the vehicle is off shift (OffShift==1) and the associated object is the vehicle (Is.Vehicle) as opposed to the entity itself and the vehicle has no riders (Vehicle.NumberRiders==0), then the token must wait at the node for Event1 to be fired. Thus, if the vehicle is off shift but has riders, the vehicle will not wait, but will drop off its riders and by default will park.
- Within the 'Vehicle1_OnShift' process, after the Assign step, place a Fire step, with the *Event Name* of 'Event1'. Here, we will fire Event1 when the off shift period is over (thus releasing the vehicle waiting in the above logic, if necessary).

Enhancing the Animation

- If you'd like the Server to appear different when it is Idle vs. Busy vs. OffShift, you need to select the Server object in the Facility Window and click on the Add Additional Symbol icon in the Ribbon. Do this twice so there are now three symbols associated with this object. Change which symbol is 'Active' with the Active Symbol icon. Once the symbol is active, you can change the appearance of that symbol.
- The *Current Symbol Index* property of the Server controls which symbol is displayed during the run. The value of this property is used in the symbol index (the numbers listed next to the symbols in the Active Symbol drop down). In this example, the following expression is used so symbol 1 is displayed when the Server is On Shift (capacity = 1) but not busy, symbol 2 is displayed when the Server is On Shift and Busy and symbol 0 is displayed when the server goes Off Shift (capacity = 0): $(\text{Server1.Capacity.Allocated} + 1) * \text{Server1.Capacity}$
- Changing the appearance of the Vehicle during the run is done in a similar fashion. This example has two symbols for the vehicle, one that is Blue and one that is Red. The *Current Symbol Index* of the Vehicle is set to the state 'OffShift', which we set within the Vehicle's add-on processes.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VehicleFixedRoute - SimBit

Problem:

I have a vehicle, such as a bus, that loops around a specific route picking up and dropping off entities at varying locations.

Categories:

Vehicles

Key Concepts:

Entity Destination Type, Fixed Route, Ride on Transporter, Sequence Table, Vehicle

Assumptions:

There are four specific pickup / drop-off locations in the fixed route for the vehicle. Entities must wait at their pickup location for the vehicle to come to them, based on the vehicle's route sequence. Therefore, entities are not picked up based on a first in first out basis for requesting the vehicle, but are picked up based on the vehicle's location on the route at the time. All entities that are picked up at a given location are dropped off at a different, fixed location in this example.

Technical Approach:

Each pickup / drop-off location in the system is represented by Source and a Sink objects. By placing these objects near each other, they can be virtually the same physical location represented by two objects. The vehicle in the system will move along a fixed route that is specified in a clockwise manner. All Source and Sink objects are then connected using unidirectional Paths to show the direction of the vehicle movement. Entities entering at the various Sources will each have specific destination location Sinks to be dropped off by the vehicle.

Details for Building the Model:

Simple System Setup

- Place four Source objects and four Sink objects in the Facility Window. Move the nodes on these objects to be close to each other, as you will be connecting them to represent the vehicle path.
- Connect the various sink and source objects with Paths in the following order – Source1-Sink1-Source2-Sink2-Source3-Sink3-Source4-Sink4-Source1. This will form a circular path between all the objects in the system. Set the Allow Passing property for all the paths to 'False' (in case there are multiple vehicles).

Defining the Vehicle Route

- Open the Data Window and select the Tables panel. Here you will define the fixed route that the vehicle will take when moving between the objects. Click on Add Sequence Table and enter the various nodes that the Vehicle will visit, in order. These include 'Output@Source1', 'Input@Sink1', 'Output@Source2', 'Input@Sink2'... 'Output@Source4', 'Input@Sink4'. When the vehicle has gotten to the last destination in the table, it will move to the first one listed and continue through the list again and again until the simulation ends. This table will automatically have the *Name* 'SequenceTable1'.

Defining the Vehicle

- Place a Vehicle in the Facility Window. Within the Vehicle's Properties window, change the *Routing Type* to 'Fixed Route' and the *Route Sequence* to the 'SequenceTable1' table that was just defined.
- The *Initial Ride Capacity* for the vehicle specifies the maximum number of entities on the vehicle simultaneously. Change this value to 'Infinity', such that the Vehicle statistics can later be evaluated to determine the minimum, maximum and average number of entities riding on the vehicle.
- Set the *Load Time* and the *Unload Time* to '0.02' minutes each to provide a brief pause while each entity is loaded or unloaded.
- We will also replace the default animated queue on the transporter (Vehicle1.RideStation.Contents) with a U shaped one that will display more entities.

Creating the Part Types

- The Source objects will generate entities to enter the system. In the example, four different types of ModelEntity objects were placed in the Facility Window, with *Name* properties of 'PartA', 'PartB', 'PartC' and 'PartD'. For each of

the Source objects, change the *Entity Type* property to generate a different entity type.

Transferring using a Vehicle

- In this example, entities get on the Vehicle at each Source and are taken by the vehicle to any destination node (to be specified by the user). Click on the Source1 TransferNode. Under the Routing Logic, specify the *Entity Destination Type* as 'Specific' and the *Node Name* as 'Input@Sink3'. In the example, we have specified that Source1 goes to Sink3, Source2 goes to Sink4, Source3 goes to Sink1 and Source4 goes to Sink2. You can specify your different sources go to whichever sink input nodes you like. This will give the entity a "destination" so that when it is moved by the vehicle along the designated path, the entity knows which node to exit, at which time it will be dropped off by the vehicle.
- Finally, in order for the entity at each Source node to be moved along the path using the vehicle, click on Source1's TransferNode and change *Ride on Transporter* to 'True'. This will allow you then to select a *Transporter Type* ('Specific') and *Transporter Name* ('Vehicle1'). Do the same for the other three transfer nodes.

Embellishments:

Within the Vehicle object, change the Initial Number In System (under Population) to '3' and the Initial Ride Capacity (under Transport Logic) to '5' to evaluate multiple vehicles that can take a fixed number of entities.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VehicleFleetManagement - SimBit

Problem:

I have multiple vehicles that are responsible for transporting entities. Each vehicle is responsible for transporting to a specific destination, for example Vehicle1 takes entities from any origin to Sink1, Vehicle2 takes entities from any origin to Sink2 and so on.

Categories:

Decision Logic -- Paths, Entity Characteristics, Vehicles

Key Concepts:

Entity Destination Type, Find Step, NodeList, OnRun Initialized Process, PlanVisit Step, Ride On Transporter, Search Step, Selection Condition, Selection Goal, Subscribe Step, Transporter Reference State, Vehicle

Assumptions:

There are three pickup and three drop-off locations for the vehicle. Entities must wait at their pickup location for the vehicle to come to them and are then taken by vehicle to a random drop-off location. The fleet management strategy is that Truck #1 can only service requests to destination Input@Sink1, Truck #2 can only service requests to destination Input@Sink2, and Truck #3 can only service requests to destination Input@Sink3.

Technical Approach:

Each pickup location is represented by a Source and each drop-off location is represented by a Sink object. A 'TruckFleetManagement' process logic is invoked whenever a new ride pickup request occurs or an entity has been unloaded from a truck. There is a delay of 'Math.Epsilon' to force the fleet management decision making to occur as a late event at the end of the current time step. The process logic then checks for any available trucks and searches the global visit request queue to attempt to assign a truck to a waiting request, based on the above assumption logic. If a pickup request has been assigned to a truck, the fleet management process logic does a PlanVisit step to notify the truck and the truck then accordingly handles the pickup.

Details for Building the Model:

Simple System Setup

- Place three Source objects and three Sink objects in the Facility window.
- Change the *Interarrival Time* for each of the Source objects to 'Random.Exponential(.5)' minutes.

Defining the Vehicle

- Place a Vehicle in the Facility window. Change the *Name* to 'Truck' and the *Initial Number in System* to '3'.
- While the Vehicle is highlighted, click on the Animation ribbon and place a Status Label above the vehicle symbol (it should be *Attached To* 'Truck'). Enter the *Expression* "'Truck#' + String.FromReal(Population.Index)'" so that the truck number will appear above the vehicle symbol. The "Truck#" is simply a string, while the function for Population.Index will return 1, 2 or 3.

Defining the Entity

- Within the Navigation window, click on the ModelEntity model, go to the Definitions window / States panel. Add a new State Variable of type Transporter Object Reference and change the *Name* to 'AssignedTruck'. This can then be referenced as 'ModelEntity.AssignedTruck' within the Model.
- In the Facility window, place a ModelEntity. While the entity is highlighted, click on the Animation ribbon and place a Status Label above the entity symbol (it should be *Attached To* 'DefaultEntity'). Enter the *Expression* 'DestinationNode' so that when the entity destination is determined, it will appear above the entity symbol.

Defining the Node Lists and Model State Variable

- Within the Definitions window, add a Node list named 'EntityDestinationNodeList'. Include in the list the three Sink options, Input@Sink1, Input@Sink2 and Input@Sink3. This list will be used within the output node of each Source to determine where the entity goes after leaving the Source.
- Add a Node list named 'EntitySourceNodeList'. Include in this list the output nodes for each of the Source objects, such as Output@Source1, Output@Source2 and Output@Source3. This list will be used within process logic to

subscribe to a process when an entity waits for a ride at the node.

- Within the Definitions window, States panel, add a new Integer state variable with the *Name* 'PickupRequestIndex'. This will be used in the TruckFleetManagement logic.

Transferring using a Vehicle

- In this example, entities get on the Vehicle at each Source and are taken by the vehicle to any destination node (randomly selected from list). Click on each of the Source object TransferNodes and, under the Routing Logic, specify the *Entity Destination Type* as 'Select From List', the *Node List Name* as 'EntityDestinationNodeList' and the *Selection Goal* as 'Random'. With this logic, entities from any of the Source objects may move to any of the Sink objects, as assigned randomly. Then, based on their assigned destination, we will use logic to determine which Truck vehicle to request.
- Also within each of the Source object TransferNodes, under the Transport Logic, specify the *Ride On Transporter* as 'True', the *Transporter Name* as 'Truck' and the *Selection Condition* as 'ModelEntity.AssignedTruck==Candidate.Vehicle'. We will be adding process logic to assign the entity's AssignedTruck value based on its destination location logic.

Process Logic for OnRunInitialized Process

- Within the Processes window, click on the Select Process button from the Process ribbon and select OnRunInitialized. Within this process, add two Search steps. In the first Search step, change the *Collection Type* to 'NodeList' and the *Node List Name* to 'EntitySourceNodeList'. Change the *Limit* to 'Infinity'. This step will provide a token related to each source in the list (Output@Source1, Output@Source2 and Output@Source3) that will exit the Found exit of the step.
- From the Found exit of the first Search discussed above, add a Subscribe step. Within the Subscribe step, specify the *Event Name* as 'TransferNode.Riding', the *Event Condition* as 'TruckFleetManagement.TokensInProcess.NumberItems==0' and the *Process Name* as 'TruckFleetManagement'. When the RiderWaiting based event for each TransferNode is entered, the process named 'TruckFleetManagement' will be activated (while the condition specifies that only one entity necessary within that process).
- Now, enter the second Search step (which should be connected from the Original exit of the first Search described above). Keep the *Collection Type* as 'EntityPopulation', and change the *Entity Type* to 'Truck' and the *Limit* to 'Infinity'. Then, associated with each of the 3 trucks in the system, a token will be generated from the Found exit.
- From the Found exit of the second Search, add a Subscribe step. Change the *Event Name* to 'Vehicle.RideStation.Exited', the *Event Condition* to 'TruckFleetManagement.TokensInProcess.NumberItems==0' and the *Process Name* to 'TruckFleetManagement'. When the truck drops off an entity at a Sink object, the 'TruckFleetManagement' process will be activated (again, given there is only one entity in that process at a time).

Process Logic for TruckFleetManagement Process

- Within the Processes window, click on the Create Process button on the Process ribbon to add a new process. Specify the *Name* as 'TruckFleetManagement'.
- Add a Delay step as the first step in the process with the *Delay Time* specified as 'Math.Epsilon'. This will cause the event to be handled last on the calendar of events that occur at the same specific time.
- Add a Search step with the *Collection Type* of 'EntityPopulation', the *Entity Type* of 'Truck', the *Match Condition* of 'Candidate.Vehicle.ResourceState == 0' and the *Limit* of 'Infinity'. This will return a token for each of the trucks that are idle (ResourceState == 0).
- From the Found exit, add a Decide step. Specify the *Condition or Probability* as 'Global.VisitRequestQueue.NumberWaiting > 0'. This will determine if there are any entities waiting to be assigned a vehicle (once assigned, they move from the global request queue to the vehicle's specific queue while awaiting pickup).
- From the True exit of the Decide step, add a Find step. Change the *Index Variable Name* to 'PickupRequestIndex'. The *Starting Index* should be '1' and the *Ending Index* should be 'Global.VisitRequestQueue.NumberWaiting'. The *Search Expression* should be 'Math.If(Vehicle.Population.Index==1, Global.VisitRequestQueue.ItemAtIndex(PickupRequestIndex).ModelEntity.DestinationNode==Input@Sink1, Vehicle.Population.Index==2, Global.VisitRequestQueue.ItemAtIndex(PickupRequestIndex).ModelEntity.DestinationNode==Input@Sink2, Global.VisitRequestQueue.ItemAtIndex(PickupRequestIndex).ModelEntity.DestinationNode==Input@Sink3)'. Since a token for each idle vehicle is sent through this Find step, note that the Math.If statement determines which population.index (1/2/3) the vehicle is and then finds the first entity in the visit request queue that has its destination node set to the corresponding Sink input node (1/2/3). If there is nothing in the visit request queue for a particular vehicle token, the Find step Index Variable Name 'PickupRequestIndex' is set to '0' and the token exits the NotFound exit of the step. If there is an entity found that matches the vehicle destination assignment, then the 'PickupRequestIndex' is set to the index in the queue that the entity resides and the token continues from the Found

exit to the next step.

- Place an Assign step from the Found exit of the Find step. Set the *State Variable Name* 'Global.VisitRequestQueue.ItemAtIndex(PickupRequestIndex).ModelEntity.AssignedTruck' equal to the *New Value* of 'Vehicle'. This will assign the entity's transporter reference state, AssignedTruck, to the appropriate Vehicle based on the entity destination.
- Add a PlanVisit step to the process after the Assign step. Under Advanced Options, select *Entity Type* to be 'Specific Object' and *Entity Object* to be 'Vehicle'. The PlanVisit step will cause the particular vehicle to search the Global.VisitRequestQueue for any requests, which then the output nodes on the Sources include the *Selection Condition* 'Candidate.Vehicle==ModelEntity.AssignedTruck' to assure the correct entity is paired with the vehicle.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VehicleMovingBackward - SimBit

Problem:

I would like to model a vehicle moving back and forth on a path and not have unrealistic “jumps” in the animation.

Categories:

Vehicles

Key Concepts:

Allow Passing, Detached Queue, Entity Destination Type, ParkingStation Queue, Ride on Transporter, Vehicle

Assumptions:

When a vehicle changes direction at a node it often appears to jump to a new position. This is because the head of the vehicle determines the position of the vehicle on a link or when it enters a node. For example, the vehicle will enter a node from the right and when the vehicle tries to turn around, it will appear to be entering the node from the left; making an undesired “flip” around the node.

Technical Approach:

The vehicle used will look the same in both directions, meaning the head and tail of the vehicle cannot be distinguished from each other. A mixture of Paths and Connectors will be used between four TransferNodes to model the course that the Vehicle will follow. In order to make it appear like the vehicle is moving backward, a path will be used to animate its normal movement between the TransferNodes at the ends of the course and a connector will be used to animate the vehicle’s turn around movement.

Details for Building the Model:

Simple System Setup

- Add a ModelEntity, Source, Sink and four TransferNodes to the Facility Window. Connect the Source to TransferNode2 and TransferNode4 to the Sink using Paths.
- Add a Path from TransferNode2 to TransferNode4 and another Path from TransferNode3 to TransferNode1.
- Add a Connector from TransferNode1 to TransferNode2 and another Connector from TransferNode3 to TransferNode4. The Connectors have been made visible in this example by drawing them without overlapping the Paths.

Animating the Vehicle

- Add a Vehicle to the Facility Window. Set the *Initial Node (Home)* to ‘TransferNode1’ and the *Idle Action* to ‘Park at Home’.
- It is very important that the distance between TransferNode1 and TransferNode2 and the distance between TransferNode3 and TransferNode4 is greater than the length of the Vehicle. This is because the Vehicle needs to have enough room to instantly turn itself around on the Path and Connector without getting stuck. Otherwise it could block itself from turning around.

Animating the Entities

- Select the Output Node on the Source and change the *Entity Destination* to ‘Specific’ and select ‘TransferNode4’ and the destination.
- Select the Path between the Source and TransferNode2 and change the *Allow Passing* property to ‘False’. This will cause the Entities to line up in order as they wait to be transported by the Vehicle.
- Select TransferNode2 and change the *Ride on Transporter* property to ‘True’ and select ‘Vehicle1’ as the *Transporter Name*.
- Select TransferNode4 and change the *Entity Destination Type* property to ‘Specific’ and the *Node Name* to ‘Input@Sink1’.

Animating the Parking Queues

- In order to make it appear that the Vehicle is parking on its route while idle, the Parking Queues for TransferNode1 and TransferNode4 must be drawn and placed directly on the route.

-
- Add two detached queues to the Facility Window and change the *Queue State* of one to 'TransferNode1.ParkingStation.InProcess' by choosing it from the drop down menu. Change the *Queue State* of the other queue to 'TransferNode1.ParkingStation.InProcess'. Then move these queues to their respective places in between the TransferNodes.

Embellishments:

Try changing the Interarrival Time of the Source or the Desired Speed of the Vehicle. Also, try different a Idle Action for the Vehicle.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VehiclesPassingOnRoadway - SimBit

Problem:

Model a simple version of a roadway with sections that allow passing and sections that do not.

Categories:

Vehicles

Key Concepts:

Allow Passing, Path, Path Decorators

Assumptions:

There are separate Source/Path/Sink sets for each direction of the road. Multiple vehicles can pass a vehicle simultaneously without regard for oncoming traffic.

Technical Approach:

Two sources, placed at opposite ends of the model, create the entities with random speeds mimicking vehicles entering separate sides of a roadway. The roadway is separated into three sections – two curved sections that do not allow passing and one straight section that does. After the vehicle has driven the entire roadway it exits the system.

Details for Building the Model:

Creating the Roadway

- Place two Sources and two Sinks in the model, with one source and sink on each side of the model. Position them so that there are 2 Source/Sink pairs representing either side of the road. Separate them so that the vehicles have enough room to illustrate the vehicle movement.
- Place 4 TransferNodes in between the Source/Sink pairs (2 for each pair) to represent the passing section.
- Move left Sink's BasicNode and the right Source's TransferNode so that that both Source/Sink Pairs have their nodes facing each other.
- Double-click on Path to create the lanes of the road. Start at one of the Sources and start clicking to create a curved section of the roadway. End this section at one of the TransferNodes.
- From this TransferNode draw a straight line connecting it to a second TransferNode.
- Then create another curved section ending at a Sink.
- Create the other lane in the same manner. Start at the other Source and create another curved section leading to a new TransferNode, followed by a straight section to the final TransferNode, then a section leading to the second Sink. Make sure to mimic the contour of the first lane.
- To each path, add a single lane path decorator and set the path width (Size – Width) to 2 meters.

Adding Entity Symbols

- Add a Vehicle to the Facility Window. Set the *Initial Node (Home)* to 'TransferNode1' and the *Idle Action* to 'Park at Home'.
- It is very important that the distance between TransferNode1 and TransferNode2 and the distance between TransferNode3 and TransferNode4 is greater than the length of the Vehicle. This is because the Vehicle needs to have enough room to instantly turn itself around on the Path and Connector without getting stuck. Otherwise it could block itself from turning around.

Animating the Entities

- Place a ModelEntity from the Project Library into the Facility Window.
- After making sure the ModelEntity is selected, click on Add Additional Symbol in the Symbols tab. Click on the button repeatedly until you have reached an appropriate level of Active Symbols (in this example there are 9 Active Symbols).
- Go through all of the Active Symbols in the dropdown list and change each vehicle symbol to a different color or different symbol.
- Set the ModelEntity's *Random Symbol* property to 'True'.

Modeling the Vehicle Movement

- Change the ModelEntity's *Initial Desired Speed* to 'Random.Uniform(1,5)' Meters per Second.
- To prevent the vehicles from passing on the curved sections of the road, click on the curved sections and change the *Allow Passing* in the Travel Logic to 'False'.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

VehicleStopsWhenServerOffShift - SimBit

Problem:

I want a Transporter to follow the same Work Schedule as a Server in my system. The Transporter should stop working immediately when the Server goes Off Shift.

Categories:

Vehicles

Key Concepts:

Add-On Process, AssociateObjectMovement, Off Shift, On Off Shift, On On Shift, On Shift, Real State Variable, Resume Step, Ride on Transporter, Schedules, Search Step, Subclass, Suspend Step, Vehicle

Assumptions:

The Vehicle object's capacity cannot be controlled by a Work Schedule so the Vehicle's logic will be suspended when the Server goes Off Shift and it will resume its logic when the Server goes back On Shift.

Technical Approach:

Subclass the standard Vehicle object and change the OnVisitingNode processes property *Public* from 'False' to 'True'. This will allow this process to appear in the selection list of the Suspend Step. The Suspend Step will suspend this process from within the OffShift Add On Process of the Server. Similarly, the Resume Step will resume the OnVisitingNode process from within the OnShift Add On Process of the Server. The Vehicle's movement is also suspended and resumed with a Suspend Step from within the same Add On Processes.

Details for Building the Model:

Simple System Setup

- Add a Source, a Server and a Sink to the Facility Window.
- Connect the objects together with Paths. Ensure that the *Type* property of the path from the Server to the Sink is set to 'Bi-directional' so the vehicle can travel back and forth.
- In order to slow down the entity arrival to best view the vehicle's behavior, change the *Interarrival Time* of the Source object to 'Random.Exponential(4)' minutes.

Subclassing the Vehicle Object

- Select MySimioProject in the Navigation Window and highlight the Models panel. On the Edit ribbon, press the Subclass From Library button and select the Vehicle. This will create a sub-classed 'MyVehicle' object.
- Highlight Model in the Navigation Window to return the main model and place the MyVehicle object from the Project Library into the Facility Window. In order to make it look like a true vehicle, when the object is selected in the Facility Window, select a vehicle symbol from the Project Symbols dropdown in the Ribbon.
- Select the Output Node of the Server and set the *Ride On Transporter* property to 'True', the *Transporter Type* to 'Specific' and then select your Vehicle for the *Transporter Name* property.
- In order to slow down the animation to best view the vehicle's behavior, change the *Desired Speed* property of the Vehicle to '0.04' (meters per second).
- Set the *Initial Node (Home)* property of the Vehicle to 'Output@Server1'.

Creating a Work Schedule

- Go to the Data Window and select the Schedules panel. Click the Work Schedule icon from the Schedule ribbon and change the *Name* to 'Schedule1' and the *Days* to '1'.
- In the Day Patterns tab, add a new one with the *Name* 'DayPattern1' and press the '+' to open the pattern.
- Add three rows to reflect the following schedule: 8:00 AM – 8:15 AM with a *Value* of '1', 8:30 AM – 8:45 Am with a *Value* of '1' and 9:00 AM – 12:00 PM with a *Value* of '1'. NOTE: The times when there aren't on shift times are considered to be off shift (capacity value of 0).
- Go back to the Work Schedules Tab and select 'DayPattern1' for Day 1.

Add the Schedule to the Server and Create logic to Stop Vehicle

- Highlight the MyVehicle object in the Navigation Window and open the MyVehicle's Processes Window. Find the process OnVisitingNode and click on the process. Click the Override icon in the top Ribbon and then change the *Public* property from 'False' to 'True'.
- In the Model's Facility Window, select the Server object and set the *Capacity Type* property to 'WorkSchedule'. Select the work schedule that was just created for the *Work Schedule* property.
- From within the Server properties window, select the dropdown icon from the Add On Process trigger *OffShift* and select 'Create New'. Do the same for the trigger *OnShift*. (if you double click on the name of the Property, i.e. On Shift, you will be taken to the Process Window)
- In the Model's Processes Window, you should see the two new processes that were created from the Server's Add On Process triggers. In the Off Shift process, place a Search Step (found under All Steps). The Search Step is needed to create a token that points at the Vehicle instance so the Vehicle can be suspended. The *Collection Type* of the Search Step is set to 'ObjectInstance' and the *Object Instance Name* is set to the name of the subclassed Vehicle. The other properties can be kept with their default values.
- Place a Suspend Step (found under All Steps) in the Found segment leaving the Search Step. The *Suspend Type* of this step is set to 'AssociatedObjectMovement'. This will stop the vehicle's movement immediately.
- Place a second Suspend Step directly after the first Suspend Step in the Found segment. The *Suspend Type* of this step is set to Process and the *Process Name* is set to the OnVisitingNode process of the Vehicle. (i.e. MyVehicle.OnVisitingNode)
- Almost the exact same logic should be put in the On Shift process, except instead of the Suspend Steps, Resume Steps should be used to both resume the vehicle's movement and the OnVisitingNode process.

Enhancing the Animation

- If you'd like the Server to appear different when it is Idle vs. Busy vs. OffShift, you need to select the Server object in the Facility Window and click on the Add Additional Symbol icon in the Ribbon. Do this twice so there are now three symbols associated with this object. Change which symbol is 'Active' with the Active Symbol icon. Once the symbol is active, you can change the appearance of that symbol.
- The *Current Symbol Index* property of the Server controls which symbol is displayed during the run. The value of this property is used in the symbol index (the numbers listed next to the symbols in the Active Symbol drop down). In this example, the following expression is used so symbol 1 is displayed when the Server is On Shift (capacity = 1) but not busy, symbol 2 is displayed when the Server is On Shift and Busy and symbol 0 is displayed when the server goes Off Shift (capacity = 0): $(\text{Server1.Capacity.Allocated} + 1) * \text{Server1.Capacity}$
- Changing the appearance of the vehicle during the run is done in a similar fashion. This example has two symbols for the vehicle, one that is Blue and one that is Red. The *Current Symbol Index* of the Vehicle is set to a State Variable called Off Shift. This state is set to '1' with an Assign Step after the Suspend Step that stops the vehicle. It is set to '0' when the Vehicle resumes its movement.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VehicleVisitsServiceCenter - SimBit

Problem:

I want to model a system that contains a vehicle that is subject to random failures. And upon failure, the vehicle finishes its delivery, if not idle, and then proceeds to a Service Center to be repaired or serviced.

Categories:

Vehicles

Key Concepts:

Capacity.Initial, Capacity.Remaining, Contents, Decide Step, Event, Failure, Failure.Active, ID, Node Property, OnInitialized Process, Override, Park Step, ParkingStation, ParkingStation.Queue, Real State Variable, Ride on Transporter, Search Step, SetNode Step, Subclass, Time To Repair, UnPark Step, Uptime Between Failures, Vehicle, Wait Step

Assumptions:

There is only one Vehicle in the system.

Technical Approach:

Changes are required to the standard Vehicle object, therefore it is subclassed into the Project Library. A new Node Property is added to this object to hold information about the service center node. From within the Vehicle object, the OnFailed, FailureOccurrenceLogic, OnRepaired and OnVisitingNode processes are updated with custom logic to indicate that the vehicle must not immediately stop in place upon Failure, but instead finish delivery, if appropriate, and then travel to the service node before starting its repair time delay.

Details for Building the Model:

Simple System Setup

- Add a Source, two Servers and a Sink to the Facility Window. Change the mean *Interarrival Time* on the Source to 'Random.Exponential(1)'.
- Connect the Source to the first Server, connect the first and second Server together and connect the second Server to the Sink with Paths. Add a Basic Node and connect it to the Output Node of Server 1 and the Input Node of Server 2.
- Make the path between the two Servers bi-directional, along with the two paths connecting the Basic Node. To make a path bi-directional, set its *Type* property to 'Bidirectional'.
- Click on the Output Node of Server 1.
 - Set its *Entity Destination* property to 'Specific' and the *Node Name* property to 'Input@Server2'. Entities will always go to Server2.
 - Set its *Ride On Transporter* property to 'True', its *Transporter Type* property to 'Specific'. You'll need to set the *Transporter Name* property to 'MyVehicle1' but this cannot be done until your custom Vehicle is in your Facility Window and this happens later.

Customize a Vehicle

Because we want our vehicle to have some unique behavior (e.g. go to a specified node before failing) we will start by creating a new vehicle object based on the existing Vehicle, then add a new property and other definitions that it will need.

- Subclass the standard Vehicle object into your Project Library. This is done by clicking on MySimioProject in the Navigation Window, then selecting the Models panel. Click on the Subclass From Library button from the ribbon and select Vehicle. This adds a subclassed MyVehicle object to the library.
- Highlight the Model in the Navigation Window and select the Definitions tab, Properties panel. Add a new Node Property (from the Object Reference icon in the ribbon) and name it ServiceNode. If you put 'Reliability' in the *Category Name* property of this new property, it will appear in that category of the Vehicle, next to the other information about Failures.
- In the Definitions Window, select the Events panel and add a new Event that is named 'ArrivedAtServiceNode'.

FailureOccurrenceLogic Process

- In the Processes Window, find the FailureOccurrenceLogic process and select Override from the ribbon. This will allow

you to make changes to this process. In the Processes Window, find the OnFailed process and select Override from the ribbon. This will allow you to make changes to this process.

- Place a SelectVisit Step in the Failed segment leaving the Allocate Step. The existing PlanVisit Step will then be in the Failed segment of this new SelectVisit step
- Place a Decide Step in the OK segment leaving the Select Visit Step. The expression should be "IsParked" to check if this vehicle is parked. Place a UnPark Step in the True segment, so the vehicle will unpark itself if it indeed is parked.

OnFailed Process

We want our vehicle to automatically move to a repair location on failure, so we need to replace the default failure logic with some custom logic.

- In the Processes Window, find the OnFailed process and select Override from the ribbon. This will allow you to make changes to this process.
- Delete the first two Suspend steps which suspend the ParentObjectMovement and OnVisitingNode. When the failure occurs, we don't want the vehicle to immediately stop. Additionally, if the vehicle is transferring an entity, we need it to continue (including going through OnVisitingNode) until the entity is dropped off.
- After the Suspend Failure Occurrence Logic step, add a Decide step to see if there are riders on the Vehicle or if its capacity is not seized, meaning that it is not on its way to get a rider. (Expression is NumberRiders == 0 & & Capacity.Remaining == Capacity.Initial)
- From the True branch of this step, place a SetNode step. The *Object Type* is 'ParentObject', the *Destination Type* is 'Specific' and the *Node Name* should be set to the new ServiceNode property. To set a property here, right click on *Node Name* and select SetReferencedProperty and then select ServiceNode.
- After the SetNode Step, place a Decide step that checks if the Vehicle is parked. (IsParked).
- From the True branch of this step, place a UnPark step. The *UnPark Type* is set to 'Parent Object'.
- After this UnPark Step, place a Fire step to fire the *Event Name* 'RemainInPlaceEnded'. This will cause the parked vehicle logic to continue in the OnVisitingNode process (where the Wait step for 'RemainInPlace' step exists).
- After the Fire step, place a Wait step. The *Event Name* is set to 'ArrivedAtServiceNode'.
- Connect the False branch of both Decide steps to the Wait step ('ArrivedAtServiceNode').
- The Wait step will flow into the Execute step ('FailedAddOnProcess') that was part of the original process. All the other steps in the process remain untouched.

OnRepaired Process

- Because the two Suspend steps in the OnFailed process were deleted, you will also need to delete the corresponding Resume steps in this process, the Resume 'ParentObjectMovement' and the Resume 'OnVisitingNode' process.

OnVisitingNode Process

As part of implementing our custom failure logic, we need to special handle the case when the vehicle arrives at the repair location and also when the vehicle is failed but arrives at the location with an entity to dropoff.

- Find the OnVisitingNode process and Override it.
- Find the IfNotVisitingNode Decide step. Place a new Decide step in the False branch, directly before the Decide (IfBusy) step. This new Decide step checks to see if the Vehicle is at the Service Node. (CurrentNode != ServiceNode) If not, it continues with the normal logic.
- In the False branch of this new Decide step, place a Park step. The *Entity Type* is 'Parent Object' and the *Node Name* is the ServiceNode property. Set this by right clicking on the *Node Name* and selecting SetReferencedProperty and the selecting ServiceNode.
- After this Park step, place a Fire Step. This will fire the event 'ArrivedAtServiceNode'. The branch then exits this Process.
- Go to the bottom, right of this process and find a Decide step titled "IfOffShift". In the False branch of this Decide step, place a Decide Step that determines if the vehicle is failed (ResourceState==List.ResourceStateName.Failed). You'll need to disconnect the False branch from it's current connection in order to add the new Decide Step.
- The True branch should contain a new Dropoff step where the *Unload Time* references 'UnloadTime' property, followed by a Wait step where *Event Name* is 'RiderUnloaded'. These can be copied / pasted from earlier logic prior to the Pickup step. The Wait step connects back around to the Dropoff step. The Wait step should have an *Exclusion Expression* of '!UnloadedAddOnProcess.HasValue' so that the vehicle doesn't wait if it has failed carrying an entity but doesn't have an unload add-on process.
- The Failed branch of the Dropoff step should connect to a Release step (also can be copied from earlier logic) where *Object Type* is 'Parent Object', and *Units per Object* is SeizedResources.CapacityOwnedOf(ParentObject). The *Owner* in Advanced Options is 'ParentObject'.
- From the Release step, place a SetNode step. *Object Type* is 'Parent Object' and the *Node Name* is the ServiceNode property.

-
- The False branch of the Decide step that was added and the segment leaving the Set Node step should both connect to the second step (Decide) in the process with Name 'IfNotVisitingNode'. The '-' and '+' keys on the keyboard can be used within the process to zoom in/out. This may be necessary to connect these steps.
 - The rest of this process remains unchanged.

Place the Vehicle into the Model

Here we will customize how the default vehicle looks.

- Before you place the Vehicle into the Facility Window of the model, you might want to change the External view of the Vehicle (how it looks in the Facility window). To do this, go to the External panel of the Definitions Window for the Vehicle object. Place a symbol of a vehicle and place the RideStation.Contents queue on the Vehicle.
- Place the new customized Vehicle into the Facility Window of the model.
- Set the *Initial Node* property to 'Output@Server1'.
- Set the *Failure Type* to 'Calendar Time Based'. Set *Uptime Between Failures* to 3 minutes. Set *Time to Repair* to 1 minute.
- Set *ServiceNode* to 'BasicNode1'.
- Click onto Output@Server1 and set the Ride On Transporter property to 'True' and the Transporter Name to the name of the customized vehicle you placed into the model.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VehicleWithVariableRideCapacity - SimBit

Problem:

I have a vehicle that can pick up a variable number of entities at any given location, based on a distribution.

Categories:

Vehicles

Key Concepts:

Add-On Process, Assign Step, Decide Step, Is.Vehicle, Math.Floor(), NumberRiders, On Evaluating Transport Request, On Exited, ReturnValue, Round, Token, Vehicle

Assumptions:

There is a single vehicle that can pick up anywhere from one to five entities at Source1.

Technical Approach:

A state variable, *RidersNeeded*, will be used to sample from a distribution to determine how many entities the vehicle will pick up at a given time for each pickup. The Initial Ride Capacity of the vehicle will be set to 5, which will be the maximum number of entities the vehicle can move. The Evaluating Transport Request add-on process will be used to compare the *RidersNeeded* value sampled with the number of current riders function of the vehicle to determine if a ride request is accepted or not.

Details for Building the Model:

Simple System Setup

- Add a Source and Sink from the Standard Library to the Facility window. Connect the objects with a Path.
- Change the *Type* of the Path to 'Bidirectional' so the vehicle may go back and forth between the Source and Sink.

Adding and Using a State Variable

- Within the Definitions tab, States panel, add a Discrete State and change the *Name* to 'RidersNeeded'.
- Set the *Initial State Value* to '1', which will allow the vehicle to pick up one entity for its first pickup.
- Within the Processes window, use the Create Process button and create a process with the *Name* 'ResetRiderNumber'. Within this process, add a Decide step and change the *Decide Type* to 'ConditionBased' and *Expression* to 'Is.Vehicle'.
- Add an Assign step to the process from the True branch of the Decide step. Within the Assign step, change the *State Variable Name* to 'RidersNeeded' and the *New Value* to 'Math.Floor(Random.Triangular(1,3,6))'. This will assign the *RidersNeeded* state to a new value each time the vehicle leaves the Source for its next pickup. The *Math.Floor* function is used to truncate the values generated from the distribution to integer values.
- Within the Facility window, highlight the output node of the source (Output@Source1) and, under the Add-On Process Triggers *Exited* property, add the process 'ResetRiderNumber' that was just defined above.

Defining the Vehicle

- Place a Vehicle object in the Facility window and change the *Initial Ride Capacity* to '5'. This will set the maximum capacity of the vehicle.
- Within the Add-On Process Triggers section, create a new process for the *Evaluating Transport Request* property called 'Vehicle1_EvaluatingTransportRequest'.
- Open the Processes window. In the Vehicle1_EvaluatingTransportRequest process, add a Decide step with the *Decide Type* of 'ConditionBased' and the *Expression* of 'Vehicle.NumberRiders<RidersNeeded'. The *Vehicle.NumberRiders* is a function that looks at the current number of entities on the vehicle. The *RidersNeeded* is the state variable that we use to determine the number of riders for a particular pickup. This process will be evaluated with each entity awaiting transport at Source1.
- From the False connection of the Decide step, add an Assign step. Within the Assign step, set the *State Variable Name* to 'Token.ReturnValue' and the *Value* to '0.0'. When the *Token.ReturnValue* is set to less or equal to 0.0, the transport request is rejected, which is what will happen for those requests above the *RidersNeeded* state value.

Embellishments:

The RidersNeeded state variable can be animated by using a Status Label from the Animation tab and changing the *Expression* to 'RidersNeeded'. This is useful in making sure that the vehicle is picking up the number of riders as determined by the RidersNeeded state variable value sampled from the distribution.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VisitAllServersInAnyOrder - SimBit

Problem:

An entity must process through ServerA, then Servers 1, 2, and 3 in any order but must do all three, then finish with ServerC. This model illustrates how a simple modification can be made to an object to permit more intelligent use, in this case allowing selection of only objects that still need the service it provides.

Categories:

Building New Objects / Hierarchy, Decision Logic -- Paths, Discrete States

Key concepts:

Entity Destination Type, State Variable, Selection Goal, Selection Condition, Candidate, NodeList

Assumptions:

The three pooled servers have the same processing time and each has a single buffer position. The entities can process through the 3 servers in any order, but must process at each of the 3 servers exactly once before moving on to Server C. The server with the smallest overload is selected. If no server is available that is still needed, the entity will wait in the Dispatching area (a node).

Technical Approach:

There are two key requirements:

- Each entity must keep track of which of the three servers it needs to visit and which have already been visited. This is done by adding three states to the ModelEntity.
- The easiest way to select between servers is to use the TransferNode's built-in Entity Destination Type and in particular it's Selection Condition. In order to take advantage of this, we created a custom MyServer by subclassing from Server. The only change to MyServer is to add a HasEntityVisited property into which can be passed the appropriate entity state corresponding to that server.

Other logic includes path weights on Path9 and Path10 for determining if entity should return to dispatching or continue to ServerC.

Embellishments:

We added 3 status labels (one for each custom state) on each entity picture so you can see which stations have been visited.

We added 3 pictures to the entity that gets successfully darker as more stations are visited so you can tell its status at a glance.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

WF_AdditionalResource - SimBit

Problem:

I want to model the ability of some process-aware information systems to allocate a work item to an additional resource.

Categories:

Worker

Key Concepts:

Functions, Lists, Server, Secondary Resources, Worker

Assumptions:

There are two entity types in the system and two Workers. A productivity function is used to calculate the processing times of each entity type.

Technical Approach:

Entities will seize randomly a Worker from a list. EntTypeB entities require 2 resources for processing. This is specified in Number of Objects property of the Secondary Resources seize in order to an EntTypeB entity seizes 2 workers.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink into the Facility window. Put the objects in series and connect them with paths.
- Place another Source object and connect the Output@Source2 to Input@Sever1 using a Path.
- Add two Worker objects into the Facility window. Set the *Park While Busy* to 'True' and the *Initial Network* as 'No Network (Free Space)'. Set 'Output@Source1' Node as the *Initial Home* for each Worker. Set the *Idle Action* and *Off Shift Action* properties of each worker to 'Part At Home'. Change Worker2 color to Red by clicking in Color in the ribbon.

Defining the Entity Types

- Place two ModelEntity objects into the Facility window and change their names to 'EntTypeA' and 'EntTypeB'. Click on EntTypeB and change its color to Red by clicking in Color in the Ribbon. Set *Entity Type* property of Source2 to 'EntTypeB', while the *Entity Type* for Source1 should be 'EntTypeA'.

Creating a List of Workers

- In the Definitions window, go to the Lists panel. Add a new Object list. Name it 'Resources' and add Worker1 and Worker2 to the list.

Creating a Function and Set Processing Times

- In the Definitions window, go to the Functions panel. Add a new Function. Name it 'Productivity' and change *Expression* property to 'Math.If(Entity.SeizedResources.NumberItems > 1, 1.5, 1)'.
- Click in Server1 and change *Processing Time* property to 'Random.Triangular(3,5,7)/Productivity'.

Seizing the Secondary Resources Within the Server

- Click on Server1 and within the Secondary Resources, Other Resource Seizes section, enter the Before Processing repeat group. Use the Add button and change *Object Type* to 'Select From List', *Object List* to 'Resources', *Request Move* to 'To Node' and *Destination* to 'Input@Server1'. Expand Advanced Options, change the value of *Number of Objects* property to 'Math.If(ModelEntity.Is.EntTypeB,2,1)' and change the value of *Selection Goal* to 'Random'. This will cause the entities of EntTypeB to require 2 workers from the set, while any other entities require only 1.
- Within the Secondary Resources section of Server1, under the Other Resource Releases section, enter the After Processing repeat group and Add a release of *Object Type* 'Select From List', *Object List Name* to 'Resources' and *Number of Objects* to 'Math.If(ModelEntity.Is.EntTypeB,2,1)' to release the resource(s) that were seized.

WF_Authorization - SimBit

Problem:

I want to model the ability of some process-aware information systems to specify the range of authorized workers able for processing entities in a given server.

Categories:

Worker

Key Concepts:

Data Table, Lists, Secondary Resources, Server, State Assignments, Table Row Referencing, Worker

Assumptions:

There are two entity types created from the same entity instance in the system. Green and Red workers are authorized to process green and red entities while the yellow worker can process both.

Technical Approach:

Lists of Workers are used to represent authorized workers per each entity type. A Data Table is used to store for each entity type its symbol reference, percentage of arrival and a reference to the list of authorized resources for processing in Server1. A table row reference is associated randomly to each created entity at the Source object by percentage of arrival. Add-on process triggers are used to seize and release the authorized workers in Server1.

Details for Building the Model:

Simple System Setup

- Place a Source, Server and Sink into the Facility window. Put the objects in series and connect them with Paths.
- Add three Worker objects into the Facility window. Set 'Output@Source1' Node as the *Initial Home* for each Worker. Set the *Idle Action* of each worker to 'Park at Home', the *Initial Network* to 'No Network (Free Space)' and the *Park While Busy* to 'True'.
- Clicking in Worker1, 2 and 3 and change their color to Green, Yellow and Red respectively by selecting Color in the ribbon.
- Place a ModelEntity into the Facility window and change its the name to 'ProcessCaseType'. Click on it and add an additional symbol. Change the color of active symbol 1 to Red in order to distinguish the entity types.

Creating Lists of Workers Representing Authorized Resources

- In the Definitions window, go to the Lists panel. Add 2 new Object lists. Name them 'AuthorizedGroup_One' and 'AuthorizedGroup_Two' respectively. Add Workers 1 and 2 into the AuthorizedGroup_One list and Workers 3 and 4 into AuthorizedGroup_Two list. These lists represent authorized resources.

Creating a Data Table

- Go to the Data window, and within the Tables panel, click the Add Data Table icon the ribbon to add a new table. Name it 'EntityData'.
- Add 3 columns that are a String, Integer and Real standard properties by selecting each data type from the Standard Property drop down in the ribbon. Change the names of each column to 'Name', 'Symbol' and 'Percentage' respectively.
- Add fourth column to the table – an Object List reference by selecting Object List from the Object Reference drop down in the ribbon. The column can be renamed to 'AuthorizedResources'.
- Fill the table with data – such as:
 - TypeA, 0, 40, AuthorizedGroup_One
 - TypeB, 1, 60, AuthorizedGroup_Two

Associating a Data Table Row Reference to the ModelEntity and Changing Its Color

- Click on the Source object and expand Table Row Referencing property category. Expand On Created Entity and set *Table Name* property to 'EntityData' and *Row Number* property to 'EntityData.Percentage.RandomRow'. This will associate a table row reference to a created entity based on the percentage of arrival specified in the table.

-
- Click on the Source Object and expand State Assignments property category. Click on the "... " button to open the Repeating Property Editor. Click the Add button and change *State Variable Name* to 'ModelEntity.Picture' and *New Value* to 'EntityData.Symbol'. This will change the color of a new created entity by its corresponding active symbol.

Specifying the Secondary Resources Required

- Click on Server1 and within the Secondary Resources, Resource for Processing properties, specify the *Object Type* as 'Select From List', *Object List Name* to 'EntityData.AuthorizedResources', *Selection Goal* as 'Random', *Request Move* as 'To Node' and *Destination Node* as 'Input@Server1'. This will require each entity to seize the appropriate resource to be used during the processing delay.
- Change the Server *Initial Capacity* to 'Infinity' to that multiple workers can process entities at the same time at this location.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

WorkerPoolWithEfficiency - SimBit

This SimBit project includes two models that demonstrate the use of the Worker pools or lists, including efficiency of the worker and skill level. Models included in this SimBit:

1. **SimpleWorkerPool** – Demonstrates how to incorporate individual worker walking speed and efficiency based on worker selection.
2. **ComplexWorkerPool** – Demonstrates how to incorporate worker skill level or ability to work at particular servers. The selection of the worker from a pool (list) is based on efficiency factor.

Model 1: SimpleWorkerPool

Problem:

I have a group of workers that have different efficiencies and would like to adjust the processing time at my workcenters based on the worker efficiency.

Categories:

Add-On Process Logic, Data Tables, Entity Characteristics, Schedules / Changeovers, Worker

Key Concepts:

Add-On Process, Assign Step, Candidate, Dynamic Label Text, Request Move, Search Step, SeizedResources, Select From List, Server, Transporter List, Worker

Assumption:

We will request a worker from the worker pool and then determine the worker efficiency from the data table.

Technical Approach:

There are three types of workers within the WorkerPool. The WorkerInfo table includes data such as the workers' walking speed and efficiency factor. Once the worker is allocated to the entity at the Server, an entity state will be used to store the efficiency factor related to the seized worker. This value is then used as a divisor within the Server's *Processing Time* entry.

Details for Building the Model:

Simple System Setup

- Place a Source, three Servers and a Sink in the Facility window. Connect the Source to each of the three Servers and connect each Server to the Sink with Paths. Place a BasicNode anywhere with the *Name* 'WorkerHomeNode'.
- Change the *Interarrival Time* on the Source to 'Random.Exponential(.5)'.

Defining the Workers and WorkerInfo Table

- Place three Worker objects in the Facility window and name them WorkerA, WorkerB and WorkerC. Change the color of both the idle and busy symbols for the workers to easily distinguish them.
- Go to the Definitions window and click on Lists in the panel. Add a Transporter list named 'WorkerPool' and add WorkerA, WorkerB and WorkerC to the list. *This can also be done by group selecting the three worker objects within the Facility window, right-clicking and selecting Add To Transporter List.*
- Go to the Data window and add a new Data Table with the *Name* 'WorkerInfo'. Add an Object Reference column of type Transporter (*Name* 'WorkerType') and add the three worker names to the table. Within the column properties, change the *Auto-set Table Reference* to 'True'. This assures that at run initialization, the worker will have an association with its specific table row, and thus all the data in that row as well. In the same table, add a Real property with the *Name* 'WalkingSpeed'. Change the *Default Value* to '0.0', the *Unit Type* to 'TravelRate' and the *Default Units* to 'Meters per Second'. Add a third column of type Expression with the *Name* 'EfficiencyFactor'. For WorkerA, add a value of '2' for WalkingSpeed and '1.5' for EfficiencyFactor. For WorkerB, add a value of '1' for WalkingSpeed and '1.2' for EfficiencyFactor. For Worker3, add a value of '0.5' for WalkingSpeed and '1' for EfficiencyFactor.
- Go to the Facility window and for each worker, set the *Initial Desired Speed* to 'WorkerInfo.WalkingSpeed' to reference the speed from the data table. Change the *Initial Node (Home)* to 'WorkerHomeNode', which is the BasicNode placed within the window earlier in the process. Change both the *Idle Action* and *OffShift Action* to 'Park At Home' so that the workers go to the home node when not working at a Server. Change the *Park While Busy* to

'True'.

Adding a ModelEntity State

- Within the Navigation window (top right), all above work has been in the 'Model'. Now, click on the ModelEntity, as a state to the entity will be added for keeping the efficiency factor. Go to the Definitions window (of ModelEntity) and select the States panel. Add a Real state variable to model entity with the *Name* 'EfficiencyFactor'. This will be used to store the efficiency factor of the worker that the entity seizes once in a server. It can be referenced as 'ModelEntity.EfficiencyFactor'.
- Within the Navigation window, highlight the Model and go to the Facility window. Place a ModelEntity object from the Project library. Change the *Dynamic Label Text* (under Animation section of properties) to 'ModelEntity.EfficiencyFactor'. This will then show the value of the worker efficiency once the entity has entered a server and seized a worker from the list.

Calculating the ModelEntity.Efficiency

- Go to the Processes window and add a new process with the *Name* 'WhichWorkerAllocated'.
- Add a Search step to the process and specify the *Collection Type* as 'SeizedResources'. This will search the associated object (entity) for all seized resources that it has, including the server itself, as well as the worker allocated (provided this process is called AFTER both resources have been seized). To filter out the server and only include the worker in the search candidates, add a *Match Condition* of 'Candidate.Object.Is.Worker'.
- Under the Found exit of the Search, add an Assign step. Here, we will use the token associated with the seized worker that is found to determine the efficiency of that worker from the WorkerInfo table. Assign the *State Variable Name* of 'ModelEntity.EfficiencyFactor' to the *New Value* of 'WorkerInfo.EfficiencyFactor'.

Changing Server Logic

- The following steps should be done for each of the three Servers. They can be done simultaneously by using Ctrl-click to highlight all three Servers.
- Within each Server, change the *Processing Time* to '1 / ModelEntity.EfficiencyFactor'. This will take the standard processing time (assumed in this case to be 1 minute) and divide by the efficiency of the worker that is allocated to the entity (in the above step).
- Under the Secondary Resources section, change the *Object Type* to 'Select From List' and *Object List Name* to 'WorkerPool'. To move the worker to the server, change the *Request Move* to 'ToNode' and the *Destination Node* to 'Server.Input'.
- Under the Add-On Process Triggers, specify 'WhichWorkerAllocated', which is the process added in above section, to the *Processing* field. Note that this add-on process will occur after the server and worker have been allocated, just prior to starting the processing time.

Enhancements (done within this model):

Add a work schedule to one or more of the workers so that one goes off-shift for a portion of time. In this model, the StandardWeek work schedule has been modified and added to WorkerA.

Model 2: ComplexWorkerPool

Problem:

I have a group of workers that have different efficiencies and skill abilities for particular workcenters. I want to pick the most efficient worker with the skill level for the workcenter and then adjust the processing time accordingly.

Categories:

Add-On Process Logic, Data Tables, Entity Characteristics, Worker

Key Concepts:

Add-On Process, Assign Step, Candidate, Dynamic Label Text, Request Move, Search Step, SeizedResources, Select From List, Selection Condition, Selection Expression, Selection Goal, Server, Table List, Transporter List, Worker

Assumption:

We will request the most efficient worker from the worker pool that has the appropriate skill requirements and then determine the worker efficiency from the data table.

Technical Approach:

There are three types of workers with information regarding each, such as walking speed, efficiency, cost and skill abilities, defined within a table. The entity will seize the most efficient worker available, given the skill level requirement for the

server. Once the worker is allocated, an entity state will be used to store the efficiency factor related to the seized worker. This value is then used as a divisor within the Server's *Processing Time* entry.

Details for Building the Model:

Simple System Setup

- Same as Model 1 above.

Defining the Workers and WorkerInfo Table

- Same as Model 1 above except do not define a Transporter List within the Definitions window, as a table list (using WorkerInfo.WorkerType as the list) will be used.
- Within the Data window, change the values in the EfficiencyFactor column so that WorkerA is '1.8', WorkerB is '1.6' and WorkerC is '2' (meaning WorkerC will be most efficient and processing time calculation smallest).
- Add an Expression type column named 'Cost' and enter '20' for WorkerA, '16' for WorkerB and '21' for WorkerC. Add three Boolean columns named 'Server1', 'Server2' and 'Server3'. These will be used to determine whether a worker is skilled on a Server. Change the following to 'True' (checked). WorkerA is skilled on all servers, WorkerB is skilled on Server1 and Server3, and WorkerC is skilled on all servers.
- In the Facility window within each of the worker instances, add the *Idle Cost Rate* and *Usage Cost Rate* (under Financials) as 'WorkerInfo.Cost'.

Adding a ModelEntity State

- Same as Model 1 above.

Calculating the ModelEntity.Efficiency

- Same as Model 1 above.

Changing Server Logic

- The following steps should be done for each of the three Servers. They can be done simultaneously by using Ctrl-click to highlight all three Servers.
- Within each Server, change the *Processing Time* to '1 / ModelEntity.EfficiencyFactor'.
- Under the Secondary Resources section, change the *Object Type* to 'Select From List' and *Object List Name* to 'WorkerInfo.WorkerType'. Instead of selecting the worker from the table in preferred order (WorkerA, WorkerB, then WorkerC), change the *Selection Goal* to 'Largest Value' and the *Selection Expression* to 'WorkerInfo.EfficiencyFactor'. This will select the worker with the largest efficiency factor (and thus most efficient based on processing time calculation). To move the worker to the server, change the *Request Move* to 'ToNode' and the *Destination Node* to 'Server.Input'.
- To incorporate the skill level selection of the worker based on the server location, change the *Selection Condition* to 'WorkerInfo.Server** == True' where ** is either 1/2/3 depending on Server Name.
- Under the Add-On Process Triggers, specify 'WhichWorkerAllocated', which is the process added in above section, to the Processing field. Note that this add-on process will occur after the server and worker have been allocated, just prior to starting the processing time.

Notes:

The *Value Expression* and *Selection Condition* properties should not use 'Candidate' when referencing the Data Table as a list. When evaluating which Worker to pick from the Data Table, the specific row of the Data Table will be considered as it evaluates each resource in the list. This allows that table row to be used when evaluating related properties, such as *Value Expression* and *Selection Condition* properties, that are also used in this secondary resource logic. The Data Table can be referenced directly as it points to a row and is not an item that needs to be considered as a Candidate.

Enhancements:

Note that the work schedule enhancement with the above Model 1 was not included in this example. Potential enhancement includes changing the *Selection Goal* and *Selection Condition* to select the lowest cost worker.

WorkersArriveLateToShift - SimBit

Problem:

I would like to model the fact that my Worker will sometimes arrive late to Shift.

Categories:

Building New Objects / Hierarchy, Custom Statistics, Decision Logic -- Processing, Discrete States, Schedules / Changeovers, Worker

Key Concepts:

Active Symbol, Assign Step, Decide Step, Floor Label, NumberOccurrences, Off Shift, On Shift, Schedules, Secondary Resources, Server, TallyStatistic, WorkSchedule, Worker

Assumptions:

This model assumes that the worker could arrive late to all shifts, including the shift that begins after the lunch hour break.

Technical Approach:

A Server seizes a Worker object before processing and releases it after processing. The standard Worker object is subclassed in order to keep the original object definition, but allowing the user to make a few changes to the logic to account for the fact that the Worker might arrive late. The new Worker's OnCapacityChanged process is Overridden and five new Steps are added to this logic. A Decide step checks the ProbabilityOfArrivingLate property and if the Worker is late, the MinutesLateToShift property is assigned to a State variable and the picture of the Worker is set to Red. The Worker delays for the assigned amount of time, a Tally step updates a TallyStatistic that keeps track of the minutes that this Worker is late to the shift. Finally the picture of the Worker is set back to Blue and they return On Shift.

Details for Building the Model:

System Setup

- Place a standard Source, Server and Sink object from the Standard Library into the Facility window and connect them with Paths. Alternatively, click on the Actions (under Add-Ins) button on the Project Home ribbon and select the Source, Server, Sink option.
- Place a Transfer Node somewhere near the Server and name it WorkForServer1. Place another Transfer Node a short distance away from the Server and name this WorkerHome. Connect these two nodes with a Path and set the Path's Type property to 'Bidirectional'. This will be the path that the Worker object takes.

Subclass a Worker

- Right click on the Worker Object in the standard library and select Subclass. This will create a new object in the Project Library called MyWorker. You'll find this new object in the Project Library located in the bottom left part of the interface, underneath the Standard Library
- The new object definition for MyWorker appears in the Navigation window in the upper right corner. Click onto MyWorker to enter into this new object. You'll be in MyWorker's Processes window at first.
- Go to this object's Definitions window and then click into the Properties panel. Create two new Expression properties by selecting Expression from the Standard Property drop down menu in the ribbon.
 - Name the first property *ProbabilityOfArrivingLate*. In the *CategoryName* property of this new expression property, type in 'ArrivingLateToShift'. This creates a new property category which helps us keep our new properties organized.
 - Name the second expression property *MinutesLateToShift*. In the *CategoryName* property of this new expression property, select 'ArrivingLateToShift' so this appears in the same category as the other new property.
- Click onto the States panel to add two new States to this object. Add a Real State by clicking on Real in the ribbon. Name this new state, "MinutesLateToShift_State". Add another Real State and name this new state "AnimationPicture". We'll discuss how these are used in the details below.
- Go to the Object's Processes window. Find the OnCapacityChanged process. Click anywhere inside this process and then click Override in the Process ribbon. The process should then turn white and you can now make changes to the process.

- Place a Decide step directly after the first Decide step in that process, which is named IfOnShift. Name the new Decide Step, "Late?". Enter the expression "1-ProbabilityOfArrivingLate" into the *Expression* property of this new Decide Step. The True segment leaving this step will tell the token to continue with the original logic (i.e. go to the Notify Step). The False segment will contain additional new logic.
- Place an Assign step in the False segment leaving that Decide step. Set the *State Variable Name* to 'MinutesLateToShift_State' and the New Value to 'MinutesLateToShift'. Note: Because the property can be entered as a Random distribution, we want to get a value from this random distribution once (now) and assign it to a State Variable. And then we'll use this State variable in the upcoming Delay Step and Tally Step. If we were to instead use the property in the Delay step and the Tally step, we'd sample the random distribution each time and therefore has different values for the Delay and the Tally, thus making our statistics wrong).
- In this same Assign step, add another Assignment where the *State Variable Name* is 'AnimationPicture' and the *New Value* is '1'. This will allow us to turn the Worker Red when it's late for its shift.
- Place a Delay step next and set the *Delay Time* property to 'MinutesLateToShift_State'.
- Place a Tally step next and select 'CreateNew' from the drop down in the *Tally Statistic Name* property. Name the new Tally Statistic 'LateToShift'. In the Value property of this step, enter 'MinutesLateToShift_State' so that we record the value of the state into the new Tally Statistic. Note: The Simbit has a BreakPoint on this Step to help the user easily see when the Worker arrives late.
- Place an Assign step and set the *State Variable Name* to 'AnimationPicture' and the *New Value* to '0'. This will allow us to change the color of the Worker back to Blue when its going back On Shift. Connect the process back up to the beginning of the Notify Step so that the token will continue with the original logic to go back OnShift.

Add to the MyWorker1 Instance

- Go back to the main model by clicking Model in the Navigation window. Drag an instance of MyWorker into the Facility window.
 - Select MyWorker1 in the window and set the following properties:
 - *Capacity Type* to 'WorkSchedule'
 - *Work Schedule* to 'StandardWeek'
 - *Initial Node (Home)* to 'WorkerHome'
 - *OffShift Action* to 'GoToHome'
- Find the new property category called "ArrivingLateToShift" where you should see the two new properties that were created.
 - Set the ProbabilityOfArrivingLate to '1' (100%)
 - Set the MinutesLateToShift to 'Random.Uniform(5,10)'
- With the MyWorker1 instance still selected, click on the Add Additional Symbol button in the Symbols ribbon. This will add another symbol for this instance. Symbol 2 of 2 should be selected so click onto the Color ribbon button and select Red and then click back onto the instance of MyWorker so that it turns Red. If you go back to symbol 1 by selecting symbol 0 in the Active Symbol ribbon drop down, you'll see that it is still Blue.
- Ensuring that the MyWorker1 instance is still selected, open its Animation property category and find the property *Current Symbol Index*. Set it to 'MyWorker.AnimationPicture' So when the logic in our MyWorker object sets the State AnimationPicture to 1, the Red symbol will be displayed and when the logic sets it back to '0', the Blue symbol will be displayed.

Embellishments:

The SimBit includes some Floor Labels to describe the model and the properties of MyWorker1. One of the Floor Labels displays the current value of the Tally Statistic we added to the MyWorker object. The expression {MyWorker1[1].LateToShift.Average * 60} displays the Average value of the Tally Statistic. Note: The expression must point to a specific member of the MyWorker1 population (in this case [1]). This is because there might be more than 1 MyWorker1 in the model at any given time and this expression needs to know which MyWorker1 to look at.

WorkerUsedForMultipleTasks - SimBit

Problem:

I have a system where my worker is responsible for both transportation type tasks, as well as inspection type tasks. Therefore, the worker must move entities from one location to another, but also must be available at the inspection area to perform stationary tasks.

Categories:

Worker

Key Concepts:

Add-On Process, BasicNode, Decide Step, Is.ModelEntity, On Entered, Request Move, Ride on Transporter, Seize Step, Worker

Assumptions:

There is one worker who is responsible for both transport and stationary tasks. The worker does not take breaks or go off-shift.

Technical Approach:

The worker will be responsible for moving entities from the Source to the Server so that the entities may continue processing at the Server. This will be done by using the Worker as a Transporter type object. Additional entities are sent through a node in need of inspection by the Worker. The Worker will move to the node for inspection and later return to the transport tasks. Add-on Process Triggers at the node will be used to Seize and Release the Worker for this stationary task.

Details for Building the Model:

System Setup

- Add a Source, a Server, and a Sink to the Facility window. Connect the Source to Server and also the Server to the Sink with Paths. Since the Worker will be moving the entities between the Source and Server, change the *Type* of path to 'Bidirectional'. Change the Source1 *Interarrival Time* property to 'Random.Exponential(.5)' minutes.
- Add a Worker object to the Facility window.
- Add a second Source, a BasicNode and a second Sink. Connect Source2 to BasicNode1 and also BasicNode1 to Sink2 with Paths. The Worker's inspection task will occur at the BasicNode. Change the Source2 *Interarrival Time* property to '5' minutes.
- Finally, add a Path between the input node of Server1 and the BasicNode1. This will allow the worker to move between the various tasks. Change the path's *Type* property to 'Bidirectional'.

Requesting the Worker for Transport Task

- Within the output node of Source1 – Output@Source1, change the Transport Logic properties to reflect that a worker is needed to move the entity from location to location. This is done by setting the *Ride on Transporter* property to 'True' and *Transporter Name* to 'Worker1'.

Requesting the Worker for Stationary Task

- Entities coming from Source2 will require the Worker for the inspection process, which will be done at BasicNode1. Within BasicNode1, create a new process in the *Entered* Add-On Process Trigger named 'BasicNode1_Entered'.
- Within the Processes window, place a Decide step into the BasicNode1_Enter process. This will be used determine whether the object entering the node is the entity (coming from Source2) or the worker (coming to do the inspection). The *Decide Type* should be 'ConditionBased' and the *Expression* should be 'Is.ModelEntity'.
- In the True segment leaving the Decide step (which will be the ModelEntity objects), place a Seize step, Delay step, then Release step.
- Within the Seize step, set the *Object Name* property to 'Worker1', the *Request Move* to 'ToNode' and the *Destination Node* to 'BasicNode1'. This will not only give control of the Worker to the entity by using the Seize, but will cause the Worker to move to the entity's location at BasicNode1.
- Within the Delay step, change the *Delay Time* to '2' and the *Units* to 'Minutes'. In the Release step, set the *Object*

Name to 'Worker1'.

- In the False segment leaving the Decide step, there will be no additional logic. Only the token associated with the incoming Worker will be coming out the False exit and no additional logic is required. However, we used the Decide so that the Worker would not also try to Seize (itself) upon entry to the node.

Placing Parking Queues for the Worker

- When Worker1 is not being utilized, it can be animated at its node location by placing a parking queue. In the Facility window, click on the Animation tab and select Detached Queue. Place a queue by left clicking once to start the queue, dragging the mouse and left clicking again to place the end of the queue. Right click to end the queue placement. The *Queue State* for the Source1 location should be 'Output@Source1.ParkingStation.InProcess'.
- Parking areas should also be added to the Server 1 ('Input@Server1.ParkingStation.InProcess') and BasicNode1 ('BasicNode1.ParkingStation.InProcess').

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

WorkerUsesWorkSchedule - SimBit

Problem:

I have a system where my worker is responsible for helping with operations at two different locations. The worker takes breaks and goes off-shift based on a work schedule.

Categories:

Worker, Schedules / Changeovers

Key Concepts:

Off Shift, On Shift, Request Move, ResourceState, Schedules, Secondary Resources, Worker, WorkSchedule

Assumptions:

There is one worker who is responsible who performs stationary tasks at two server locations. The worker goes on break at a different location.

Technical Approach:

The Worker will be responsible for helping processing at both Server1 and Server2. Add-On Process Triggers will be used to specify that the Worker is required at node locations associated with the Servers. A Work Schedule will be defined to specify when the worker is On Shift and Off Shift.

Details for Building the Model:

Simple System Setup

- Add a Source, two Servers, and a Sink to the Facility window. Connect the Source to each Server and also both Servers to the Sink with Paths. Set the *Allow Passing* property on all paths to 'False'.
- Change the Source1 *Interarrival Time* property to '10' minutes. Change the Server1 and Server2 *Processing Time* properties to 'Random.Triangular(6,8,10)'. Keep all *Time Units* as 'Minutes'.
- Add a Worker object to the Facility window.
- Add three BasicNode objects to the Facility Window. Place one directly below Source1, one directly above Source2 and the third one to the right and middle of the other 2 BasicNodes. These will be used as the locations between which the worker moves. Connect the three paths in a triangle by using Paths and change the *Type* of each to 'Bidirectional'.

Requesting the Worker for Stationary Tasks

- Worker1 will be required at for processing at both Server1 and Server2.
- Within Server1, enter the Secondary Resources / Resources for Processing section of properties and set the *Object Name* property to 'Worker1', the *Request Move* to 'ToNode' and the *Destination Node* to 'BasicNode1'. This will not only give control of the Worker to the entity, but will cause the Worker to move to BasicNode1.
- Within Server2, do the same thing except set the *Node Name* to 'BasicNode2' so the Worker moves to the BasicNode2 node to process at Server2.

Defining the Worker's Work Schedule

- Within the Data tab, Schedules panel, click on the Work Schedule button to create a new work schedule with the *Name* 'Schedule1'. Change the *Days* to '1'.
- Add a Day Pattern with the *Name* 'DayPattern1' and click the '+' to enter the information for the day pattern.
- Add four rows to reflect the on shift times, which will all have a *Value* of '1'. The times are 12:00 AM – 12:30 AM, 1:00 AM – 2:00 AM, 2:30 AM – 4:00 AM and 4:30 AM – 12:00 PM. All other times are assumed the capacity value of 0 or off shift.
- IMPORTANT NOTE: With Worker object schedules, the Value field within the Day Pattern should be set at either 0 or 1 (but not greater than 1). This is because the capacity of the dynamic set of Worker objects is based on the Initial Number in System property defined within the Worker. The work schedule is used to define only On Shift and Off Shift times for the 'group' of workers.
- Back in the Work Schedule tab, select 'DayPattern1' for Day 1.

Defining the Worker's Characteristics

- Within the Worker1 object, change the *Capacity Type* to 'Work Schedule' and the *Schedule Name* to 'Schedule1'.
- Within the Routing Logic properties, add an *Initial Node (Home)* property of 'BasicNode3'. Worker1 will start out the simulation at this location. When the worker is idle, it will stay where it is located, so set the *Idle Action* property to 'Remain In Place'. When the worker goes off shift, however, we will move it back to BasicNode3, so change the *Off Shift Action* to 'Go To Home'.

Optional: Changing Worker Animation Symbol

- Change the *Current Symbol Index* property of the worker (Animation) to 'Worker.ResourceState' such that the value of the worker's resource state will be used to animate the worker in various colors depending upon the state that it is in (i.e., Idle, Busy, OffShift, etc.).
- The resource state values for a Worker are Idle = 0, Busy = 1, OffShift = 4, OffShiftBusy = 6, Transporting = 7 and OffShiftTransporting = 9. In this example, the worker doesn't do any transporting, thus states Transporting and OffShiftTransporting are not used. Highlight the Worker1 object and select the Add Additional Symbols button from the Symbols tab. To animate all possible states, there should be 7 symbols (numbered 0 – 6).
- Change the color of the various symbols to reflect the states by clicking on Active Symbol (1 of 7) and selecting the symbol to change. Once the Active Symbol has changed, the Color can be changed by selecting the appropriate color and then clicking on Worker1.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

WorkerUsesWorkSchedule_InterruptWorkingOffShift - SimBit

Problem:

I would like a Server and the secondary resource, a Worker, to stop processing an entity when the Worker goes off shift. When the Worker comes back on shift, the Server and Worker will continue processing the interrupted entity for the remaining processing time.

Categories:

Add-On Process Logic, Schedules / Changeovers, Worker

Key Concepts:

Add-On Process, Assign Step, Candidate, Interrupt Step, Math.If, Off Shift, Picture, Priority, Processing, Ranking Rule, Real State Variable, Release Step, ResourceOwners, Save Remaining Time, Search Step, Secondary Resources, SeizedResources, Status Pie, Transfer Step, Token, Token State, Worker, WorkSchedule

Technical Approach:

Entities are sent to one of two servers, and each server requires the worker to perform a process. Every 30 minutes, the worker goes off shift for 30 minutes. If the worker goes off shift while processing an entity, the process is interrupted, the entity priority is increased, and the entity goes back to the server's input buffer. When the worker goes back on shift, the unfinished entity is processed first for its remaining processing time because it has the highest priority.

Details for Building the Model:

Simple System Setup

- Place a Source, 2 Servers, a Sink, a Worker and 3 BasicNodes into the Facility window. Place BasicNode1 next to Server1, BasicNode2 next to Server2 and BasicNode3 between BasicNodes 1 and 2.
- Using Paths, connect Source1 to both Servers and each Server to the Sink.
- Connect BasicNode1 to BasicNode2, BasicNode2 to BasicNode3, and BasicNode3 to BasicNode1 (creating a triangle). Change the *Path Type* property to 'Bidirectional' for each path connecting the three BasicNodes.
- Place a ModelEntity object from the Project Library into the Facility window.
- Within the Source, change the *Interarrival Time* to '10' minutes.
- Edit the Worker, change the *Initial Node (Home)* property to 'BasicNode3'. Change the *Idle Action* to 'Remain In Place' and the *Off Shift Action* property to 'Go To Home'. Change the *Ranking Rule* property to 'Largest Value First' and set the *Ranking Expression* as 'ModelEntity.Priority'.
- Change Server1 *Ranking Rule* property to 'Largest Value First' and set the *Ranking Expression* as 'ModelEntity.Priority'. Also expand the Secondary Resources property category, and in Resource for Processing set the *Object Name* property to 'Worker1'. Change the *Request Move* to 'To Node' and the *Destination Node* to 'BasicNode1'.
- Repeat the above step for Server2, but set the *Destination Node* property to 'BasicNode2' instead of 'BasicNode1'.

Work Schedules

- Go to the Data Window and click on the Schedules View. Change the first row *Name* on the Day Patterns tab to 'DayPattern1' and expand DayPattern1. In the *Start Time* column set the rows as 12:00 AM, 1:00 AM, 2:00 AM,...,11:00 AM. Set all the rows in the *Duration* column to '30 minutes' and set the *End Time* column as 12:30 AM, 1:30 AM, 2:30 AM,...,11:30 AM respectively. The *Value* of the Worker Capacity during these times is 1 (on shift), otherwise it is 0 (off shift).
- Change the first row *Name* on the Work Schedule tab to 'Schedule1', set the *Days* column to '1', and set the *Day 1* column to 'DayPattern1'.
- Click on the Worker1 object in the Facility window and change the *Capacity Type* property to 'WorkSchedule' and change the *Work Schedule* to 'Schedule1'.

ModelEntity

- Within the Navigation window, click on ModelEntity. On the ModelEntity Definitions window, click the States panel and add a new State (type Real) with the *Name* 'RemainingProcessingTime'. Change the *Unit Type* property to 'Time'. This will be referenced as 'ModelEntity.RemainingProcessingTime' in the main model.
- In the Model Facility window, change the *Processing Time* property of both Servers to 'Math.If(ModelEntity.RemainingProcessingTime > 0, ModelEntity.RemainingProcessingTime, Random.Triangular(16, 18, 20))'.

Add On Process Logic for Worker1

- In the Definitions window, under the Tokens panel, click Add Token to add a new token called 'MyToken1'. Add two Object Reference States and call them 'WorkerInstance' and 'ServerInstance'.
- In the Processes Window, click Create Process and rename the process 'ResourceOffShift'. In the process logic do the following:
 - Open the process properties by clicking the white space in the process. In the Advanced Options, change the *Token Class Name* property to 'MyToken1'. This process will use MyToken1 instead of the base token so the two object reference states can be used.
 - Place an Assign step and set the *State Variable Name* property to 'MyToken1.WorkerInstance' and set the *New Value* to 'Worker'.
 - Place a Search step after the Assign step and change its *Collection Type* to 'ResourceOwners'. This will search for the entity that seized the worker for processing.
 - On the Found segment of the Search step, place another Search step. Change its *Collection Type* to 'SeizedResources' and its *Match Condition* property to 'Candidate.Object != MyToken1.WorkerInstance'. This will search for the objects seized by the associated object (the Entity) but only find objects that are not the Worker.
 - In the Found segment of the second Search step, place an Assign step and set the *State Variable Name* to 'MyToken1.ServerInstance' and *New Value* to 'Server'.
 - Place an Interrupt step after the Assign step. Change the *Process Name* to 'MyToken1.ServerInstance.Server.OnEnteredProcessing' and under Advanced Options, set *Save Remaining Time* to 'ModelEntity.RemainingProcessingTime'.
 - On the Interrupted segment place an Assign step. Change the *State Variable Name* property to 'ModelEntity.Priority' and the *New Value* to '2'.
 - Place a Transfer step after the Assign step. Change the *From* property to 'CurrentStation', the *To* property to 'Station' and the *Station Name* to 'MyToken1.ServerInstance.Server.InputBuffer'.
 - On the OK segment leaving the Transfer step, place a Release step. Click the 3 dots on Resource Releases and add 2 Resource Releases. In the first one, set the *Object Name* to 'MyToken1.WorkerInstance' and in the second one, set the *Object Name* to 'MyToken1.ServerInstance'.
- In the Facility window, select Worker1 and expand the Add-On Process Triggers property category. Set *Off Shift* to process 'ResourceOffShift'. This process will be triggered whenever the Worker changes its state to OffShift.

Animation

- Select the Entity and click the Add Additional Symbol button on the Symbols Ribbon to add a second entity picture. Click the Active Symbol button and select the symbol index 1. Click the Color dropdown and select Red. Then click the Entity Instance to turn the entity red.
- Select a Server object and expand the Add-On Process Triggers property category. Add a process in *Processing* by double clicking the name *Processing*. This automatically creates a new process and moves to the Processes Window. This new process will be triggered whenever the Server starts processing.
 - Place an Assign step in the process and change the *State Variable Name* property to 'ModelEntity.Picture' and the *New Value* to '1'. That will help to visualize the entity that is being processed and locate the interrupted entity when the worker goes off shift.
 - Add the name of the process created above to the other Server object in the *Processing* Add-On Process Trigger.
- Add a Status Pie to the Facility window and change the *Data Type* property to 'ListState' and the *List State* to 'Worker1[1].ResourceState'. Also change its *Text Scale* to '0.75'.
- In the Run ribbon, set the *Speed Factor* to '30'.

WritingToAFile - SimBit

Problem:

I'd like to write values to a file during the run.

Categories:

File Management

Key Concepts:

Assign Step, Event, File Element, Monitor, Write Step, Write Step

Assumptions:

This model writes to a .csv file.

Technical Approach:

This model has a state variable that keeps track of the number of entities in the queue for Server 1. A monitor watches this variable and writes the value of the variable and the current simulation time to a .csv file every time the value of that variable changes.

Details for Building the Model:

Simple System Setup

- Add a Source, Server and Sink object to the Facility Window.

Creating a New State Variable

- Create an Add On Process that is triggered when an entity enters the Input Buffer of the Server.
 - Add an Assign Step to this process and in the *State Variable* drop down, select Create New.
 - Name the new state variable 'ServerQueue'.
 - The *New Value* property should be set to 'ServerQueue + 1'.
- Create an Add On Process that is triggered when an entity hits the Processing station of the Server.
 - Add an Assign Step to this process and select 'ServerQueue' as the *State Variable*.
 - The *New Value* property should be set to 'ServerQueue - 1'.

Creating a Monitor Element

- Create a new Monitor Element from the Definition Window's Element panel. The *Monitor Type* is 'DiscreteStateChange' and the *State Variable* should be set to 'ServerQueue'.

Creating a File Element

- While in the Elements panel, create a new File Element from the User Defined ribbon icon. Set the *FilePath* to a location where you have permission to create a new file. The example uses the file name 'WritingToAFile.csv'.

Adding Logic to the Model

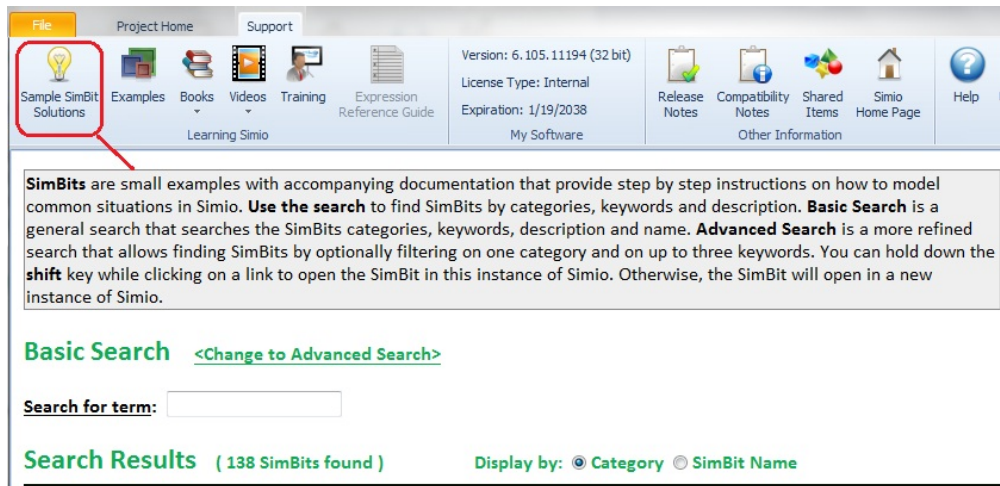
- In the Processes Window, add a new process by clicking the Create Process icon in the ribbon.
 - Set the *Triggering Event* for this new process to be 'Monitor.Event' – where "Monitor" is replaced with the actual name of your Monitor element.
 - Add a Write Step to this process. It can be found in the User Defined Steps. Select the new File element that was created. Add two items to the Items property. The first item should be the 'TimeNow' function, which gives the current simulation time. And the second item should be the 'ServerQueue' state variable.
 - The format property is set to '{0}, {1}'. The {0} is replaced with the value of the first item, in this case, 'TimeNow'. The {1} is replaced with the value of the second item, in this case, 'ServerQueue'. The values are written to the file each time the Monitor element fires its event, which is each time the value of ServerQueue changes.

Sample SimBit Solutions

Sample SimBit Solutions

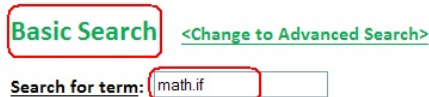
The SimBits icon can be found within the Support ribbon in Simio. By clicking on this button, a window is opened that provides a search mechanism for the SimBit examples within Simio. SimBits are small examples with accompanying documentation that provide step by step instructions on how to model common situations in Simio.

Clicking on a SimBit model name within the Search window will open the SimBit in new Simio window. The new instance of Simio opens in Unlicensed Evaluation mode. To open a SimBit model within the existing instance of Simio, hold down the Shift key while clicking on the SimBit name. Any existing model you have open will be closed. If any changes had been made to the model, the user will have the option of saving before the SimBit is opened.



Basic Search

The Basic Search a general search that allows the users to search categories, Simbits name and Simbits related keywords and descriptions at the same time. The user can enter any word or set of words in the search field. The search engine will look for all the Simbits that contain the word(s) within its keywords, categories or description.



Advanced Search

The Advanced Search is a more refined search that allows the users to look up Simbits by filtering on one category and up to three keywords. The Category field includes twenty four (24) different general areas under which a SimBit may be classified. Some SimBits are shown under multiple categories. The main [SimBits](#) help page shows the various categories and the SimBits within them. The three Keyword fields include more than 250 words that may be used to find a specific modeling concept or Simio step, function or property.

The default behavior searches the SimBits for the **Category AND [Keyword1 OR Keyword2 OR Keyword3]** specified. A User may choose not to enter a category and only search by one or more keywords. This is accomplished by leaving the category field blank.

The "Require all keywords (and)" button allows for "AND" logic search that will list the results of searching for Keyword1 AND Keyword2 AND Keyword3. Thus the default search logic is replaced by **Category AND [Keyword1 AND Keyword2 AND Keyword3]**. The search engine will ignore any empty fields when performing the search, be it a keyword or category.

The 'Reset' button to the right of the keywords will reset all keywords and the category to blank fields.

In the following example, the Category is 'Decision Logic -- Processing' and two keywords are used 'Add-On Process' and 'ResourceState'. This will search only the SimBits within the specified category looking for either of the the keywords specified. If the "Require all keywords (and)" button was on, the search would require both the 'Add-On Process' and 'ResourceState' keyword to be in the SimBit's description, name and/or key concepts section.

Advanced Search [<Change to Basic Search>](#)

Search for:

Decision Logic -- Processing

Custom Statistics
Data Tables
Decision Logic -- Paths
Decision Logic -- Processing
Discrete States
Enterprise

and

Keyword1

Add-On Process
Accumulating
Active Symbol
Add On Process
Add-On Process
AddRow Step
After Processing

Keyword2

ResourceState
Replication
Request Move
Resource
ResourceState
Resources
Response

Keyword3

Accumulating
Active Symbol
Add On Process
Add-On Process
AddRow Step
After Processing

Reset

☐ Require all keywords (and)

Search Results

The number of SimBits found by the search is displayed in the Search Results section. Note that even if a SimBit is displayed in multiple categories, it is only counted as a single SimBit found.

The user is given the option of displaying the list of results by Category or Simbit Name. When displaying the results by Category, some simbits will be displayed more than once as one SimBit can be classified under different categories. The name of the SimBit is listed, along with the general description. Additional information, including the categories, keywords and general description of the SimBit are also shown.

Search Results (20 SimBits found)

Display by: ☒ Category ☐ SimBit Name

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

533 / 1555

Examples

An Example is a complete Simio model that illustrates how to use Simio. Each example has description that explains the details of the model within a *.pdf file that will open automatically when the example is loaded. The example *.spfx files can be found in C:\Program Files\Simio LLC\Simio\Examples.

Below is a list of the Examples.

[AirportTerminal](#)

[ClaudesPizza](#)

[DDMRPExample](#)

[EngineRepairUsingExtrasLibrary](#)

[ExtrasLibrary](#)

[HospitalEmergencyDepartment](#)

[InfectiousDiseasePlanning](#)

[ManufacturingAssembly](#)

[MiningExample](#)

[MultiEchelonSupplyChain](#)

[PackagingSystem](#)

[SampleCoWithDemandDrivenMRP](#)

[SchedulingBatchBeverageProduction](#)

[SchedulingBicycleAssembly](#)

[SchedulingBicycleAssembly_UpdateCreateMOandPOs](#)

[SchedulingDiscretePartProduction](#)

[SchedulingLaborEfficiencies](#)

[SchedulingParallelRouteController](#)

[WarehouseExample](#)

Note that the SchedulingDiscretePartProduction (formerly named RPExample) and other scheduling examples were developed in Simio RPS Edition. On-line help is available on all Simio RPS features discussed in Scheduling examples within the [Scheduling in Simio RPS](#) help sections.

There is a folder within the C:\Program Files\Simio LLC\Simio\Examples area named **ProcessImprovementPrinciples**. This folder contains many small models that correspond to the Process Improvement Principles discussed in the book that is available through the Support ribbon, Books section.

The screenshot shows the Simio Books section with a ribbon at the top containing 'Books', 'Videos', 'Training', and 'Expression Reference Guide'. The 'Books' ribbon is active, displaying a list of books:

- Rapid Modeling Solutions: Introduction to Simulation and Simio**
Learn about Simulation and Simio
- Planning and Scheduling With Simio**
An introduction to Risk-based Planning and Scheduling (RPS)
- Process Improvement Principles**
A guide for using simulation to improve your business
- Deliver On Your Promise**
How Simulation-Based Scheduling Will Change Your Business
- Reference Guide**
Simio Reference Guide
- API Reference Guide**
API Guide for advanced users.
- Online Books**
Simio publications.

A detailed view of the **Process Improvement Principles** book is shown on the right:

Process Improvement Principles

This book introduces 25 process improvement principles which can be applied by managers to improve the design and operation of their production systems. It also describes four case studies illustrating how these principles have been used in real companies.

A list of the 25 principles is provided:

- Principle01VariationDegradesPerformance
- Principle02IncreasingUtilizationIncreasesWIP
- Principle03ConWIP
- Principle04SingleQueue
- Principle05SPTversusFIFO
- Principle06VariabilityDownstream
- Principle07FastServersDownstream
- Principle08BufferSpace
- Principle09BufferingTheBottleneck
- Principle10FeedBottleneck
- Principle11MinimizeChangeover
- Principle12TaskSplitting
- Principle13FlexibleWorkers
- Principle14FlexibleBuffers
- Principle15FlexibleServers
- Principle16TransferBatching
- Principle17PM
- Principle18ReducingNumberOfSteps
- Principle19DecreasingNumberOfTask
- Principle20SlackBasedRules
- Principle21SmallerBatchSize
- Principle22IncreasingCapacityEarly

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AirportTerminal - Example

General Description

This model represents an Airport Terminal. The general process flow is as follows:

- Passengers enter the airport
- Passengers move to one of three check-in areas (2 self check-in and 1 full service counter)
- Passengers move into security check
- Bags are 'separated' from passengers and continue through xray
- Passengers go through xray area
- Bags and passengers are reunited
- Passengers move to one of two large waiting areas for plane departure

* This example includes an 'Experiment' that utilizes input variables to run various configuration scenarios.

Detailed Description

Model Properties

Within the model's properties, there are 4 options that can be set, including the number of self CheckinKiosks, CheckInClerks, IDChecks and ScannerStations. These 4 properties can be varied within the experiment to evaluate various scenarios. Additionally, the value of these properties may change the logic within the model itself.

In addition to the 4 visible properties discussed above, there are a number of hidden properties (Visible property set to 'False') that are used to track the costing information. These are used in the calculations for the Cost, Revenue and Profit output statistics for the simulation model.

Passenger Arrival

Two sources create passengers into the airport – one (EarlyPassengers) that creates 20 groups of between 1-3 to start up the system, the other (Entrance) on an arrival rate basis. Passengers move with their bags from the entry doorway to one of three areas – two self checkin areas (with up to 4 servers each) and one full service counter (with up to 3 checkin clerks).

As the passengers enter the airport, there are three paths. The selection weight on the two of the paths is based on the value of the properties for CheckInKiosks and CheckInClerks. The value of IDCheck passes directly into the capacity of the IDCheck server. The ScannerStations capacity (1 or 2) will turn on/off the path that is leading to the second scanner station. This is done through an add on process for the path and will turn the direction to 'none' if there is only a single scanning station to prevent passengers from entering that path.

Security Area

Upon creation of the passenger, a bag was created and they were batched. After checkin, all passengers move to the security area, where they are moved through one of two xray areas. Bags are 'separated' from the passengers and continue through their own xray process. A separator is used at the scanners to split the passenger and his/her bag. Bags and passengers are reunited after the xray processes. Batch steps at the end of the scanners for both parent and member are used.

Plane Departure Waiting Areas

The passengers 'wait' in the two large waiting areas for a plane. Timer elements are used to schedule plane events, causing the passengers to leave the waiting area once the event occurs.

Experiment

This example is unique in that it has a defined experiment that includes 4 controls, 3 responses and 1 constraint. The controls are the properties discussed above and include the capacities for the main resources within the model. The responses include the costing information related to those resources and are calculated based on the number of each of those resources. The constraint requires there to be at least one clerk for every two checkin kiosks.

If you do not have an OptQuest license, you will not be able to run this Experiment with the OptQuest add-in. If you are interested to learn more about OpQuest, please contact sales@simio.com.

Send comments on this topic to [Support](#)

ClaudesPizza - Example

General Description

This model represents a Pizza by the slice restaurant. The general process flow is as follows:

- Group of customers enter the restaurant
- Individual customer places an order
- Customer pays for order
- Employee places slices in the oven to reheat
- Customer waits for Pizza to come out of the oven
- Once reheating is complete the customer waits for the rest of the group (or joins waiting group)
- Group sits down at table (or waits for an available table)
- Group eats pizza
- Throws away trash at the trash can
- Leave restaurant

*When the Trash Cans have 15 or more Dirty Trays, an employee will collect them and bring them back to the counter.

Detailed Description

Creating Unique Individuals in Groups

Groups enter the restaurant, each group consists of 1 to 4 customers, and each customer will order 1 to 3 pieces of pizza. Upon entering the restaurant (Before Exiting the Source) we assign CustomerEntity.GroupSize, CustomerEntity.GroupNumber, CustomerEntity.EatingTime, and CustomerEntity.Picture. Then, in the Output@Source Node we create a process that creates the rest of the group via a Create step set to Create Copies. We then re-assign a Picture so they are not the same throughout the group. This process allows the group to have the same values for the GroupSize, GroupNumber, and EatingTime which all come into play later in the model. Then, at various points in the model we assign State values for the number of pieces of pizza and if the customer is considered a parent. By Creating Copies before assigning these States, we allow some parts of the entities to be the same, and others to differ.

Creating the Pickup Counter Object

After the Pizza has been taken out of the oven, the pieces go onto a Tray according to the State TrayEntity.OrderSize that is based off the state CustomerEntity.Ordersize (which is assigned when the customer enters the Path leading to the Cash Register). Then the customer carries the Tray (that is carrying the Pizza) to the Table. This can be done using 2 Combiner Objects, or in this model we made one simple object that combines all three Entities.

There are three Entities that are going to enter this object with two separate Batch Logic Elements that are to be used to join the Entities together - first the Pizza gets batched to the Tray, then the Tray gets batched to the Customer. The PickupCounter Object uses Station Elements to do the batching, each Station is designed to do a portion of the batching.

There are 3 Input Nodes: one for Tray, Pizza, and Customer. Each one of these nodes corresponds to an InputBuffer where the object waits for the corresponding Parent/Member for its batch logic. After the Tray and Pizza are combined, the pair is transferred to an intermediate station where it attempts to Batch the Pizza/Tray as a Member with the Customer. After everything is batched together, the Customer is transferred to the OrderFulfilled node.

Creating the Trash Can Object

The Trash Can is Similar to the Counter except it was created using Standard Library Objects instead of processes. The trash can consists of 2 separators, the first one separates the customer from the Tray and Pizza, and the second separates the tray and pizza – with the pizza going into a Sink to destroy it.

Then, we defined an external view with one input node and 2 output nodes: one for the customer that leads to the Exit, and one for the tray which (in the model) is set to RideOnTransporter = True so that the worker can come pick up the waiting trays.

Creating the Table Objects

After the group is together (leaves the WaitForGroup Combiner), the group is routed to an available table. After routing to the table, the group is Unbatched so that they can each sit in their own seat at the table. One difficulty is that if the group is smaller than 4 members, there is still remaining capacity at the table and some groups could be routed to the table even though it is technically occupied. This requires some add-on processes to a standard Server Object. To avoid having to create add-on processes at each table, we created a new table object.

This object consists of two stations, SplitStation and EatingStation. When the SplitStation is entered, we Unbatch the group and transfer the Parents and Members to the EatingStation. After the transfer we assigned the SplitStation.CurrentCapacity to 0 – this means that the table is no longer considered available and no customers can be routed to this table, even if there is available capacity.

Then, after Delaying for EatingTime, we assign the Capacity of the SplitStation back to its original value, signifying that the table is available for new customers.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

DDMRPExample - Example

DDMRPExample - Single and Multiple Factory Examples

General Description

This example uses Simio RPS Edition to build a data-generated model for orders, within multiple manufacturing facilities and a distribution center, using Demand-Driven Material Requirements Planning (DDMRP) methodologies.

Simio RPS Edition is a simulation tool for developing applications in Risk-based Planning and Scheduling. RPS is the dual use of a simulation model to generate both a detailed resource constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty. DDMRP calculators and Inventory Review Log used within this example are restricted to Simio RPS Edition licensing. *Note that the ability to change or run this example requires an RPS license. Users with other licensing may only review the example, data tables and results.* For more information or questions about Simio Editions, contact sales@simio.com.

To understand the data schema and scheduling results in this example it is recommended to read the "Planning and Scheduling with Simio" document in C:\Program Files\Simio LLC\Simio. This problem description assumes familiarity with the standard data schema and scheduling concepts that are presented in that document.

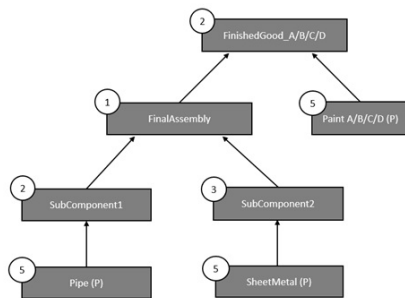
This example is intended to be a guide for implementing DDMRP in Simio. If you would like to learn more about DDMRP methodology in general, Demand-Driven Material Requirements Planning (DDMRP) concepts can be found at [Demand Driven Material Requirements Planning \(DDMRP\) \(demanddrivenmrp.com\)](http://DemandDrivenMaterialRequirementsPlanning.com). Additional information about the Demand Driven Institute can be found at [The Demand Driven Institute - World Leader in Demand Driven Education](http://TheDemandDrivenInstitute.com). Simio is certified as a [DDMRP Compliant Software vendor DDMRP Compliant Software \(demanddriveninstitute.com\)](http://TheDemandDrivenInstitute.com).

It is intended to create a six-week production schedule that fully accounts for the resources in the system and decoupled buffers at strategic points within manufacturing and warehouse inventory locations.

This example project includes two separate models of similar processing within the factory manufacturing. These models include a simple example, OneFactoryExample, that produces two types of finished goods within a factory with a warehouse. Each finished good consists of a final assembly and associated paint type (purchased material) for the finished good (A/B). Final assemblies are made of two manufactured subcomponents, Subcomponent1 and Subcomponent2. Purchased materials of Pipe and SheetMetal are required for processing the subcomponents, respectively. All orders are fulfilled from the single factory/warehouse. In the second model, there are two factories that produce four types of finished goods (A/B/C/D), as well as warehouses associated with each manufacturing factory and a distribution center that holds those same finished goods.

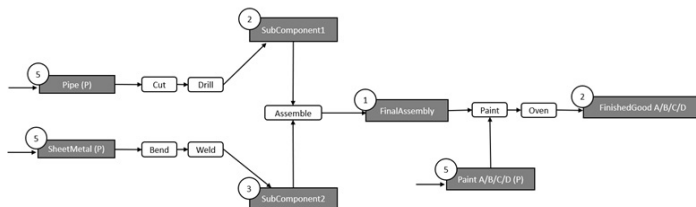
The diagram below shows the product structure and lead times associated with each material in the system.

DDMRP Product Structure and Lead Times



The below Routing diagram shows manufacturing steps associated with each material, including the finished goods, final assembly, and subcomponents. (NOTE: The first model, OneFactoryExample, includes only FinishedGood_A and FinishedGood_B, as well as PaintA and PaintB).

Routing BOM and Lead Times



Each of the manufacturing facilities, or factories, consists of functionally grouped workstations for each of the materials manufactured, as shown in the diagram above. Subcomponent1 material processes through Cut and Drill workcenters, while Subcomponent2 required Bend and Weld workcenters for processing. Both Subcomponent1 and Subcomponent2 are required for the FinalAssembly material to process at the Assemble area. FinalAssembly material and one type of paint (A/B/C/D) are then combined and processed through the Paint and Oven workcenters to become FinishedGoodA / FinishedGoodB / FinishedGoodC / FinishedGoodD. Each workstation has a staging area from which orders are routed to one or more machines based on dispatching logic. Within the OneFactoryExample, there are single machines at Cut, Drill, Bend, Paint and Oven, and two Weld machines. Within each factory of the TwoFactoryAndDCExample, there are two machines in Cut, two in Drill, two in Bend, three in Weld, two in Assemble, two in Paint and one Oven. Decoupling points used for holding inventory of each completed material (or raw material) can be easily specified. Demand-Driven MRP replenishment policies and DDMRP inventory levels can be specified at each decoupling point, or alternative replenishment policies (Min/Max for example) can be used.

The template used to start building this model provides custom Object definitions created by using the Subclass feature on Standard Library Objects. These are called SchedSource, SchedServer, etc. and have been specifically designed for use in scheduling models. Additionally, the Site object was designed specifically for multi-site ordering and transfers.

The staging areas are modeled using a SchedTransferNode while the machines are SchedServer Objects. In this document, let "machines" be used to refer to all SchedServer Objects in the model. Machines may require a Secondary Resource, modeled using a SchedWorker in this case, to complete one or more of its Processing Tasks. Additionally, each machine may have a sequence dependent setup time specified in a changeover matrix and based on material color. Material colors are defined in a string list named MaterialColor with values 'Turquoise', 'Gold', 'Blue' and 'Pink', as specified in the Materials table.

The subcomponent and finished goods materials produced in each facility are described above. Each material within a given site has its own routing, unique setup/processing times, and material requirements at each machine within its routing. The ISA95SchemaProductBasedRoutings template (File -> New From Template) discussed in the Planning and Scheduling with Simio document is initially used to organize the production data for this example, with the difference being that the Routings are inventory-based (material/site combination) and not simply material-based.

Additional tables are used specifically for Demand-Driven MRP decoupling point inventory areas. A DDMRP ribbon includes specific buttons for calculating Average Daily Usage, Decoupled Lead Times, Buffer Zone Sizes and Qualified Spike Demand. The Average Daily Usage, or ADU, Calculator utilizes data from the Demand table, as well as Inventories table parameters. The Decoupled Lead Times Calculator utilizes data from the Inventories table lead times and decoupling points as well as the ProductStructure table. The Buffer Zone Sizes Calculator utilizes data from a combination of parameters including ADU, Decoupled Lead Times and the Planned Adjustment Factors table. The Qualified Spike Demand Calculator uses data from the SalesOrders table, as well as Inventories table parameters. Finally, the Calculate All button performs all calculations for the forementioned buttons in the order specified.

These example models illustrate the concept of "data-generated" modeling where the model is created automatically by storing Object data in tables, namely Resources, Employees and Vehicles. In this example, the objects that are created from these tables are already mapped to the other standard scheduling data tables.

Detailed Description

The model was originally started by using the File > New From Template > ISA95SchemaProductBasedRoutings template. From this template, the basic structure of the Resources, Routing Destinations, Materials, Manufacturing Orders (now Sales Orders), Routings and Bill of Materials tables were developed. As with any model developed with the ISA95Schema templates, these base tables were then slightly modified to accommodate the Demand-Driven MRP example. Notable changes to the few tables listed include a Site Id for the main equipment in the Resources table, lot sizing information in the Materials table, inventory-based routings (instead of material-based routings), and the Sales Orders table (originally called Manufacturing Orders) simply ships finished goods materials. Inventory replenishment, as defined through a new Inventories table, along with custom process logic, is used to generate three output data tables, including Manufacturing Orders, Purchase Orders and Stock Transfer Orders.

Resources, Employees and Vehicles Tables

Note that the Resources table can be used to generate any 'Object Type' including SchedServers, SchedTransferNodes (as in the example) but also SchedWorkers and/or SchedVehicles. We have broken out the Employees and Vehicles generation into their own separate tables, with custom object properties specifically related to each. For example, the Employees table includes Home Location, as well as Initial Speed. An Employees Schedule can be used to specify shift schedules and availability for each type of worker in the system. The Employee Details table allows for specific Workcenter/Resource allocations and efficiencies for the employees. Thus, employees can have specific resources at which they are 'qualified' to work and associated labor efficiency for that resource. (*Note: The OneFactoryExample does not include workers, while the TwoFactoryAndDCExample includes workers and their schedules / details). The Vehicles table includes custom properties such as initial ride capacity, load and unload times, dwelling information (waiting at a site for a given amount of time for X orders), as well as home location and initial speed. A separate Resource Calendar table, as well as Resource Exceptions are used to specify resource availability and planned downtimes, respectively.

Sites and Suppliers Tables

The Sites table and an associated Site object are used to generate locations for materials within the system. Each Material defined within the Materials table can then be associated with one or more Site. The Inventories table, discussed below, then tracks the combination of material/site in the system. The Sites table allows for Maximum Capacity Specifications, as well as a Destination Vehicle for the site (used when material is being moved from that site to another site). While this example includes maximum capacity values, it currently uses that information only for the Warehouse Capacity dashboard. Custom logic could be added to delay/remove orders when site capacity exceeds specifications.

Inventories and DDMRP Specific Tables

The Inventories table is not specifically a table for DDMRP, however, the Inventory element (generated automatically from the table data) includes the replenishment policy for the material/site combination and may be specified as Demand-Driven MRP. The Inventories table structure also includes DDMRP related parameters, such as Lead Time, Minimum Order Quantity, Desired Order Cycle, Buffer Profile information, Average Daily Usage horizon/weight factors and Spike horizon parameters. If an inventory is specified as Demand-Driven MRP replenishment, these parameters as data within the tables discussed below, will be used to generate the time-indexed output tables specific to demand driven inventory calculations.

The ProductStructure table is like the Bill of Materials table and can be created by simply importing BillOfMaterials table, filtering by MaterialUse of 'Consume' and copy/pasting the Parent Material Name and Component Material Name fields. Note that it is assumed that product structure for a given Material name is consistent between Site locations. In this example, the BillOfMaterials table is Routing-based (and associated Routings table is Inventory-based), whereas the ProductStructure table is only Material based.

The Demand table includes prior demand (for 'Past' Average Daily Usage calculator) as well as future demand. It differs from Sales Orders in that it includes the full bill of materials explosion of demand for all inventories, including subcomponents and purchased parts. While this Demand table isn't considered a time-indexed table, it is time-based and should include the Date for each demand entry. A missing date indicates a value of '0' and will impact the Average Daily Usage calculations.

PlannedAdjustmentFactors are specifically for adjusting the demand, red/yellow/green zones or lead time factors associated with buffer zone sizes. This is a time-indexed table and based on the StartDate and EndDate for the adjustments.

The BufferProfiles table contains information related to the Lead Time Factor and Variability Factor used in Buffer Zone Sizes calculations. The Key column in this table is referenced from within the Inventories table for each inventory (material/site) defined. Default data for purchased, manufactured, distributed and intermediate part types has been defined for lead time categories (short/medium/long) and variability categories (low/medium/high). Additional profiles may be added.

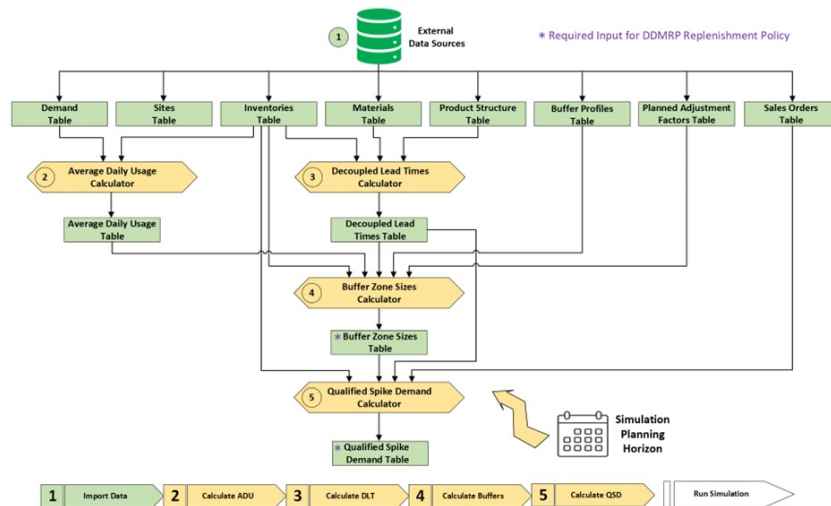
DDMRP Calculated Output Tables

The DDMRP calculators are used to generate the associated output tables. The next section discusses the calculator in detail. The AverageDailyUsage table is time-indexed and is used for calculating the BufferZoneSizes table. The DecoupledLeadTimes table is based on the Lead Times and Decoupling Point Boolean within the Inventories table. This information will also be used for calculating the BufferZoneSizes table. The BufferZoneSizes and QualifiedSpikeDemand tables are also time-indexed, and their data is fed directly into the Inventory elements that are generated via the Inventories table (see below, all replenishment policy property values are input here – do not change these, this is FYI only).

The screenshot displays the 'Replenishment Policy' dialog box in the Simio software. The dialog is titled 'Default Values Of Dynamic Properties - Repeating Property Editor'. It features a list of items on the left, including 'RedZoneSize: BufferZoneSizes.RedZoneSize.TimeIndexedValue', 'YellowZoneSize: BufferZoneSizes.YellowZoneSize.TimeIndexedValue', 'GreenZoneSize: BufferZoneSizes.GreenZoneSize.TimeIndexedValue', 'QualifiedSpikeDemand: QualifiedSpikeDemand.Quantity.TimeIndexedValue', 'ReorderCondition: Inventories.ReorderPointOrCondition', 'ReorderPoint: Inventories.ReorderPointOrCondition', 'ReorderQuantity: Inventories.ReorderQuantityOrUpToLevel', and 'OrderUpToLevel: Inventories.ReorderQuantityOrUpToLevel'. The 'General' tab is selected, showing a 'Name' field with 'RedZoneSize' and a 'Value' field with 'BufferZoneSizes.RedZoneSize.TimeIndexedValue'. The 'Properties' section on the right shows 'Default Value' as 'None' and 'Default Values Of Dynamic Properties' as '8 Rows'. The 'Appearance' section shows 'Display Name' as 'Replenishment Policy'. The 'Operational Planning' section shows 'Tables' as 'True', 'Visible' as 'False', 'Editable' as 'False', 'Category Name' as 'Basic Logic', 'Gantt Charts' as 'False', 'Visible' as 'False', 'Target Views' as 'False', 'Visible' as 'False', and 'General' as 'ReplenishmentPolicy'.

Simulation Input Data Preparation Using DDMRP Calculators

The diagram below shows the relationship between the various data input tables, calculators and output tables.



DDMRP Calculators

The following includes more detailed information about the DDMRP calculators, associated parameters and actual calculations.

ADU CALCULATOR

A past ADU is calculated using the following equation:

$$\text{Past ADU} = \frac{\text{Sum of Past Horizon Demand}}{\text{Past Horizon}}$$

A forward ADU is calculated using the following equation:

$$\text{Forward ADU} = \frac{\text{Sum of Forward Horizon Demand}}{\text{Forward Horizon}}$$

A blended ADU is calculated using the following weighted average equation:

$$\text{Blended ADU} = \frac{(\text{Past ADU} * \text{Past Weight}) + (\text{Forward ADU} * \text{Forward Weight})}{\text{Past Weight} + \text{Forward Weight}}$$

DECOUPLED LEAD TIME CALCULATOR

Decoupled lead time (DLT) is a qualified cumulative lead time concept in DDMRP. For manufactured items, it can be defined as: The longest cumulative coupled lead time chain in the product structure, limited and defined by the placement of decoupling point buffers within that structure.

Any manufactured item with at least one coupled component will always have a longer decoupled lead time than its manufacturing lead time. If an inventory item is not manufactured (i.e., is replenished by purchase or stock transfer orders) then the decoupled lead time is simply the purchasing or transfer lead time.

BUFFER ZONE SIZES CALCULATOR

The equations used to calculate the buffer zone sizes are as follows:

$$\text{ADU} = \text{ADU} * \text{Demand Adjustment Factor}$$

$$\text{DLT} = \text{DLT} * \text{Lead Time Adjustment Factor}$$

$$\text{Red Zone Base} = \text{ADU} * \text{DLT} * \text{Lead Time Factor}$$

$$\text{Red Zone Safety} = \text{Red Zone Base} * \text{Variability Factor}$$

$$\text{Red Zone Size} = (\text{Red Zone Base} + \text{Red Zone Safety}) * \text{Red Zone Adjustment Factor}$$

$$\text{Yellow Zone Size} = \text{ADU} * \text{DLT} * \text{Yellow Zone Adjustment Factor}$$

$$\text{Green Zone Size} = \text{MAX}(\text{Minimum Order Quantity}, \text{ADU} * \text{Desired Order Cycle}, \text{ADU} * \text{DLT} * \text{Lead Time Factor}) * \text{Green Zone Adjustment Factor}$$

The calculated buffer red, yellow, and green zone sizes are rounded to the nearest integer value.

QUALIFIED SPIKE DEMAND CALCULATOR

The order spike horizon (a future time window starting tomorrow) is calculated using the following equation:

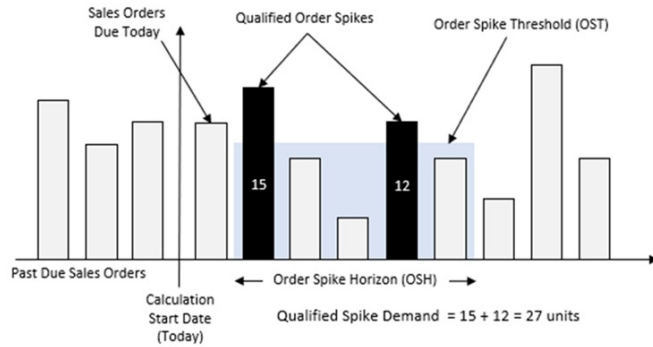
$$\text{Order Spike Horizon (OSH)} = (\text{DLT} * \text{Spike Horizon Factor}) + \text{Spike Horizon Constant}$$

The OSH is rounded to the nearest integer value.

The order spike threshold is calculated using the following equation:

$$\text{Order Spike Threshold (OST)} = (\text{Red Zone Size} * \text{Spike Threshold Factor}) + \text{Spike Threshold Constant}$$

The calculated qualified spike demand values are rounded to the nearest integer value.



Additional Tables for Dashboards

Additional output tables for dashboards have been generated for the purpose of displaying information on costing, including costing by material site and by resource.

Processes Window

The following is a brief description of the processes that are included with the DDMRP Example.

OnRunInitialized – This process executes the SupplierLeadTimes process at the start of the simulation run.

SupplierLeadTimes – This process searches through the Suppliers table and will populate the In_LeadTime (in stock) and Out_LeadTime (out of stock) for the Sites in the model.

DailyTimer_Cost and **DailyTimer_ITRandCPM** – This process is called daily (from DailyTimer.Event) and is used to calculate daily costing for costing by materialsite and costing by resource. It also executes the DailyTimer_ITRandCPM process, used for calculating DDMRP specific parameters of ITR (inventory target ratio) and Cpm (Taguchi capability measure that indicates how well the process is behaving within the specifications along with how close the process average is to the actual target value).

OnCustomerOrder – This process consumes the inventory for the order.

Component_MfgEnd – This process calculates the completion time for manufactured parts and decreases the work in process.

Manufacture_Subcomponents – This process is called from the Inventories table for manufactured subcomponents (user may customize their own process, if desired). This process will split the recommended order quantity for DDMRP components into lot sizes (as specified in the Materials table) and add rows to the ManufacturingOrders table for any subcomponents that are ordered. It will also create the corresponding order entity and transfer it to the MoArrivals node which then transfers it to the inventory's first sequence-based routing step. Note that the Token.MaterialOrder detail is used within this process for any replenishment orders.

MfgStarted – This process is called from the Site object for finished goods and will add rows to the ManufacturingOrders table for those finished good materials.

ResupplyStarted – This process is called from the Site object for stock transfer orders and will add rows to the StockTransferOrders table for those finished good materials.

UpdateNextDelivery – This process is called when a delivery is received (transfer completed) from another Site. For purchase orders arriving to sites, it updates the arrival date information. For stock transfer orders, it updates the completion time.

MoArrivals_Entered – This process sets the destination inventory location for those orders that do not automatically have it set from a stock transfer.

Produce_RawMaterials – This process is called from the Inventories table for purchased parts (user may customize their own process, if desired). This process will add rows to the PurchaseOrders table for any purchased parts that are ordered. It will also create the corresponding PO entity, delay for the lead time and Supplier output node for transfer to the appropriate site.

EvaluateSupplierLeadTime – This process is called from the OnInventoryReplenishment process (see below description) to evaluate in stock/on order/out of stock status for inventory of finished goods.

OnInventoryReplenishment – This process is called from the Inventories table for finished goods parts (user may customize their own process, if desired). This process determines lot size and supplier information (per lead times) and then sends a token into the Fulfillment station of the Site object (from which the order will be fulfilled – either manufactured or transferred. Note: See the Site object OnFulfillment process (triggered by FulfillmentStation.Entered) for more details.

CheckSequence – This process is an add-on process trigger within the output node of the SchedServer object (Entered) and determines if the order (entity) has any additional routing steps, if not, the order is completed and destroyed. This eliminates the need for a separate 'end' type step/station for the routings in the sequence.

AtEndOfRun – This process sorts and configures the Cpm calculations at the end of the simulation run using the Cpm/CpmSorted tables for end of run statistics and dashboard information. A tally named Cpm_Overall is also calculated for all decoupling point inventories.

Appendix A – Data Tables

The following is a summary of the default tables for the Demand-Driven MRP template.

Resources:

A list of 'resources' or SchedServer objects that are in the manufacturing facility, can also include SchedTransferNodes and SupplierSource objects.

Column Name	Description
Resource Name	The unique name of the resource. This is the object name.
Description	A description of this resource.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Site Id	The Site object associated with this resource.
Object Type	This references an object type (sub-classed object) of the object.
XLocation	This is the X location of the object in the model.
ZLocation	This is the Z location of the object in the model.
Work Schedule	Work schedule assigned to the resource.
Capacity	Capacity of the resource.
IdleCostRate	The hourly idle cost rate when a resource is idle.
CostRate	The hourly rate for the resource while being utilized.

Changeover Matrix	Changeover Matrix used by the resource.
Supplier Vehicle	This is the vehicle/worker that is used by a SupplierSource object for transport to a site.
Changeover Efficiency	State column, which is a factor used to increase or decrease the total time taken for a setup.
Current Idle Cost	Costing information tracked for daily idle statistics (an output).
Current Usage Cost	Costing information tracked for daily utilization statistics (an output).

Routing Destinations:

A list of each possible destination node for each resource. The entity is routed to one of the nodes specified in this list based on the selection rule.

Column Name	Description
Resource Name	This is the transfer node used to route each entity to their destinations.
Node	Possible destination from this transfer node.

Materials:

A list of materials that can be produced at any manufacturing facility.

Column Name	Description
Material Name	The unique name of this material.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Material Class	A description of this material.
Infinite Availability	Boolean to determine whether the material availability should be considered infinite (and thus will not constrain the system per Bill Of Materials).
Material Cost	The cost per unit of this material.
Material Color	The color index for computing color-dependent changeovers.
Gantt Color	The color used for drawing orders for this material in the Gantt.
Material State Statistic	State statistics are used to plot material quantities over time on the Gantt.
Minimum Split Lot Size	Minimum lot size for determining order lot size from recommended order quantity.
Maximum Split Lot Size	Maximum lot size for determining order lot size from recommended order quantity.
Rounding Split Lot Size	Rounding split lot size for determining order lot size from recommended order quantity.
Use Remainder Instead of Rounding Lot Size	Boolean to determine whether remainder will be used when lot sizing for full recommended order quantity. If this Boolean is checked, all smaller lots will equal the recommended order quantity, which otherwise may be slightly smaller value.
Symbol Index	Entity symbol index used for orders of given material name.
Raw Material Supplier	Name of the SupplierSource for which a raw material is sourced (see Purchase Orders table). Used for raw materials only.

Routings:

A list of job routings for each inventory (material/site combination) specifying the resource and processing time for each task required to produce the material.

Column Name	Description
Routing Key	The unique name for this routing step.
Sequence	The transfer node (list of resources) or resource where this step is to be processed.
Inventory Name	The foreign key property specifying the inventory (material/site) for the specified routing step(s).
Route Number	The routing number.
SetupTime	Discrete setup time for this step. (see Definitions > SequenceDependentSeutpMatrix for more information).
ProcessTime	Processing time for this step. In this example, the process time is multiplied by the ManufacturingOrders.Quantity, thus process time is 'per unit'.

Bill Of Materials:

A list of component materials that are required at a specific routing sequence location to produce a material.

Column Name	Description
Routing Key	The material routing step where the material action takes place.
Parent Material	The name of the parent material that is associated with the routing step (used for reference only).
Component Material	The name of the component material to be consumed or produced.
Required Quantity	The quantity of the material that is either consumed or produced. This value based on an order quantity = 1. The required quantity is multiplied by the ManufacturingOrders.Quantity.
Required Lot Id	Specified the lot id required for consumption and the lot id provided for production.
Material Use	Specifies if the material is to be consumed or produced.

Product Structure:

A list of pairs of parent/component materials that are used for determining decoupled lead times.

Column Name	Description
Parent Material Name	The name of the parent material associated with the routing step (used for reference only).
Component Material Name	The name of the component material to be consumed or produced.

Sales Orders:

A list of all sales orders to be processed during this planning period.

Column Name	Description
Enabled	Boolean that determines if the order will be enabled/used in the simulation run.
Order Id	A unique string name assigned to this sales order.
Inventory Name	A foreign key reference to the Inventories table for the material/site from which the order will be shipped.
Release Date	The date-time this order is released to production.
Due Date	The date-time this order is due.
Priority	The scheduling priority for this order.
Quantity	The material quantity to be produced for this order.
Adjusted Release Date	This field is an expression field used by the source to determine when to release orders considering any orders whose release date is earlier than the start date (will attempt to ship those as well using Math.Max expression.
Ship Date	The ship date for this order (an output) based on the generated schedule.
Production Cost	The production cost for this order (an output) based on the generated schedule.
Target Ship Date-Value	The target ship date for this order.
Target Ship Date-Status	The target ship date status; OnTime, Late, or Incomplete.
Target Cost-Value	The target cost for this order.
Target Cost-Status	The target cost status: OnBudget, Overrun, or Incomplete.

Note that the *Ship Date* and *Production Cost* are both output columns in the Sales Orders table; they are written from the simulation model to the table as the schedule is generated by the deterministic simulation run. Also note that the last four columns are targets.

Inventories:

A list of inventories that are specific materials at each site, along with many associated DDMRP parameters.

Column Name	Description
Inventory Name	The unique name of this inventory ("material_site" plus "site_material" recommended).
Material Name	The name of the material for this material/site inventory.
Site Object Name	The name of the site for this material/site inventory.
Initial Quantity	The quantity of material present in the inventory at this site at the beginning of the simulation run.
Review Period	The frequency of inventory review to determine whether a replenishment order is required. If the review period is 'Continuous', then an inventory review is triggered once at the start of the simulation and then whenever the inventory position decreases. If the review period is 'Timer', then an inventory review is triggered whenever the specified timer fires its event.
Review Timer Name	The name of the timer that is used to trigger inventory reviews. For any Demand-Driven MRP replenishment, it's recommended using the DailyTimer.
Replenishment Policy	The replenishment policy checked at the time of an inventory review to determine whether a replenishment order is required and, if so, the recommended order size. If the Decoupling Point Boolean is 'True', this should be set to 'Demand-Driven MRP'.
Reorder Point or Condition	For Reorder Point/Reorder Quantity or Min/Max replenishment policy, the minimum threshold for the inventory position that signals the need to place a replenishment order. For Custom Reorder Condition, the condition that, if true, signals the need to place a replenishment order.
Reorder Quantity / Up-To-Level	For Reorder Point/Reorder Quantity or Custom Reorder Condition replenishment policy, the fixed reorder quantity when placing a replenishment order. For Min/Max or Order-Up-To-Level, the target level to return the inventory position to if below that level.
On Replenishment Order Process	The name of the process that will be executed to handle an inventory replenishment order. Note that whenever this process is triggered by the replenishment policy, the inventory's quantity on order will have been automatically increased by the size of the recommended order size. (Note: the created token's material order detail reference will provide the detail of the replenishment request.)
Sourcing Policy	Currently, it can be specified as PreferredCapableToPromise or SmallestLeadTime. The example will be the highest priority supplier that can deliver within the planned lead time window, if none are within the window select the smallest lead time.
Manufactured	Boolean specifying whether or not the inventory (material/site) is manufactured or not. This entry is used within the DDMRP calculators.
Decoupling Point	Boolean specifying whether or not the inventory (material/site) is decoupled or not. This

	entry is used within the DDMRP calculators.
LeadTime (Days)	The number of days of lead time to either manufacture, purchase or transfer the inventory (material/site). This entry is used within the DDMRP calculators to determine Decoupled Lead Time.
Minimum Order Quantity	The minimum order quantity for the inventory (material/site). This entry is used within the DDMRP calculators for determining the Green Zone size (if large enough, will factor into this size).
Desired Order Cycle (Days)	This order cycle determines how often an order should be placed when using DDMRP calculators. It may factor into the Green Zone size.
Buffer Profile Key	The key column specified within the Buffer Profiles table that will indicate the lead time and variability factors within DDMRP calculators.
Estimated ADU	This field will be used for a Demand-Driven MRP based inventory if there is not Demand information within the Demand table.
ADU Calculation Type	Average Daily Usage (ADU) calculation type can be Forward, Past, Blended or None. For Forward calculations, the Forward Horizon (Days) is used. For Past calculations, the Past Horizon (Days) is used. For Blended, both the Forward Horizon (Days) and Past Horizon (Days) are used, as well as the Past Weight and Forward Weight. For None, no ADU will be calculated.
Past Horizon (Days)	The number of past days to consider when performing a 'Past' or 'Blended' ADU calculation.
Forward Horizon (Days)	The number of forward days to consider when performing a 'Forward' or 'Blended' ADU calculation.
Past Weight	The weight associated with the 'past' data when performing a 'Blended' ADU calculation.
Forward Weight	The weight associated with the 'forward' data when performing a 'Blended' ADU calculation.
Spike Horizon Factor	The spike horizon factor is used within the Qualified Spike Demand DDMRP calculator and is multiplied by the DLT (decoupled lead time) within the Order Spike Horizon.
SpikeHorizonConstant (Days)	The spike horizon constant is an optional constant used within the Qualified Spike Demand DDMRP calculator and is added to the (DLT * Spike Horizon Factor) within the Order Spike Horizon calculation.
Spike Threshold Factor	The spike threshold factor is used within the Qualified Spike Demand DDMRP calculator and is multiplied by the Red Zone Size within the Order Spike Threshold.
Spike Threshold Constant	The spike threshold constant is an optional constant used within the Qualified Spike Demand DDMRP calculator and is added to the (Red Zone Size * Spike Threshold Factor) within the Order Spike Threshold calculation.
On Hand Tally	The name of the automatically generated tally for calculating on hand values for inventory target ration (ITR) over time.
On Hand Squared Tally	The name of the automatically generated tally for calculating on hand squared values for inventory target ration (ITR) over time.
Cpm	Daily updated with Cpm (Taguchi Capability Matrix) values until the end of the simulation run (an output).
Next Scheduled Delivery	Tracks the next scheduled delivery of materials (an output).
CpmSorted	Boolean used for tracking and sorting Cpm (Taguchi Capability Matrix) at end of simulation run – used for dashboard creation of green/yellow/red areas.

Sites:

A list of distribution sites that are either stand alone or attached to a production facility/factory.

Column Name	Description
Site Object Name	The unique name of the site. This is the object name for the Site.
Latitude	The object's latitude, in degrees (-90 to +90).
Longitude	The object's longitude, in degrees (-180 to +180).
Factory Start Node	For those sites with attached factories, the name of the node for the factory. In this example/template, the MoArrivals node should be used.
Maximum Work In Process	Associated with the Site object, this field is currently not used and set to Infinity.
Maximum Capacity	The maximum capacity of the site, which is used in the Warehouse Capacity dashboard to display the % of inventory at that site by material. Note that materials are not stopped from transferring or manufacturing due to this capacity (additional custom logic could be added), but this value is used for dashboards only.
Destination Vehicle	The name of the vehicle/worker that transfers inventory from this site to another site.

Manufacturing Orders (Output Table):

A list of all manufacturing orders generated during the simulation run to manufacture, which includes finished goods (within a factory at a given site) or component materials.

Column Name	Description
Order Id	A unique string name assigned to this manufacturing order.
Inventory Name	A foreign key reference to the Inventories table for the material/site for which the order will be manufactured.
Material Name	The material name associated with this manufacturing order.
Site Id	The site id associated with this manufacturing order.
Quantity	The number of inventory parts associated with this order. Note: This value is impacted by the Lot Sizing associated with the Material and may/may not match the recommended order quantity.

Type	This example currently only generates DemandDriven type manufacturing orders.
Release Date	The date of release of the manufacturing order to the manufacturing/factory floor and is based on the review date when the recommended order quantity was generated (typically daily).
Due Date	The due date of the order, based on the release date + lead time of the inventory (material/site).
Completion Date	The date when the manufacturing order was completed.
Priority	Priority of the manufacturing order (not currently used, as manufacturing orders are not directly associated with Sales Orders with priority).
Complete	Boolean indicating if the manufacturing has been completed.
Planning Priority	Priority based on DDMRP planning, including NetFlow / (GreenZoneSize + YellowZoneSize + RedZoneSize).
Percent Red Zone	Priority based on DDMRP execution, including QuantityInStock / RedZoneSize.

Purchase Orders (Output Table):

A list of all purchase orders generated during the simulation run ship from one of the suppliers, given that the *InfiniteRawMaterials* property of the model is 'False'. If this property is 'True', raw materials will be assumed always available and thus, no purchase orders will be generated.

Column Name	Description
Order Id	A unique string name assigned to this purchase order.
Material Name	The material name associated with this purchase order.
Site Id	The site id associated with this purchase order.
Quantity	The number of inventory parts associated with this order. Note: This value will not have any applied lot sizing but will strictly be based on the recommended order quantity determined by the replenishment policy.
Order Date	The date that order was released for the processing at raw material supplier to start.
Due Date	The due date of the purchase order, based on the order date + lead time of the inventory (material/site).
Lead Time (Hours)	Lead time of the purchased material inventory, based on Inventories table.
Arrival Date	The date when the purchase order was received at the destination factory/site.
Duration (Hours)	The total duration, in hours, from order date until arrival date, which includes the lead time plus any transportation.
Raw Material Supplier	The supplier where the raw material will be purchased. This is determined by the Materials table property 'RawMaterialSupplier' for the given material.
Current Stock Level	At the time of order date, the current level of inventory for the given material/site. This value is for display purposes and is currently not used in any decision making.

Stock Transfer Orders (Output Table):

A list of all stock transfer orders generated during the simulation run to move between sites, which currently only includes finished goods. In this example, DC1 site is a warehouse without an associated factory, thus when inventory is required at that location, it must transfer from one of the other warehouses with associated manufacturing/factory.

Column Name	Description
Order Id	A unique string name that is assigned to this manufacturing order.
Source Site ID	The site from which the stock transfer order will initiate.
Destination Site ID	The site to which the stock transfer order will be completed.
Material Name	The material name associated with this stock transfer order.
Quantity	The number of inventory parts associated with this stock transfer order. Note: This value is impacted by the Lot Sizing associated with the Material and may/may not match the recommended order quantity.
Start Date	The start date of the transfer from the source site to the destination site.
Due Date	The due date of the order, based on the start date + lead time of the inventory (material/site) at the destination site.
Completion Date	The date when the stock transfer order was completed (arrived at destination).
Duration (Hours)	The total duration, in hours, from start date until arrival date, which includes the lead time plus any transportation.
Current Stock Level	At the time of stock transfer date, the current level of inventory at the destination site for the given material. This value is for display purposes and is currently not used in any decision making.

Suppliers:

This table includes a list of the various inventories (material/site) that can be transferred from one Site to another (stock transfer orders).

Column Name	Description
Inventory ID	This field indicates the inventory (material/site) from which the customer Sales Order will be shipped. At least one entry for each finished goods inventory (shipped as Sales Orders) should be entered.

Supplier Inventory ID	This field indicates the supplier inventory (material/site) that MAY be used if the inventory is not available from the associated Inventory ID. When a Supplier Inventory ID is blank, no transfer order will be attempted, and it is assumed that the inventory will be replenished from an associated manufacturing facility/factory. If multiple Supplier Inventory IDs are specified, then the Inventories table Sourcing Policy property will be used to determine how to decide between the multiple suppliers. Associated lead times for in-stock and out-of-stock may be used.
Minimum Order Quantity	The minimum order quantity for the transfer orders.
In_LeadTime (Hours)	The in-stock lead time that is calculated for this inventory transfer, as calculated at the start of the simulation run.
Out_LeadTime (Hours)	The out-of-stock lead time that is calculated for this inventory transfer, as calculated at the start of the simulation run.
Employees:	
This table includes a list of the various employees in the system. These employees may be used to transfer material between locations or more typically, assist with setup/processing type work at a SchedServer.	
Column Name	Description
Employee Name	The unique name of the employee (worker) resource. This is the object name.
Description	A description of this worker.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Object Type	The object type associated with this employee, typically SchedWorker.
Site Id	The site object where this worker is associated.
XLocation (Meters)	This is the X location of the object in the model.
ZLocation (Meters)	This is the Z location of the object in the model.
Capacity	Capacity of the worker (number of individual units of employee name).
CostRate	Cost of the employee, per hour, charged for both idle and usage time.
Home Location	Initial physical location of the employee, generally a node object.
InitialSpeed (Meters per Second)	Initial walking speed of the employee between object (server) locations.
Employees Schedule:	
This table includes the work schedules of the various employees defined in the Employees table. Employee schedules will repeat weekly. The Data > Work Schedules area for the EmployeesTableSchedule references this table and can be edited if necessary.	
Column Name	Description
Employee Name	The reference key to the name of the employee (worker) resource. Multiple rows may be associated with a given employee (first shift hours may include an entry for morning and another for afternoon, with a break during lunch for example).
Start Date	The start date/time of the worker shift.
End Date	The end date/time for the worker shift.
Capacity	The capacity of the worker during the start > end date/time specified. If a time period is not specified, the capacity of 0 is assumed (breaks, offshift, for example).
Employee Details:	
This table includes employee qualifications and labor efficiency for given workcenters.	
Column Name	Description
Workcenter Name	The name of the Transfer Node 'area' of similar processing workstations for which a given employee can work.
Resource Name	The name of the specific SchedServer (or similar server) at which the employee can work.
Labor Name	The name of the employee associated with the given workcenter and resource. Multiple employees may have the same workcenter and resource names and then the worker will be selected among the 'group' of employees.
Labor Efficiency	The efficiency applied to the setup and/or processing time when this worker is used at a given workcenter and resource.
Vehicles:	
This table includes a list of the various Vehicles in the system. These Vehicles are typically used to move inventory from one location to another, for example, from a supplier to factory or warehouse to distribution center (site to site).	
Column Name	Description
Vehicle Name	The unique name of the vehicle. This is the object name.
Description	A description of this vehicle.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Object Type	The object type associated with this vehicle, typically SchedVehicle.
XLocation (Meters)	This is the X location of the object in the model.
ZLocation (Meters)	This is the Z location of the object in the model.
Initial Quantity	Initial number of Vehicles of this type in the system.
Initial Ride Capacity	The initial carrying capacity of this vehicle (the number of orders that may be transported at a time).
LoadTime (Minutes)	The time required for this vehicle object to load an order (i.e., load time per order or entity).

UnloadTime (Minutes)	The time required for this vehicle object to unload an order (i.e., unload time per order or entity).
Minimum Dwell Time Type	Specifies the minimum dwell time requirement for this vehicle object when loading and unloading orders at a node.
MinimumDwellTime (Hours)	The specific minimum amount of time that this vehicle object is required to wait, or 'dwell' at a node to load and unload entities.
DwellEventName	The name of the event whose occurrence will indicate the expiration of the minimum wait time requirement for this vehicle object when loading and unloading orders at a node.
InitialSpeed (Miles per Hour)	The initial desired speed value for this type of vehicle.
Home Location	The initial home node location for vehicles of this type at the beginning of the simulation run.
CostRate	The cost per unit time that is charged to the cost of the vehicle when it is idle, utilized for a non-transport task or transport task.

Dispatching Rules:

This table includes the dispatching rules utilized for dispatching multiple orders at a given SchedTransferNode (prior to multiple SchedServer selection).

Column Name	Description
Dispatching Rule	The criteria used to select the next order from a transfer node queue (at start of workcenter group of SchedServer objects). Note that using a particular dispatching rule may require some specific model data about the candidate entities (orders), such as due dates, job routings, expected setup or operation times. See Simio's dynamic selection rule documentation for more information.
Attribute Value Expression	The expression used to get the attribute value for each candidate entity (order). In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (orders), for example, Candidate.Entity.Priority.
Campaign Value Expression	The expression used to get the campaign value for each candidate entity (order). In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (orders), for example, Candidate.Entity.Priority.

Resource Calendar:

This table includes the resource calendar (for Resources table objects). Resource schedules will be repeated weekly. The Data > Work Schedules area for the ResourcesTableSchedule references this table and can be edited if necessary.

Column Name	Description
start_calendartime	The start date/time of the resource calendar shift.
end_calendartime	The end date/time for the resource calendar shift.

Resource Exceptions:

This table may include any exceptions to the standard resource calendar shown above. This may include different entries for various SchedServers, and typically may be used for preventive maintenance or downtime events.

Column Name	Description
Resource Name	The reference key to the name of the resource from the Resources table.
PM_Start Time	The start date/time of the preventive maintenance of the resource.
PM_End Time	The end date/time for the preventive maintenance of the resource.

Demand:

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The demand should be daily time based and may include past data as well as future demand. If a date is not specified in the data, a '0' quantity is assumed. Note: this should also include a full explosion of bill of material component part demand over the time periods as well.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this demand.
Date	The date/time for this inventory/quantity combination. Any missing dates (daily) will be assumed to be '0' quantity when calculating Average Daily Usage.
Quantity	The number of units required of a given inventory on the specified date.

Planned Adjustment Factors:

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. It can include multiple inventory entries with varying start and end dates.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this planned adjustment.
Start Date	The start date of the planned adjustment.
End Date	The end date of the planned adjustment.
Demand Adjustment Factor	Adjustment factor that will be multiplied by the ADU when calculating the buffer zone sizes.
Red Zone Adjustment Factor	Adjustment factor that will be multiplied by the red zone size (includes both the red zone safety + red zone base).
Yellow Zone Adjustment Factor	Adjustment factor that will be multiplied by the yellow zone size (includes ADU * DLT).
Green Zone Adjustment Factor	Adjustment factor that will be multiplied by the green zone size (includes the max calculation of minimum order quantity, ADU * desired order cycle, ADU * DLT * lead time factor).
Lead Time	Adjustment factor that will be multiplied by the decoupled lead time DLT.

Adjustment Factor

Buffer Profiles:

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Buffer Profiles table defines the various part types with associated lead time and variability categories. These are then referenced within the Inventories table.

Column Name	Description
Key	The name of the buffer profile key (referenced within Inventories table).
Part Type	Descriptor of the type of part, such as purchased, distributed, intermediate, or manufactured.
Lead Time Category	Descriptor for the type of lead time category (short, medium or long).
Variability Category	Descriptor for the type of variability category (low, medium, or high).
Lead Time Factor	Lead time factor that is used in calculating the Red Zone Base and Green Zone Size within the Buffer Zone Sizes calculator.
Variability Factor	Variability factor that is used in calculating the Red Zone Safety within the Buffer Zone Sizes calculator.

Average Daily Usage (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Average Daily Usage DDMRP calculator utilizes the Demand table, in addition with Inventories table parameters to generate this time-indexed table.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this average daily usage.
Date	The date/time for this inventory/quantity combination.
ADU	The average daily usage as calculated for this inventory/date.

Decoupled Lead Times (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Decoupled Lead Times DDMRP calculator Lead Time and Decoupling Point Boolean within the Inventories table, along with the Product Structure table to calculate the decoupled lead times for each inventory.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this average daily usage.
Decoupled Lead Time (Days)	Decoupled lead time calculated for the associated inventory.

Buffer Zone Sizes (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Buffer Zone Sizes DDMRP calculator utilizes many parameters (discussed above in DDMRP Calculators section) to determine the time-indexed daily values for each inventory. This information is fed directly into the Inventories element.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this buffer zone size.
Date	The date/time for this inventory/quantity combination.
Red Zone Size	The red zone size calculated for the associated inventory/date.
Yellow Zone Size	The yellow zone size calculated for the associated inventory/date.
Green Zone Size	The green zone size calculated for the associated inventory/date.

Qualified Spike Demand (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Qualified Spike Demand DDMRP calculator utilizes the Sales table as well as parameters within the Inventories table to determine qualified spike values for each inventory/date.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this qualified spike demand.
Date	The date/time for this inventory/quantity combination.
Quantity	The qualified spike demand quantity calculated for the associated inventory/date.

Costing (Output Table):

This table provides general costing information by site for dashboard reports.

Column Name	Description
DateTime	Date/time for costing information which is calculated daily.
Location	Site location for the total cost information.
Total Cost	Total cost of inventory at the associated site for the day (date/time).

Costing by Material Site (Output Table):

This table provides costing information by material site for dashboard reports.

Column Name	Description
DateTime	Date/time for costing information which is calculated daily.
Site	Site location for the total material cost information.
Material Name	Name of the material for which the cost is associated.
Total Material Cost	Total cost of inventory at the associated site for the day (date/time).

Costing by Resource (Output Table):

This table provides costing information by resource for dashboard reports.

Column Name	Description
-------------	-------------

DateTime	Date/time for costing information which is calculated daily.
Site	Site location for the total resource cost information.
Resource Name	Name of the resource for which the cost is associated.
Idle Cost	Total idle cost for the site/resource at the given datetime.
Usage Cost Charged	Total usage cost charged for the site/resource at the given datetime.

Cpm and CpmSorted (Output Tables):

These tables are generated daily to calculate the Cpm (Taguchi performance index) used for Dashboard Reports. They report the statistical tally and tally squared values based on on-hand inventories, optimal high/low values and target.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

EngineRepairUsingExtrasLibrary - Example

Engine Repair Using Extras Library

Note: The Extras Library is a collection of new object definitions included with the Simio installation. This library is installed with the Simio examples, typically found at the following directory: C:\Program Files\Simio LLC\Simio\Examples\ExtrasLibrary.spfx

Purpose

This example model demonstrates the use of the Extras Library objects. The Crane, Robot, Elevator, Rack, and Lift Truck are utilized throughout this system. This model highlights animation capabilities and illustrates the visual interaction between the Extras Library objects and Standard Library objects. For more technical information regarding the Extras Library objects, please see their corresponding SimBits.

General Description

This model represents an engine repair line. Engines requiring repairs arrive from a truck dock and are placed in a holding area. Cranes will move these engines from station to station. The first stop is an uncasing station that removes the engine's outer case. The engine then moves to an inspection station where diagnostics are run, and the parts needed for repair are recorded. After, the engine will move into a middle holding area and then to a repair station when one is available. At the repair station, kits with the required parts are delivered and the engine is repaired. Once repaired, if there is room in the middle zone, the engine will move there, then to travel straight to testing, else it will be moved into one of the empty spaces at the end of the line to wait. Repaired engines will be tested at the test station, re-cased at the re-casing station, and then moved out for shipping at the truck dock.

The parts needed for engine repair are pulled from containers which sit on pallet racks in the storage area. A forklift will pick up the container off a shelf and move it to a drop off zone. In this building, the storage facility is located on a different level. Once dropped off by a forklift, the storage container must ride down an elevator to the lower level. From the elevator, a worker moves the container to a conveyor system. This conveyor system moves the containers past robots which will pick up the needed parts out of the container and then place the part in a small kit bin. At the end of the container conveyor, new parts are added back into the storage container and sent up to the upper level on another elevator where they are then returned to the shelf by a forklift. At the end of the kit conveyor, a worker will deliver the kits to the engine repair stations.

Approach

Initialization WIP

The *OnRunInitialized* process is used to create WIP throughout the system. A List containing the three repair stations and the two holding spaces prior to the repair stations is populated with engines requiring repair. Kits and Containers are also populated on the conveyors and in the drop zones.

Cranes and Engine Areas

The entire engine area is contained in one Bay with 11 zones. The four Cranes in this Bay are used to move the engines from each location, waiting areas and stations. The Cranes are not able to move past each other but could push one out of the way if needed. Some zones can only be reached by one Crane, while other zones must be shared by multiple Cranes.

The engines are moved out of a truck by a Worker and placed in a waiting area to be picked up by a Crane. When the uncasing station is available, a Crane will move a waiting engine into it. After uncasing, the engine moves to inspection. Post inspection, the engine will move to another waiting area with the intent to move into one of the three repair stations when available. After the engine has been overhauled at one of the repair stations, the engine needs to be tested. If possible, the engine will be moved into the middle waiting area to then head straight to the testing station. If there is no space in the middle holding area, the engine will be held in the extra spaces at the far end of the bay. After testing, the engine is re-cased and sent to the drop-off zone to be reloaded on the truck by a Worker.

Each location where an engine can sit is modeled as a Server object with zero input or output buffer space. This means that the Crane will not move the engine unless the next location is available. Through use of Node Lists, Transporter Lists, and Selection Conditions, the engines are routed to specific locations using specific Cranes. To view the routing logic, enable Node visibility in the Visibility menu and select each Server's output Node.

Parts Storage and Transportation

For this example, the storage containers are randomly pulled from the Rack. Using the Rack's *Shelf Storage Time* property, each storage container will sit on the Rack for a random time before requesting a ride from the Lift Truck. The Lift Truck will

drop the storage container off at a drop zone where a Worker will move the storage container to the Elevator and wait to load it. The container rides the Elevator down where a different Worker unloads it and moves it to the Conveyor.

A custom Elevator submodel was created to allow for easy placement of multiple objects. The submodel contains multiple Elevators and Elevator Nodes as well as a Worker and other standard Nodes and Paths. The Worker will deliver the Storage Container Entity to one of the Elevators in a cyclical pattern.

Automated Kitting

The storage containers and kits are each loaded onto their own conveyor and travel down, passing by multiple robots. The robot will pick up parts out of the storage container and move them into the kitting bin.

A custom robot submodel was made to allow for easy placement of multiple objects. This submodel contains a Robot, Separator, and Combiner. The Separator and Combiner are used in conjunction with the conveyors in the main model. They separate Part Entities from the Storage Container Entity and then combine the Part Entity with the Kit Entity. The Robot is the vehicle which transports the part from the storage container to the kit bin.

In this example, robots are not looking for a specific bin, part, or kit. The robot will randomly select if a part should be separated from the storage container by using a random *New Entity Quantity* property on the Separator. Similarly, the Combiner uses a random *Batch Quantity* property to determine if the kit should wait for a part at this robot station. The Combiner also uses a *Release Batch Early Triggers* so that it does not hold up the conveyor or cause a gridlock if it waits too long.

Additional Notes

Prior to Run Time, many of the symbols in the model are missing. This is due to a new paradigm introduced with the Extras Library: compound objects. The Crane, Lift Truck, Rack, and Robot are all made up of multiple objects, typically additional entities. For example, the Rack's Shelf and the Robot's Upper Arm are both entities created at Run Time.

You will notice in the Elevator and Robot submodels, the entity instances for those objects are not set to be externally visible. If they were visible, we would see an additional entity instance along with the realization of these objects during Run Time.

The Shelves on each Rack are not visible because they are not created until Run Initialization. The Rack uses its Shelf properties such as Number of Shelves, Shelf Capacity, and Height of First Shelf to dynamically create the Shelves at runtime, allowing for experimentation with Shelf parameters.

In this instance, there are also no rails for the Cranes in the Crane and Engine Area. Animating these rails was omitted intentionally to avoid obstructing the view of this area.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

HospitalEmergencyDepartment - Example

General Description

This model represents a small Emergency Department (ED) consisting of a waiting area, a registration desk, a triage room, a radiology station, a billing area, 6 beds and 6 rooms that are used for patients that are admitted into the hospital.

Patients enter the ED through the front door entrance and go directly to the registration desk. After they are registered, they wait in the waiting area to be sent to the triage room. After they are seen in the triage room, they wait in the waiting area for an available bed. When a bed and a nurse are both available, the nurse greets the patient and walks them to the bed. The patient waits in the bed for an available doctor, who is accompanied by a nurse. They finish treating the patient and the patient either visits billing before exiting the ED or is sent to the Radiology room. If a patient is seen at Radiology, they are either sent to billing and then sent home or they are admitted into a room for a longer stay. Once admitted into a room, the patient is again visited by a doctor and a nurse and then released to billing and then home.

Patients also enter the ED from an ambulance. The ambulance can carry up to 2 patients. They arrive at the emergency entrance and are greeted by a nurse who will walk them directly into a room. After being treated by a doctor, an emergency patient will either visit billing or the Radiology department.

There are several statistics collected in this model. The details of these statistics are discussed below. The Facility window contains two different floor labels with statistics and the Console window contains floor labels, status pie charts and status plots.

A small experiment can be used to vary the number of nurses, the number of doctors and the number of radiologists to see the impact on Average Wait Time for Doctor, Number of Patients in System and Average Length of Stay.

Detailed Description

System Initialization

The system does not begin empty. It is initialized so that there are patients in the system at the beginning of the model run. This is controlled by a process called "OnInitialization" which is executed when the main model is first initialized. This process searches a Data Table called 'Initialization' and uses the content of this table to create a set number of patients, with certain priorities and place them in a specific location (station) within the model. This is done with the combination of a Search step, Create step, Assign step and Transfer step.

Patient Arrivals

The large main door is the Source of the main arrivals of patients. This Source uses a Rate Table to determine the rate of arrivals into the system. The Source also references a Data table named 'PatientPriority' which determines the type of patient to create and therefore which symbol to display for the entity.

Arrivals are created for the Emergency entrance from a Source that can be found off on the "road" to the side of the ED near the ambulance. This Source also references the PatientPriority table to assign a priority to a patient. There is an Add-On Process on this Source that checks to see if all the Rooms are full after the patient is created. If there is no room in this Emergency Department for this patient, a state variable is incremented to keep track of the number of Diversions from this hospital and then this patient is destroyed and not sent into the ED. But for most patients created, they request a ride on the ambulance vehicle and are dropped off at the emergency entrance where they are met by a nurse who walks them to a room.

Registration, Waiting Area and Triage

The registration desk is a standard Server with *Initial Capacity* of '3'. After the patients are registered, they visit the Server named 'WaitingAreaBeforeTriaged', which has the symbol of a set of waiting rooms chairs. They visit Triage when it is available and then sit at the Server named 'WaitingArea_PostTriaged' until a bed and nurse are available. The nurse arrives to walk the patient to the bed. The nurse is a standard Worker object that is requested in the TransferNode 'ToBeds', by a patient.

Beds, Rooms, Radiology and Billing

A Bed is an object that is part of the Project Library. It is a subclassed version of the standard Simio Server object. A patient will arrive at a bed and lay down using the assignment of State Variable Name 'ModelEntity.Animation' with a New Value of 'Sleep'.

Each instance of the Bed requests a Doctor to visit using the *Resource For Processing* section of the *Secondary Resources* properties. The *Before Processing* property in the *Other Resource Seizes* properties requests a Nurse to visit this bed. Once

the Doctor and the Nurse arrive, the processing begins at this Bed and then both the Doctor and Nurse are released. Link Weights on the Paths leaving the output node of each Bed determine the percentage of patients that visit Radiology vs Billing.

A Room object is very similar to a Bed object. Patients are only routed to a Room if they are sent there after visiting Radiology or if they arrive through the Emergency entrance.

The Radiology department is a standard Server object, with the capacity of '1'. The Billing desk is also a standard Server object. All patients leaving the Billing desk travel to the front door and exit the system.

Statistics

The following statistics are collected and displayed in the Facility window. Utilization statistics, status charts and plots can be found in the Console window (Definitions tab, Console panel). The Math.Round function is used within the floor labels to truncate display of real numbers to a single decimal point.

- **Average Wait Time to See Doctor:** The wait time is calculated with a Tally Statistic on the Bed object and this overall value is found by taking an average of the wait times of all the beds.
- **Number of Patients Admitted:** This is calculated with a Tally Statistic that records the number of patients that are admitted into a Room.
- **Percentage of Beds and Rooms Occupied:** This is calculated with a State variable that tracks the number of Beds and Rooms occupied at the current time.
- **Number of Diversions From Hospital:** This is calculated with a State variable that tracks the number of times that a patient wanted to enter the ED through the Emergency entrance but could not visit this hospital because all the Rooms were full.
- **Average Number of Patients in System:** This is a function on the ModelEntity, named Population.NumberInSystem.Average
- **Average Wait Time for a Bed (min):** This is calculated with a Tally Statistic that records the amount of time that a patient waits for a Bed.
- **Average Length of Stay (min):** This is calculated with a Tally Statistic that records the amount of time that a patient is in a Bed or a Room.
- **Total Leave Without Being Seen (LWBS):** This is calculated with a State variable that tracks the number of patients who arrive in the main entrance but leave because the waiting area is too full or the average wait time is too long.

Cost

Each Bed object, Room object, Nurse, Doctor and the Radiologist object have their Parent Cost Center property set to 'CostOfStaff'. This will ensure that all the costs associated with these objects will get rolled up to the CostOfStaff cost center. The Nurse, Doctor and Radiologist objects have their Usage Cost Rate and Idle Cost Rate properties populated with a cost so that the model will calculate the cost of the staff both when they are idle and being utilized. Each time a staff member is used by either a Bed or a Room object, the usage cost of the staff member is rolled up to the cost center of the Bed or Room object, which is why these objects also have their Parent Cost Center set to 'CostOfStaff'. The total cost for this cost center can be found in the Results Pivot Grid. CostOfStaff is also used as a Response in the Experiment.

Experiment

An Experiment exists on the main model that varies the input of three Controls; 'Number of Nurses' and 'Number of Doctors'. These controls appear in the experiment because they are all properties on the main model. The 'Number of Nurses' property is referenced from the *Initial Capacity* property on the Nurse Worker object. The 'Number of Doctors' property is referenced from the *Initial Capacity* property on the Doctor Worker object.

There are three Responses defined in this experiment. 'CostOfStaff.Cost' indicates the cost that has been associated to the Cost Center named CostOfStaff, while 'Average Length of Stay' contains the expression 'LengthOfStayStatistic.Average * 60', which references the Tally Statistic that tracks this duration. 'Average Leave Without Being Seen' references the State Statistic that references the average number of patients that leave the system because the wait was too long.

Selecting the 'Average Length of Stay' response will show it has an *Upper Bound* of '36'. Notice on the Experiment Properties (right click on experiment in the navigation window), you'll see that the *Objective Type* property is set to 'Multi-Objective Weighted', which will tell the OptQuest Add-In to take into account the three responses when determining the best scenario, based on the objectives and weights of each response.

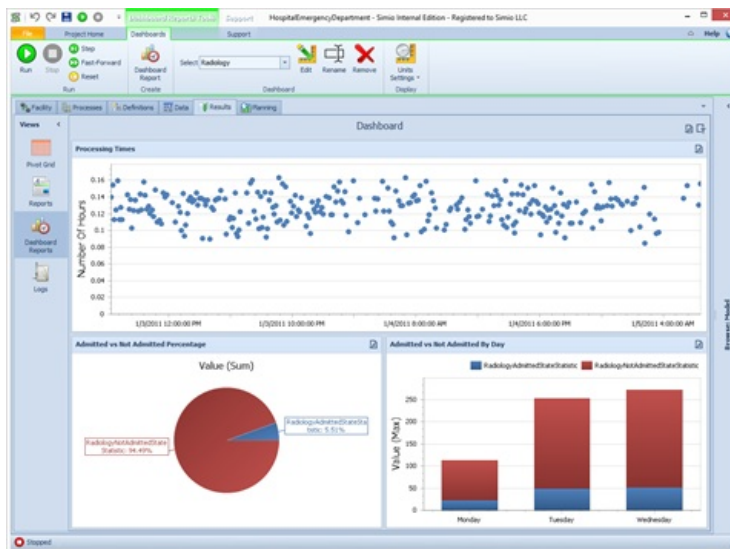
See the [Experiments](#) page, Experiment Dashboard Reports section, for more information on building a Dashboard Report for this hospital example.

Dashboards

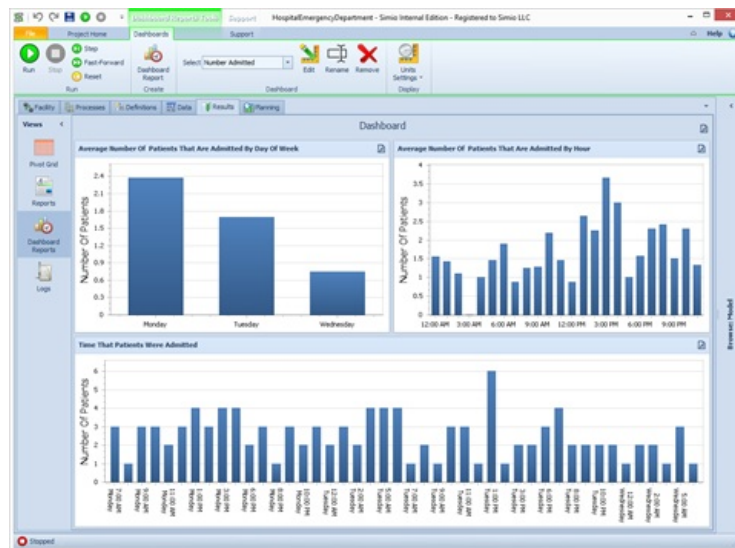
Patients Leave Without Being Seen – Shows patients that leave the hospital over time compared to current waiting times for beds and rooms.



Radiology – Show the time for each patient to be seen in Radiology as well as admitted vs non admitted comparisons.



Number Admitted – Shows the average number of patients admitted in the hospital by day of week and time of day...Also shows the time of day that the patients were admitted.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

InfectiousDiseasePlanning - Example

Infectious Disease Planning

Purpose

During an infectious disease outbreak, the strain on health systems and healthcare professionals is unprecedented. There is uncertainty in what demand a healthcare facility might see, and when it might surge. In response, the Simio team has created a model to assess the capacity of critical resources for a hospital. The model uses a data-driven approach so users could tailor the data to meet specifications of their service area and healthcare provider. It also demonstrates, more generally, how a simulation model can act as decision support tool and provide value in dealing with change.

A healthcare provider's ability to combat an infectious disease is directly impacted by their critical resource availability. All inpatient cases will require a hospital bed, while machines like ventilators are crucial for patients that require intensive care. The ability to anticipate demand for these resources in a facility is vital for a proactive planning strategy. With the model, you will be able to experiment with the number of beds, ventilators, and personal protective equipment (PPE) items such as masks, gloves, and gowns, to estimate the capacities and quantities needed for care delivery to the infected population. This model may be instructional in its current state or used as a starting point to model a specific service area population and healthcare provider.

Approach

Initialization and Tracking Infected Cases

The model accepts several user-adjustable parameters including the hospital's *Service Area Population*, the initial number of *Reported Cases*, the *Contagion Factor (Ro)*, and a *Social Distancing Factor*, to calculate how the infection will spread over the planning horizon: 20 weeks by default. The values of these parameters (or model "Controls") may be set in the Model Properties, shown in the Properties window, upon clicking blank space in the Facility window or right-clicking the name of the model in the Navigation window towards the top right.

An OnRunInitialized Process called 'DailyAssessmentOfInfectedCases' will initialize the number of infectious cases in the system by assuming that the *Service Area Population* minus *Reported Cases* is healthy. This process uses an Execute step to start the 'TrackingInfectedCases' process which creates Tokens for each new infected case. The Tokens are delayed stochastically until the onset of infection, at which time they are mapped to a row in the *Table Population Mix* data table using a Set Row step. Notice that this step uses the RandomRow function to assign a row (*Age Group*) using the probabilities in the *Population Mix* column as selection weights. Then the table data is used, throughout the rest of the process, to determine whether hospitalization and critical care are needed for each patient.

As new cases occur, a percentage of infected cases will remain asymptomatic or otherwise not require hospitalization as specified by the table data. In 'DailyAssessmentOfInfectedCases', the model will create new cases based on the following expression: 'Infected Not Hospitalized Population' * *Contagion Factor (Ro)* * (1 - *Social Distancing Factor*). Note that the *Social Distancing Factor* should be interpreted implicitly for its effect on *Contagion Factor (Ro)*. For example, *Social Distancing Factor* '0.5' effectively halves the effect of *Contagion Factor (Ro)* based on the previous equation – but the factor should not be interpreted as a percentage of people who social distance.

Hospitalized Cases and Arrivals

The objects shown in the model's Facility view represent the healthcare provider's facility. Of the cases that require hospitalization, a portion will need intensive care as specified by the *Table Population Mix* data. Notice that the likelihood of both hospitalization and critical care (ventilator) increase with *Age Group*. Keep in mind that the table data could be updated in accordance a healthcare providers service area.

The 'TrackingInfectedCases' process uses a Create step called 'CreatePatient' to generate entities that require hospitalization. Recall that each entity belongs to a certain age group in the *Table Population Mix* data. The Simio team chose to create entities, only for hospitalized cases, and track all other population members using Tokens since the focus of the model is on the healthcare provider and its critical resource management.

Hospital Processing

The hospitalized cases enter the model as a Patient Entity and are sent to the Hospital Server. At the Hospital, they will claim a unit of the Hospital's capacity representing a bed. If they need a ventilator, they will attempt to seize the secondary Ventilator Resource. Then the patient remains at the hospital for their assigned length of stay.

The model keeps track of the PPE material used by patients each day and uses a continuous review reorder point / reorder quantity replenishment policy to restock on masks, gloves, and gowns. Order lead time is also accounted for in the replenishment policy via the 'ProducePPEMaterial' process. All PPE data is found in the 'Table PPE Supplies' data table

which also could be manipulated to experiment with stock levels and replenishment policies that suit your facility.

While running the model, interactive buttons are available to change the current number of beds and ventilators. This allows you to dynamically experiment with the model as events unfold. For example, the hospital could have reallocated more beds for infection cases in the middle of the planning horizon. The charts and labels in the model will update to reflect changes made using the buttons.

Statistics

The model tracks important metrics such as the number of times a critical resource was not available. A variable counts each time that an infected person was turned away from the Hospital due to a lack of beds. Another variable counts how many times a person had a bed, needed a ventilator, and there were no ventilators available. When there is a shortage of any type of PPE material, the deficit is also tracked and saved.

Experiments

Two experiments are defined in the model to demonstrate how changing the model Controls will impact the Responses, in this case, counts of resource and material deficits. Various Output Statistics (see Definitions -> Elements) and a Simio expression are used to define the Responses. Review them by selecting a Response and observing its properties. For instance, the HospitalBedDeficit Response reports the NumberBalked, from the Hospital Input Buffer. Patients that were unable to get a hospital bed balk immediately from the Hospital Input Buffer. The other deficit counts were collected by State Variables reported out by Output Statistic Elements.

The experiment called SocialDistancingImpact performs Sensitivity analysis on *Social Distancing Factor*. Scenarios 1-6 consider a *Hospital Bed Capacity* of 100, and Scenarios 7-12 double the bed capacity. One objective of this experimentation may be to identify parameters that estimate no bed or ventilator deficiencies, then evaluate the PPE separately. Upon running the experiment, notice that the *Ventilator Capacity* is ampler in Scenarios 1-4 (than Scenarios 7-10) since beds are the primary constraint. In Scenarios 7-10 ventilators, and PPE are more strained as the hospital can treat double the patients at full capacity.

The SocialDistancingImpact results suggest that a *Social Distancing Factor* somewhere between the values of 0.3 and 0.4 produce a '0' for both the HospitalBedDeficit and VentDeficit responses. To examine this further, a second experiment called ResourceCapacityAnalysis fixes *Social Distancing Factor* at '0.35' and analyzes the impact of varying *Hospital Bed Capacity* and *Ventilator Capacity* values. A potential enhancement would be to specify, for instance, initial quantities of PPE as referenced properties (Controls) and perform additional experimentation.

Model Assumptions

- The input for the number hospital beds is the number the hospital can allow strictly for infected patients. Expect to keep beds reserved for other emergencies.
- When the hospital has no available beds for infected cases, the cases turned away are sent to another hospital or are self-quarantining. They do not infect others.
- Once a person recovers, they are no longer in the pool of people who can become infected. It is assumed the infectious disease strain cannot mutate quick enough to cause reinfection.
- Patients who need a ventilator are assumed to be in the ICU and require the ventilator the entire duration of their hospital stay.
- Patients who need a ventilator and cannot acquire a ventilator are prematurely discharged.

Note: A similar model was used in a May 2020 Simio Webinar called "Redefining Your Business Planning Strategy". Some enhancements were made to the latest version that may reflect slightly different results than the Webinar version.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Manufacturing Assembly - Example

General Description

This model represents an assembly line manufacturing Lamps. The general process flow is as follows:

- Pallets containing lamp bases are withdrawn from inventory by a fork lift
- Individual lamps bases are sent to preparation (Prep)
- Lamps bases and shades are assembled
- Lamps are sent to repair if any defects are detected
- Finished lamps are packaged
- Packages of lamps are shipped randomly through 2 docks

Detailed Description

Model States

This model has two defined model states – 'NumberInPrep' and 'TotalShipped'. These states are meant to respectively track the number of lamps bases in preparation and the total number of assembled lamps shipped.

Lamps bases Arrival

Lamps bases are withdrawn from two inventories which are modeled as sources. On the model initialization, the lamp bases are taken from inventory 1. During the model run, if the the number of lamps bases in Prep is less than 2 the lamp bases are randomly created at either inventory 1 or 2. This is modeled through the use of a Monitor element on the 'NumberInPrep' state variable with a *Threshold Value* of '2' and a 'Negative' *Crossing Direction*. When this threshold value becomes less than 2 (i.e 1) the monitor will fire an Event triggering Process1 to create new bases at the inventories.

Pallets Discharge

Each pallet holds 10 lamp bases. The pallet's contents are emptied at the Separator object at the beginning of the line. The bases are sent down the Conveyor while the pallets are taken to a storage rack that is represented with a Sink.

Lamps Bases Preparation

The number of lamp bases in preparation is tracked through a user defined state that is incremented and decremented respectively as lamps bases enter and leave the Prep server. After preparation, the lamps bases are randomly sent to one of the two existing combiners.

Lamps Assembly

The lamps shades are generated by two sources each linked to a combiner. Each base is combined with one shade. The lamp base represents the combiner parent while the lamp shade represents the combiner member.

Lamps Packaging

All lamps assembled at Combiner2 go directly to packaging. 70% of lamps assembled at Combiner1 are sent directly to packaging, while the remaining 30% are taken by the repair staff to the RepairStation. After the lamps are repaired, the repair staff takes them to packaging.

Lamps Shipping

After the packaging, the assembled lamps are randomly taken by the DockResource to one of the two available docks that are modeled as Sinks. The random selection is enforced through the use of a 'Random' *Selection Goal* at the output node of the packaging server where the *Entity Destination Type* is set to 'Select From List' and the *Node List Name* is a list containing the docks' input nodes.

Useful Tips

Note the use of processes to change the appearance of the entity after each station

- At the separator member output node to model the lamp bases
- At the separator parent output node to model the pallets
- At the combiners output to model the assembled lamps
- At the packaging output node to model the ready to ship package

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

MiningExample - Example

General Description

This example represents an ore mine. The general process flow is as follows:

- Ore is extracted from three different Chutes.
- A MineTrain will pick up the ore and drop it off at the Dump.
- Three MineTrains are available and will pick up ore according to when the ore was created.

* A two scenario experiment is used with two distinct values of the control 'OverallHoursBetweenLoads'.

Detailed Description

Ores Extraction

The ores are extracted from three different chutes modeled as sources. Each source's interarrival time is defined as an expression using the property 'OverallHoursBetweenLoads'. Each created entity is assigned a priority based on the time it was created through assigning the *New Value* to 'Run.TimeNow'.

All resources output nodes *Entity Destination Type* are set to 'Input@Dump' so that all the entities are dropped off at the dump.

Ores Transportation

Three mine trains are available for the transportation of the extracted ore. The trains are modeled as a Vehicle with an *Initial Number in System* equals to '3'. The trains Transport Logic *Task Selection Strategy* is based on 'Smallest Priority'. Because the Ore's priority was set to the time it was created, the trains will pick up the ore in the order it was created.

Bypass Siding

Because the Bi-Directional track only allows traffic moving in one direction at a time, we must make use of a bypass area to allow for passing. This area consists of a separate track for each direction of traffic. It allows for the train to exit the bi-directional portion of the track and park temporarily to avoid another train moving in the opposite direction. If this section were not there, then the trains would collide and it would result in a deadlock.

*Please see the SimBit 'BidirectionalPaths' for a more detailed description of bypass sidings.

Custom Statistics

The 'TimeBetweenLoads' is a user defined Tally Statistic to track the time between loads at the dump. A Tally step is used where the *Value Type* is assigned to 'TimeBetween' to record the time between loads arrival to the dump.

Useful Tips

The *Network Turnaround Method* for the MineTrain vehicle has been set to 'Reverse' to allow for the train to reverse while on the path – as opposed to exiting and re-entering the path in a forward direction.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

MultiEchelonSupplyChain - Example

General Description

This Simio example has been adapted from Melamed, B. & Altiock, T. (2007) *Simulation Modeling and Analysis with ARENA*. Chapter 12, Example 12.3: A Multiechelon Supply Chain.

The modeled system is a single-product, multi-echelon supply chain consisting of a retailer (R) that faces a customer demand stream, a distribution center (DC) that replenishes the retailer, a manufacturing plant (P) that replenishes the DC, and a raw material supplier (S). The manufacturing plant interacts with two buffers: an input buffer (IB) storing incoming raw material, and an output buffer (OB) storing outgoing finished product. The system is depicted schematically in Figure 1.

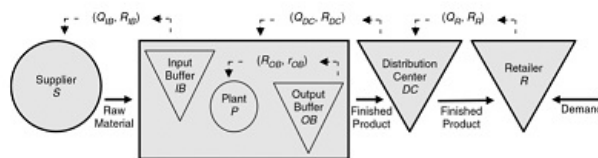


Figure 1 - A Multi-Echelon Supply Chain

The system is subject to the following assumptions:

1. The time between customer order arrivals at the retailer is exponentially distributed with a mean of 1.5 hours. The demand quantity of each customer order is one product unit. Any customer demands at the retailer that cannot be immediately satisfied from on-hand inventory are lost.
2. Continuous review replenishment policies are used to manage the inventory at each site in the supply chain network. The inventory control parameters are as follows:

Table 2 – Inventory Control – Initial Quantities & Replenishment Policies

Material	Site	Initial Quantity	Replenishment Policy
Finished Product	Retailer	15	Reorder Point/Reorder Qty where Reorder Point = 5, Reorder Quantity = 10
Finished Product	Distribution Center	30	Reorder Point/Reorder Qty where Reorder Point = 10, Reorder Quantity = 20
Finished Product	Plant	30	Min/Max where Reorder Point = 10, Order-Up-To-Level = 30
Raw Material	Plant	23	Reorder Point/Reorder Qty where Reorder Point = 10, Reorder Quantity = 13
Raw Material	Supplier	Infinity	None

3. The plant manufactures one product unit at a time. The bill of materials needed to make 1 unit of finished product is 1 unit of raw material. The processing time per unit is triangularly distributed with a minimum, mode, and maximum of 0.8, 1.8, and 2.2 hours respectively.
4. If a stock out occurs at the distribution center or plant, then replenishment orders are backordered. Order fulfillment (shipment) is always deferred until the full order becomes available. In other words, there is no shipping of partial orders.
5. All transportation time delays in the system are drawn from the distribution Erlang(1,2) hours.

The Simio Project File

The *MultiEchelonSupplyChain* Simio project file (.spfx) contains two models: *BasicApproach* & *DataTableDrivenApproach*. These two models simulate the same example system and produce identical animation and statistical results. The key difference between the two models is the latter illustrates how to define all required model input in a set of relational data tables. The third model contains additional data tables and inputs to demonstrate a similar system using the Demand Driven MRP methodology. Some of the data, including processing time at the plant and transportation times vary slightly from the first two models to more effectively demonstrate the movement of material using a demand driven MRP approach with buffers at the various locations.

Basic Modeling Approach Overview

The Facility View

The library objects used are illustrated in *Figure 2*:

1. **BasicNode** objects have been placed to represent the four physical locations in the supply chain network: the supplier, plant, distribution center, and retailer. Each basic node is supplemented by Stacked Bar animation and Status Labels to show current inventory levels.
2. **TimePath** objects have been drawn between the basic nodes to model the transportation time delays between sites.
3. A **Server** object connected to a **Sink** object has been added. The server is used to transform raw material into finished product at the plant. Its processing is modeled as a task sequence with a conditional, processing count-based loopback to simulate the manufacturing of one product unit at a time.
4. A **ModelEntity** object named ProductionEntity has been added to represent a manufacturing order at the plant. Production entities are processed through the plant's server.
5. A **ModelEntity** object named ShipmentEntity has been added to represent a transportation shipment of raw material or finished product. Shipment entities use the drawn time paths for travel. The symbol of the ShipmentEntity has been changed to a box.

Note that user-defined *Material*, *OrderQuantity*, & *ProducedQuantity* state variables have been added to the **ModelEntity** object definition to store either ProductionEntity or ShipmentEntity details.

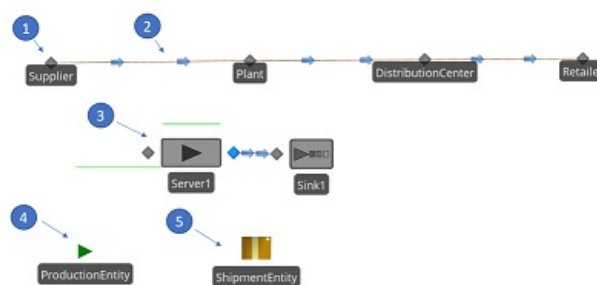


Figure 2 – Objects in the Facility View

Additional animation-related items provided in the Facility View include a grid that displays inventory metrics during the simulation run. This grid was drawn using Floor Labels and Status Labels. Below the grid, a Status Plot was added to show inventory on hand over time at both the retailer and distribution center. Finally, a Floor Label placed near the retailer location displays total satisfied and lost demand.

The Definitions -> Elements View

The elements defined in the model are as follows:

1. Two **Material** elements defining the raw material and finished product. The *Location Based Inventory* property for both elements is set to True. The *Bill of Materials* for one unit of finished product is one unit of raw material. *Figure 3* shows the properties for the material element that represents finished product.



Figure 3 – Material Element Properties – Finished Product

2. Five **Inventory** elements defining the five distinct stocks of material in the system: raw material at the supplier, raw material at the plant, finished product at the plant, finished product at the distribution center, and finished product at the retailer. *Figure 4* shows the properties for the inventory element that represents finished product at the retailer. Note that the inventory control parameters for all the inventory elements are as listed in *Table 1*.

Properties: FinishedProductAtRetailer (Inventory Element)	
Basic Logic:	
Material Name	FinishedProduct
Site Object Name	Retailer
Initial Quantity	15
Review Period	Continuous
Replenishment Policy	Reorder Point/Reorder Qty
Reorder Point	5
Reorder Quantity	10
On Replenishment Order Process	FinishedProductAtRetailer_OnReplenishmentOrder
Advanced Options	
Allocation Ranking Rule	FirstInFirstOut
Assume Infinite Availability If	
Allow Partial Allocation If	
Allow Backorder Policy	Never

Figure 4 - Inventory Element Properties - Finished Product At Retailer

3. A single **Timer** element named CustomerOrderTimer that is used to signal customer order arrivals at the retailer. This timer is configured to fire its event every Exponential(1.5) hours.

The Processes View

The processes defined in the model are as follows:

1. **OnCustomerOrder** Process – This process is triggered whenever the CustomerOrderTimer has fired its event, signaling a customer order arrival at the retailer. The process logic here simply tries to consume 1.0 unit of finished product at the retailer. Note that since the *Allow Backorder Policy* property of the FinishedProductAtRetailer inventory element is set to 'Never', any material consumption request that cannot be immediately satisfied from on-hand inventory is treated as lost demand.
2. **OnReplenishmentOrder** Processes – These processes are triggered whenever replenishment is required at an inventory site and are used to model inventory sourcing behavior including ProductionEntity or ShipmentEntity creation (if necessary). Note that these processes are linked to the inventory replenishment policies via the *On Replenishment Order Process* property on each inventory element.

For example, the process logic that handles the replenishment of finished product at the retailer is shown in *Figure 5*. When the execution of that specific process is initiated, the created token first attempts to consume the required quantity of finished product at the distribution center, with the material consumption request being backordered if there is not enough on-hand inventory. The built-in function *Token.MaterialOrderDetail.Quantity* is used to get the recommended replenishment order size. Once the material consumption request has been filled, a new ShipmentEntity is created and assigned its shipment details - the material and order quantity. That entity then has its destination node set to the retailer and is transferred to the distribution center node (as an outbound shipment) so it can proceed with its travel from the distribution center to the retailer using the drawn time path in the Facility View.

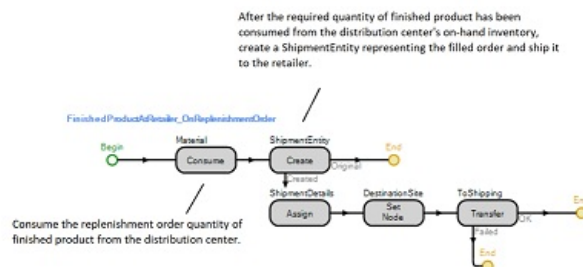


Figure 5 - Process Logic - Replenishing Finished Product at Retailer

As another example, the process logic that handles replenishment of finished product at the plant is shown in *Figure 6*. Here, the inventory sourcing assumption is 'Make'. When the execution of that specific process is initiated, a new ProductionEntity is created and assigned its production details - the material and order quantity. The entity is then transferred to the plant's Server object in the Facility View to manufacture the inventory.

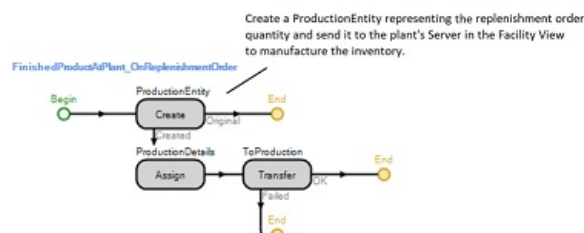


Figure 6 - Process Logic - Replenishing Finished Product at Plant

3. **ShippingReceivingLogic** Process – This process is triggered whenever a ShipmentEntity enters one of the BasicNode objects in the Facility View that represents an inventory site location. The process logic is shown in *Figure 7*. Here, if the entity is an inbound shipment, then the received material is added to the destination inventory (using a Produce

step) and the entity is then destroyed. Otherwise, if an outbound shipment, then the entity simply continues its travel out of the node.

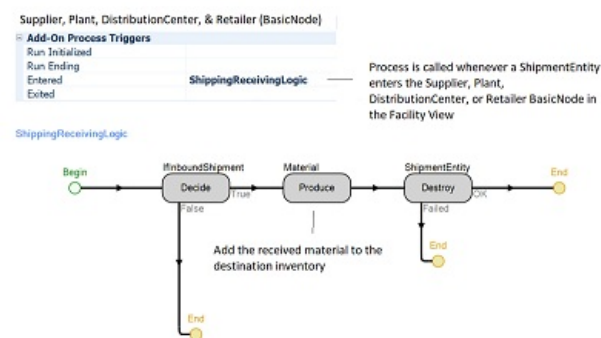


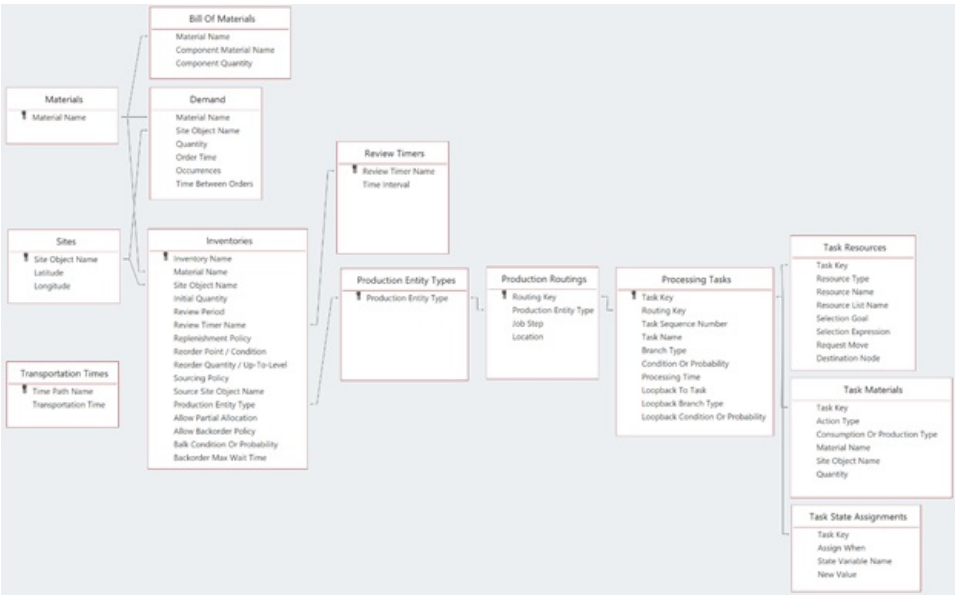
Figure 7 - Process Logic - Shipping & Receiving

Data Table Driven Approach Overview

The *DataTableDrivenApproach* model provided in the *MultiEchelonSupplyChain* project file is the same system and produces identical animation and statistical results as the *BasicApproach* model covered previously in this document. The key difference is most of the required model input is now defined in a set of relational data tables and there are also some differences particularly in the process logic design to better fit the defined data table schema.

Note that although the example system being modeled here is rather simple, a much more complex supply chain simulation could potentially be accommodated using the same presented data table structure.

Figure 8 shows the relationships of the data tables that are defined in the model.



The summary of each data table is as follows:

PK => Primary Key, FK => Foreign Key

Materials Table Schema

Defines the materials that are consumed or produced in the system.

PK/FK	Column Name	Property Type	Description
PK	Material Name	Material Element (Auto-Create)	The name of the material.

Sites Table Schema

Defines the physical inventory sites in the system.

PK/FK	Column Name	Property Type	Description
PK	Site Object Name	Object {Auto-Create}	The name of the fixed object that is an inventory site.
	Latitude	Real	The latitude location in the Facility View.
	Longitude	Real	The longitude location in the Facility View.

Demand Table Schema

Defines the end-customer demand streams in the system.

PK/FK	Column Name	Property Type	Description
FK	Material Name	Foreign Key (Table Key = Materials.MaterialName)	The name of the material.
FK	Site Object Name	Foreign Key (Table Key = Sites.SiteObjectName)	The name of the inventory site.
	Quantity	Expression	The customer order quantity.
	Order Time	DateTime	The time of the first customer order.
	Occurrences	Integer	The number of times that a customer order will occur.
	Time Between Orders	Expression	The time between customer orders if the Occurrences is greater than one.

Inventories Table Schema

Defines the storage buckets for holding stocks of material in the system.

PK/FK	Column Name	Property Type	Description
PK	Inventory Name	Inventory Element {Auto-Create}	The name of the inventory.
FK	Material Name	Foreign Key (Table Key = Materials.MaterialName)	The name of the material held in the inventory.
FK	Site Object Name	Foreign Key (Table Key = Sites.SiteObjectName)	The name of the inventory site.
	Initial Quantity	Expression	The quantity of material present in the inventory at the beginning of the simulation run.
	Review Period	Enumeration (Enum Type = InventoryReviewPeriod)	The frequency of inventory review to determine whether a replenishment order is required.
FK	Review Timer Name	Foreign Key (Table Key = ReviewTimers.ReviewTimerName)	The name of the timer used to trigger the inventory reviews, if the Review Period is Timer.
	Replenishment Policy	Replenishment Policy	The replenishment policy checked at time of an inventory review to determine whether a replenishment order is required and, if so, the recommended order size.
	Reorder Point / Condition	Expression	The minimum threshold for the inventory position or custom condition that signals the need to place a replenishment order.
	Reorder Quantity / Up-To-Level	Expression	The fixed reorder quantity or target level to return the inventory position to if at or below the reorder point.
	Sourcing Policy	List (List Name = SourcingPolicy, is user defined list with choices None, Single Source, or Make)	The method used to replenish the inventory. Replenishment may come from either a single source or a manufacturing process (Make).
FK	Source Site Object Name	Foreign Key (Table Key = Sites.SiteObjectName)	The source inventory site if the Sourcing Policy is Single Source.
FK	Production Entity Type	Foreign Key (Table Key = ProductionEntityTypes.ProductionEntityType)	The type of production entity to create if the Sourcing Policy is Make.
	Allow Partial Allocation	Enumeration (Enum Type = BooleanType)	Indicates whether a request for material from the inventory can immediately consume only a portion of its requested quantity if the full quantity is not available.
	Allow Backorder Policy	Enumeration (Enum Type = AllowBackorderPolicy)	Indicates whether a request for material from the inventory can be backordered if its full requested quantity is not immediately satisfied from inventory on hand.
	Balk Condition Or Probability	Expression	The balk at backorder condition or probability specified as an expression, if the Allow Backorder Policy is Conditional or Probabilistic. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
	Backorder Max Wait Time	Expression	Optional limit on the total time that a backorder will wait for its full requested quantity of material from the inventory.

Review Timers Table Schema

Defines the timers that drive periodic inventory reviews in the system.

PK/FK	Column Name	Property Type	Description
PK	Review Timer Name	Timer Element (Auto-Create)	The name of the timer that will trigger inventory reviews.
	Time Interval (Hours)	Expression	The time between inventory reviews.

Production Entity Types Table Schema

Defines the production entity types in the system.

PK/FK	Column Name	Property Type	Description
PK	Production Entity Type	Entity (Object Type = ModelEntity, Auto-set Table Row Reference = True)	The name of the production entity type.

Production Routings Table Schema

Defines the routings for the production entity types in the system.

PK/FK	Column Name	Property Type	Description
PK	Routing Key	Integer	The routing key.
FK	Production Entity Type	Foreign Key (Table Key = ProductionEntityTypes.ProductionEntityType)	The name of the production entity type.
	Job Step	Integer	The job step in the routing.
	Location	Sequence Destination	The location of the job step.

Processing Tasks Table Schema

Defines the processing tasks for the production entity types in the system.

PK/FK	Column Name	Property Type	Description
PK	Task Key	Integer	The task key.
FK	Routing Key	Foreign Key (Table Key = ProductionRoutings.RoutingKey)	The routing key.
	Task Sequence Number	Sequence Number	Sequence number used to define the task precedence constraints.
	Task Name	String	Name field for the task.
	Branch Type	Enumeration (Enum Type = TaskBranchType)	Indicates whether the task is always performed, or whether it is instead treated as the first task of a conditional or probabilistic branch in the process workflow.
	Condition Or Probability	Expression	The branch condition or probability specified as an expression, if the Branch Type is not Always. If a condition then enter the logical condition. If a probability then enter the chance of performing the task as a value between 0.0 (0%) and 1.0 (100%)
	Processing Time (Hours)	Expression	The time required to perform the task.
	Loopback To Task	Sequence Number	The task sequence number to loopback to.
	Loopback Branch Type	Enumeration (Enum Type = LoopbackBranchType)	Indicates whether the loopback is a conditional or probabilistic branch in the task workflow.
	Loopback Condition Or Probability	Expression	The loopback condition or probability specified as an expression. If a condition then enter the logical condition. If a probability then enter the chance of performing the loopback as a value between 0.0 (0%) and 1.0 (100%)

Task Resources Table Schema

Defines the resource requirements for processing tasks defined in the Processing Tasks table.

PK/FK	Column Name	Property Type	Description
FK	Task Key	Foreign Key (Table Key = ProcessingTasks.TaskKey)	The task key.
	Resource Type	Enumeration (Enum Type = ObjectSeizeType)	The type of resource that is required.
	Resource Name	Object	The name of the resource that is required.
	Resource List Name	Object List	The name of the resource list from which a resource is required.
	Selection Goal	Enumeration (Enum Type = SeizeSelectionGoal)	The rule used to select a resource from multiple candidates.
	Selection Expression	Expression	The selection expression if the Selection Goal is Largest Value or Smallest Value.
	Request Move	Enumeration (Enum Type = SeizeRequestVisitType)	Indicates whether a move to a specified node will be requested from the resource.
	Destination Node	Node	The name of the destination node that the resource will be requested to move to.

Task Materials Table Schema

Defines the materials requirements for processing tasks defined in the Processing Tasks table.

PK/FK	Column Name	Property Type	Description
FK	Task Key	Foreign Key (Table Key = ProcessingTasks.TaskKey)	The task key.
	Action Type	Enumeration (Enum Type = MaterialActionType)	The type of material-related action required (Consume or Produce).
	Consumption Or Production Type	Enumeration (Enum Type = BOMActionRules)	Indicates whether to consume or produce a single material or a bill of materials.
FK	Material Name	Foreign Key (Table Key = Materials.MaterialName)	The name of the material which is to be either specifically consumed or produced, or whose bill of materials is to be consumed or produced.
FK	Site Object Name	Foreign Key (Table Key = Sites.SiteObjectName)	The name of the inventory site.
	Quantity	Expression	The quantity to be consumed or produced.

Task State Assignments Table Schema

Defines state variable assignments related to processing tasks defined in the Processing Tasks table.

PK/FK	Column Name	Property Type	Description
FK	Task Key	Foreign Key (Table Key = ProcessingTasks.TaskKey)	The task key.
	Assign When	List (List Name = TaskStateAssignmentTrigger, is user defined list with choices Task Ready, Starting Task, or Finished Task)	Indicates when to perform the state assignment.
	State Variable Name	State Property	The name of the state variable that will be assigned a new value.
	New Value	Expression	The new value to assign.

Transportation Times Table Schema

Defines transportation times for TimePath objects placed in the system.

PK/FK	Column Name	Property Type	Description
PK	Time Path Name	Object (Auto-set Table Row Reference = True)	The name of the time path.
	Transportation Time (Hours)	Expression	The transportation time.

Data Table Driven – Demand Driven MRP Approach

The model using the Demand Driven MRP approach is also data table driven with most of the data tables and associated data, similar to the Data Table Driven Approach model. Some of the data, including processing time at the plant and transportation times vary slightly from the first two models to more effectively demonstrate the movement of material using a demand driven MRP approach with buffers at the various locations. The updated data, as well as new data table structures are listed below:

- 1) The time between customer order arrivals at the retailer is now based on a Sales Orders data table where daily orders are shown. In the previous models, orders were exponentially distributed with a mean of 1.5 hours (or approx. 16 per day). Given that rate, the Sales Orders table daily order quantity is between 14 and 18 for the first period, and a small 'spike' in orders begins at the start of week 4, where the order size ranges from 18 to 20 for a 6-day period, then continues between 14 and 18 for the rest of the simulation. Note that the Demand-Driven MRP replenishment policy REQUIRES a Sales Orders table for all DDMRP calculations.

Sales Orders Table Schema

Defines the end-customer orders daily.

PK/FK	Column Name	Property Type	Description
FK	Inventory Name	Foreign Key (Table Key = Inventories.InventoryName)	The name of the inventory of the order.
FK	Material Name	Foreign Key (Table Key = Materials.MaterialName)	The name of the material type of the order.
FK	Site Object Name	Foreign Key (Table Key = Sites.SiteObjectName)	The name of the inventory site.
	Quantity	Real	The customer order quantity.

Order Time	DateTime	The arrival time of the customer order.
Due Date	DateTime	The due date of the customer order.

1. The review period within the ReviewTimers data table is 12 hours, whereas the review period within the first two models was continuous.
2. To be more consistent with the 1-day lead times for the product between site locations, the TransportationTimes data has been changed to be Random.Triangular(13,15,17) Hours for both the SupplierToPlant and DistributionCenterToRetailer times, and to Random.Triangular(12,13,14) Hours for the PlantToDistributionCenter. Note also that the ProcessingTime within the ProcessingTasks data table has changed to Random.Triangular(.5,8,1) hours.
3. The initial quantity within the Inventories table for Finished Product at Retailer has been increased from 23 to 30 to provide starting inventory for the Sales Orders to pull from at the start of the simulation. Note that within the Inventories table, many more properties are available that are used specifically for the Demand-Driven MRP replenishment policy, including Manufactured and DecouplingPoint booleans, LeadTime, MinimumOrderQuantity, DesiredOrderCycle values, as well as Average Daily Usage calculation parameters (ADUCalculationType, past and forward horizons and weights, as well as spike threshold information. These are all REQUIRED for DDMRP calculations.
4. Demand-Driven MRP specific tables, such as ProductStructure, Demand, PlannedAdjustmentFactors, BufferProfiles, AverageDailyUsage, DecoupledLeadTimes, BufferZoneSizes and QualifiedSpikeDemand have been added, as well as links within the Data Connectors area for the 4 DDMRP calculated tables. Note that all of these tables are REQUIRED for DDMRP calculations.
5. The SystemStats table has been added as an updated feature within Simio that allows for table data to be displayed in the Facility window through a Status Table. The statistics shown in the first two models have been duplicated using this Status Table (found on the Animation tab).
6. The Demand table data has been added to reflect the Sales Order data within each step in the process daily. Note that the Demand table structure is now completely different than the Demand table in the second model discussed above.

Demand Table Schema

Defines the end-customer demand streams in the system.

PK/FK	Column Name	Property Type	Description
FK	Inventory Name	Foreign Key (Table Key = Inventories.InventoryName)	The name of the inventory.
	Date	DateTime	The datetime for the inventory quantity.
	Quantity	Real	The daily customer order quantity.

1. Four dashboards have been included in this model. The Buffer Status Planning, Planning Screen and Buffer Run Chart dashboards are all Demand Driven MRP related dashboards and are based on the Inventory Review Log. The Materials dashboard is based on the Material Usage Log.

See Simio help page on 'Demand Driven Material Requirements Planning' for more information on DDMRP calculators, data table structures and additional information on Demand Driven MRP replenishment. The DDMRPExample included with the Simio software also includes several examples.

References

Altiock, T., & Melamed, B. (2007). Simulation modeling and analysis with Arena. Burlington, MA: Elsevier.

PackagingSystem - Example

General Description

This model represents a packaging system. The general process flow is as follows:

- Four types of products are created.
- Products are transported on conveyors.
- Pallets are transported from the Pallet Supply to the palletizer through a conveyor.
- The packaging is done at the palletizer.
- Packages are transported through a conveyor for shipment.

Products

Four Sources create four types of products each flowing on a separate conveyor. The merging takes place in three stages.

- The Soda and Toy products merger
- Grab Bags and Golf Clubs merger
- Complete merge of the four products at the merging zone

After the last merger, the products continue their way to the Palletizer on a conveyor.

Pallets

The Source PalletSupply creates pallets that are transported to the Palletizer through a Power and Free Monorail that is modeled as a Conveyor. The conveyor travel logic *Entity Alignment* is set to 'Cell Location' to represent the hooks position along the conveyor.

Palletizer

Packaging is done at the Palletizer which is modeled using a Combiner. In this model, products represent the Combiner members while the pallet represents the Combiner parent. A Pallet holds six products.

Shipping

Pallets are transported through a Conveyor to the Shipping station where they get destroyed. The Shipping station is modeled as a Sink.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SampleCoWithDemandDrivenMRP - Example

SampleCo with Demand Driven MRP

General Description

This example uses Simio RPS Edition to build a data-generated model for multiple order types with the focus on managing shared resources using dispatching and prioritization between the Make to Stock (MTS) and Make to Order (MTO) materials. The Make to Stock orders use the Demand-Driven Material Requirements Planning (DDMRP) methodologies, while the Make to Order product is manufactured when the order is placed. Because of the contention of multiple shared resources for the various orders, various dispatching rules will be used for comparison of red zone penetration for DDMRP orders and time zone penetration for custom orders.

Simio RPS Edition is a simulation tool for developing applications in Risk-based Planning and Scheduling. RPS is the dual use of a simulation model to generate both a detailed resource constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty. DDMRP calculators and Inventory Review Log used within this example are restricted to Simio RPS Edition licensing. *Note that the ability to change or run this example requires an RPS license. Users with other licensing may only review the example, data tables and results.* For more information or questions about Simio Editions, contact sales@simio.com.

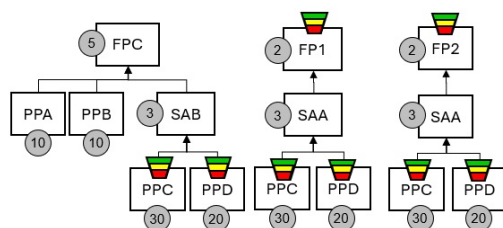
To understand the data schema and scheduling results in this example it is recommended to read the "Planning and Scheduling with Simio" document in C:\Program Files\Simio LLC\Simio. This problem description assumes familiarity with the standard data schema and scheduling concepts that are presented in that document.

If you would like to learn more about DDMRP methodology in general, Demand-Driven Material Requirements Planning (DDMRP) concepts can be found at DemandDrivenMaterialRequirementsPlanning.com (demanddrivenmrp.com). Additional information about the Demand Driven Institute can be found at TheDemandDrivenInstitute.com (thedemanddriveninstitute.com). Simio is certified as a [DDMRP Compliant Software](http://DDMRPCompliantSoftware.com) vendor DDMRP Compliant Software (demanddriveninstitute.com).

The example includes a single manufacturing facility that produces three types of finished goods, two make to stock products, FP1 and FP2, as well as one make to order product, FPC. Subcomponents SAA and SAB are manufactured in the facility where SAA is used in both FP1 and FP2, while SAB is used within FPC. Purchased materials of PPA and PPB are used in the manufacturing of FPC, while PPC and PPD are used within the manufacturing of the subcomponents SAA and SAB.

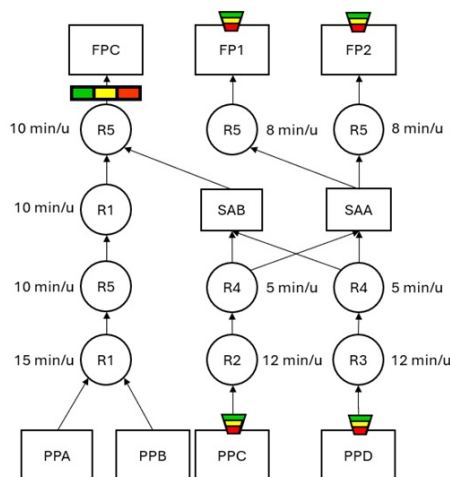
The diagram below shows the product structure and lead times associated with each material in the system.

Product Structure and Lead Times



The below diagram shows manufacturing steps associated with each material, including the finished goods, final assembly, and subcomponents.

Routing BOM and Processing Times



There are five (5) resources in the manufacturing facility, including R1, R2, R3, R4 and R5. FPC is a make to order finished good. It requires the raw materials PPA and PPB, as well as the subcomponent SAB, which is made of raw materials PPC and PPD. Between all materials in the bill of materials, FPC requires processing at all 5 resources. FP1 and FP2 are make to stock finished goods. They require the subcomponent SAA, which is made of raw materials PPC and PPD. These finished goods require processing at all resources except R1.

Decoupling points used for holding inventory of each completed material (or raw material) can be easily specified. Demand-Driven MRP replenishment policies and DDMRP inventory levels can be specified at each decoupling point, or alternative replenishment policies (Min/Max for example) can be used. In this example, decoupling points or buffers, are located at PPC and PPD raw materials, as well as the finished goods FP1 and FP2. Note that a 'time buffer' will be used for the custom order FPC, which is discussed in more detail in the *Detailed Description* section, *Dispatching Rules and Drum Dispatching Rules*.

The template used to start building this model provides custom Object definitions created by using the Subclass feature on Standard Library Objects. These are called SchedSource, SchedServer, etc. and have been specifically designed for use in scheduling models. Additionally, the Site object was designed specifically for multi-site ordering and transfers. In this example, only one Site object is used as the warehouse at the manufacturing facility.

The staging areas are modeled using a SchedTransferNode while the machines are SchedServer Objects. In this document, let "machines" be used to refer to all SchedServer Objects in the model. Machines may require a Secondary Resource, modeled using a SchedWorker in this case, to complete one or more of its *Processing Tasks*. Additionally, each machine may have a sequence dependent setup time specified within the TblSetupTransitions table. In this example, the setup times are shown as model properties for each machine (resource) and are based on changing from one material name to another (i.e., FP1 to FP2 to FPC at R5 for example).

The subcomponent and finished goods materials produced in each facility are described above. Each material has its own routing, unique setup/processing times, and material requirements at each machine within its routing. The ISA95SchemaProductBasedRoutings template (File -> New From Template) discussed in the Planning and Scheduling with Simio document is initially used to organize the production data for this example.

Additional tables are used specifically for Demand-Driven MRP decoupling point inventory areas. A DDMRP ribbon includes specific buttons for calculating Average Daily Usage, Decoupled Lead Times, Buffer Zone Sizes and Qualified Spike Demand. The Average Daily Usage, or ADU, Calculator utilizes data from the Demand table, as well as Inventories table parameters. The Decoupled Lead Times Calculator utilizes data from the Inventories table lead times and decoupling points as well as the ProductStructure table. The Buffer Zone Sizes Calculator utilizes data from a combination of parameters including ADU, Decoupled Lead Times and the Planned Adjustment Factors table. The Qualified Spike Demand Calculator uses data from the SalesOrders table, as well as Inventories table parameters. Finally, the Calculate All button performs all calculations for the forementioned buttons in the order specified. While the user can certainly select the DDMRP ribbon and manually select these buttons, the model (and experiment if used) automatically performs all DDMRP calculations upon running a simulation model (note the 'Automatic' on the Binding Options for each of the DDMRP calculated tables).

This example illustrates the concept of "data-generated" modeling where the model is created automatically by storing Object data in tables, namely Resources, Employees and Vehicles. In this example, the objects that are created from these tables are already mapped to the other standard scheduling data tables.

Detailed Description

The model was originally started by using the File > New From Template > ISA95SchemaProductBasedRoutings template. From this template, the basic structure of the Resources, Routing Destinations, Materials, Manufacturing Orders (now Sales Orders), Routings and Bill of Materials tables was developed. As with any model developed with the ISA95Schema templates, these base tables were then slightly modified to accommodate the Demand-Driven MRP example. Notable changes to the few tables listed include a Site Id for the main equipment in the Resources table, lot sizing information in the Materials table, and the Sales Orders table (originally called Manufacturing Orders). The Sales Orders table simply ships finished goods materials for those that have a replenishment policy, whereas if the order has no review period and isn't a decoupling point, the Check_CustomOrder process will create orders for finished goods and raw materials using the bill of materials structure. Inventory replenishment, as defined through a new Inventories table, along with custom process logic, is used to generate three output data tables, including Manufacturing Orders, Purchase Orders and Stock Transfer Orders. In this example, there will be no stock transfer orders, as all orders are shipped through a single site, Warehouse1.

Resources, Employees and Vehicles Tables

Note that the Resources table can be used to generate any 'Object Type' including SchedServers, SchedTransferNodes (as in the example) but also SchedWorkers and/or SchedVehicles. We have broken out the Employees and Vehicles generation into their own separate tables, with custom object properties specifically related to each. For example, the Employees table includes Home Location, as well as Initial Speed. An Employees Schedule can be used to specify shift schedules and availability for each type of worker in the system. The Employee Details table allows for specific Workcenter/Resource allocations and efficiencies for the employees. Thus, employees can have specific resources at which they are 'qualified' to work and associated labor efficiency for that resource. The Vehicles table includes custom properties such as initial ride capacity, load and unload times, dwelling information (waiting at a site for a given amount of time for X orders), as well as home location and initial speed. Note that there are no vehicles currently within this example. A separate Resource Calendar table, as well as Resource Exceptions are used to specify resource availability and planned downtimes, respectively.

Sites and Suppliers Tables

The Sites table and an associated Site object are used to generate locations for materials within the system. Each Material defined within the Materials table can then be associated with one or more Site. The Inventories table, discussed below, then tracks the combination of material/site in the system. The Sites table allows for Maximum Capacity Specifications, as well as a Destination Vehicle for the site (used when material is being moved from that site to another site). While this example includes maximum capacity values, it currently uses that information only for the Warehouse Capacity dashboard. Custom logic could be added to delay/remove orders when site capacity exceeds specifications.

Dispatching Rules and Drum Dispatching Rules

There are two data tables that are used for dispatching rules for the server resources in the system. R1, R2, R3 and R4 processing areas (and associated SchedTransferNode objects) reference the DispatchingRules data table through the Routing Logic > Dispatching Rules property. These dispatching rules include the first rule which will make sure that all work in process (WIP) orders are started first at the specified locations (SalesOrder_WorkInProcess and MfgOrder_WorkInProcess data tables). The second dispatching rule is LeastSetupTime, followed by EarliestDueDate and LeastSlackTime. With dispatching rules, when there is a tie between orders to be processed on a given machine, the next rule will be used (thus, if remaining orders don't meet the WIP expression, LeastSetupTime will be used. If all setups are the same for the orders in the waiting the machine, then the order's EarliestDueDate will be used, and so on).

There is also a Drum_DispatchingRules table that includes different rules and is used for the R5 server (and associated R5 node which is a Drum_SchedTransferNode). This node references the Drum_DispatchingRules data table through the Routing Logic > Dispatching Rules property. Before discussing these rules, it's important to discuss the custom order FPC and the associated "time buffer", as that will be used in the dispatching rules. FPC orders are made to order and thus, when an order is released using the Sales Order table, if it is for the FPC material, custom logic within the Check_CustomOrder process will generate the specific orders for the top level FPC material required, as well as the component materials SAB, and raw materials PPA and PPB. Note that the lead times for PPA and PPB are both 10 days. The lead time for SAB is only 3 days, and the lead time for FPC is 5 days, once SAB, PPA and PPB are available (see the diagram above titled Product Structure and Lead Times). Thus, the overall lead time for manufacturing FPC is 15 days. Note that within the Sales Orders table, the release date of an FPC material order is 15 days prior to the Due Date (whereas the release date for FP1 and FP2 orders is the same day). Also take note of the column within the Sales Orders table named Time Buffer Due Date. This state column will be calculated based on the TimeBuffer_LeadTime% model property (currently set to 20, or 20%) and the overall lead time. Thus, with the 15-day overall lead time for FPC materials, the Time Buffer Lead Time will reduce the lead time by 20%, thereby those time buffer calculated lead times will be 15 - 15*2 days (see TimeBufferDueDates process) or 12 days. The amount of time between the time buffer date and the due date (3 days) for FPC materials will then be used to compare against the red zone penetration for Demand-Driven MRP orders FP1 and FP2 when dispatching at the main constraint resource ("drum") R5.

Additionally, be sure to understand the Buffer Zones within DDMRP, in particular the Red Zone (see Buffer Zone Sizes Calculator section below, as well as Demand Driven Material Requirements Planning section of Simio Help). Given the time buffer for Make to Order materials (FPC) discussed above, and the Red Zone for Make to Stock materials (FP1 and FP2), the drum dispatching rules used for server R5 are now discussed.

Within the Drum_DispatchingRules data table, the first rule will make sure that all work in process (WIP) orders are started first at the specified locations (SalesOrder_WorkInProcess and MfgOrder_WorkInProcess data tables), which is the same as the DispatchingRules table. Then, the second rule evaluates the status of the orders in queue based on the type of order and uses either the time buffer due date and due date status for custom orders or the quantity in stock and red zone size for DDMRP orders to evaluate the SmallestAttributeValue based on an expression. Note that this expression is dynamically evaluating the orders in the queue each time an order is selected using the dispatching rule.

The Buffer Status Planning dashboard, as well as the Time Buffer Status dashboard, are useful for providing graphical information for all finished good materials.

Inventories and DDMRP Specific Tables

The Inventories table is not specifically a table for DDMRP, however, the Inventory element (generated automatically from the table data) includes the replenishment policy for the material/site combination and may be specified as Demand-Driven MRP. The Inventories table structure also includes DDMRP related parameters, such as Lead Time, Minimum Order Quantity, Desired Order Cycle, Buffer Profile information, Average Daily Usage horizon/weight factors and Spike horizon parameters. If an inventory is specified as Demand-Driven MRP replenishment, these parameters as data within the tables discussed below, will be used to generate the time-indexed output tables specific to demand driven inventory calculations.

The ProductStructure table is like the Bill of Materials table and can be created by simply importing BillOfMaterials table, filtering by MaterialUse of 'Consume' and copy/pasting the Parent Material Name and Component Material Name fields. Note that it is assumed that product structure for a given Material name is consistent between Site locations. In this example, the BillOfMaterials table is Routing-based (and associated Routings table is Inventory-based), whereas the ProductStructure table is only Material based.

The Demand table includes prior demand (for 'Past' Average Daily Usage calculator) as well as future demand. It differs from Sales Orders in that it includes the full bill of materials explosion of demand for all inventories, including subcomponents and purchased parts. While this Demand table isn't considered a time-indexed table, it is time-based and should include the Date for each demand entry. A missing date indicates a value of '0' and will impact the Average Daily Usage calculations.

PlannedAdjustmentFactors are specifically for adjusting the demand, red/yellow/green zones or lead time factors associated with buffer zone sizes. This is a time-indexed table and based on the StartDate and EndDate for the adjustments.

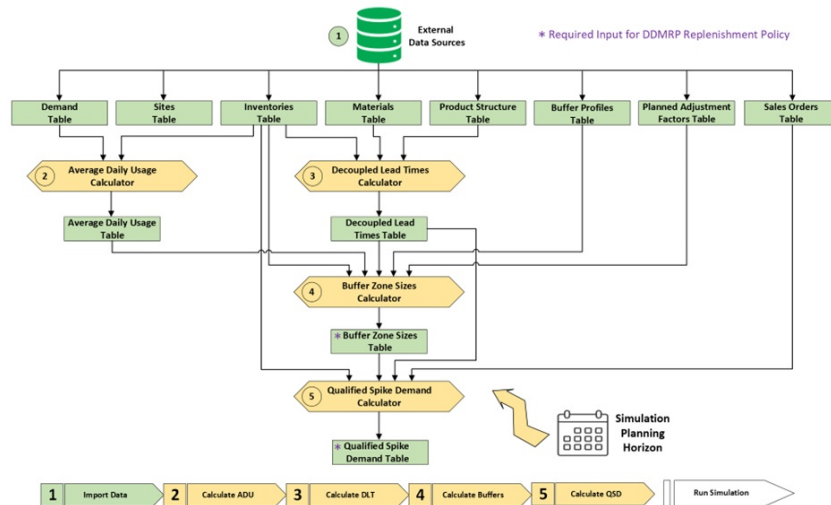
The BufferProfiles table contains information related to the Lead Time Factor and Variability Factor used in Buffer Zone Sizes calculations. The Key column in this table is referenced from within the Inventories table for each inventory (material/site) defined. Default data for purchased, manufactured, distributed and intermediate part types has been defined for lead time categories (short/medium/long) and variability categories (low/medium/high). Additional profiles may be added.

DDMRP Calculated Output Tables

The DDMRP calculators are used to generate the associated output tables. The next section discusses the calculator in detail. The AverageDailyUsage table is time-indexed and is used for calculating the BufferZoneSizes table. The DecoupledLeadTimes table is based on the Lead Times and Decoupling Point Boolean within the Inventories table. This information will also be used for calculating the BufferZoneSizes table. The BufferZoneSizes and QualifiedSpikeDemand tables are also time-indexed, and their data is fed directly into the Inventory elements that are generated via the Inventories table (see below, all replenishment policy property values are input here – do not change these, this is FYI only).

Simulation Input Data Preparation Using DDMRP Calculators

The diagram below shows the relationship between the various data input tables, calculators and output tables.



DDMRP Calculators

The following includes more detailed information about the DDMRP calculators, associated parameters and actual calculations.

ADU CALCULATOR

A past ADU is calculated using the following equation:

$$\text{Past ADU} = \frac{\text{Sum of Past Horizon Demand}}{\text{Past Horizon}}$$

A forward ADU is calculated using the following equation:

$$\text{Forward ADU} = \frac{\text{Sum of Forward Horizon Demand}}{\text{Forward Horizon}}$$

A blended ADU is calculated using the following weighted average equation:

$$\text{Blended ADU} = \frac{(\text{Past ADU} * \text{Past Weight}) + (\text{Forward ADU} * \text{Forward Weight})}{\text{Past Weight} + \text{Forward Weight}}$$

DECOUPLED LEAD TIME CALCULATOR

Decoupled lead time (DLT) is a qualified cumulative lead time concept in DDMRP. For manufactured items, it can be defined as: The longest cumulative coupled lead time chain in the product structure, limited and defined by the placement of decoupling point buffers within that structure.

Any manufactured item with at least one coupled component will always have a longer decoupled lead time than its manufacturing lead time. If an inventory item is not manufactured (i.e., is replenished by purchase or stock transfer orders) then the decoupled lead time is simply the purchasing or transfer lead time.

BUFFER ZONE SIZES CALCULATOR

The equations used to calculate the buffer zone sizes are as follows:

$$\begin{aligned} ADU &= ADU * Demand Adjustment Factor \\ DLT &= DLT * Lead Time Adjustment Factor \\ Red\ Zone\ Base &= ADU * DLT * Lead Time Factor \\ Red\ Zone\ Safety &= Red\ Zone\ Base * Variability Factor \\ Red\ Zone\ Size &= (Red\ Zone\ Base + Red\ Zone\ Safety) * Red\ Zone\ Adjustment Factor \\ Yellow\ Zone\ Size &= ADU * DLT * Yellow\ Zone\ Adjustment Factor \\ Green\ Zone\ Size &= MAX(Minimum\ Order\ Quantity, ADU * Desired\ Order\ Cycle, ADU * DLT \\ &\quad Lead\ Time\ Factor) * Green\ Zone\ Adjustment Factor \end{aligned}$$

The calculated buffer red, yellow, and green zone sizes are rounded to the nearest integer value.

QUALIFIED SPIKE DEMAND CALCULATOR

The order spike horizon (a future time window starting 'tomorrow') is calculated using the following equation:

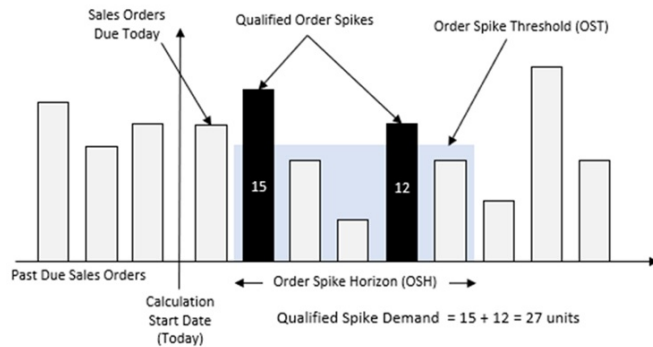
$$Order\ Spike\ Horizon\ (OSH) = (DLT * Spike\ Horizon\ Factor) + Spike\ Horizon\ Constant$$

The OSH is rounded to the nearest integer value.

The order spike threshold is calculated using the following equation:

$$Order\ Spike\ Threshold\ (OST) = (Red\ Zone\ Size * Spike\ Threshold\ Factor) + Spike\ Threshold\ Constant$$

The calculated qualified spike demand values are rounded to the nearest integer value.



Additional Tables for Dashboards

Additional output tables for dashboards have been generated for the purpose of displaying information on costing, including costing by material site and by resource.

Processes Window

This example was developed from the original DDMRPExample provided with the Simio software. Additional processes were added, including Check_CustomOrder (called from within the SO_Generation source object), as well as R5Node_Entered (for time buffer evaluation of FPC), TimeBufferDueDates and parallel route controller processes. For more information on parallel route controller logic, see the SchedulingParallelRouteController example provided with the Simio software.

Appendix A – Data Tables

The following is a summary of the default tables for the Demand-Driven MRP template.

Resources:

A list of 'resources' or SchedServer objects that are in the manufacturing facility, can also include SchedTransferNodes and SupplierSource objects.

Column Name	Description
Resource Name	The unique name of the resource. This is the object name.
Description	A description of this resource.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Site Id	The Site object associated with this resource.
Object Type	This references an object type (sub-classed object) of the object.
XLocation	This is the X location of the object in the model.
ZLocation	This is the Z location of the object in the model.
Work Schedule	Work schedule assigned to the resource.
Capacity	Capacity of the resource.
IdleCostRate	The hourly idle cost rate when a resource is idle.
CostRate	The hourly rate for the resource while being utilized.
Changeover Matrix	Changeover Matrix used by the resource.
Supplier Vehicle	This is the vehicle/worker that is used by a SupplierSource object for transport to a site.
Changeover Efficiency	State column, which is a factor used to increase or decrease the total time taken for a setup.

Current Idle Cost	Costing information tracked for daily idle statistics (an output).
Current Usage Cost	Costing information tracked for daily utilization statistics (an output).

Routing Destinations:

A list of each possible destination node for each resource. The entity is routed to one of the nodes specified in this list based on the selection rule.

Column Name	Description
Resource Name	This is the transfer node used to route each entity to their destinations.
Node	Possible destination from this transfer node.

Materials:

A list of materials that can be produced at any manufacturing facility.

Column Name	Description
Material Name	The unique name of this material.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Material Class	A description of this material.
Infinite Availability	Boolean to determine whether the material availability should be considered infinite (and thus will not constrain the system per Bill Of Materials).
Material Cost	The cost per unit of this material.
Material Color	The color index for computing color-dependent changeovers.
Gantt Color	The color used for drawing orders for this material in the Gantt.
Material State Statistic	State statistics are used to plot material quantities over time on the Gantt.
Minimum Split Lot Size	Minimum lot size for determining order lot size from recommended order quantity.
Maximum Split Lot Size	Maximum lot size for determining order lot size from recommended order quantity.
Rounding Split Lot Size	Rounding split lot size for determining order lot size from recommended order quantity.
Use Remainder Instead of Rounding Lot Size	Boolean to determine whether remainder will be used when lot sizing for full recommended order quantity. If this Boolean is checked, all smaller lots will equal the recommended order quantity, which otherwise may be slightly smaller value.
Symbol Index	Entity symbol index used for orders of given material name.
Raw Material Supplier	Name of the SupplierSource for which a raw material is sourced (see Purchase Orders table). Used for raw materials only.

Routings:

A list of job routings for each inventory (material/site combination) specifying the resource and processing time for each task required to produce the material.

Column Name	Description
Routing Key	The unique name for this routing step.
Sequence	The transfer node (list of resources) or resource where this step is to be processed.
Inventory Name	The foreign key property specifying the inventory (material/site) for the specified routing step(s).
Route Number	The routing number.
SetupTime	Discrete setup time for this step. (see Definitions > SequenceDependentSeutpMatrix for more information).
ProcessTime	Processing time for this step. In this example, the process time is multiplied by the ManufacturingOrders.Quantity, thus process time is 'per unit'.

Bill Of Materials:

A list of component materials that are required at a specific routing sequence location to produce a material.

Column Name	Description
Routing Key	The material routing step where the material action takes place.
Parent Material	The name of the parent material that is associated with the routing step (used for reference only).
Component Material	The name of the component material to be consumed or produced.
Required Quantity	The quantity of the material that is either consumed or produced. This value based on an order quantity = 1. The required quantity is multiplied by the ManufacturingOrders.Quantity.
Required Lot Id	Specified the lot id required for consumption and the lot id provided for production.
Material Use	Specifies if the material is to be consumed or produced.

Product Structure:

A list of pairs of parent/component materials that are used for determining decoupled lead times.

Column Name	Description
-------------	-------------

Parent Material Name	The name of the parent material associated with the routing step (used for reference only).
Component Material Name	The name of the component material to be consumed or produced.
Sales Orders:	
A list of all sales orders to be processed during this planning period.	
Column Name	Description
Enabled	Boolean that determines if the order will be enabled/used in the simulation run.
Order Id	A unique string name assigned to this sales order.
Inventory Name	A foreign key reference to the Inventories table for the material/site from which the order will be shipped.
Release Date	The date-time this order is released to production.
Due Date	The date-time this order is due.
Priority	The scheduling priority for this order.
Quantity	The material quantity to be produced for this order.
Adjusted Release Date	This field is an expression field used by the source to determine when to release orders considering any orders whose release date is earlier than the start date (will attempt to ship those as well using Math.Max expression.
Ship Date	The ship date for this order (an output) based on the generated schedule.
Production Cost	The production cost for this order (an output) based on the generated schedule.
Target Ship Date-Value	The target ship date for this order.
Target Ship Date-Status	The target ship date status; OnTime, Late, or Incomplete.
Target Cost-Value	The target cost for this order.
Target Cost-Status	The target cost status: OnBudget, Overrun, or Incomplete.
Note that the <i>Ship Date</i> and <i>Production Cost</i> are both output columns in the Sales Orders table; they are written from the simulation model to the table as the schedule is generated by the deterministic simulation run. Also note that the last four columns are targets.	
Inventories:	
A list of inventories that are specific materials at each site, along with many associated DDMRP parameters.	
Column Name	Description
Inventory Name	The unique name of this inventory ("material_site" plus "site_material" recommended).
Material Name	The name of the material for this material/site inventory.
Site Object Name	The name of the site for this material/site inventory.
Initial Quantity	The quantity of material present in the inventory at this site at the beginning of the simulation run.
Review Period	The frequency of inventory review to determine whether a replenishment order is required. If the review period is 'Continuous', then an inventory review is triggered once at the start of the simulation and then whenever the inventory position decreases. If the review period is 'Timer', then an inventory review is triggered whenever the specified timer fires its event.
Review Timer Name	The name of the timer that is used to trigger inventory reviews. For any Demand-Driven MRP replenishment, it's recommended using the DailyTimer.
Replenishment Policy	The replenishment policy checked at the time of an inventory review to determine whether a replenishment order is required and, if so, the recommended order size. If the Decoupling Point Boolean is 'True', this should be set to 'Demand-Driven MRP'.
Reorder Point or Condition	For Reorder Point/Reorder Quantity or Min/Max replenishment policy, the minimum threshold for the inventory position that signals the need to place a replenishment order. For Custom Reorder Condition, the condition that, if true, signals the need to place a replenishment order.
Reorder Quantity / Up-To-Level	For Reorder Point/Reorder Quantity or Custom Reorder Condition replenishment policy, the fixed reorder quantity when placing a replenishment order. For Min/Max or Order-Up-To-Level, the target level to return the inventory position to if below that level.
On Replenishment Order Process	The name of the process that will be executed to handle an inventory replenishment order. Note that whenever this process is triggered by the replenishment policy, the inventory's quantity on order will have been automatically increased by the size of the recommended order size. (Note: the created token's material order detail reference will provide the detail of the replenishment request.)
Sourcing Policy	Currently, it can be specified as PreferredCapableToPromise or SmallestLeadTime. The example will be the highest priority supplier that can deliver within the planned lead time window, if none are within the window select the smallest lead time.
Manufactured	Boolean specifying whether or not the inventory (material/site) is manufactured or not. This entry is used within the DDMRP calculators.
Decoupling Point	Boolean specifying whether or not the inventory (material/site) is decoupled or not. This entry is used within the DDMRP calculators.
LeadTime (Days)	The number of days of lead time to either manufacture, purchase or transfer the inventory (material/site). This entry is used within the DDMRP calculators to determine Decoupled Lead Time.
Minimum Order	The minimum order quantity for the inventory (material/site). This entry is used within the

Quantity	DDMRP calculators for determining the Green Zone size (if large enough, will factor into this size).
Desired Order Cycle (Days)	This order cycle determines how often an order should be placed when using DDMRP calculators. It may factor into the Green Zone size.
Buffer Profile Key	The key column specified within the Buffer Profiles table that will indicate the lead time and variability factors within DDMRP calculators.
Estimated ADU	This field will be used for a Demand-Driven MRP based inventory if there is not Demand information within the Demand table.
ADU Calculation Type	Average Daily Usage (ADU) calculation type can be Forward, Past, Blended or None. For Forward calculations, the Forward Horizon (Days) is used. For Past calculations, the Past Horizon (Days) is used. For Blended, both the Forward Horizon (Days) and Past Horizon (Days) are used, as well as the Past Weight and Forward Weight. For None, no ADU will be calculated.
Past Horizon (Days)	The number of past days to consider when performing a 'Past' or 'Blended' ADU calculation.
Forward Horizon (Days)	The number of forward days to consider when performing a 'Forward' or 'Blended' ADU calculation.
Past Weight	The weight associated with the 'past' data when performing a 'Blended' ADU calculation.
Forward Weight	The weight associated with the 'forward' data when performing a 'Blended' ADU calculation.
Spike Horizon Factor	The spike horizon factor is used within the Qualified Spike Demand DDMRP calculator and is multiplied by the DLT (decoupled lead time) within the Order Spike Horizon.
SpikeHorizonConstant (Days)	The spike horizon constant is an optional constant used within the Qualified Spike Demand DDMRP calculator and is added to the (DLT * Spike Horizon Factor) within the Order Spike Horizon calculation.
Spike Threshold Factor	The spike threshold factor is used within the Qualified Spike Demand DDMRP calculator and is multiplied by the Red Zone Size within the Order Spike Threshold.
Spike Threshold Constant	The spike threshold constant is an optional constant used within the Qualified Spike Demand DDMRP calculator and is added to the (Red Zone Size * Spike Threshold Factor) within the Order Spike Threshold calculation.
On Hand Tally	The name of the automatically generated tally for calculating on hand values for inventory target ration (ITR) over time.
On Hand Squared Tally	The name of the automatically generated tally for calculating on hand squared values for inventory target ration (ITR) over time.
Cpm	Daily updated with Cpm (Taguchi Capability Matrix) values until the end of the simulation run (an output).
Next Scheduled Delivery	Tracks the next scheduled delivery of materials (an output).
CpmSorted	Boolean used for tracking and sorting Cpm (Taguchi Capability Matrix) at end of simulation run – used for dashboard creation of green/yellow/red areas.

Sites:

A list of distribution sites that are either stand alone or attached to a production facility/factory.

Column Name	Description
Site Object Name	The unique name of the site. This is the object name for the Site.
Latitude	The object's latitude, in degrees (-90 to +90).
Longitude	The object's longitude, in degrees (-180 to +180).
Factory Start Node	For those sites with attached factories, the name of the node for the factory. In this example/template, the MoArrivals node should be used.
Maximum Work In Process	Associated with the Site object, this field is currently not used and set to Infinity.
Maximum Capacity	The maximum capacity of the site, which is used in the Warehouse Capacity dashboard to display the % of inventory at that site by material. Note that materials are not stopped from transferring or manufacturing due to this capacity (additional custom logic could be added), but this value is used for dashboards only.
Destination Vehicle	The name of the vehicle/worker that transfers inventory from this site to another site.

Manufacturing Orders (Output Table):

A list of all manufacturing orders generated during the simulation run to manufacture, which includes finished goods (within a factory at a given site) or component materials.

Column Name	Description
Order Id	A unique string name assigned to this manufacturing order.
Inventory Name	A foreign key reference to the Inventories table for the material/site for which the order will be manufactured.
Material Name	The material name associated with this manufacturing order.
Site Id	The site id associated with this manufacturing order.
Quantity	The number of inventory parts associated with this order. Note: This value is impacted by the Lot Sizing associated with the Material and may/may not match the recommended order quantity.
Type	This example currently only generates DemandDriven type manufacturing orders.
Release Date	The date of release of the manufacturing order to the manufacturing/factory floor and is based on the review date when the recommended order quantity was generated (typically daily).
Due Date	The due date of the order, based on the release date + lead time of the inventory (material/site).

Completion Date	The date when the manufacturing order was completed.
Priority	Priority of the manufacturing order (not currently used, as manufacturing orders are not directly associated with Sales Orders with priority).
Complete	Boolean indicating if the manufacturing has been completed.
Planning Priority	Priority based on DDMRP planning, including NetFlow / (GreenZoneSize + YellowZoneSize + RedZoneSize).
Percent Red Zone	Priority based on DDMRP execution, including QuantityInStock / RedZoneSize.
Purchase Orders (Output Table):	
A list of all purchase orders generated during the simulation run ship from one of the suppliers, given that the <i>InfiniteRawMaterials</i> property of the model is 'False'. If this property is 'True', raw materials will be assumed always available and thus, no purchase orders will be generated.	
Column Name	Description
Order Id	A unique string name assigned to this purchase order.
Material Name	The material name associated with this purchase order.
Site Id	The site id associated with this purchase order.
Quantity	The number of inventory parts associated with this order. Note: This value will not have any applied lot sizing but will strictly be based on the recommended order quantity determined by the replenishment policy.
Order Date	The date that order was released for the processing at raw material supplier to start.
Due Date	The due date of the purchase order, based on the order date + lead time of the inventory (material/site).
Lead Time (Hours)	Lead time of the purchased material inventory, based on Inventories table.
Arrival Date	The date when the purchase order was received at the destination factory/site.
Duration (Hours)	The total duration, in hours, from order date until arrival date, which includes the lead time plus any transportation.
Raw Material Supplier	The supplier where the raw material will be purchased. This is determined by the Materials table property 'RawMaterialSupplier' for the given material.
Current Stock Level	At the time of order date, the current level of inventory for the given material/site. This value is for display purposes and is currently not used in any decision making.
Stock Transfer Orders (Output Table):	
A list of all stock transfer orders generated during the simulation run to move between sites, which currently only includes finished goods. In this example, DC1 site is a warehouse without an associated factory, thus when inventory is required at that location, it must transfer from one of the other warehouses with associated manufacturing/factory.	
Column Name	Description
Order Id	A unique string name that is assigned to this manufacturing order.
Source Site ID	The site from which the stock transfer order will initiate.
Destination Site ID	The site to which the stock transfer order will be completed.
Material Name	The material name associated with this stock transfer order.
Quantity	The number of inventory parts associated with this stock transfer order. Note: This value is impacted by the Lot Sizing associated with the Material and may/may not match the recommended order quantity.
Start Date	The start date of the transfer from the source site to the destination site.
Due Date	The due date of the order, based on the start date + lead time of the inventory (material/site) at the destination site.
Completion Date	The date when the stock transfer order was completed (arrived at destination).
Duration (Hours)	The total duration, in hours, from start date until arrival date, which includes the lead time plus any transportation.
Current Stock Level	At the time of stock transfer date, the current level of inventory at the destination site for the given material. This value is for display purposes and is currently not used in any decision making.
Suppliers:	
This table includes a list of the various inventories (material/site) that can be transferred from one Site to another (stock transfer orders).	
Column Name	Description
Inventory ID	This field indicates the inventory (material/site) from which the customer Sales Order will be shipped. At least one entry for each finished goods inventory (shipped as Sales Orders) should be entered.
Supplier Inventory ID	This field indicates the supplier inventory (material/site) that MAY be used if the inventory is not available from the associated Inventory ID. When a Supplier Inventory ID is blank, no transfer order will be attempted, and it is assumed that the inventory will be replenished from an associated manufacturing facility/factory. If multiple Supplier Inventory IDs are specified, then the Inventories table Sourcing Policy property will be used to determine how to decide between the multiple suppliers. Associated lead times for in-stock and out-of-stock may be used.

Minimum Order Quantity	The minimum order quantity for the transfer orders.
In_LeadTime (Hours)	The in-stock lead time that is calculated for this inventory transfer, as calculated at the start of the simulation run.
Out_LeadTime (Hours)	The out-of-stock lead time that is calculated for this inventory transfer, as calculated at the start of the simulation run.
Employees:	
This table includes a list of the various employees in the system. These employees may be used to transfer material between locations or more typically, assist with setup/processing type work at a SchedServer.	
Column Name	Description
Employee Name	The unique name of the employee (worker) resource. This is the object name.
Description	A description of this worker.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Object Type	The object type associated with this employee, typically SchedWorker.
Site Id	The site object where this worker is associated.
XLocation (Meters)	This is the X location of the object in the model.
ZLocation (Meters)	This is the Z location of the object in the model.
Capacity	Capacity of the worker (number of individual units of employee name).
CostRate	Cost of the employee, per hour, charged for both idle and usage time.
Home Location	Initial physical location of the employee, generally a node object.
InitialSpeed (Meters per Second)	Initial walking speed of the employee between object (server) locations.
Employees Schedule:	
This table includes the work schedules of the various employees defined in the Employees table. Employee schedules will repeat weekly. The Data > Work Schedules area for the EmployeesTableSchedule references this table and can be edited if necessary.	
Column Name	Description
Employee Name	The reference key to the name of the employee (worker) resource. Multiple rows may be associated with a given employee (first shift hours may include an entry for morning and another for afternoon, with a break during lunch for example).
Start Date	The start date/time of the worker shift.
End Date	The end date/time for the worker shift.
Capacity	The capacity of the worker during the start > end date/time specified. If a time period is not specified, the capacity of 0 is assumed (breaks, offshift, for example).
Employee Details:	
This table includes employee qualifications and labor efficiency for given workcenters.	
Column Name	Description
Workcenter Name	The name of the Transfer Node 'area' of similar processing workstations for which a given employee can work.
Resource Name	The name of the specific SchedServer (or similar server) at which the employee can work.
Labor Name	The name of the employee associated with the given workcenter and resource. Multiple employees may have the same workcenter and resource names and then the worker will be selected among the 'group' of employees.
Labor Efficiency	The efficiency applied to the setup and/or processing time when this worker is used at a given workcenter and resource.
Vehicles:	
This table includes a list of the various Vehicles in the system. These Vehicles are typically used to move inventory from one location to another, for example, from a supplier to factory or warehouse to distribution center (site to site).	
Column Name	Description
Vehicle Name	The unique name of the vehicle. This is the object name.
Description	A description of this vehicle.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Object Type	The object type associated with this vehicle, typically SchedVehicle.
XLocation (Meters)	This is the X location of the object in the model.
ZLocation (Meters)	This is the Z location of the object in the model.
Initial Quantity	Initial number of Vehicles of this type in the system.
Initial Ride Capacity	The initial carrying capacity of this vehicle (the number of orders that may be transported at a time).
LoadTime (Minutes)	The time required for this vehicle object to load an order (i.e., load time per order or entity).
UnloadTime (Minutes)	The time required for this vehicle object to unload an order (i.e., unload time per order or entity).
Minimum Dwell Time Type	Specifies the minimum dwell time requirement for this vehicle object when loading and unloading orders at a node.

MinimumDwellTime (Hours)	The specific minimum amount of time that this vehicle object is required to wait, or 'dwell' at a node to load and unload entities.
DwellEventName	The name of the event whose occurrence will indicate the expiration of the minimum wait time requirement for this vehicle object when loading and unloading orders at a node.
InitialSpeed (Miles per Hour)	The initial desired speed value for this type of vehicle.
Home Location	The initial home node location for vehicles of this type at the beginning of the simulation run.
CostRate	The cost per unit time that is charged to the cost of the vehicle when it is idle, utilized for a non-transport task or transport task.

Dispatching Rules:

This table includes the dispatching rules utilized for dispatching multiple orders at a given SchedTransferNode (prior to multiple SchedServer selection).

Column Name	Description
Dispatching Rule	The criteria used to select the next order from a transfer node queue (at start of workcenter group of SchedServer objects). Note that using a particular dispatching rule may require some specific model data about the candidate entities (orders), such as due dates, job routings, expected setup or operation times. See Simio's dynamic selection rule documentation for more information.
Attribute Value Expression	The expression used to get the attribute value for each candidate entity (order). In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (orders), for example, Candidate.Entity.Priority.
Campaign Value Expression	The expression used to get the campaign value for each candidate entity (order). In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (orders), for example, Candidate.Entity.Priority.

Resource Calendar:

This table includes the resource calendar (for Resources table objects). Resource schedules will be repeated weekly. The Data > Work Schedules area for the ResourcesTableSchedule references this table and can be edited if necessary.

Column Name	Description
start_calendartime	The start date/time of the resource calendar shift.
end_calendartime	The end date/time for the resource calendar shift.

Resource Exceptions:

This table may include any exceptions to the standard resource calendar shown above. This may include different entries for various SchedServers, and typically may be used for preventive maintenance or downtime events.

Column Name	Description
Resource Name	The reference key to the name of the resource from the Resources table.
PM_Start Time	The start date/time of the preventive maintenance of the resource.
PM_End Time	The end date/time for the preventive maintenance of the resource.

Demand:

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The demand should be daily time based and may include past data as well as future demand. If a date is not specified in the data, a '0' quantity is assumed. Note: this should also include a full explosion of bill of material component part demand over the time periods as well.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this demand.
Date	The date/time for this inventory/quantity combination. Any missing dates (daily) will be assumed to be '0' quantity when calculating Average Daily Usage.
Quantity	The number of units required of a given inventory on the specified date.

Planned Adjustment Factors:

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. It can include multiple inventory entries with varying start and end dates.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this planned adjustment.
Start Date	The start date of the planned adjustment.
End Date	The end date of the planned adjustment.
Demand Adjustment Factor	Adjustment factor that will be multiplied by the ADU when calculating the buffer zone sizes.
Red Zone Adjustment Factor	Adjustment factor that will be multiplied by the red zone size (includes both the red zone safety + red zone base).
Yellow Zone Adjustment Factor	Adjustment factor that will be multiplied by the yellow zone size (includes ADU * DLT).
Green Zone Adjustment Factor	Adjustment factor that will be multiplied by the green zone size (includes the max calculation of minimum order quantity, ADU * desired order cycle, ADU * DLT * lead time factor).
Lead Time Adjustment Factor	Adjustment factor that will be multiplied by the decoupled lead time DLT.

Buffer Profiles:

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Buffer Profiles table defines the various part types with associated lead time and variability categories. These are then referenced within the Inventories

table.

Column Name	Description
Key	The name of the buffer profile key (referenced within Inventories table).
Part Type	Descriptor of the type of part, such as purchased, distributed, intermediate, or manufactured.
Lead Time Category	Descriptor for the type of lead time category (short, medium or long).
Variability Category	Descriptor for the type of variability category (low, medium, or high).
Lead Time Factor	Lead time factor that is used in calculating the Red Zone Base and Green Zone Size within the Buffer Zone Sizes calculator.
Variability Factor	Variability factor that is used in calculating the Red Zone Safety within the Buffer Zone Sizes calculator.

Average Daily Usage (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Average Daily Usage DDMRP calculator utilizes the Demand table, in addition with Inventories table parameters to generate this time-indexed table.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this average daily usage.
Date	The date/time for this inventory/quantity combination.
ADU	The average daily usage as calculated for this inventory/date.

Decoupled Lead Times (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Decoupled Lead Times DDMRP calculator Lead Time and Decoupling Point Boolean within the Inventories table, along with the Product Structure table to calculate the decoupled lead times for each inventory.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this average daily usage.
Decoupled Lead Time (Days)	Decoupled lead time calculated for the associated inventory.

Buffer Zone Sizes (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Buffer Zone Sizes DDMRP calculator utilizes many parameters (discussed above in DDMRP Calculators section) to determine the time-indexed daily values for each inventory. This information is fed directly into the Inventories element.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this buffer zone size.
Date	The date/time for this inventory/quantity combination.
Red Zone Size	The red zone size calculated for the associated inventory/date.
Yellow Zone Size	The yellow zone size calculated for the associated inventory/date.
Green Zone Size	The green zone size calculated for the associated inventory/date.

Qualified Spike Demand (Table with Calculator Outputs):

This table is used with Demand-Driven MRP (DDMRP) inventory type materials. The Qualified Spike Demand DDMRP calculator utilizes the Sales table as well as parameters within the Inventories table to determine qualified spike values for each inventory/date.

Column Name	Description
Inventory Name	Name of the inventory (material/site combination) for this qualified spike demand.
Date	The date/time for this inventory/quantity combination.
Quantity	The qualified spike demand quantity calculated for the associated inventory/date.

Costing (Output Table):

This table provides general costing information by site for dashboard reports.

Column Name	Description
DateTime	Date/time for costing information which is calculated daily.
Location	Site location for the total cost information.
Total Cost	Total cost of inventory at the associated site for the day (date/time).

Costing by Material Site (Output Table):

This table provides costing information by material site for dashboard reports.

Column Name	Description
DateTime	Date/time for costing information which is calculated daily.
Site	Site location for the total material cost information.
Material Name	Name of the material for which the cost is associated.
Total Material Cost	Total cost of inventory at the associated site for the day (date/time).

Costing by Resource (Output Table):

This table provides costing information by resource for dashboard reports.

Column Name	Description
DateTime	Date/time for costing information which is calculated daily.
Site	Site location for the total resource cost information.
Resource Name	Name of the resource for which the cost is associated.

Idle Cost	Total idle cost for the site/resource at the given datetime.
Usage Cost Charged	Total usage cost charged for the site/resource at the given datetime.

Cpm and CpmSorted (Output Tables):

These tables are generated daily to calculate the Cpm (Taguchi performance index) used for Dashboard Reports. They report the statistical tally and tally squared values based on on-hand inventories, optimal high/low values and target.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

SchedulingBatchBeverageProduction - Example

General Description

This example uses Simio RPS Edition to schedule a batch processing system that mixes and fills a beverage product. Although Simio RPS Edition is required to build this model, it may be viewed using Simio Design or Professional Edition. If in Trial Edition, to give you the best planning and scheduling experience, please contact sales@simio.com.

To understand the data schema and scheduling results in this example you should first read the *Planning and Scheduling* with Simio document found in C:\Program Files\Simio LLC\Simio. This problem description assumes that you are familiar with the standard data schema and scheduling concepts that are presented in that document.

In this example, we wish to generate a 30-day production schedule for this facility that fully accounts for the limited resources in the system. This example contains orders for both manufactured material and finished material. Raw materials are also modeled as a constraint in this system. The manufactured materials, named Green Bulk, Red Bulk and Blue Bulk, must be mixed at a mixing machine and then put into a Tank. The finished good materials, such as Green Pack, Red Pack and Blue Pack, begin at a Filler machine and are then packed at a Packing machine. Workers are required at each of these steps in the process and the appropriate manufactured material must be available in the Tanks in order for the finished goods produce to be processed. Raw materials, such as bottles and labels, are also modeled in this system and are required as part of the finished goods filling process.

Detailed Description

Arrival of Manufacturing Orders

The ManufacturingOrderArrivals source creates ManufacturingOrderEntity type entities, which represent manufacturing orders. This source produces arrivals according to the ReleaseDate column in the Manufacturing Orders table. The system does not begin empty. It is initialized with a few orders already in the system and having completed a portion of their processes. These orders are listed in the table named WorkInProcess, which also indicates the current step in the routing, the quantity that has already been completed and any cost that has been accrued up to this point in time. The WIP orders that have started on a resource will be schedule before the NEW orders.

Manufactured Material (GreenBulk, RedBulk, BlueBulk)

If an order is a manufactured material, it will first route to a Mixing machine and then to a TankFill machine, as outlined in the Routings table. In the Mixing TransferNode, the entity checks to see if the raw materials that are required to be consumed at the Mixing Server are available. If they are not available, the entity will wait at the Mixing TransferNode, instead of waiting at the Mixing Server.

Once the materials are available, they can be selected to be run on a Mixing Server. The Mixing TransferNode uses a *Route Request Dynamic Selection Rule* to determine the sequence of jobs for Mixing1, Mixing2 and Mixing3. The *Route Request Dynamic Selection Rule* is set to a Referenced Property so that a user can experiment with using different routing rules. By default, this *Mixing Selection Rule* property is set to 'Standard Dispatching Rule' which has many child properties. For the *Standard Dispatching Rule*, the *Dispatching Rule* property is set to 'LeastSetupTime' and the *Tie Breaker Rule* property is set to 'EarliestDueDate'. Using these *Standard Dispatching Rule* properties, entities will be selected based on least amount of setup time on a Mixing server. If more than one entity has the same least setup time, the entity with the earliest due date will be selected.

Before Setup occurs at each Mixing Server, the process AddManufacturingOrdersOutput adds a row in the table named ManufacturingOrdersOutput and records the start time of this order on this Mixing Server. The endtime is recorded after this order is finished on this Mixing server. After processing, the "WaitingForTankFillProduce" process is called from the WaitingForTankFillProduce task on the Mixing Server. This process will create the downstream entity and route it to the TankFill TransferNode. The original entity will then wait until the 'Produce' task is completed on the TankFill Server.

The order will route to the TankFill Transfer node, where it will select to route to either the TankFill1 or TankFill2 workcenter by selecting from the node list TankFillList. Similar to what occurs at the Mixing workcenter, before Setup occurs at each TankFill workcenter, the process AddManufacturingOrdersOutput adds a row in the table named ManufacturingOrdersOutput and records the start time of this order on this TankFill machine. The endtime is recorded after this order is finished on this TankFiller workcenter. After the order is completed at the TankFiller, it is routed to the server named HoldingTankStorageServer where it waits until it is needed by an entity at the Filler machine. At that point, the entity representing this manufactured material order will get transferred to the DisposeBulkOrders sink, where the current time and cost is recorded in the Manufacturing Orders table for this manufactured material order.

After the order is completed at the TankFiller, it is routed to the server named HoldTank. First, the material for the mixing entity is produced and it is given a lot id. Since it is given a lot id, only entities consuming the same material and lot id is possible. After the produce, the 'WaitForTankRelease' process is called. Using this process, the entity will wait in the Server its until it is needed by an entity at the Filler. It is not until a pack order consumes the bulk that the entity produced and the trigger is called to complete the 'WaitForTankRelease' process that the entity in the Hold Tank Server and the Tank are released.

Finally, after the entity exists to HoldTank Server that it is routed to the OrderDispose Sink. At the OrderDispose Sink, the current time and cost is recorded in the Manufacturing Orders table a for this manufactured material order. The entity is then disposed.

Finished Material (GreenPack, RedPack, BluePack)

If an order is a finished material, it will first route to a Filler Server (PackFill) and then to a Packer Server, as outlined in the Routings table. In the PackFill TransferNode, entity checks to see if the raw materials that are required to be consumed at the Filler Server are available. If they are not available, the entity will wait here at the PackFill TransferNode, instead of waiting at the Filler Server.

Once the materials are available, they can be selected to be run on a Filler Server. The PackFill TransferNode uses a *Route Request Dynamic Selection Rule* to determine the sequence of jobs for Filler1 and Filler2. The *Route Request Dynamic Selection Rule* is set to a Referenced Property so that a user can experiment with using different routing rules. By default, this *Packing Selection Rule* property is set to 'Standard Dispatching Rule' which has a number of child properties. For the *Standard Dispatching Rule*, the *Dispatching Rule* property is set to LargestPriorityValue and the *Tie Breaker Rule* property is set to EarliestDueDate. Using these *Standard Dispatching Rule* properties, entities will be selected will based on largest priority value. If more than one entity has the same largest priority value, the entity with the earliest due date will be selected.

At the Filler, bill of materials is first consumed. After an order is finished with the consume task on the Filler server, it executes a process named 'Filler_ConsumeFinishedTask'. This process populates the TankReleaseOrderId state and then fires the 'TankRelease' event. This will release the mixing order from the Hold Server and release its Tank. Next, the process task requires a secondary resource. The RoutingsSecondaryWorkers will be used to identify which workers to consider for the task. If more than one worker pool is defined in the RoutingsSecondaryWorkers table, the first available worker pool will be selected.

Once the process step is completed, an entity representing the Finished Material order will leave the Filler Server and get routed to a Packing TransferNode. When a Packer Server is available, the entity is routed to the Packer. After processing at a Packer Server, the entity is routed to the AddToStorage TransferNode, where it will select the appropriate storage location based on its picture, which indicates if it is a RedPack, GreenPack or BluePack. The ship date in the Manufacturing Orders table for this finished material order is updated when the entity enters the Storage Server.

Data Tables

Resources:

A list of resources that are in the manufacturing facility.

Column Name	Description
Resource Name	The unique name of the resource. This is the object name.
Description	A description of this resource
Routing Group	Routing groups are auto created for each row, but only the routing group in the Filling and Packing transfer nodes are used. This model has 3 types of look ahead logic that are implemented using the Routing Groups. By default, "None" is used. The "Simple" and "Complex" look aheads are described in more detail in the next section.
Routing Destinations	Name of the Routing Destinations table and column reference used within the <i>Routing Group Name</i> custom routing group in the associated node.
Selection Rule	Name of the custom routing group's <i>Route Request Dynamic Selection Rule</i> .
XLocation	This is the X location of the object in the model.
ZLocation	This is the Z location of the object in the model.
Object Type	This reference an object type in the Object Type List.
Work Schedule	Work schedule assigned to the resource.
Cost Rate	The hourly rate for the resource either idle or while being utilized.
Changeover Matrix	Changeover Matrix used by the resource.
Secondary Resource Name	This is the resource (object) that is seized by the primary resource while processing (NOTE: This field is not being used in this example. The Resource Secondary Workers table is being used instead).
Freeze Schedule	This option specifies the resources where the freeze schedule logic will be used. The freeze schedule logic is described in more detail in the next section.
Use Look Aheads	This option specifies whether to use the look ahead logic for that resource within the OnConfirmingDestinationNode process logic.

A compelling advantage of the Simio scheduling solution is the flexibility offered by a custom-built model of the production system. However, for simple applications that do not require this flexibility, the Resource table can be used to eliminate the need to manually build the Simio model. The Object Type column, in combination with the 'AutoCreateInInstance' option on the Resource Name column, will automatically generate the appropriate object at the x/z location specified.

Materials:

A list of material that can be produced at the manufacturing facility.

Column Name	Description
Material Name	The unique name of this material.
Material Class	A description of this material
Material Cost	The cost per unit of this material.
Material Color	The color index for computing color-dependent changeovers.
Gantt Color	The color used for drawing orders for this material in the Gantt.
Material State Statistic	State statistic used to plot material quantities over time on the Gantt.

Note that the Gantt color defines the bar color to use for drawing entities associated with this material in the Entity Workflow Gantt. All orders for this material will be drawn using this color. The Material Color is used in the sequence dependent changeover matrix for determining the color dependent changeover time. By default, this table is configured for changeovers that are dependent on color. This column could be changed to Size or another attribute that determines the sequence dependent changeover time.

Material Lots:

A list of material lots and quantities for materials in the facility.

Column Name	Description
Material Name	A foreign key reference to the Materials table for the material being produced.
Lot Id	This is a string determining what lot the material. This field is not required. If there is no lot control, just leave column blank.
Quantity	The initial quantity of material for the lot.

Manufacturing Orders:

A list of all production orders to be processed during this planning period.

Column Name	Description
Order Id	A unique string name assigned to this manufacturing order.
Material Name	A foreign key reference to the Materials table for the material being manufactured.
Release Date	The date-time this order is released to production.
Due Date	The date-time this order is due.
Order Status	The order status specified as New or WIP (work in process).
Priority	The scheduling priority for this order.
Quantity	The material quantity to be produced for this order.
Ship Date	The ship date for this order (an output) based on the generated schedule.
Production Cost	The production cost for this order (an output) based on the generated schedule.
Target Ship Date-Value	The target ship date for this order
Target Ship Date-Status	The target ship date status: OnTime, Late, or Incomplete.
Target Cost-Value	The target cost for this order.
Target Cost-Status	The target cost status: OnBudget, Overrun, or Incomplete.

Note that the *Ship Date* and *Production Cost* are both output columns in the Manufacturing Orders table; they are written from the simulation model to the table as the schedule is generated by the deterministic simulation run. Also note that the last four columns are targets.

Routings:

A list of job routings for each material specifying the resource and processing time for each task required to produce the material.

Column Name	Description
Routing Key	The unique name for this routing step.
Material Name	A foreign key reference to the Materials table for the manufactured material.
Sequence	The transfer node (list of resources) or resource where this step is to be processed.
Route Number	The routing number.
SuccessorRouteNumber	The routing number of next time dependent route step. If value = 0, no time dependency. This is only used when Look Ahead control on the model equals Complex.
Start Offset	The time after the start of the current routing number until the next routing number can start. If value = 0, the next route step need to wait until the current route step is completed before starting. This used when the UseStartOffsets control on the model equals True.
Setup Time	Discrete setup time for this step.
Process Time	Processing time for this step. In this example, the process time is multiplied by the ManufacturingOrders.Quantity.

Routing Secondary Workers:

A list of routing secondary workers that will be seized along with the primary resource.

Column Name	Description
RoutingKey	A foreign key reference to the Routings table for the routing key that the worker is being used.
Workers	Entity population (Worker) that will be seized as a secondary resource at the primary resource. In this example, the primary resource will only seize from one worker pool. If the Task Resource Referenced Table is populated on the server to this table, all workers listed for the routing key will need to be seized.

Bill of Materials:

A list of component materials that are required at a specific routing sequence location to produce a material.

Column Name	Description
Routing Key	The material routing step where the material action takes place.
Component Material	The name of the material.
Required Quantity	The quantity of the material that is either consumed or produced. This value based on an order quantity = 1. The required quantity is multiplied by the ManufacturingOrders.Quantity.
Required Lot Id	Specified the lot id required for consumption and the lot id provide for production.
Material Use	Specifies if the material is to be consumed or produced.

The Routing Key is a foreign key reference to Routing table. Note that there is a many-to-one relationship that supports multiple materials consumed/produced for the same Routing Key. All the materials that are listed in this table with the same Routing Key will be consumed/produced at that routing step.

Work In Process:

A list of current orders that are in the system, along with their current status.

Column Name	Description
Order Id	Foreign key reference to the Order Id in the Manufacturing Orders table.
Current Route Number	The current route number for this order.
Current Resource	The current resource that this order is being processed on.
FractionOfSetupCompleted	Percentage of setup completed (values between and including 0 and 1)
Completed Quantity	The number of material units that have been processed on this resource.
Accrued Cost	The activity-based cost accrued by this order at this point.

The information in this table is typically imported from the MES system and defines the current state of all the active orders that are being processed in the system. The Order Id is a foreign key reference to the orders that are listed in the Manufacturing Orders table with their Order Status specified as WIP. The information in this table defines the current route number and resource where the order is being processed, along with the quantity completed at this resource. The Accrued Cost is also specified in applications where activity-based costing is being used to track the production cost for the orders.

Manufacturing Orders Output:

An Output Table defining order start and end time on each resource used for creating table reports. The output is also exported to "publish the schedule". See Published Schedule table.

Column Name	Description
Order Id	Foreign key reference to the Order Id in the Manufacturing Orders table.
Routing Key	Foreign key reference to the Routing Key in the Routings table.
Scheduled Resource	The resource where this order is to be processed.
Scheduled Start Time	The start time at this resource for this order.
Scheduled End Time	The end time at this resource for this order.
Scheduled Quantity	The quantity schedule at the resource for this order.

The foreign key reference for the Order Id and Routing Key ties this output data to the other tables in the model. This makes it easy to create custom Table Reports on the relations data set that display relevant information on the orders and routings as well as the scheduled start and end time for each order on each resource in the system.

Published Schedule

The Published Schedule is typically bound to the last published schedule information. Typically, to publish a schedule the Manufacturing Orders Output table will be exported to a CSV file. Then, the Published Schedule will be bound to this CSV file and imported each time the schedule is run.

Column Name	Description
Order Id	String value representing the Order Id in the MFG Orders table.
Routing Key	String value representing the Routing Key in the Routings table.
Scheduled Resource	String value representing the Resource Name in the Resources table.
Scheduled Start Time	The start time at this resource for this order.
Scheduled End Time	The end time at this resource for this order.

The Published Schedule is used for Freezing the schedule. See Freezing Schedule section below.

Resource Group Events

This table is used by the auto-creation of Routing Group elements in the Resources table. This table is used to define the Event and the Event Condition when the routing group will be fired.

Column Name	Description
Event	Event property to will trigger the routing group.
Event Condition	Condition that must be met to fire the routing group.

FutureRoutingGroupEvents

This output table is used for the Complex look ahead logic. This table logs the time when future events will happen. The table is used to determine whether a future event has already been scheduled for the StartingTransferNode and EventTime.

Column Name	Description
TimeNow	Time that group event was created.
StartingTransferNode	Starting transfer node where group event was created.
TransferNode	Transfer node that caused the group event.
EventTime	Time of the future event.
EventDelayTime	Amount of time to delay until the future event.

Mixing / Packing / Tank Fill / Filling / Add To Storage Routing Destinations

These five routing destination tables include lists of each possible destination node for each resource at that particular processing area. The entity is routed to one of the nodes specified in this list based on the selection rule.

Column Name	Description
Resource Name	This is the transfer node used to route each entity to their destinations. Foreign key that references the Resource Name within the Resources table.
Node	Possible destination from this transfer node.

FreezeRoutingDependencies

This output table is used OnRunInitialized_FreezeSchedule process that gets called at the start of run. This table shows the Routing Dependencies that are initialized by the Freeze Schedule. Once the rows are populated into this table, each row in the table is then used to create stipulations using the Stipulate step.

Column Name	Description
Node List	Node list of the stipulation.
CandidateEntityName	String name of the candidate entity.
FollowsEntityName	String name of the entity that the candidate entity must follow.
PriorUsageCount	Number of prior usage counts to consider the stipulation.

FreezeListRequirements

This output table is used OnRunInitialized_FreezeSchedule process that gets called at the start of run. This table shows the List Requirements that are initialized by the Freeze Schedule. Once the rows are populated into this table, each row in the table is then used to create stipulations using the Stipulate step.

Column Name	Description
CandidateEntityName	String name of the candidate entity.
NodeList	Node list of the stipulation.
RequiredNode	Required node for the candidate entity.

ResourceReservations

This output table is used for the Complex look ahead logic. This stores the reservation information for the primary and secondary resources for each order id and routing key. Once the reservation information is used and the operation is schedule, the reservation is removed from the table.

Column Name	Description
TimeNow	Time Reservation was created.
TransferNode	Transfer Node that reservation will be used.
Resource	Resource Reserved.
IsPrimary	Is Reservation for Primary Resource.
OrderId	Reservation for Order Id
StartingRouteKey	Reservation reserved at Starting Routing Key
RoutingKey	Reservation for Routing Key.
ReserveStartTime	Start time of the reservation.
ReserveEndTime	End time of the reservation.

Model Properties

Here are the model properties in the model:

Property Name	Description
Selected Routing Type	This property is used by the Configure Scheduling Resources to determine what type of routings that the objects should be setup to use. The possible options are Product Based or Order Based
Object Types XML Location	This is the file location of the XML file used to map the object properties based on object type and routing type.
Mixing Selection Rule	Dynamic selection rule used to schedule Mixing and TankFill.
Packing Selection Rule	Dynamic selection rule used to schedule Filling and Packing.
Tank Qtys	Number of tanks in the system.
Workers1stShiftQty	Number of 1 st shift workers in the system.
Workers2ndShiftQty	Number of 2nd shift workers in the system.
Workers3rdShiftQty	Number of 3rd shift workers in the system.
LookAheads	This is a list property used to determine what type of look heads the Filling and Packing routing groups will use. The possible values are None, Simple and Complex.
UseStartOffsets	Boolean property to determine whether the Routings.StartOffset value should be used in the model.

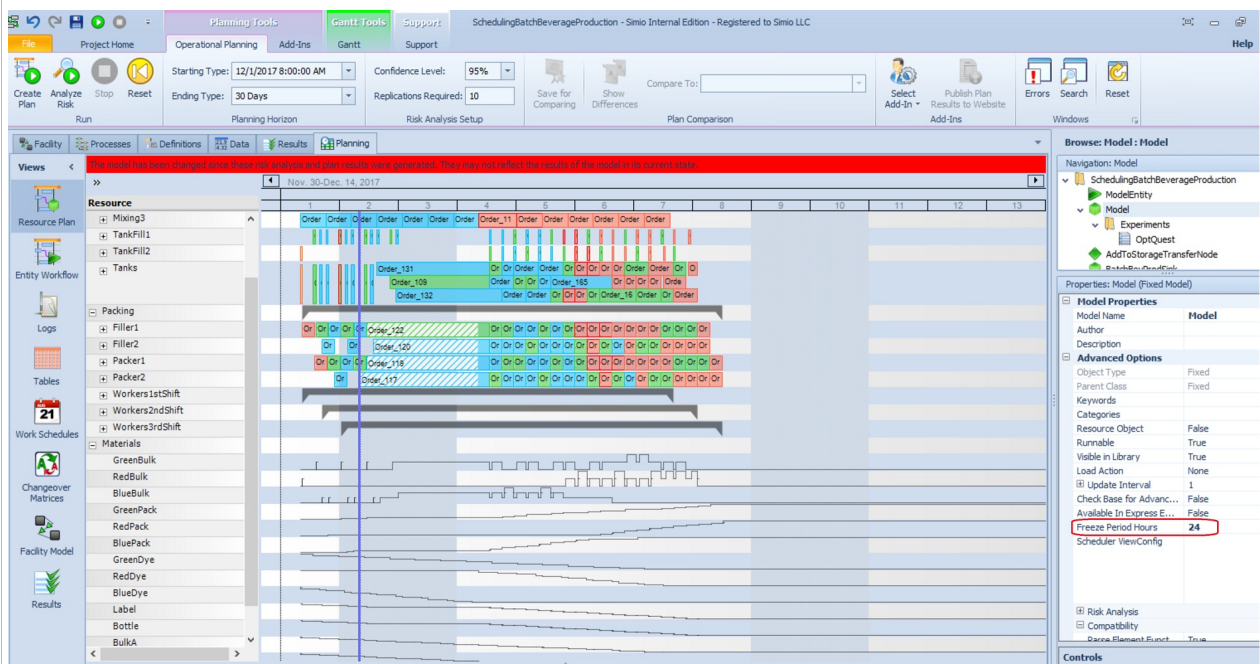
Freeze Schedule

Freeze schedule is used to generate the same schedule that was generated the last time the schedule was published. Freeze schedule ensures that the short-term plan that was laid out is not changed the next time the schedule is run. The freeze schedule also takes WIP into consideration.

There is a property defined on the model called 'Freeze Period Hours'. It is found under Advanced Options. The Freeze Period Hours is the number of hours from the start of the plan run that the schedule is considered frozen. The end of the freeze period is shown in 'Run.FreezePeriod' state.

The 'OnRunInitialized_FreezeSchedule' process is used freeze the schedule. The process searches all the resources in the Resources table. Then on each resource, it first freezes the Work In Process. This is done by adding rows into the FreezeRoutingDependencies and FreezeListRequirements tables. Once the work in process is recorded, if a resource has it

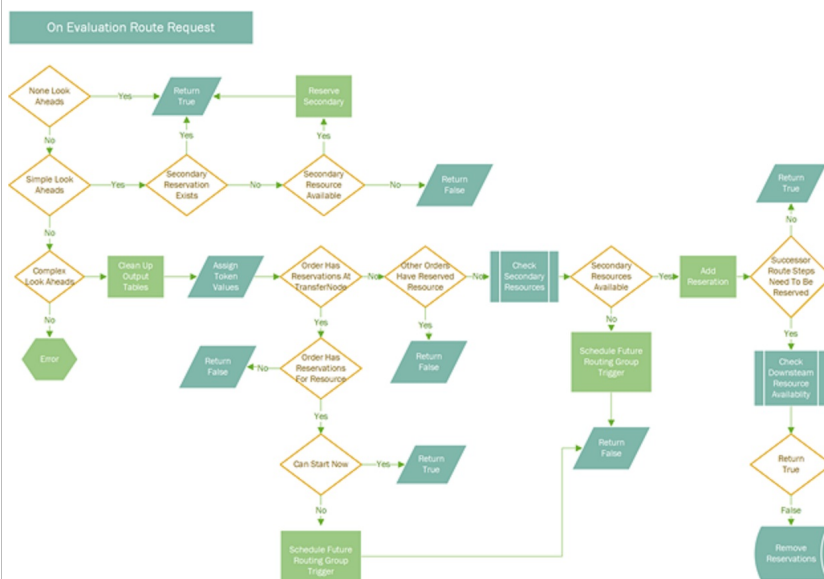
Once all the rows of the resources table have been searched, the FreezeRoutingDependencies and FreezeListRequirements tables are searched and their values recorded in the model using the Stipulate step. The Stipulate step will use these dependencies / requirements when scheduling the entities in the transfer node. The Stipulations are first taken into consideration. Second, any drag and drops from the Gantt chart. Lastly, the selection rules on are used.



LookAheads are used by the routing groups to determine whether secondary resources and downstream operation will be available once the entity is routed to a primary resource. By default, the routing groups only look ahead to the primary resources to ensure that they are available using the *Blocked Destination Rule*. In this example, since the Servers have an *Input Buffer Capacity = 0*, the routing groups will only route an entity to the server if there is capacity available for processing. When the *LookAheads* control on the model is 'None', the routing groups works this way.

When the *LookAheads* control on the model is 'Complex', the routing group will check to ensure that the secondary resource is also available as well as whether any downstream primary and secondary resource are available. The downstream operations are indicated using the *Routeings.SuccessorRouteNumber* field. If this field is populated with a value greater than zero, the complex lookaheads will search for the downstream operation and make sure their primary and secondary resources are available. This process is recursive. If the 2nd operation has a *SuccessorRouteNumber*, the complex lookaheads will search for downstream operation for its *SuccessorRouteNumber*.

If a resource cannot be reserved, the complex lookaheads determine when to try and schedule the entities again. The time to reschedule is either based on availability of the resource or when a reservation expires. Once the time is determined, the starting routing group is triggered again to try and schedule the entities. The time of when the starting routing group will be triggered is recorded in the `FutureRoutingGroup` events table.



The next step is to enhance the model with some custom dashboards. We will import three standard dashboards to our

model that are designed to work with the default data schema. These dashboards are saved in XML format and included in the DiscretePartProductionFiles folder that is saved with this example. These dashboards display material, order details, and a dispatch list for use by operators.

To import these dashboards go to Dashboards Report view of the Results window and select the Dashboards ribbon. Click in the Import button and browse to the folder, and select the three dashboard xml files named DispatchList, Materials, and OrderDetails.

Set the Model Run Time

Our final step is the set the run parameters for the model. From the Operational Planning ribbon in the Planning window specify the starting time as 12/1/2017 8am and select the ending time as 30 days.

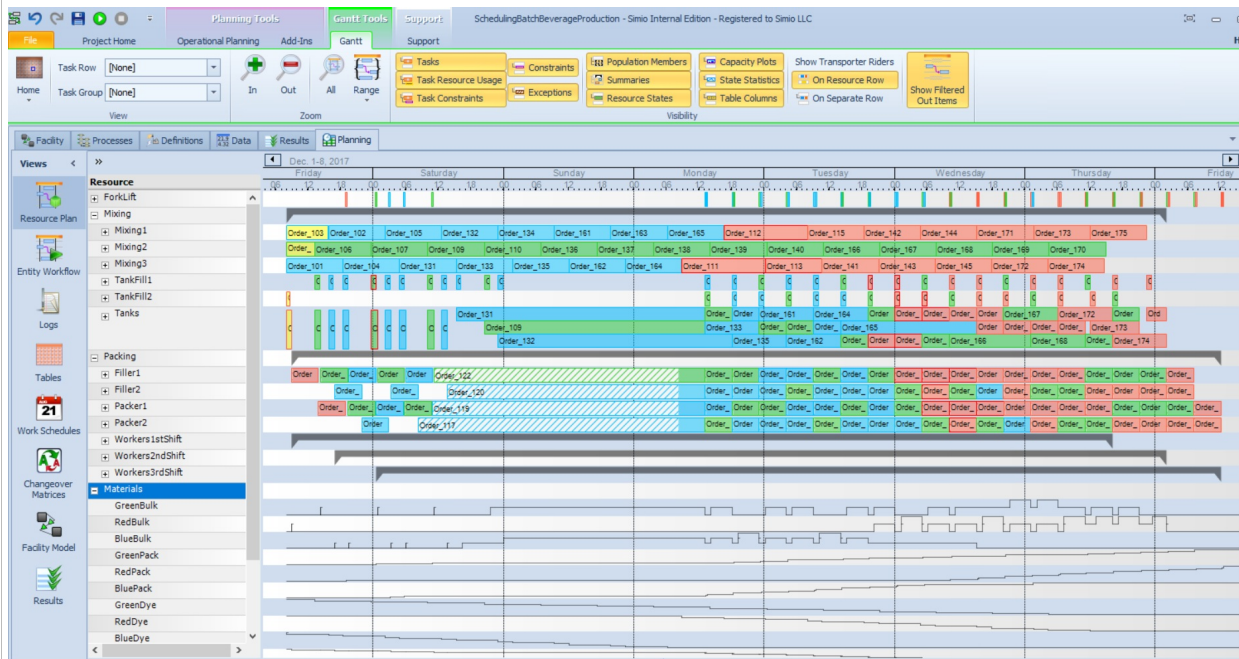
Experiments

There are two experiments on this model. One experiment is setup to use OptQuest to minimize cost by the setting of the model properties listed above. The Lateness response references a Tally Statistic that is recorded on the input node of the DisposeBulkOrders sink object. It records any late orders and by how much time they are late. There is also a response for Number Completed because it is critical that all orders are processed so the user does not want to choose a scenario where some orders are not filled.

The second experiment is used to compare the alternative routing logic at both the Mixing machines and the Filler machines by changing the values of the Mixing Rule and the PackFillRule.

The Scheduling Interface

The model can now be used to generate a plan. Go to the Planning tab and click Create Plan in the Operational Planning Ribbon. This will generate the deterministic plan, which will populate the Resource Plan, Entity Workflow, Logs, Tables and Results. Click onto Analyze Risk in the Operational Planning Ribbon and multiple replications, with variability, will be run to assess the risk of the plan. The Entity Workflow, Tables and Results will be updated with risk analysis for all the Targets defined in the model.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SchedulingBicycleAssembly - Example

General Description

The example uses Simio RPS Edition to schedule a bicycle assembly line. Simio RPS Edition is a simulation tool for developing applications in Risk-based Planning and Scheduling (RPS). RPS is the dual use of a simulation model to generate both a detailed resource constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty. Although Simio RPS Edition is required to build this model, it may be viewed using Simio Professional or Design Edition. If in Trial Edition, to give you the best planning and scheduling experience, please contact sales@simio.com.

To understand the data schema and scheduling results in this example you should first read the *Planning and Scheduling with Simio* document found in C:\Program Files\Simio LLC\Simio. This problem description assumes that you are familiar with the standard data schema and scheduling concepts that are presented in that document.

The system to be scheduled in this example is a bicycle assembly facility. The facility is a job shop that produces finished goods on a final assembly line as well as parts or components for final assembly based on the customer orders. The schedule also includes a number of purchased parts required for final assembly. We wish to generate a 7-day production schedule for this facility that fully accounts for the limited resources and parts in the system.

The facility consists of three (3) functional areas named *Component Manufacturing*, *Assembly Line*, and *Shipping and Receiving*. These areas are described in more detail below.

The *Component Manufacturing* area consist of (5) work centers (Cut, Shape, Weld, Machine and Paint) with each work center containing 2 processing areas. The entry point to the work center is modeled a TransferNode with the corresponding processing areas are modelled using Server objects. The TransferNodes are used to select the order to process based on a standard dispatching rule.

Each Server also requires an Operator that is modeled as a Worker named 'Operators' with an initial capacity of five (5) and managed by a work schedule. Initially, the openShifts have a work schedule named '5Day8Hour1stShift' which is 5 days a week for 8 hours during first shift. The Paint Servers have sequence dependent setup times where the setup time is specified in a changeover matrix based on the item color. Possible item colors are defined in a string list named 'MaterialColor' with values None, Green, and White. The *Component Manufacturing* area produces six (6) manufactured items for final assembly (*GreenFinishedFrame*, *WhiteFinishedFrame*, *GreenFinishedFork*, *WhiteFinishedFork*, *GreenFinishedHandleBar* and *WhiteFinishedHandleBar*). Each of these items has its own routing and unique processing time and material requirements at each Server within its routing.

The *Assembly Line* area produces six (6) finished goods to satisfy the customer orders for Walmart, Costco and Ambridge Bike Shop (*EcoBikeSTDGreen*, *EcoBikeSTDWhite*, *EcoBikeEXTGreen* and *EcoBikeEXTWhite*). The *Assembly Line* area consist two (2) identical assembly lines with four (4) work centers in each line named Mainframe, Power, Final and Test. The selection of the two assembly lines is controlled by a TransferNode called AssemblyNode to allocate the selected line from a list called AssemblyList.

Each of the four (4) machines per assembly line is modeled using a Server object with a capacity of one (1). All the Servers also require an Assembler that is modeled as a Worker named Assemblers. There are 2 pools for Assemblers; Assemblers1stShift and Assemblers2ndShift. One assembly pool for first shift and one pool for second shift. The initial capacity of each pool is ten (10). Initially, the Assemblers1stShift pool uses the '5Day12Hour1stShift' work schedule and Assemblers2ndShift pool uses the '5Day12Hour2ndShift' work schedule.

The MainFrame Servers use task sequences to first consume the correct components. These final assembly components are listed per final product in the Bill Of Materials table. Once material is consumed, one of two tasks are completed. If the number of Assemblers1stShift in the system is <= 8, one assembler is used for the task and the processing time is set to 12 minutes. If the number of Assemblers1stShift in the system is > 8, two assemblers are used for the task and the processing time is set to 6 minutes. A condition on the 20 Sequence Number tasks are used to select which number of operators and processing times. In addition, the task sequences call 2 processes named 'MainFrame_StartingTask' and 'MainFrame_FinishedTask' that are used to control animation that delivers stock and receiving material to the AssemblyLine using Forklifts.

The *Shipping and Receiving* area consists of a PackAndShip Combiner that assembles final product, as well as a Receiving (Server) area that stores material and a PurchasedMaterials Source that generates initial product based on the PurchaseOrders table.

The B2MML-based data schema discussed in the *Planning and Scheduling with Simio* document is used to hold the production data for this example.

Detailed Description

System Initialization

The system begins with no Work In Progress (WIP) and starts with an empty production line. It is initialized by reading the orders, both for components and finished goods, from the ManufacturingOrders table. In the OnRunInitialized process, we initialize ManufacturingOrders table, then iterate through each Materials row and its associated Material.Lots row to sum total quantity involved. If it's a ManufacturingMaterial then create that quantity of MfgOrders and send to Stock. Otherwise create that quantity of PurchaseOrders and send to Receiving.

Arrival of Manufacturing Orders

The manufacturing orders fall into two (2) categories being Production Orders for manufactured components required for final assembly as well as the Manufacturing Orders for final assembly that includes both the manufactured components as well as procured items required to produce the finished products. These manufacturing orders are released into the system and scheduled based on the release and due dates provided in the ManufacturingOrders table. This is modeled by using a Source object (MRPOrders) that is reading the ArrivalsTable.

Manufactured Material (*GreenFinishedFrame*, *WhiteFinishedFrame*, *GreenFinishedFork*, *WhiteFinishedFork*, *GreenFinishedHandleBar* and *WhiteFinishedHandleBar*)

If an order is a manufactured material, it will route to a combinations of workcenters (Cut, Shape, Machine, Weld and Paint) as outlined in the Routings table. The output node of the Source object (MRPOrders) is routing the orders to the relevant workcenters based on the required Sequence per component as setup in the Routing table. At the Paint workcenters, the setup times are dependent on the color of the component and it is controlled by a changeover matrix to provide the required setup time based on the color change required for the next item to be painted. At three (3) of the workcenters (Cut, Shape and Paint) the transfer node is used for the entity to check if the raw materials that are required to be consumed at the workcenter are available. If they are not available, the entity will wait at the relevant transfer node, instead of waiting at the workcenter.

Before processing occurs at each workcenter, the process AddManufacturingOrdersOutput adds a row in the table named ManufacturingOrdersOutput and records the start time of this order on the workcenter. The end time is recorded after this order is finished on the workcenter.

After the manufacturing order is completed at the last operation in the manufacturing process the sequence will route the entity to the workcenter called Stock in the Shipping and Receiving area to record the completion time for availability for use at the Assembly Line for the production of the finished material. At that point, the entity representing this manufactured material order will get transferred to the Sink1 object to be disposed of.

Finished Material (*EcoBikeSTDGreen*, *EcoBikeSTDWhite*, *EcoBikeEXTGreen* and *EcoBikeEXTWhite*)

If an order is a finished material, it will first route to the Separator object (AssemblyLine) in the Assembly area to create two entities, one to represent the order for consolidation at Shipping and one to drive the assembly line, providing the processing and material information required. The entity created to drive the assembly process is routed via a path to a transfer node (AssemblyNode) where it is routed to one of two assembly lines reading the destination from a node list called AssemblyList. The two assembly lines are identical with 4 workcenters each (Mainframe, Power, Final and Test) to complete the final assembly of the finished material and then routed to a Pack and Ship Station modelled by a Combiner object to match the customer order with the actual material manufactured to complete the customer order. The entity then exits the system through the Sink object called Customer.

Purchased Materials (Seat, BrakeSet, WheelSet, Motor, FenderSet, StdBattery, ExtBattery)

The Source object *PurchasedParts* creates the entities representing the purchase order for materials. This source object point to the appropriate row in the PurchasedMaterials table so the entity has a reference to the purchased material items' arrival date and time. The material then becomes available in the system for consumptions by the Assembly line.

Data Tables

The following is a summary of the default tables that are based on the B2MML standard. It is important to note that the base model can also be generated by importing these files in CSV format as described in the *Planning and Scheduling with Simio* document and explained in detail in the Discrete Part Production example also included as one of the scheduling examples as part of the Simio 9 install.

Resources:

A list of all resources that are in the manufacturing facility.

Column Name	Description
Resource Name	The unique name of the resource. This is the object name.
Description	A description of this material
Xlocation	This is the X location of the object in the model.
Zlocation	This is the Z location of the object in the model.
Area	A string used to define the area in the facility where the resource resides.
Workcell	A string used to define the workcell in the facility where the resource resides.
Object Type	This reference an object type in the Object Type List.
Work Schedule	Work schedule assigned to the resource.
Cost Rate	The hourly rate for the resource either idle or while being utilized.
Changeover Matrix	Changeover Matrix used by the resource.
Secondary Resource Name	This is the resource (object) that is seized by the primary resource while processing (NOTE: This field is not being used in this example. The Resource Secondary Workers table is being used instead).

A compelling advantage of the Simio scheduling solution is the flexibility offered by a custom-built model of the production system. However, for simple applications that do not require this flexibility the Resource table can be used to eliminate the need to manually build the Simio model. The Object Type column, in combination with the 'AutoCreateInstance' option on the Resource Name column will automatically generate the appropriate object at the x/z location specified.

Resource Secondary Workers:

A list of resource secondary workers that will be seized along with the primary resource.

Column Name	Description
Resource	A foreign key reference to the Resources table for the resource that the worker is being used.
Workers	Entity population (Worker) that will be seized as a secondary resource at the primary resource. In this example, the primary resource will only seize from one worker pool. If the Task Resource Referenced Table is populated on the server to this table, all workers listed for the resource will need to be seized.

Materials:

A list of materials that can be produced at this manufacturing facility.

Column Name	Description
Material Name	The unique name of this material.
Material Class	A description of this material
Material Cost	The cost per unit of this material.
Material Color	The color index for computing color-dependent changeovers.
Gantt Color	The color used for drawing orders for this material in the Gantt.
Picture Id	The entity symbol index for orders for this material in the Facility view.

Note that the Gantt color defines the bar color to use for drawing entities associated with this material in the Entity Workflow Gantt. All orders for this material will be drawn using this color. The Picture Id determines which entity symbol should be used for drawing the entity that is associated with this material in the Facility view of the model. All entities representing orders for this material will be depicted in the Facility view using the specified Picture Id. The Material Color is used in the sequence dependent changeover matrix for determining the color dependent changeover time. By default, this table is configured for changeovers that are dependent on color. This column could be changed to Size or another attribute that determines the sequence dependent changeover time.

Material Lots:

A list of material lots and quantities for materials in the facility.

Column Name	Description
Material Name	A foreign key reference to the Materials table for the material being produced.
Lot Id	This is a string determining what lot the material. This field is not required. If there is no lot control, just leave column blank.
Quantity	The initial quantity of material for the lot.

Manufacturing Orders:

A list of all production orders to be processed during this planning period.

Column Name	Description
Order Id	A unique string name assigned to this manufacturing order.
Customer	A string name that identified the customer. For manufacturing material, the customer is "Stock".
Material Name	A foreign key reference to the Materials table for the material being manufactured.
Release Date	The date-time this order is released to production.
Due Date	The date-time this order is due.
Order Status	The order status specified as New or WIP (work in process).
Priority	The scheduling priority for this order.
Quantity	The material quantity to be produced for this order.
Ship Date	The ship date for this order (an output) based on the generated schedule.
Production Cost	The production cost for this order (an output) based on the generated schedule.
Target Ship Date-Value	The target ship date for this order
Target Ship Date-Status	The target ship date status; OnTime, Late, or Incomplete.
Target Cost-Value	The target cost for this order.
Target Cost-Status	The target cost status: OnBudget, Overrun, or Incomplete.

Note that the *Ship Date* and *Production Cost* are both output columns in the Manufacturing Orders table; they are written from the simulation model to the table as the schedule is generated by the deterministic simulation run. Also note that the last four columns are targets.

Purchase Orders:

A list of raw materials that are purchased at this production facility and are scheduled to arrive.

Column Name	Description
PO Number	A unique string name assigned to this purchase order.
Material Name	The unique name of this material.
Arrival Date	A description of this material
Quantity	The initial quantity of material available.

Depending on the application there can be various purchased raw material items that are required as part of the manufacturing process. These items with their arrival dates and quantities can be imported into the model.

Routings:

A list of job routings for each material specifying the resource and processing time for each task required to produce the material.

Column Name	Description
Routing Key	The unique name for this routing step.
Material Name	A foreign key reference to the Materials table for the manufactured material.
Sequence	The transfer node (list of resources) or resource where this step is performed.
Route Number	The routing number.
Setup Time	Discrete setup time for this step.
Process Time	Processing time for this step. In this example, the process time is multiplied by the ManufacturingOrders.Quantity.

Depending upon the application this table might be extended with additional columns specifying a setup time, operator, tooling, etc. The Routing Key is a key column that requires a unique name and is typically specified as a material name with an underscore and step number appended. For example, FinishedGoodA_10 would be step 10 for producing FinishedGoodA. The step numbers are typically specified as 10, 20, 30, etc. The sequence is the resource location where this step is performed. This is typically the input node of a specific server or a TransferNode from which downstream servers are dynamically selected. For example the TransferNode might be located at a group of five similar machines, and then the entity is dynamically routed to one of the five machines based on availability and a selection rule – e.g. smallest changeover time.

Bill of Materials:

A list of component materials that are required at a specific routing sequence location to produce a material.

Column Name	Description
Routing Key	The material routing step where the material action takes place.
Component Material	The name of the material.
Quantity	The quantity of the material that is either consumed or produced.
Required Lot Id	Specified the lot id required for consumption and the lot id provide for production.
Material Use	Specifies if the material is to be consumed or produced.

The Routing Key is a foreign key reference to Routing table. Note that there is a many-to-one relationship that supports multiple materials consumed/produced for the same Routing Key. All the materials that are listed in this table with the same Routing Key will be consumed/produced at that routing step.

Work In Process:

A list of current orders that are in the system, along with their current status.

Column Name	Description
Order ID	Foreign key reference to the Order ID in the Manufacturing Orders table.
Current Route Number	The current route number for this order.
Current Resource	The current resource that this order is being processed on.
FractionOfSetupCompleted	Percentage of setup completed (values between and including 0 and 1)
Completed Quantity	The number of material units that have been processed on this resource.
Accrued Cost	The activity-based cost accrued by this order at this point.

The information in this table is typically imported from the MES system and defines the current state of all the active orders that are being processed in the system. The Order Id is a foreign key reference to the orders that are listed in the Manufacturing Orders table with their Order Status specified as WIP. The information in this table defines the current route number and resource where the order is being processed, along with the quantity completed at this resource. The Accrued Cost is also specified in applications where activity-based costing is being used to track the production cost for the orders.

Manufacturing Orders Output:

An Output Table defining order start and end time on each resource used for creating Table Reports.

Column Name	Description
Order ID	Foreign key reference to the Order Id in the Manufacturing Orders table.
Scheduled Resource	The resource where this order is to be processed.
Scheduled Start Time	The start time at this resource for this order.
Scheduled End Time	The end time at this resource for this order.
Scheduled Quantity	The quantity schedule at the resource for this order.

The foreign key reference for the Order Id ties this output data to the other input tables for the model. This makes it easy to create custom Table Reports on the relations data set that display relevant information on the orders as well as the scheduled start and end time for each order on each resource in the system.

Routing Destinations:

A list of each possible destination node for each resource. The entity is routed to one the nodes specified in this list based on the selection rule.

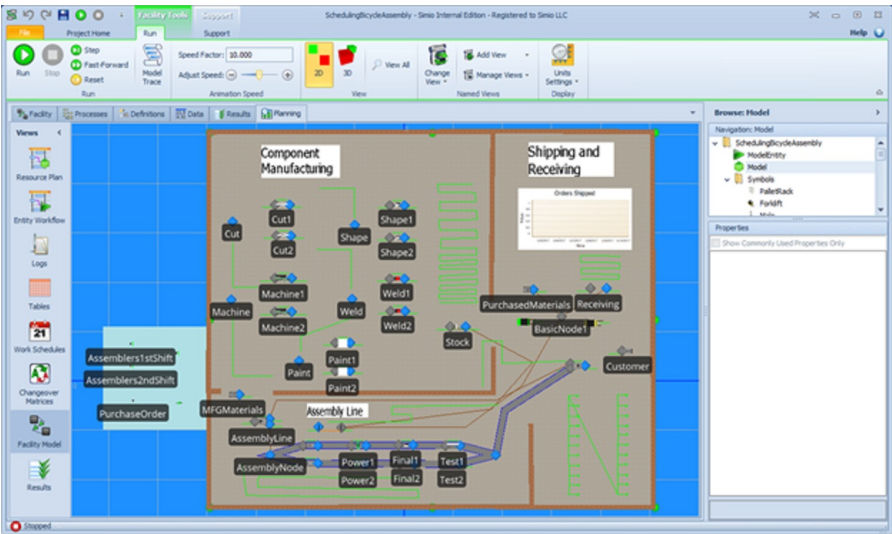
Column Name	Description
Resource Name	This is the transfer node used to route each entity to their destinations. Foreign key that references the Resource Name within the Resources table.
Node	Possible destination from this transfer node.

The Scheduling Interface

Once the model has been completed and the data imported (already completed in the example model) can the scheduler start to use the system to generate a schedule by selecting the Planning tab. Following below is an explanation of some of the different views available under the planning tab.

Facility View

Once the model has been completed it can be viewed and run in the facility view to observe the actual process in a visual way representing the facility as shown below:



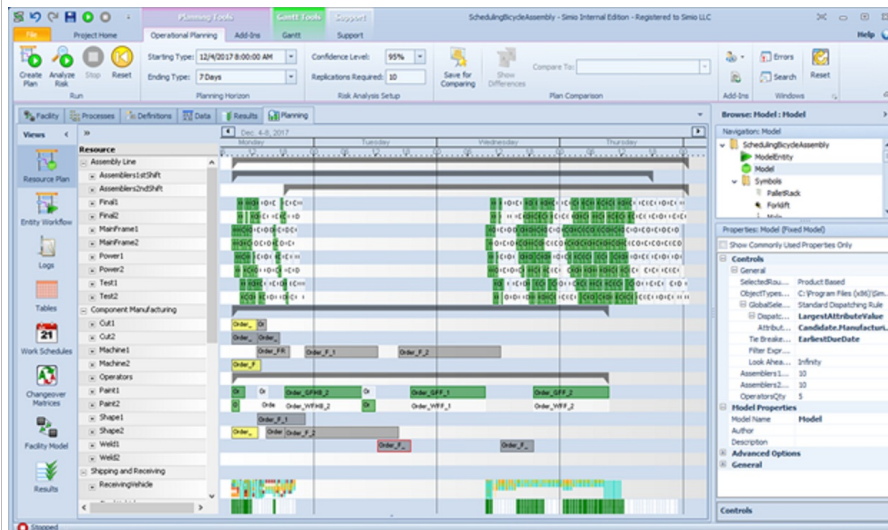
Resource Plan

The Resource Plan provides a summary of how each resource in the system is utilized by each entity/order that is processed by that resource. The Resource Plan presents this information in both a graphical view (Gantt) and tabular view (Usage and State Logs). The resource view show the resource usage for both the parts manufacturing as well as the final assembly areas. Also available on the Gantt chart view is the material levels for both the manufactured and the purchased material. By selecting the "plus" sign associated with each resource in the resource list on the left of the Gantt chart will expand the view showing the orders that are constrained by the resource as a result of material, capacity or operator constraints as well as the resource state (idle, busy, etc.).

The thick black lines on the Gantt chart indicate the various summary views and by selecting the "plus" or "minus" sign for that row will expand or collapse the view associated with that specific section of the chart to select only the view of interest to the planner. The time horizon on the Gantt view can be changed by using the zoom function after selecting the Gantt

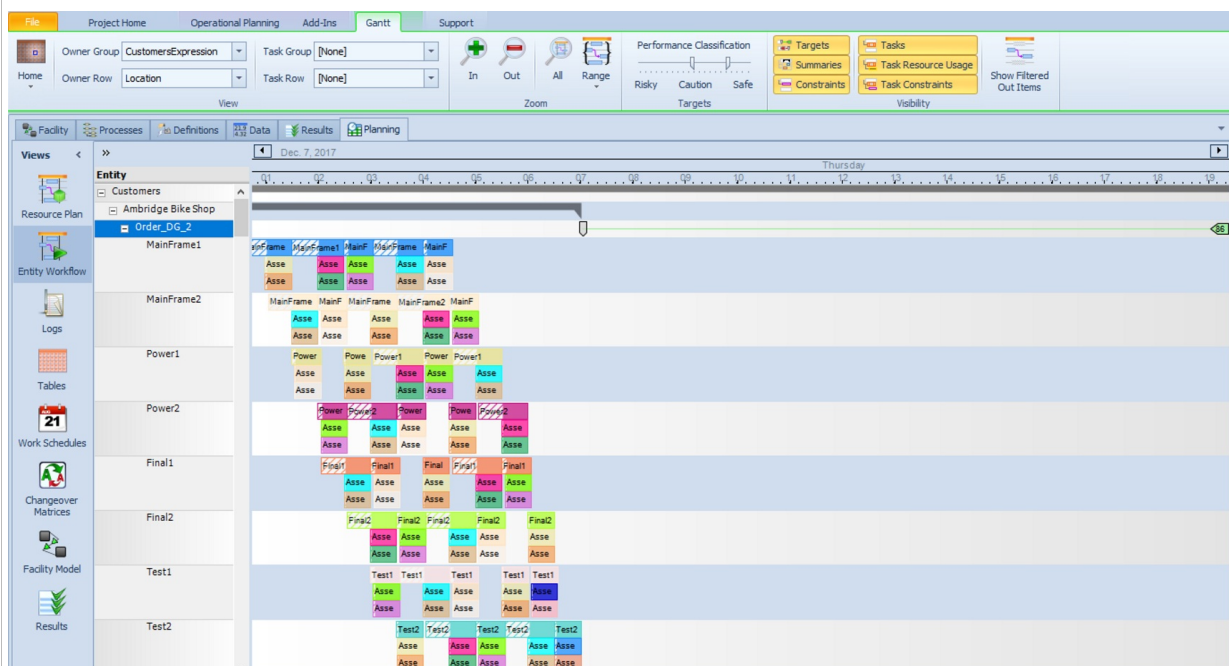
view from the top ribbon.

By double clicking on any of the orders the view will change to the Entity Workflow view to observe the constraints associated with each specific order.

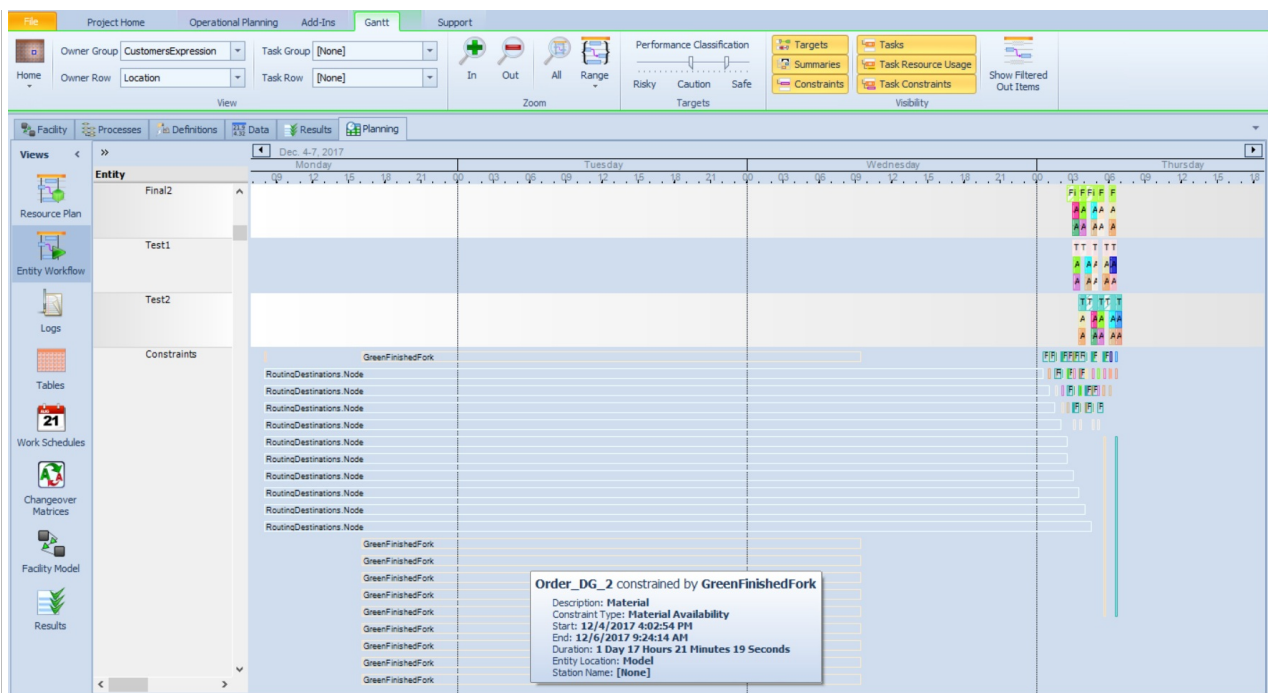


Entity Workflow

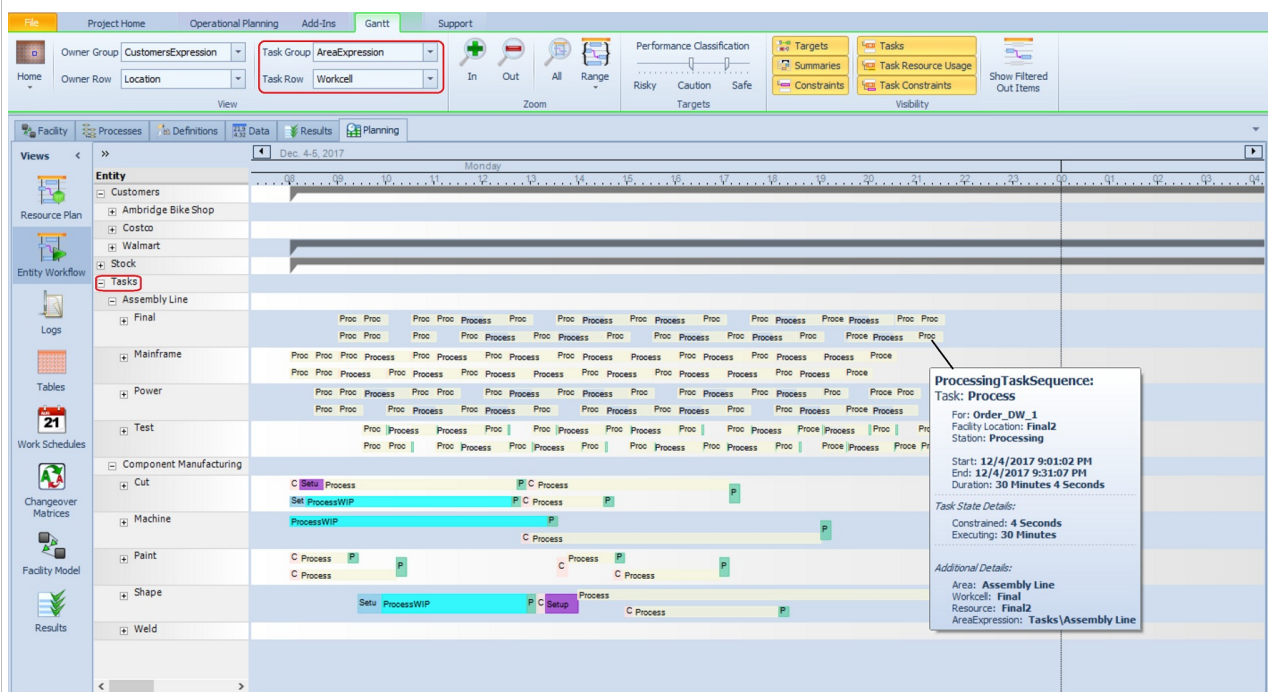
By selecting the Entity Workflow tab on the left planner can view all the orders in the system. This includes both the manufactured material orders as well as the finished material orders. By expanding the view for the various order types by selecting the "plus" or "minus" sign on the order list at the left-hand side of the Gantt chart will show all the customer orders as indicated by the screenshot below.



By further expanding the view for each order will also show the resource and material constraints associated with each order providing direct feedback to the planner as to the reasons for material lateness.

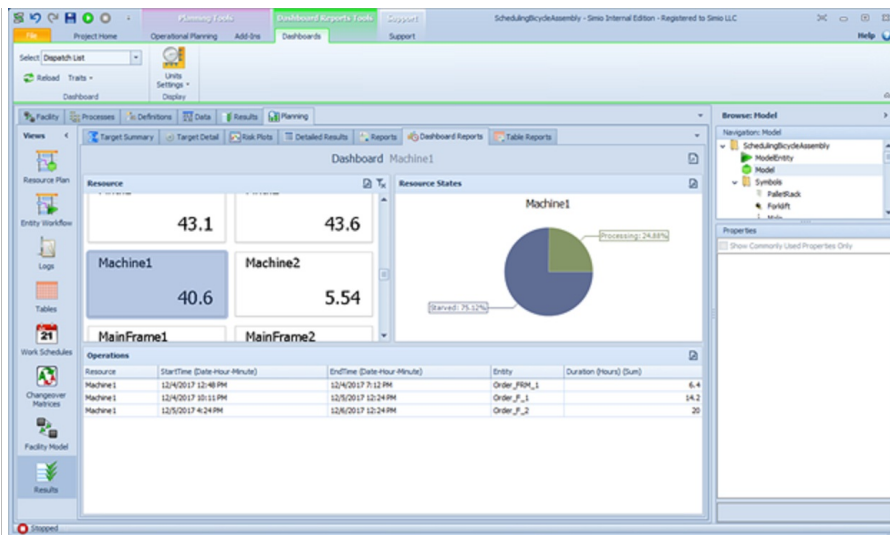


Tasks can also be separated out within Entity Workflow Gantt using the Task Group and Task Row (with custom columns from the Task Log).



Results

By selecting the results tab the planner will get access to a host of information regarding the schedule that has been produced by the system. These results include a pivot table, Dashboard reports, Risk plots, Table reports, Target detail and more. Below is an example of a resource state dashboard view.



These results tab can be used to do a detail analysis of the schedule results to fully understand the system behavior to try alternative options to improve the plan such as increasing resource capacity or providing additional material, etc.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SchedulingBicycleAssembly_UpdateCreateMOandPOs - Example

Scheduling Bicycle Assembly Line with Model Calculated Manufacturing Orders and Purchase Orders Creation

General Description

The example uses Simio RPS Edition to schedule a bicycle assembly line. Simio RPS Edition is a simulation tool for developing applications in Risk-based Planning and Scheduling (RPS). RPS is the dual use of a simulation model to generate both a detailed resource constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty. Although Simio RPS Edition is required to build this model, it may be viewed using Simio Professional or Design Edition. If in Trial Edition, to give you the best planning and scheduling experience, please contact sales@simio.com.

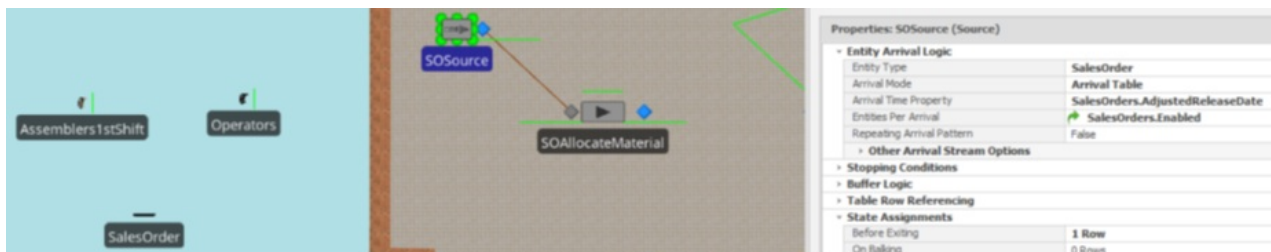
This example is based on the SchedulingBicycleAssembly. The main difference between this example of and the SchedulingBicycleAssembly is the SchedulingBicycleAssembly assumes purchase orders and component orders are generated by outside system and imported into the model. The materials are not pegged. The material is allocated to the orders on a first-come first-served basis.

This example generates the purchase orders and manufacturing orders with the model. The SalesOrders will first try and fulfill its demand based on available material. If the material is not available, the ManufacturingOrders and PurchaseOrders are generated to fulfill the demand. The sales orders are selected for material allocation based on an allocation selection rule. Once the material is allocated or an order is generated, the materials are pegged to the sales order.

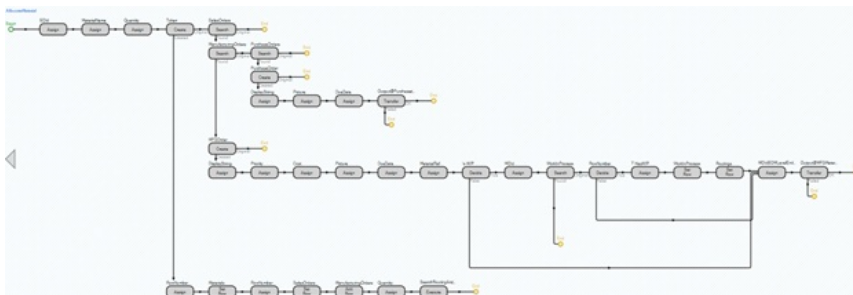
Detailed Description

The main differences between the SchedulingBicycleAssembly_UpdateCreateMOandPOs and the SchedulingBicycleAssembly model are:

First, there is a new table for SalesOrders. The SalesOrders table is the order of the model are generated from. There is a new Source called SOSource in the model. This is source uses the SalesOrders table to create SalesOrder entities. The SalesOrder entities are routed to the SOAllocateMaterial server where the ManufacturingOrders and PurchaseOrders are generated to fulfill the SalesOrder. If multiple SalesOrders arrive to the SOAllocateMaterial server at the same time, the AllocateSelectionRule is used to determine which SalesOrder to allocate first.

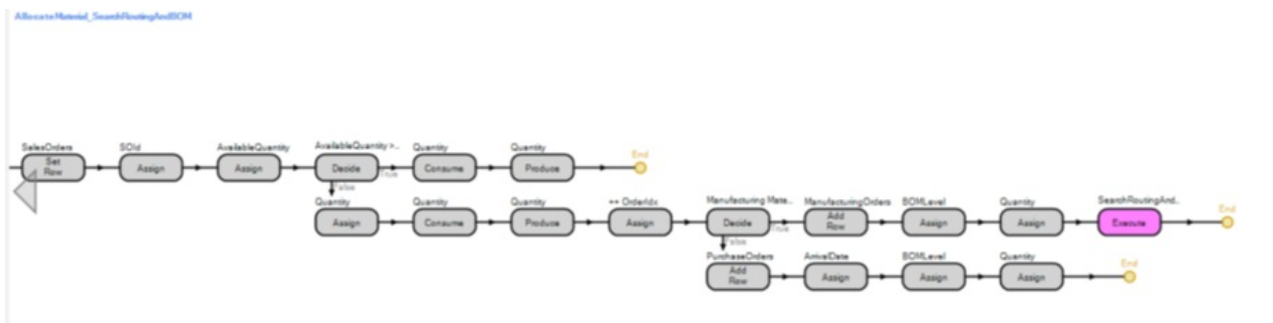


For processing on the SOAllocateMaterial server, a process named "AllocateMaterial" is run. This process will first iterate through the BOM of the material defined on the SalesOrder to either allocate materials that are already on-hand or create ManufacturingOrders and/or PurchaseOrders for the materials for the orders. This is done using the "AllocateMaterial_SearchRoutingAndBOM" process. Once material is allocated and/or ManufacturingOrders / PurchaseOrders are generated, the "AllocateMaterial" process will create the entities for the ManufacturingOrders and PurchaseOrders, if needed.



































The "AllocateMaterial_SearchRoutingAndBOM" process is called recursively to check for component need for ManufacturingOrders. The process will first try and allocation material already in stock. If the material is available, it is allocated to the SalesOrder. If the material is not available, then either a ManufacturingOrder or a PurchaseOrder is

created to fulfill the need based on the material type. This happens for multiple levels of the BOM structure in this example. There are 3 BOM levels of ManufacturingOrders (BOMIndex 0, 1, 2) and 3 BOM levels of PurchaseOrders (BOMIndex 1,2,3) are generated in this example.



The result is the SalesOrders tables has child tables with the associated ManufacturingOrders and PurchaseOrders that were created during the AllocateMaterials processes. These orders will need to be completed and material produce to fulfill the material requirement needs of its associated higher level Manufacturing Orders. Here are the ManufacturingOrders and PurchaseOrders created for SOId = "001005".

Manufacturing Orders		Resources Secondary Workers		Materials	Material Lots	Routings	Bill Of Materials	Manufacturing Orders Output		Routing Destinations	Sales Orders
Enabled		 SOId	Customer	Material Name	Release Date	Due Date	Order Status	Priority	Quantity	Adjusted Release Date	
→ 5	--	<input checked="" type="checkbox"/>	001005	Ambridge Bike Shop	EcoBikeSTDGreen	12/4/2017 8:00:00 AM	12/7/2017 4:00:00 PM	New	1	15	Math.If(SalesOrders.OrderStatus
Manufacturing Orders Output		Manufacturing Orders		Purchase Orders							
SO	 MO Id	SO Id	Material Name		BOM Level	Quantity	Entity Idx	Has WIP			
→ 1	001005 - EcoBikeSTDGreen		 001005	EcoBikeSTDGreen	0	15	15	<input type="checkbox"/>			
2	001005 - GreenFinishedFrame - 62		 001005	 GreenFinishedFrame	1	15	0	<input type="checkbox"/>			
3	001005 - GreenFinishedFork - 64		 001005	 GreenFinishedFork	1	15	0	<input type="checkbox"/>			
4	001005 - Fork - 65		 001005	 Fork	2	15	0	<input type="checkbox"/>			

Manufacturing Orders		Resources Secondary Workers		Materials	Material Lots	Routings	Bill Of Materials	Manufacturing Orders Output		Routing Destinations		Sales Orders
	Enabled	 SOId	Customer	Material Name	Release Date	Due Date	Order Status	Priority	Quantity	Adjusted Release Date		
→ 5	 <input checked="" type="checkbox"/>	001005	Ambridge Bike Shop	EcoBikeSTDGreen	12/4/2017 8:00:00 AM	12/7/2017 4:00:00 PM	New	1	15	Math.If(SalesOrders.OrderStatus		
<div><div>Manufacturing Orders Output</div><div><div>Manufacturing Orders</div><div>Purchase Orders</div></div></div>												
	 PO Number	SO Id	Material Name	Arrival Date	BOM Level	Quantity						
→ 1	001005 - GreenPaint - 63	 001005	 GreenPaint	12/5/2017 8:00:00 AM	2	15						
2	001005 - SteelTube - 66	 001005	 SteelTube	12/5/2017 8:00:00 AM	3	15						
3	001005 - GreenPaint - 67	 001005	 GreenPaint	12/5/2017 8:00:00 AM	2	15						
4	001005 - SteelTube - 70	 001005	 SteelTube	12/5/2017 8:00:00 AM	3	5						
5	001005 - GreenPaint - 71	 001005	 GreenPaint	12/5/2017 8:00:00 AM	2	15						
6	001005 - Seat - 72	 001005	 Seat	12/6/2017 8:00:00 AM	1	15						
7	001005 - Brake - 73	 001005	 Brake	12/6/2017 8:00:00 AM	1	30						
8	001005 - Wheel - 74	 001005	 Wheel	12/6/2017 8:00:00 AM	1	30						
9	001005 - StdBattery - 75	 001005	 StdBattery	12/6/2017 8:00:00 AM	1	15						
10	001005 - Motor - 76	 001005	 Motor	12/6/2017 8:00:00 AM	1	6						

The pegging requires a specific Lot ID for a material to be consumed. The Lot ID is specified as BillOfMaterials.RequiredLotId which is a reference to the BillOfMaterials table that references the SalesOrders.SOId.

Manufacturing Orders		Resources	Secondary Workers	Materials	Material Lots	Routings	Bill Of Materials
	Routing Key	Component Material		Required Quantity		Required Lot Id	Material Use
→ 1	Frame_40	Frame		1		SalesOrders.SOIId	Produce
2	Fork_50	Fork		1		SalesOrders.SOIId	Produce
3	HandleBar_30	HandleBar		1		SalesOrders.SOIId	Produce
4	GreenFinishedFrame_10	Frame		1		SalesOrders.SOIId	Consume
5	GreenFinishedFrame_20	GreenFinishedFrame		1		SalesOrders.SOIId	Produce
6	WhiteFinishedFrame_10	Frame		1		SalesOrders.SOIId	Consume
7	WhiteFinishedFrame_20	WhiteFinishedFrame		1		SalesOrders.SOIId	Produce
8	GreenFinishedHandleBar_10	HandleBar		1		SalesOrders.SOIId	Consume
9	GreenFinishedHandleBar_20	GreenFinishedHandleBar		1		SalesOrders.SOIId	Produce
10	WhiteFinishedHandleBar_10	HandleBar		1		SalesOrders.SOIId	Consume
11	WhiteFinishedHandleBar_20	WhiteFinishedHandleBar		1		SalesOrders.SOIId	Produce
12	GreenFinishedFork_10	Fork		1		SalesOrders.SOIId	Consume
13	GreenFinishedFork_20	GreenFinishedFork		1		SalesOrders.SOIId	Produce
14	WhiteFinishedFork_10	Fork		1		SalesOrders.SOIId	Consume
15	WhiteFinishedFork_20	WhiteFinishedFork		1		SalesOrders.SOIId	Produce
16	EcoBikeSTDGreen_10	GreenFinishedFrame		1		SalesOrders.SOIId	Consume
17	EcoBikeSTDGreen_10	GreenFinishedFork		1		SalesOrders.SOIId	Consume
18	EcoBikeSTDGreen_10	GreenFinishedHandleBar		1		SalesOrders.SOIId	Consume
19	EcoBikeSTDGreen_10	Seat		1		SalesOrders.SOIId	Consume
20	EcoBikeSTDGreen_10	Brake		2		SalesOrders.SOIId	Consume
21	EcoBikeSTDGreen_10	Wheel		2		SalesOrders.SOIId	Consume
22	EcoBikeSTDGreen_10	StdBattery		1		SalesOrders.SOIId	Consume
23	EcoBikeSTDGreen_10	Motor		1		SalesOrders.SOIId	Consume

Both the Consume tasks and Produce tasks on the Servers are mapped to the BillOfMaterials.RequiredLotId property that will enable the pegging of the materials for a given SOId.

Processing Tasks - Registering Property Editor

Items:

- 30.1, Consume, Conditional, Manufacture
- 30.1, Produce, Always, Sequential, Depend
- 30.1, Produce, Always, Specific, Time, Rand
- 30.2, Setup, Conditional, Manufacture
- 30.2, Process, Always, Specific, Time, J
- 40, Produce, Always, Specific, Time, Rand

Properties:

Resource Requirements

Resource Type: Select From List

Resource List Name: ResourcesSecondaryWorkers, Workers

Selection Goal: Preferred Order

Required Name: To Rule

Destination Node: Server, Input

Off Shift Rule: Switch Resources If Possible

Required Quantity & Constraints

Advanced Options

Material Requirements

Action Type: BillOfMaterials.Materialize

Consumption Type: Material

Production Type: Material

Material Name: BillOfMaterials.ComponentMaterial

Inventory Site Type: None

Required Quantity & Constraints

Quantity: Math, 30(BillOfMaterials.Materialize == Ensm.MaterializeType.Consume, BillOfMaterials.RequiredQuantity * ManufacturingOrders.Que...

Lot ID: BillOfMaterials.RequiredLotId

Must Simultaneously Consume: False

Advanced Options

State Assignments

Add-On Process Triggers

Lot ID

Optional string value indicating the lot identifier of the consumed or produced material.

Buttons: Add, Delete, Close

Processing Tasks - Registering Property Editor

Items:

- 30.1, Consume, Conditional, Manufacture
- 30.1, Produce, Always, Sequential, Depend
- 30.1, Produce, Always, Specific, Time, Rand
- 30.2, Setup, Conditional, Manufacture
- 30.2, Process, Always, Specific, Time, J
- 40, Produce, Always, Specific, Time, Rand

Properties:

Resource Requirements

Resource Type: Select From List

Resource List Name: ResourcesSecondaryWorkers, Workers

Selection Goal: Preferred Order

Required Name: To Rule

Destination Node: Server, Input

Off Shift Rule: Switch Resources If Possible

Required Quantity & Constraints

Advanced Options

Material Requirements

Action Type: BillOfMaterials.Materialize

Consumption Type: Material

Production Type: Material

Material Name: BillOfMaterials.ComponentMaterial

Inventory Site Type: None

Required Quantity & Constraints

Quantity: Math, 30(BillOfMaterials.Materialize == Ensm.MaterializeType.Produce, BillOfMaterials.RequiredQuantity * ManufacturingOrders.Que...

Lot ID: BillOfMaterials.RequiredLotId

Must Simultaneously Consume: False

Advanced Options

State Assignments

Add-On Process Triggers

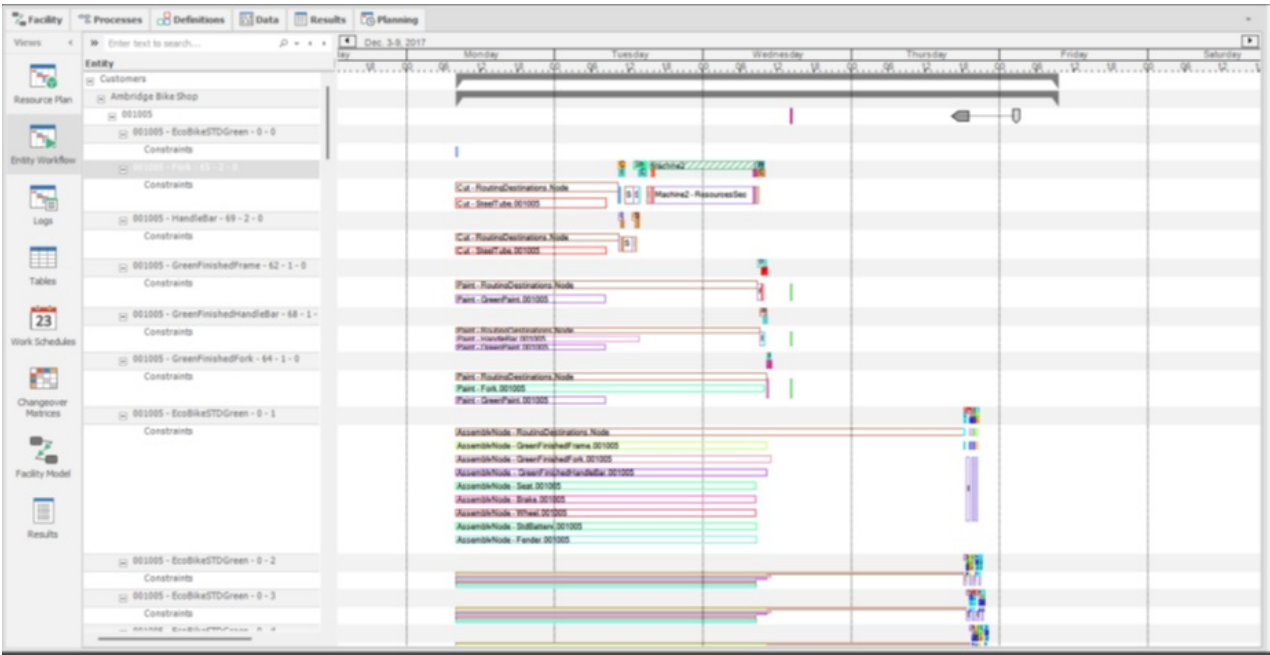
Lot ID

Optional string value indicating the lot identifier of the consumed or produced material.

Buttons: Add, Delete, Close

As a result, all the material constraints in the scheduled are pegged to the SOId. This is a great way to ensure that

component manufacturing orders and component purchase orders are linked and prioritized throughout the facility by the top-level demand.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SchedulingDiscretePartProduction - Example

General Description

The example uses Simio RPS Edition to schedule discrete part production. Simio RPS Edition is a simulation tool for developing applications in Risk-based Planning and Scheduling. RPS is the dual use of a simulation model to generate both a detailed resource constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty. Although Simio RPS Edition is required to build this model, it may be viewed using Simio Design or Professional Edition. If in Trial Edition, to give you the best planning and scheduling experience, please contact sales@simio.com.

To understand the data schema and scheduling results in this example you should first read the *Planning and Scheduling with Simio* document in C:\Program Files\Simio LLC\Simio. This problem description assumes that you are familiar with the standard data schema and scheduling concepts that are presented in that document.

The system to be scheduled in this example is a discrete part manufacturing facility. The facility is a job shop that produces finished goods. We wish to generate a 30-day production schedule for this facility that fully accounts for the limited resources in the system.

The facility consists of functionally grouped machine groups named *Cut*, *Weld*, *Shape*, and *Finish*, with two machines within each machine group. Each of these machines is modeled using a *Server*, with a *TransferNode* that is named for the machine group (e.g. *Cut*) and is used to dynamically route based on a scheduling rule to the selected machine (*Cut1* or *Cut2*) within the group. The *Cut* and *Weld* Servers require a secondary resource that is modeled as a *Worker*. The *Worker* named *Team1* operates at *Cut1* and *Cut2* Servers while *Team2* Worker operates at *Weld1* and *Weld2* Servers. The *Cut* Servers have a sequence dependent setup time where the setup time is specified in a changeover matrix based on material color. Possible material colors are defined in a string list named *MaterialColor* with values *Other*, *Red*, *Green*, and *Blue*.

There are three finished goods (*FinishedGoodA*, *FinishedGoodB*, and *FinishedGoodC*) that are produced in this facility, and each has its own routing and unique setup and processing time and material requirements at each *Server* within its routing. The B2MML-based data schema discussed in the *Planning and Scheduling with Simio* document is used to hold the production data for this example.

Although we are modeling the material consumption at each *Server*, in this example we are not explicitly modeling the material resupply logic. The consumed materials are defined in the *Material Lots* table and include *MaterialX* and *MaterialY*, and each has sufficient levels to supply production during the planning horizon. The *Bicycle* example model that is installed with Simio illustrates modeling of the material resupply based on a *Purchased Material* arrival table.

In this example we are provided a set of CSV files for automatically generating the major components of the model, and populating the required data tables. This illustrates the concept of "data-generated" modeling where the model is created automatically by populating the object data into the *Resource* table. In this example, the objects that are created from this table are already mapped to the scheduling tables.

Detailed Description

For this example, we provide the following additional data files that we will use for building the model and populating the data tables:

File Name	Description
Resources.csv	A list of resources (objects) in the facility.
RoutingDestinations.csv	A list of node destinations from transfer node.
Manufacturing Orders.csv	A list of orders to be produced during this planning period.
Materials.csv	A list of materials that are defined in the system.
Material Lots.csv	A list of material lots that are consumed and may limit production.
Routings.csv	The routing that is required to produce each material.
BillOfMaterials.csv	The materials required at each routing step.
WorkInProcess.csv	The initial state of the work in process.
DispatchList.xml	An xml file of the dispatch dashboard schema for import.
Materials.xml	An xml file of the materials dashboard schema for import.
Orders.xml	An xml file of the orders detail dashboard schema for import.
Dispatch List Report.repx	A table report file for a dispatch list report for import.
Order Details.repx	A table report file for an order details report for import.

These files are located in the *DiscretePartProductionFiles* folder that is located in the same directory as this example.

To build this model we execute the following steps:

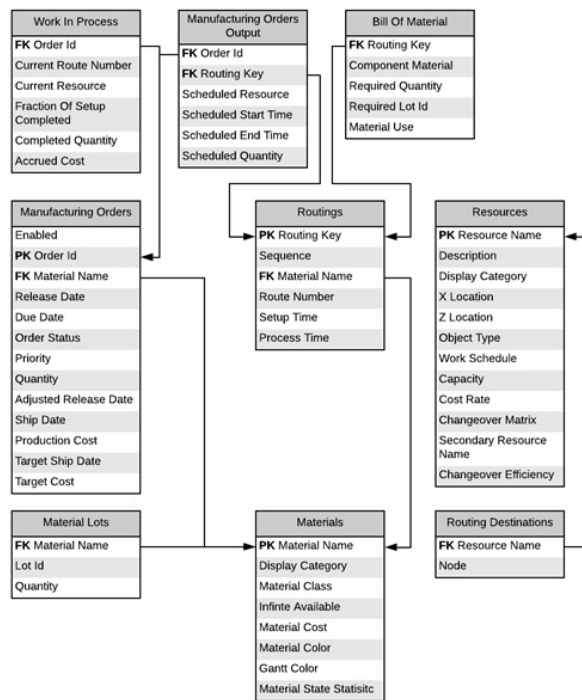
1. Click the *File > New From Template* option and select *ISA95 Schema Product Based Routings* template. This will create the tables and the subclass objects used for scheduling.
2. Import the *Resources.csv* into the *Resources* table. This will generate the objects for the model.
3. Import the remaining csv files into the data tables.
4. Add custom Dashboards and Reports.
5. Enhance the model logic as necessary (in this example no enhancements are made).

We will now describe each of these steps in detail.

[Open the ISA95 Schema Product Based Routings Template](#)

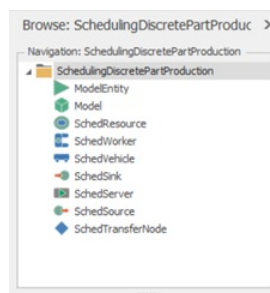
Our first step is to to File > New From Template option and select the *ISA95 Schema Product Based Routings* template option. This will open a project that includes the *Resources*, *Routing Destinations*, *Materials*, *MaterialLots*, *Manufacturing Orders*, *Routings*, *Bill of Materials*, *Work In Process* and *Manufacturing Orders Output* data tables; and the related string lists. These data tables will be populated at a later step by importing data from csv files. The *Order Status* string list is used to define orders as one of two types: *New* orders that are being released to the facility, and *WIP* orders that are currently in process. The *Material Color* string list is a default string list for use with change dependent setup times. In a typical application this list would be renamed to size, part group, etc., based on the attribute that triggers a dependent setup change. In this example, we will use the default *Material Color* list to define the changeover attribute.

The following entity relationship diagram depicts the data tables that are used in this example, along with the relations between the tables (Note: see Appendix A – Data Tables for a detailed description of the fields in each table). This data schema assumes a product-based routing where the routing is defined for each material that can be produced in the facility, and the manufacturing orders refer to each material being produced for that order. It is also possible to configure the tables where a unique routing is specified for each order. This later approach is useful in an engineered to order manufacturing environment.



Although the default Gantt display and Logs work fine for many applications, it is convenient to be able to extend the Logs with additional information, and to use this extra information to augment the Gantt display. This add-in will also add three new custom columns to the Resource Usage Log for this purpose. The first specifies the name of the material being produced on the resource. The second is the desired fill color for drawing bars on the Gantt. The third is the desired outline color for drawing these same bars.

In addition to creating the tables, the template includes the sub-classed objects that are used for scheduling. These objects are the same as their Standard Library objects except the default value for the object properties are mapped to the Scheduling tables. In the case of the Server, task sequences are also added to model the setup and processing phases of the server. As the objects are added to the model, they will already be mapped to use the data in the scheduling tables. All the objects in the Navigation window that start with "Sched" are the sub-classed scheduling objects.



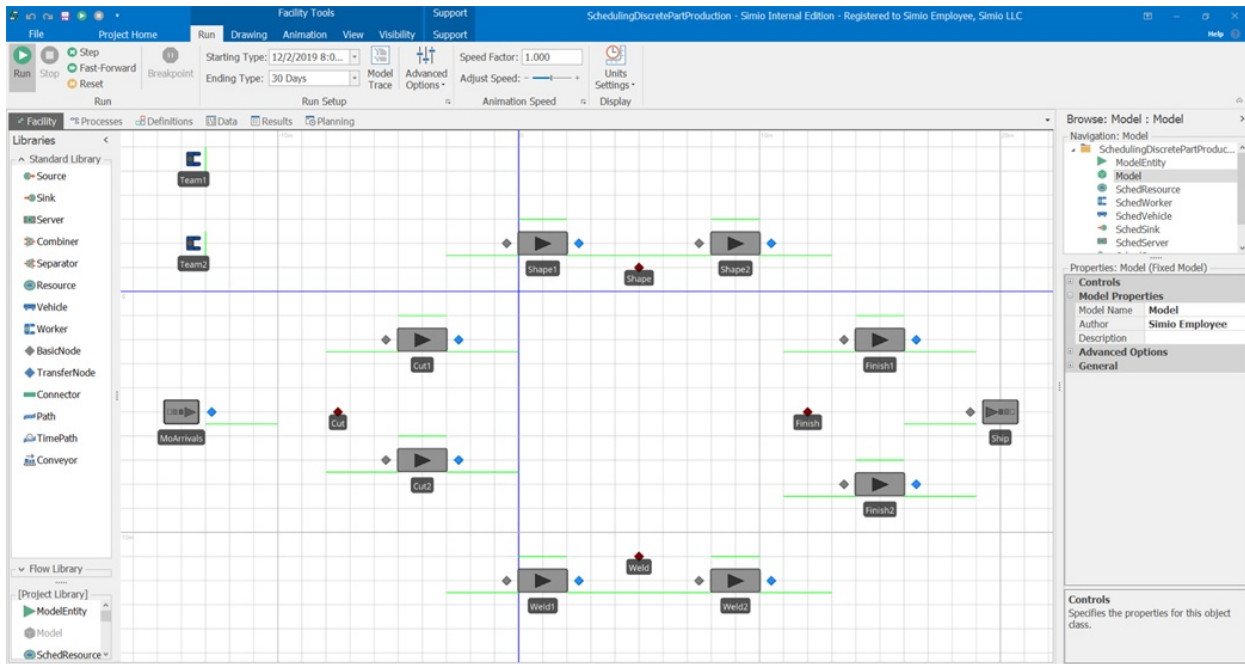
Import the Resources

Next we will populate the Resources table by importing the data from the Resources.csv data file. To do so we first bind to the file by selecting *Bind To CSV* on the Content ribbon of the Data window, and then browse to and select the Resources.csv file in the Examples folder:

C:\Program Files\Simio LLC\Simio\Examples\DiscretePartsProductionFiles

We then set our binding option to manual, and then click on *Import* to import the resource data into the table. As the data is imported into the table, the objects are automatically added to the model. They are placed in the model at the

XLocation and ZLocation coordinates in the Resources table. The resulting model is depicted below:



Import the remaining csv files

Next we will import the remaining csv files to populate the remaining set of data tables. In each case, we bind to the appropriate csv file, select the binding option to manual, and then import the data. During this process you may generate errors because of references to undefined items, but these errors will all be removed once all the data has been imported and the references are fully resolved. Also, note that the Manufacturing Orders Output table has no imported data rows; data is added to this table as the simulation executes. The following shows the imported relational data for the Material table.

Material Name	Display Category	Material Category	Material Name	Route Number	SetupTime (Hours)	ProcessTime (Hours)	Material Color	Gantt Color	Material State Statistic
FinishedGoodA_10	Materials/Finished Goods	Finished Goods	FinishedGoodA	10	3	0.4	0 Green	LimeGreen	FinishedGoodASS
FinishedGoodA_20	Materials/Finished Goods	Finished Goods	FinishedGoodA	20	0	0.7	0 Green	LimeGreen	FinishedGoodASS
FinishedGoodA_30	Materials/Finished Goods	Finished Goods	FinishedGoodA	30	0	0.44	0 Green	LimeGreen	FinishedGoodASS
FinishedGoodA_40	Materials/Finished Goods	Finished Goods	FinishedGoodA	40	0	0.4	0 Green	LimeGreen	FinishedGoodASS
FinishedGoodA_50	Materials/Finished Goods	Finished Goods	FinishedGoodA	50	0	0	0 Green	LimeGreen	FinishedGoodASS
FinishedGoodB	Materials/Finished Goods	Finished Goods	FinishedGoodB				0 Red	Tomato	FinishedGoodBSS
FinishedGoodC	Materials/Finished Goods	Finished Goods	FinishedGoodC				0 Blue	DeepSkyBlue	FinishedGoodCSS
MaterialX	Materials/Raw Materials	Raw Materials	MaterialX				60 Other	Orange	MaterialXSS
MaterialY	Materials/Raw Materials	Raw Materials	MaterialY				80 Other	Purple	MaterialYSS

Add Custom Dashboards and Reports

Our next step is to enhance the model with some custom dashboards and table reports. We will import three standard dashboards to our model that are designed to work with the default data schema. These dashboards are saved in XML format and included in the DiscretePartProductionFiles folder that is saved with this example. These dashboards display material, order details, and a dispatch list for use by operators. We will also import a custom dispatch list table report and a customer orders details table report.

To import these dashboards, go to Dashboards Report view of the Results window and select the Dashboards ribbon. Click in the Import button and browse to the folder, and select the three dashboard xml files named Dispatch List, Materials, and Order Details. To import the two reports, go to the Table Reports view under the Results window and import the Dispatch List Report.repx for Manufacturing Orders Output table, and the Orders Details.repx for the Manufacturing Orders table.

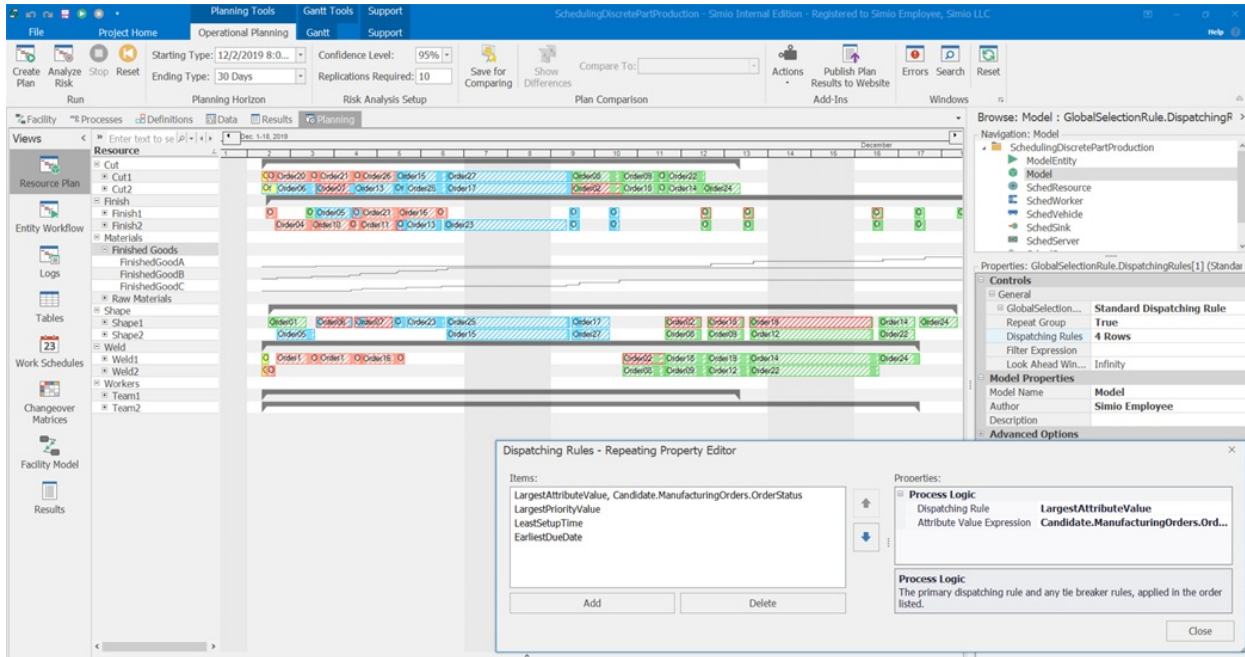
Enhance the Model Logic

Although a default model can be rapidly built using the available add-ins, it's sometimes desirable to edit this default model to add additional logic or detail to the Facility model. Examples of possible enhancements include moving operators (using Worker objects) that travel between Servers, complex material handling devices such as AGVs or conveyor systems, as well as custom decision logic for selecting between orders or Servers. Note that the full modeling power of Simio is available to us to customize the model as needed. In this example, we will keep the basic model that is automatically created for us by the scheduling add-ins.

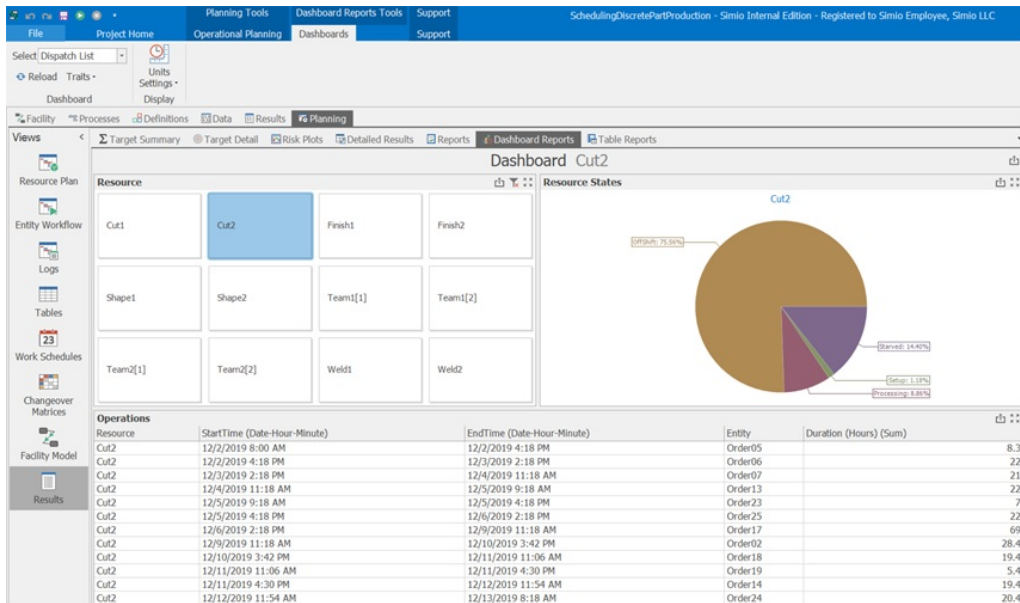
Generating the Schedule

Once our model is complete we can generate the schedule by clicking on Create Plan on the Operational Planning ribbon in the Planning window. This generates the deterministic schedule by automatically removing the randomness from the model. We can then generate the risk analysis for this schedule by clicking on the Analyze Risk button on the same ribbon.

Our model was created using a global scheduling rule that can be easily changed in the model properties. By default, orders are processed first-in-first-out, however this global rule can be easily changed to any of the standard dispatching rules supported by Simio. The following depicts a zoom-in portion of the Resource Plan for this example using the LargestAttributeValue rule based on OrderStatus (meaning that WIP will be processed first, see Definitions > OrderStatus list which is used in ManufacturingOrders table). Largest Priority values will be processed next, ties broken by LeastSetupTime and then EarliestDueDate. These rules can easily be modified by selecting a rule and using the up/down arrows within the repeat group.



As discussed in the *Planning and Scheduling with Simio* document, there are many different ways to view the schedule and the associated risk. For example, the following is the dashboard for the dispatch list that we imported using the XML file. This dashboard shows the resource utilization and dispatch list for the currently selected resource (Cut2).



Data Tables

The following is a summary of the default tables that are based on the B2MML standard.

Resources:

A list of resources that are in the manufacturing facility.

Column Name	Description
Resource Name	The unique name of the resource. This is the object name.
Description	A description of this resource.
Display Category	The machine and worker groups.
Object Type	This reference an object type (sub-classed object) of the object.
XLocation	This is the X location of the object in the model.
ZLocation	This is the Z location of the object in the model.
Work Schedule	Work schedule assigned to the resource.
Capacity	The capacity of the resource.
Cost Rate	The hourly rate for the resource either idle or while being utilized.
Changeover Matrix	Changeover Matrix used by the resource.
Secondary Resource Name	This is the resource (object) that is seized by the primary resource while processing.
Changeover Efficiency	The calculated efficiency as it changes during the run.

Routing Destinations:

A list of each possible destination node for each resource. The entity is routed to one the nodes specified in this list based on the selection rule.

Column Name	Description
Resource Name	This is the transfer node used to route each entity to their destinations.
Node	Possible destination from this transfer node.

Materials:

A list of materials that can be produced at this manufacturing facility.

Column Name	Description
Material Name	The unique name of this material.
Material Class	A description of this material
Material Cost	The cost per unit of this material.
Material Color	The color index for computing color-dependent changeovers.
Gantt Color	The color used for drawing orders for this material in the Gantt.
Material State Statistic	State statistic used to plot material quantities over time on the Gantt.

Material Lots:

A list of material lots and quantities for materials in the facility.

Column Name	Description
Material Name	The unique name of this material.
Display Category	The category of the material.
Material Class	A description of this material.
Infinite Available	Boolean for if the material is infinitely available.
Material Cost	The cost per unit of this material.
Material Color	The color index for computing color-dependent changeovers.
Gantt Color	The color used for drawing orders for this material in the Gantt.
Material State Statistic	State statistic used to plot material quantities over time on the Gantt.

Manufacturing Orders:

A list of all production orders to be processed during this planning period.

Column Name	Description
Enabled	Boolean for if this order is enabled.
Order Id	A unique string name assigned to this manufacturing order.
Material Name	A foreign key reference to the Materials table for the material being manufactured.
Release Date	The date-time this order is released to production.
Due Date	The date-time this order is due.
Order Status	The order status specified as New or WIP (work in process).
Priority	The scheduling priority for this order.
Quantity	The material quantity to be produced for this order.
AdjustedReleaseDate	This field is an expression field used by the source to determine when to release orders. This expression is used to release the WIP orders slight before (Math.Epsilon) before the NEW orders. It also handles when the release data is before the start of the model (e.g. Math.Max(ManufacturingOrders.ReleaseDate, Math.Epsilon)).
Ship Date	The ship date for this order (an output) based on the generated schedule.
Production Cost	The production cost for this order (an output) based on the generated schedule.
Target Ship Date-Value	The target ship date for this order.
Target Ship Date-Status	The target ship date status; OnTime, Late, or Incomplete.
Target Cost-Value	The target cost for this order.
Target Cost-Status	The target cost status: OnBudget, Overrun, or Incomplete.

Note that the *Ship Date* and *Production Cost* are both output columns in the Manufacturing Orders table; they are written from the simulation model to the table as the schedule is generated by the deterministic simulation run. Also note that the last four columns are targets.

Routings:

A list of job routings for each material specifying the resource and processing time for each task required to produce the material.

Column Name	Description
Routing Key	The unique name for this routing step.
Sequence	The transfer node (list of resources) or resource where this step is to be processed.
Material Name	A foreign key reference to the Materials table for the manufactured material.
Route Number	The routing number.
Setup Time	Discrete setup time for this step.
Process Time	Processing time for this step. In this example, the process time is multiplied by the ManufacturingOrders.Quantity.

Bill of Materials:

A list of component materials that are required at a specific routing sequence location to produce a material.

Column Name	Description
Routing Key	The material routing step where the material action takes place.
Component Material	The name of the material.
Required Quantity	The quantity of the material that is either consumed or produced. This value based on an order quantity = 1. The required quantity is multiplied by the ManufacturingOrders.Quantity.
Required Lot Id	Specified the lot id required for consumption and the lot id provide for production.
Material Use	Specifies if the material is to be consumed or produced.

Work In Process:

A list of current orders that are in the system, along with their current status.

Column Name	Description
Order Id	Foreign key reference to the Order Id in the Manufacturing Orders table.
Current Route Number	The current route number for this order.
Current Resource	The current resource that this order is being processed on.
FractionOfSetupCompleted	Percentage of setup completed (values between and including 0 and 1)
Completed Quantity	The number of material units that have been processed on this resource.
Accrued Cost	The activity-based cost accrued by this order at this point.

Manufacturing Orders Output:

An output table defining order start and end time on each resource used for creating table reports.

Column Name	Description
Order Id	Foreign key reference to the Order Id in the Manufacturing Orders table.
Routing Key	Foreign key reference to the Routing Key in the Routings table.
Scheduled Resource	The resource where this order is to be processed.
Scheduled Start Time	The start time at this resource for this order.
Scheduled End Time	The end time at this resource for this order.
Scheduled Quantity	The quantity schedule at the resource for this order.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SchedulingLaborEfficiencies - Example

General Description

This example uses Simio RPS Edition to build a data-generated model for scheduling orders, at a manufacturing facility, requiring laborers with varying efficiencies. Simio RPS Edition is a simulation tool for developing applications in Risk-based Planning and Scheduling. RPS is the dual use of a simulation model to generate both a detailed resource constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty. Although Simio RPS Edition is required to build this model, in its entirety, it may be viewed using Simio Design or Professional Edition. If in Trial Edition, to give you the best planning and scheduling experience, please contact sales@simio.com.

To understand the data schema and scheduling results in this example it is recommended to read the "Planning and Scheduling with Simio" document in C:\Program Files\Simio LLC\Simio. This problem description assumes familiarity with the standard data schema and scheduling concepts that are presented in that document.

The facility is a job shop that produces four types of finished goods. It is intended to create a 6-month (180 day) production schedule that fully accounts for the limited resources in the system. The facility consists of four functionally grouped workstations named Press, Anodize, Mill and Paint. Each workstation has a staging area from which orders are routed to a few machines. There are four machines in Press, two in Anodize, two in Mill, and one in Paint. The template used to start building this model provides custom Object definitions created by using the Subclass feature on Standard Library Objects. These are called SchedSource, SchedServer, etc. and have been specifically designed for use in scheduling models.

The staging areas are modeled using a SchedTransferNode while the machines are SchedServer Objects. In this document, let "machines" be used to refer to all SchedServer Objects in the model. Each machine requires a Secondary Resource, modeled using a SchedWorker in this case, to complete its Processing Tasks. Additionally, each machine has a sequence dependent setup time specified in a changeover matrix and based on material color. Material colors are defined in a string list named MaterialColor with values 'Green', 'Red', 'Blue' and 'Other'.

There are four finished goods (FG1, FG2, FG3 and FG4) produced in this facility. Each has its own routing, unique setup/processing times, and material requirements at each machine within its routing. The ISA95SchemaProductBasedRoutings template (File -> New From Template) discussed in the Planning and Scheduling with Simio document is used to organize the production data for this example. Some other data tables are defined and used in conjunction with those generated by the template.

Although material consumption is modeled at each machine, this example does not explicitly model the material resupply logic. The consumed materials are defined in the Material Lots table and include 'Tool1' and 'Tool2'. It is assumed that each material has enough levels to supply production during the planning horizon.

This example illustrates the concept of "data-generated" modeling where the model is created automatically by storing Object data in a table called Resources. In this example, the Objects that are created from this table are already mapped to the other standard scheduling data tables.

Detailed Description

To build this model, execute the following steps after opening Simio (Version 11.193 or newer required):

1. Open the ISA95 Schema Product Based Routings template
2. Import the Resources table (Auto-Create Objects)
3. Import the remaining csv files
4. Create the Labor table schema, import data, and configure Objects accordingly
5. Define the Work Schedules
6. A few quick revisions
7. Create Manufacturing Orders Sec Res Output table
8. Create Press Capability And Times table schema, import data
9. Design Add-On Process logic
10. Generate the Schedule
11. Create OptQuest Experiment

Each of these steps will be described in detail. Note that the csv files may be obtained by opening the SchedulingLaborEfficiencies model and using Export (Data tab -> Content ribbon) for each table therein. This document is nearly a complete set of instructions on how to build this model, and a learning tool to better understand aspects of the

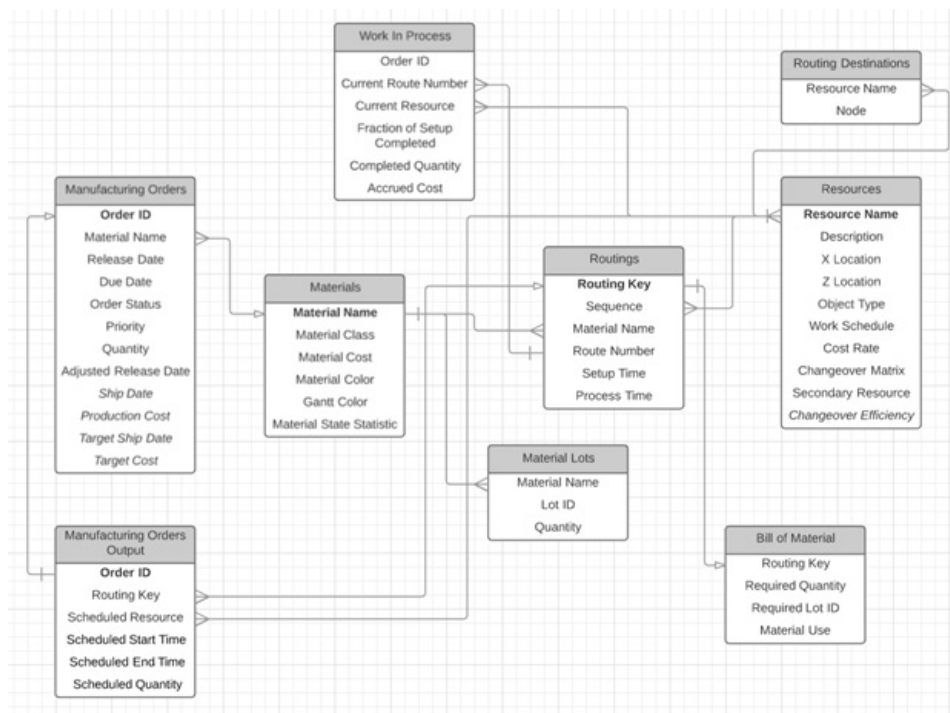
(already built) example model. As a convention, property names are typed in *italics* while property values, table column names and instance names are typed in 'single quotes'. Object, table and process names are simply typed in uppercase as they appear in the model.

Open the ISA95 Schema Product Based Routings Template

The ISA95 Schema Product Based Routings is opened at File > New From Template. This opens a project including the Resources, Routing Destinations, Materials, Material Lots, Manufacturing Orders, Routings, Bill of Materials, Work In Process, and Manufacturing Orders Output data tables; and the related string lists. These data tables will be populated at a later step by importing data from csv files.

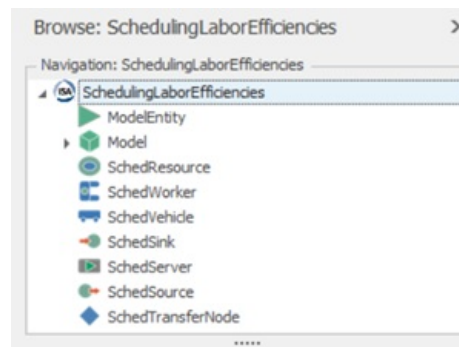
The Order Status string list is used to define orders as one of two types: 'New' orders that are being released to the facility, and 'WIP' orders that are currently in process. The Material Color string list is a default string list for use with change dependent setup times. In a typical application, this list would be renamed to size, part group, etc., based on the attribute that triggers a dependent setup change. In this example, the default Material Color list is used to define the changeover attribute.

The following entity relationship diagram depicts the template-generated data tables that are used in this example, along with the relations between them (Note: see Appendix A – Data Tables for a detailed description of the fields in each table). This data schema assumes a product-based routing where the routing is defined for each material that can be produced in the facility, and the manufacturing orders refer to each material being produced for that order. It is also possible to configure the tables where a unique routing is specified for each order. This later approach is useful in an engineered to order manufacturing environment.



Although the default Gantt display and Logs work fine for many applications, it is convenient to be able to extend the Logs with additional information, and to use this extra information to augment the Gantt display. In this case, the template adds three new custom columns to the Resource Usage Log for this purpose. The first specifies the name of the material being produced on the resource. The second is the desired fill color for drawing bars on the Gantt. The third is the desired outline color for drawing these same bars.

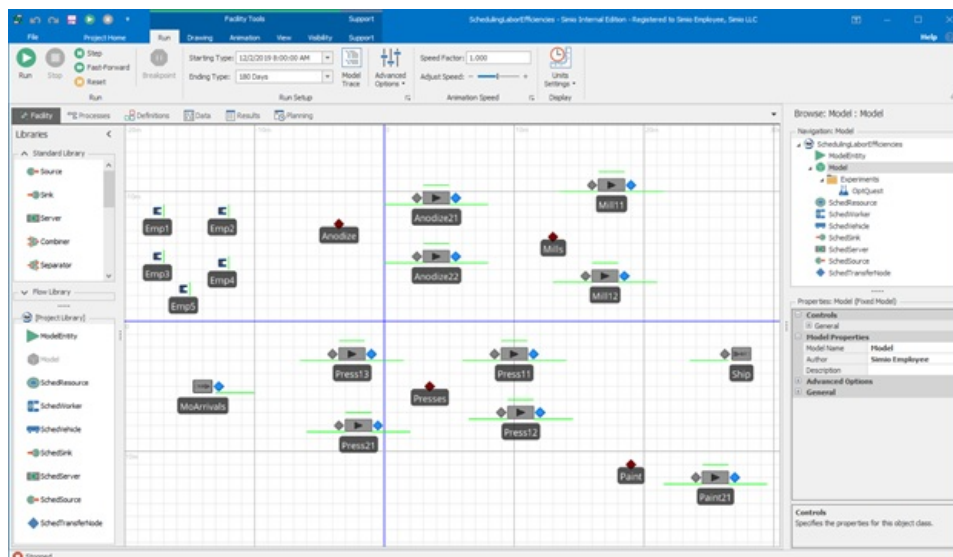
In addition to creating the tables, the template includes the custom subclassed Object definitions that have been defined specifically for Simio scheduling models. These objects are comparable to the Standard Library Objects that they were subclassed from, but some properties are different, or removed altogether, and several property default values are mapped to the scheduling tables generated by the template. For the machines, task sequences are added (to the SchedServer definition) to model setup and processing phases. As Objects are added to the model, via the Auto-Create feature, they will already be configured to use data from the scheduling tables due to their property default values. The custom scheduling Object definitions are found in the Navigation window, and all start with "Sched".



Import the Resources table (Auto-Create Objects)

Recall that, in order to build this model from scratch, all data tables in the SchedulingLaborEfficiencies model must be exported (Data tab -> Content ribbon) to csv. First, the Resources table is populated by importing data from the Resources.csv data file. To do so first bind the file, to Simio, by navigating to the Data window, selecting the Resources table and then selecting Create Binding on the Content ribbon. Then browse to, and select, the Resources.csv file saved locally.

Next use the Binding Options dropdown to select Manual, and then use the Import Table dropdown to select Import Current Table. Going forward, this general procedure is used to bind a file and import data from it. Once the table is populated with data, the objects are automatically instantiated in the model Facility window. They are placed in the Facility window at the XLocation and ZLocation coordinates in the Resources table. The resulting model setup is depicted below:



Import the remaining csv files

Next, the remaining csv files are imported to populate the remaining set of data tables. In each case, bind to the appropriate csv file, set the binding option to manual, and then import the data. During this process errors may be generated because of references to undefined items, but they will all be eliminated once the process has been run and the references are fully resolved. Also, note that the Manufacturing Orders Output table has no imported rows of data, because rows are added to this table as the simulation executes. The following shows the imported relational data for the Materials table. A String State Variable column named 'ValidPressResources' is added to the Materials table.

Material Name	Display Category	Material Class	Material Cost (Default Currency)	Material Color	Gantt Color	Material State	Valid Press Resources
FG1	Materials/Finished Goods	Finished Goods	0	Green	Yellow	FG1SS	
FG2	Materials/Finished Goods	Finished Goods	0	Red	Tomato	FG2SS	
FG3	Materials/Finished Goods	Finished Goods	0	Blue	DeepSkyBlue	FG3SS	
FG4	Materials/Finished Goods	Finished Goods	0	Other	Orange	FG4SS	
Tool1	Materials/Tooling	Tooling	5	Other	Gray	Tool1SS	
Tool2	Materials/Tooling	Tooling	5	Other	Gray	Tool2SS	

Create the Labor table schema, import data, and configure Objects accordingly

The Labor table specifies which Secondary Resource is required for Processing Tasks, at each machine, and a Secondary Resource Efficiency value expressed as a fraction. Resource Efficiency, in Simio, is implemented such that the actual task duration is the planned task duration divided by the efficiency for each task requiring a Secondary Resource.

In general, refer to Appendix A – Data Tables for a complete list of table and column names along with descriptions of them. Notice that the 'ResourceName' column is a Foreign Key Property with its Table Key Property to 'Resources.ResourceName'. Also, the 'SecondaryResourceEfficiency' column has its Default Value set to '1'.

Next, bind to Labor.csv and import its data (for the remaining tables this step is not explicitly stated). Notice that multiple Secondary Resources might be eligible to perform tasks at a certain machine in the 'ResourceName' column, while others are not eligible. Ineligibility to perform a certain task may generally be specified by an Efficiency value of 0, or in this case, the exclusion of a certain Secondary Resource in the Secondary Resource Name column – for a given machine. This will be clearer after configuring the machines to reference the data in this table.

Select the 'Press13' machine and open the Repeating Property Editor for its Processing Tasks. Notice the following changes made for each of the six tasks therein:

Resource Requirements	
Object Type	Select From List
Object List name	Labor.SecondaryResourceName
Selection Goal	Largest Value
Value Expression	Labor.SecondaryResourceEfficiency
Request Move	To Node
Destination Node	SchedServer.Input
Off Shift Rule	Switch Resources If Possible
Advanced Options	
Number Of Objects	1
Units Per Object	1
Selection Condition	
Resource Efficiency	Labor.SecondaryResourceEfficiency
Must Simultaneously Seize	False
Immediately Try Seize	False
Keep Reserved If	True
Reservation Timeout	Math.Epsilon
Immediately Try Allocate When Released	False

In this case, a table column is used as a list, and more information about this is presented in the "Data Tables" help topic, specifically "Using a Table Column as a List" subtopic. With this configuration, the machines will attempt to seize the most efficient, available Secondary Resource. The ResourceSelectionConcepts SimBit includes more examples of data-driven approaches to modeling Resource Efficiency. Since the Labor table 'ResourceName' column is a Foreign Key reference to Resources.ResourceName (which has *Auto-set Table Row Reference* of 'True'), each machine knows to only search the Labor.SecondaryResourceName rows that pertain to it.

After all *Processing Tasks* are updated for 'Press13', right-click and 'Update 'SchedServer' Property Defaults From This' to extend these changes to the other machines. Then verify that all other machines now have their *Processing Tasks* revised to account for Resource Efficiency.

Define the Work Schedules

One Pattern Based Work Schedule and two Table Based Work Schedules are defined. Navigate to Work Schedules in the Data tab. A Day Pattern called 'AllDay' is used in the 'AllDaySchedule' Work Schedule whose pattern contains just 1 day. 'AllDay' specifies a 24hr shift – Duration set to '1 day'.

Two data tables called DaySchedule and NightSchedule specify the work periods, and corresponding capacity value (in this case, a Boolean Property – numerically 0 or 1), to be used in two corresponding Repeating Table Based Work Schedules. These are created and configured, in the Table Based tab (Data tab -> Work Schedules view) to reference the data tables. Notice that the properties of each Table Work Schedule are set to reference the appropriate columns in DaySchedule and NightSchedule. Also note that the *Repeating* property is set to 'True' with an *Interval* of '1' (Weeks).

A few quick revisions

These details serve to resolve the errors mentioned in section 3. In the Resources table, notice that the 'WorkSchedule' column has *Default Value* of 'AllDayWeek' and *Required Value* set to 'False'. In the Material Lots table, the Change Type feature (Schema ribbon) is used to make the 'Quantity' column an Expression Property. The same Change Type adjustment is made to the 'RequiredQuantity' column in the Bill Of Materials table. Then, in the Definitions tab, a Standard Property named 'ToolQty' is defined, and its *Default Value* is set to '3'.

Create Manufacturing Orders Sec Res Output table

An Output Table called Manufacturing Orders Sec Res Output is used to record information, throughout the run, about the Secondary Resources and to specifically highlight details about the Resource Efficiency features. Recall that an Output Table contains only State columns and typically uses Add-On Process logic to add rows throughout the run and populate the table with output data. Think about how this is different than simply adding a State column, to a table already populated with data, and when each approach might be useful. Consider referencing the Manufacturing Orders table for an example of using State columns in a standard data table.

Create Press Capability And Times table schema, import data

The last data table called Press Capability And Times specifies which machines, in the Presses workstation, can process the various Finished Goods – and the corresponding processing times. Notice how the Foreign Key column properties are configured. And 'ProcessTime' has *Unit Type* 'Time' with *Default Units* set to 'Hours'.

Design Add-On Process logic

Particularly in this section, it is recommended to follow along in the SchedulingLaborEfficiencies model provided. If building the model from scratch, consider opening it side-by-side with the example. This section will give an overview of the processes, and their purposes, but will not provide explicit step-by-step design instructions. They are presented chronologically in terms of execution during the run.

OnRunInitialized

In the Processes tab, an OnRunInitialized Add-On Process is created. It uses data in the Press Capability And Times table to assign values in the 'ValidPressResources' column of the Materials table, for each finished good. This is achieved using a Search Step to find rows in the Materials table. There is no *Match Condition* so notice (using Model Trace, and perhaps a Breakpoint on the Search Step) that all six rows in the Materials table are found. For each found item a new Token exits the Found branch and, in this case, initiates another Search in the Press Capability And Times table. It can be informative to Step through the Trace and observe how the 'ValidPressResources' column (Materials table) is populated.

SchedServer_BeforeProcessing

Create a ModelEntity Real State Variable called 'ProcessTime' – beware of the common mistake to define this in Model, rather than ModelEntity, in the Navigation window! The next Add-On Process is called SchedServer_BeforeProcessing and it is assigned the *Category* 'Press Add-On Processes'. This Process occurs when an entity has been allocated Server capacity, but before entering (or ending transfer into) its processing station. It serves to check whether an Entity is at one of the Press machines, and if so, assign its ProcessTime from the Press Capability And Times table. Otherwise, ProcessTime is assigned from the Routings table. After designing this process, use ctrl+click to multi-select all the machines and assign it as the value of *Before Processing* (under Add-On Processes). Now this Add-On Process is triggered at each machine.

LaborAllocated

This process serves to populate the Manufacturing Orders Sec Res Output table. The Add Row Step specifies the *Table Name* 'Manufacturing Orders Sec Res Output' and that the Token should carry the row reference to the added row. In perusing the Assign Steps, notice that the Output Table is populated with information from some of the data tables, the SchedWorker Object itself, the ModelEntity who seizes the SchedWorker, and the system in general. Since this process is ultimately triggered on *Allocated*, the Token.AssociatedObject is the ModelEntity who seized SchedWorker and the Token.ContextObject is the specific realization of SchedWorker who was seized. Therefore, information about both Objects is accessible in the process. The PerPartProcessing column reflects the effect of Efficiency on Processing Time. To see it clearly, filter the ParentName column to a specific SchedServer, and observe that lower efficiency corresponds to longer Processing Time.

LaborReleased

First create two model Entity String State Variables called 'OrderId' and 'RoutingKey'. This process simply populates the ScheduledEndTime column of the Manufacturing Orders Sec Res Output table. The Search Step starts with the last row in the table and finds the row whose OrderId matches that of the Entity who released the SchedWorker. It also verifies that

the RoutingKey matches as a given OrderId has multiple RoutingKeys throughout the simulation. Recognize that the LaborAllocated and LaborReleased processes are triggered on *Allocated* and on *Released* for all five SchedWorker instances.

OnRunEnding

First go to the Tokens view in the Definitions tab and use the Add Token button to create a new custom Token. Name it 'OrderToken' and add two String State Variables called 'ParentName' and 'OrderId'. Back in the Processes tab an OnRunEnding process is created. Notice that the *Token Class Name*, for the process itself, is 'OrderToken'. This indicates that the custom Token, just created, is used to execute this process. Note that this process is only visible in model Trace upon hitting the Stop button after run completion.

Embellishments: Dispatching Rule, Tally Statistic

The GlobalSelectionRule property is deleted and two Enumeration Properties named 'DispatchingRule' and 'TieBreakerRule' are defined. The *Enum Type* property, for each of these, is set to 'SimioSelectionRules.StandardDispatchingRuleDefinition+DispatchingRule'. Consider experimenting with different values of *Enum Type* and observing how the dropdown options for *Default Value* change. For 'DispatchingRule' the *Default Value* is set to 'LeastSetupTime' and for 'TieBreakerRule' *Default Value* is set to 'EarliestDueDate'.

Notice that, for the Anodize SchedTransferNode (under Routing Logic -> Other Routing Out Options), *Route Request Dynamic Selection Rule* is set to 'Standard Dispatching Rule'. *Dispatching Rule* and *Tie Breaker Rule* reference the recently defined Enumeration Properties. The right-click and 'Update 'SchedTransferNode' Property Defaults From This' is used to extend these changes to all other SchedTransferNodes.

Also, for only the 'Presses' SchedTransferNode, the *Selection Condition* (under Entity Destination Type) is appended with '&& String.Contains(Materials.ValidPressResources, Candidate.Resources.ResourceName.ObjectName) == True'. This checks the Materials table 'ValidPressResources' column to ensure that a Press is valid for the (Finished Goods) 'MaterialName' of the Order that is requesting routing.

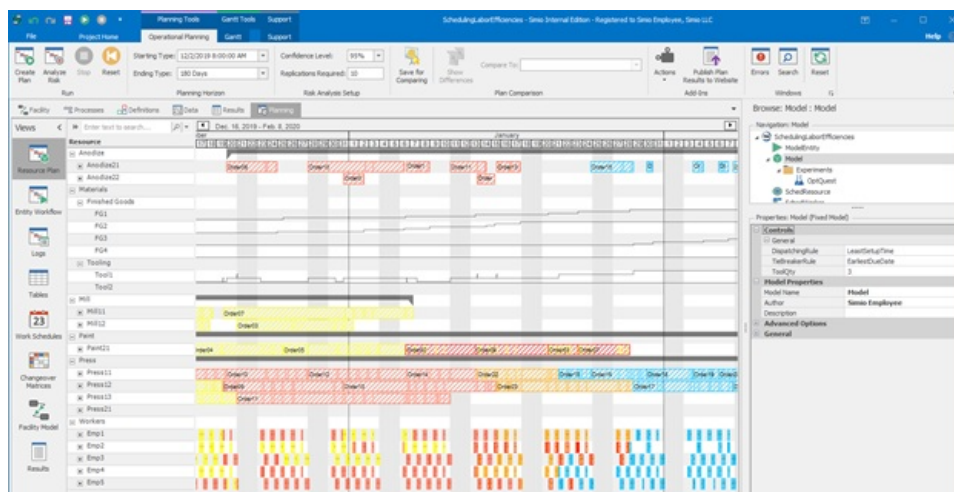
Notice the items in the Repeating Property Editor, of *Before Processing* State Assignments, for the machines. One item assigns 'ModelEntity.OrderId' a *New Value* of 'ManufacturingOrders.OrderId' and the other assigns 'ModelEntity.RoutingKey' a *New Value* of 'Routings.RoutingKey'. This adjustment is made for one machine, then 'Update 'SchedServer' Property Defaults From This' extends the changes to all machines.

Select a SchedWorker instance and notice its *Dispatching Rule* of 'SmallestAttributeValue', with *Attribute Value Expression* to 'Candidate.ModelEntity.DirectDistanceTo.Object(SchedWorker)'. This dispatches the SchedWorker to the physically closest Entity within its Allocation Queue. Additionally, right click *Tie Breaker Rule* and use Set Referenced Property to select 'TieBreakerRule'. Again, these updates are made to just one SchedWorker instance, then 'Update 'SchedWorker' Property Defaults From This' extends the changes to all SchedWorkers.

Generating the Schedule

Once the model is complete the schedule can be generated by clicking on Create Plan on the Operational Planning ribbon in the Planning tab. This generates the deterministic schedule by automatically removing the randomness from the model. Then risk analysis can be generated for this schedule by clicking on the Analyze Risk button on the same ribbon.

The model was created using a Standard *Dispatching Rule* of 'LeastSetupTime', and this can be easily changed in the model Properties. Some other options include 'FirstInQueue', 'ShortestProcessingTime', 'LongestTimeWaiting', etc. This is the primary criteria used to dynamically select an Entity from competing requests waiting to be assigned an available destination, while the secondary criteria is specified by the *TieBreakerRule* also accessible in model properties. The following depicts a zoomed-in portion of the Resource Plan for this example using the Least Setup Time rule.



Create OptQuest Experiment

A Tally Statistic Element called 'Lateness' is defined. For the Input@Ship Node, observe the *On Entering* Repeating Property Editor under Tally Statistics. Each time an Entity enters the node, an observation of 'Lateness' is tallied along with the *Expression* 'TimeNow - ManufacturingOrders.DueDate'. Observe the 'OptQuest' experiment. The OptQuest Add-In may be selected from the experiment Design ribbon. It has three Responses: 'AverageLateness', 'Cost' and 'NumberCompleted'.

Notice the property values for each of the three Responses, especially the *Weight* property which corresponds to the optimization objective function. Note that, in the Experiment Properties, *Objective Type* is set to 'Multi-Objective Weighted' so each Response (considering both *Weight* and *Objective*) is included in the optimization objective.

OptQuest uses an optimization algorithm to search for the input controls that will produce the best result for the objective function. OptQuest works by obtaining results, for a set of input controls, then running again with different controls and comparing the corresponding objective values repeatedly until it meets one of the terminating criteria – either reaching the maximum number of iterations or determining that the objective has stopped improving.

Appendix A – Data Tables

The following is a summary of the default tables that are based on the B2MML standard.

Resources:

A list of resource that are in the manufacturing facility.

Column Name	Description
Resource Name	The unique name of the resource. This is the object name.
Description	A description of this resource
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
XLocation	This is the X location of the object in the model.
ZLocation	This is the Z location of the object in the model.
Object Type	This references an object type (sub-classed object) of the object.
Work Schedule	Work schedule assigned to the resource.
Cost Rate	The hourly rate for the resource either idle or while being utilized.
Changeover Matrix	Changeover Matrix used by the resource
Secondary Resource Name	This is the resource (object) that is seized by the primary resource while processing.
Changeover Efficiency	A factor used to increase or decrease the total time taken for a setup.

Routing Destinations:

A list of each possible destination node for each resource. The entity is routed to one the nodes specified in this list based on the selection rule.

Column Name	Description
Resource Name	This is the transfer node used to route each entity to their destinations.
Node	Possible destination from this transfer node.

Materials:

A list of materials that can be produced at this manufacturing facility.

Column Name	Description
Material Name	The unique name of this material.
Display Category	Optional text used for hierarchically arranging resources in the Resource Plan window.
Material Class	A description of this material
Material Cost	The cost per unit of this material.
Material Color	The color index for computing color-dependent changeovers.
Gantt Color	The color used for drawing orders for this material in the Gantt.
Material State Statistic	State statistic used to plot material quantities over time on the Gantt.
Valid Press Resource	A string state column stating which Press Resources are valid for a given Material Name (based on data in Press Capability And Times table).

Material Lots:

A list of material lots and quantities for materials in the facility.

Column Name	Description
Material Name	A foreign key reference to the Materials table for the material being produced.
Lot Id	This is a string determining what lot the material. This field is not required. If there is no lot control, just leave column blank.
Quantity	The initial quantity of material for the lot.

Manufacturing Orders:

A list of all production orders to be processed during this planning period.

Column Name	Description
Order Id	A unique string name assigned to this manufacturing order.
Material Name	A foreign key reference to the Materials table for the material being manufactured.
Release Date	The date-time this order is released to production.
Due Date	The date-time this order is due.
Order Status	The order status specified as New or WIP (work in process).
Priority	The scheduling priority for this order.
Quantity	The material quantity to be produced for this order.
AdjustedReleaseDate	This field is an expression field used by the source to determine when to release orders. This expression is used to release the WIP orders slight before (Math.Epsilon) before the NEW orders. It also handles when the release data is before the start of the model (e.g. Math.Max(ManufacturingOrders.ReleaseDate, Math.Epsilon)).
Ship Date	The ship date for this order (an output) based on the generated schedule.
Production Cost	The production cost for this order (an output) based on the generated schedule.
Target Ship Date-Value	The target ship date for this order
Target Ship Date-Status	The target ship date status; OnTime, Late, or Incomplete.
Target Cost-Value	The target cost for this order.
Target Cost-Status	The target cost status: OnBudget, Overrun, or Incomplete.

Note that the *Ship Date* and *Production Cost* are both output columns in the Manufacturing Orders table; they are written from the simulation model to the table as the schedule is generated by the deterministic simulation run. Also note that the last four columns are targets.

Routings:

A list of job routings for each material specifying the resource and processing time for each task required to produce the material.

Column Name	Description
Routing Key	The unique name for this routing step.
Sequence	The transfer node (list of resources) or resource where this step is to be processed.
Route Number	The routing number.
Setup Time	Discrete setup time for this step.
Process Time	Processing time for this step. In this example, the process time is multiplied by the ManufacturingOrders.Quantity.

Bill Of Materials:

A list of component materials that are required at a specific routing sequence location to produce a material.

Column Name	Description
Routing Key	The material routing step where the material action takes place.
Component Material	The name of the material.
Required Quantity	The quantity of the material that is either consumed or produced. This value based on an order quantity = 1. The required quantity is multiplied by the ManufacturingOrders.Quantity.
Required Lot Id	Specified the lot id required for consumption and the lot id provide for production.
Material Use	Specifies if the material is to be consumed or produced.

Work In Process:

A list of current orders that are in the system, along with their current status.

Column Name	Description
Order Id	Foreign key reference to the Order Id in the Manufacturing Orders table.
Current Route Number	The current route number for this order.
Current Resource	The current resource that this order is being processed on.
Fraction Of Setup Completed	Percentage of setup completed (values between and including 0 and 1)
Completed Quantity	The number of material units that have been processed on this resource.
Accrued Cost	The activity-based cost accrued by this order at this point.

Manufacturing Orders Output:

An output table defining order start and end time on each resource used for creating table reports.

Column Name	Description
Order Id	Foreign key reference to the Order Id in the Manufacturing Orders table.
Routing Key	Foreign key reference to the Routing Key in the Routings table.
Scheduled Resource	The resource where this order is to be processed.
Scheduled Start Time	The start time at this resource for this order.
Scheduled End Time	The end time at this resource for this order.
Scheduled Quantity	The quantity schedule at the resource for this order.

Labor:

A list of all resources and the secondary resources eligible to work at each one. Secondary resource efficiency, for a given resource, is also specified.

Column Name	Description
Resource Name	Foreign key reference to the Resource Name in the Resources table.
Secondary Resource Name	This is the resource (object) that is seized by the primary resource while processing.
Secondary Resource Efficiency	Optional value, expressed as a fraction, which the planned task duration is divided by to get actual task duration using the seized resource(s).

Day Schedule:

A list of work periods, their duration, and the OnShift capacity to be used for a repeating table-based Work Schedule.

Column Name	Description
Start Date	Resolves the starting date and time of a work period
End Date	Resolves the end date and time of a work period
Available	A Boolean which resolves the (capacity) value of a work period

Night Schedule:

A list of work periods, their duration, and the OnShift capacity to be used for a repeating table-based Work Schedule.

Column Name	Description
Start Date	Resolves the starting date and time of a work period
End Date	Resolves the end date and time of a work period
Available	A Boolean which resolves the (capacity) value of a work period

Manufacturing Orders Sec Res Output:

An Output table to record information, throughout the run, about the Secondary Resources used for each order while specifically highlighting details about the Resource Efficiency features.

Column Name	Description
Order Id	Foreign key reference to the Order Id in the Manufacturing Orders table.
Routing Key	Foreign key reference to the Routing Key in the Routings table.
Scheduled Resource	The resource where this order is to be processed.
Scheduled Start Time	The start time at this resource for this order.
Scheduled End Time	The end time at this resource for this order.
Scheduled Quantity	The quantity schedule at the resource for this order.
Parent Name	The SchedServer resource where work is located.
Efficiency	The efficiency of the resource used to process this order.
Expected Processing Time	The order quantity times the processing time per unit divided by Efficiency.
Scheduled Duration	The cumulative scheduled duration for the specific Order Id and Scheduled Resource.

Press Capability And Times:

A list of the press resources eligible to work on each type of finished good and the corresponding processing times.

Column Name	Description
Material	Foreign key reference to the Material Name in the Materials table
Res	Foreign key reference to the Resource Name in the Resources table
ProcessTime	The process time for each finished good / press pair

Send comments on this topic to [Support](#)

SchedulingParallelRouteController - Example

Scheduling Parallel Route Controller

General Description

The example uses Simio RPS Edition to schedule manufacturing orders using a parallel route controller. Simio RPS Edition is a simulation tool for developing applications in Risk-based Planning and Scheduling. RPS is the dual use of a simulation model to generate both a detailed resource constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty. Although Simio RPS Edition is required to build this model, it may be viewed using Simio Design or Team Edition. If in Trial Edition, to give you the best planning and scheduling experience, please contact sales@simio.com.

The parallel route controller is used to define the route that a manufacturing order will take is based on task sequences to take advantage its ability to route both serial and parallel task (operations). The routes use the sequence number method to determine the sequence of serial and parallel operations should be routed through the manufacturing facility.

This example is based on the SchedulingDiscretePartProduction. The data and table structure between the 2 examples are almost identical. It is suggested that the SchedulingDiscretePartProduction example and documentation is used to understand the base modeling concept of this example.

The main differences between the SchedulingParallelRouteController and the SchedulingDiscretePartProduction model are:

- Routing tables has an additional column that contains the Sequence Number. The Sequence Number is used by the parallel route controller to determine the routing of options through the facility.
- The data in the Routing table has been updated to show examples of different sequential and parallel routing configurations.
- ManufacturingOrders table has an additional state column that lists the FirstRouteNumber. This column gets populate at the MOArrivals source and is get set to the RouteNumber of the current route row. For WIP orders, it might not be the first row based on the On Create Entity row reference in the source.
- ImmediateSuccessorSequences output table was added the model to understand the relationships between the sequence number. This table is just used as a reference in this model to help determine the appropriate sequence numbers to route operations through the facility. The output table gets populated during initialization through the "PopulateImmediateSuccessorSequences" process.

The columns are defined in the ImmediateSuccessorSequences table are as follows.

Column Name	Description
Material Name	Name of the material.
Sequence Number	The sequence number of the operation.
Successor Sequence Number	The sequence number of the next operation.

The manufacturing orders are not routed directly to the transfer nodes and servers in this example. The manufacturing orders are routed as follows:

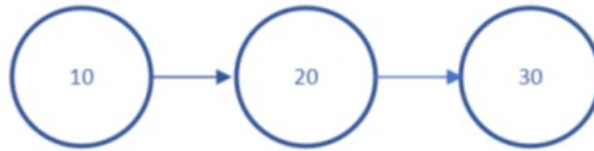
- The manufacturing orders are routed directly to the parallel route controller.
- The parallel route controller will then create operations (entities) that will be routed to the transfer nodes.
- Once an operation gets routed to a server and its operation is completed, the operation is destroyed. There is an event triggered by the destroying entity that will enable the parallel route controller to process any downstream operation(s).
- If there are no more operations to be routed, the manufacturing order will be routed to the Ship (sink) object.

Detailed Description

For this example, the Routings table has been configured to handle the following routings based on Materials. Here are the Routings that have been defined by product. The numbers in the circles are the Sequence Number in the Routings table.

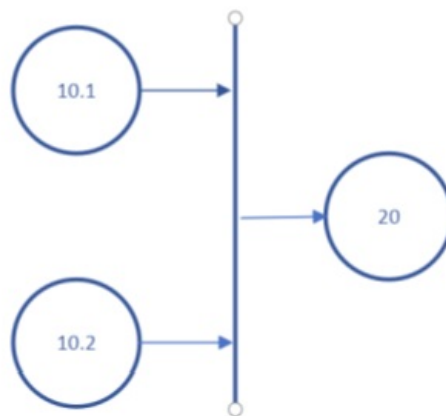
FinishedGoodsA

This is the simplest route in this example. This is a sequential route with 3 operations.



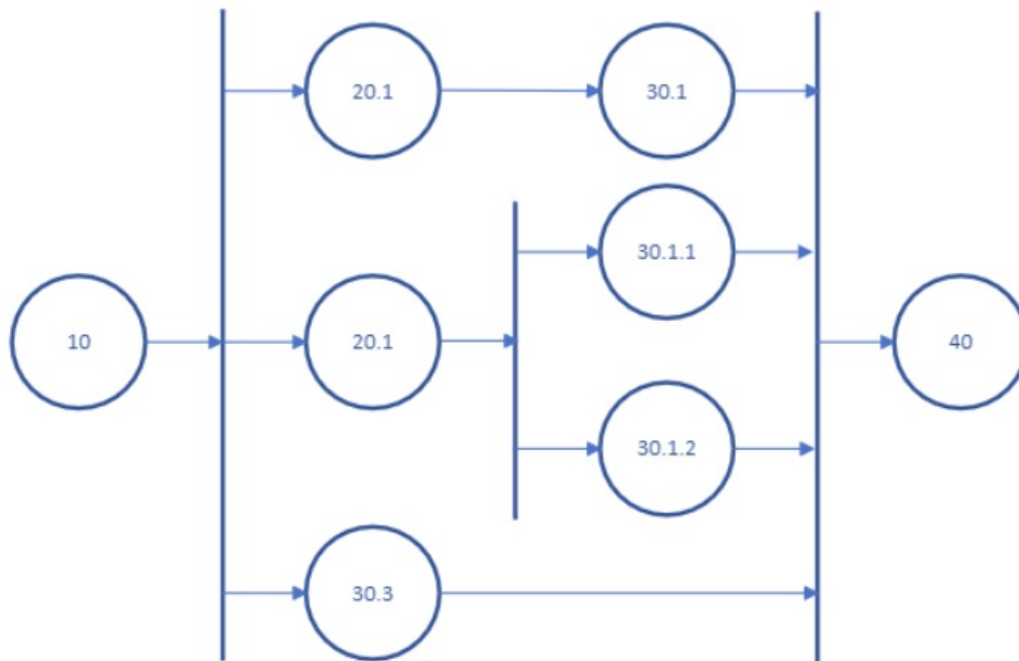
FinishedGoodsB

This route has the first 2 operations being run in parallel. Both need to be completed before starting the last operation.

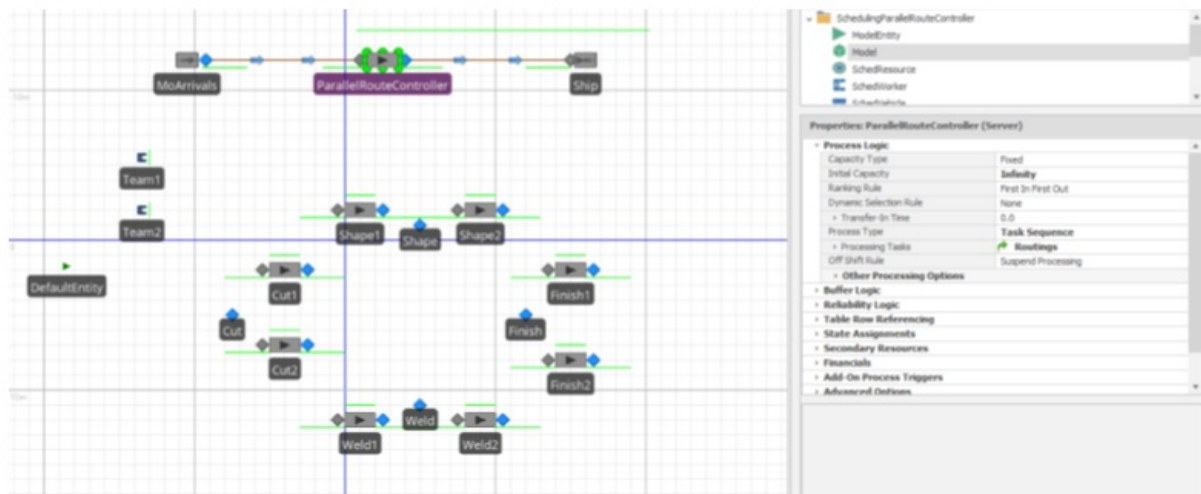


FinishedGoodC

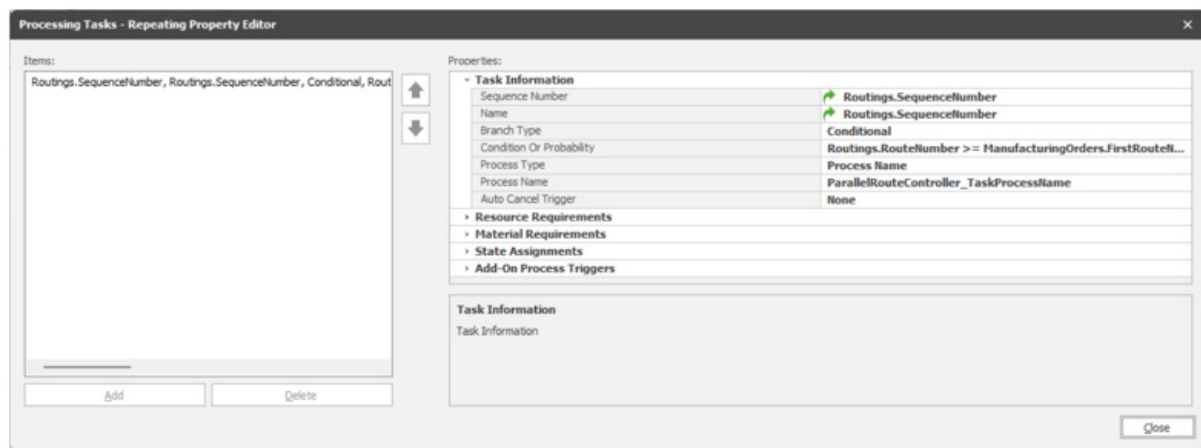
This is the most complex route in this example. The first operation needs to be completed before the second set of operations can start. The second set of operations can run in parallel. Operation 30.3 can start the same time as Operation 20.1 and 20.2 since it is a different branch. All upstream operations need to be completed before Operation 40 can start.



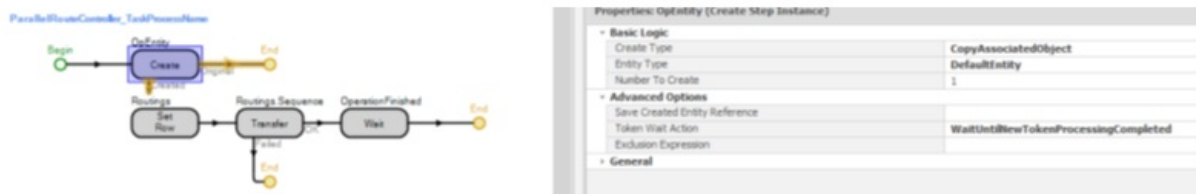
As for the model, a Server object is used as the Parallel Route Controller. The Parallel Route Controller has infinite capacity. It uses a Process Type = Task Sequence and the Processing Tasks are mapped to the Routing table.



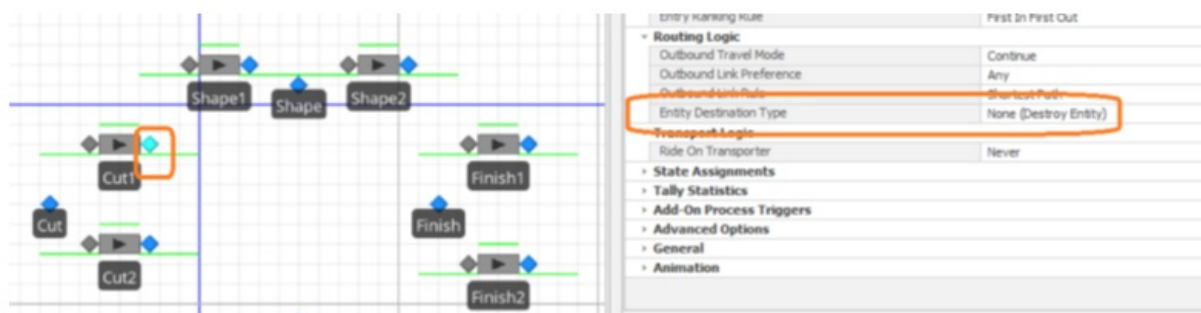
From the Processing Task dialog, the Routing.SequenceNumber is mapped. There is a conditional expression to support WIP. If the operation has already been completed, it will bypass the task (operation). If the operation is not bypassed, the Processing Task calls a "ParallelRouteController_TaskProcessName". This process needs to complete before the operations is determined to be completed.



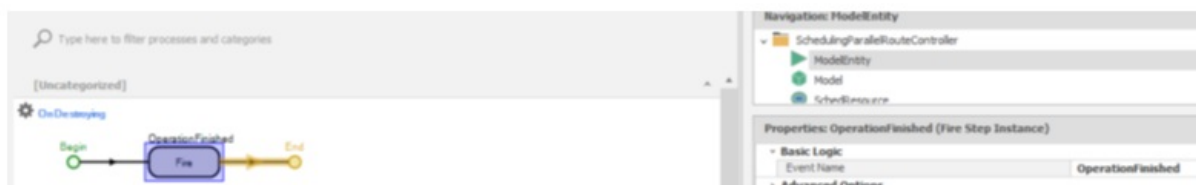
The "ParallelRouteController_TaskProcessName" process is shown below. The process will create an operation entity, set the row of the entity equal to the current Route row, transfer the entity to the Routing.Sequence and then wait until the "ModelEntity.OperationFinished" event is fired. The wait step will ensure that Operation does not complete until the operation entity has been destroyed.



The Entity Destination Type of each output node on the SchedServers in model is set to "None (Destroy Entity)"



The OnDestroying process within the Model Entity will fire its OperationFinished event.



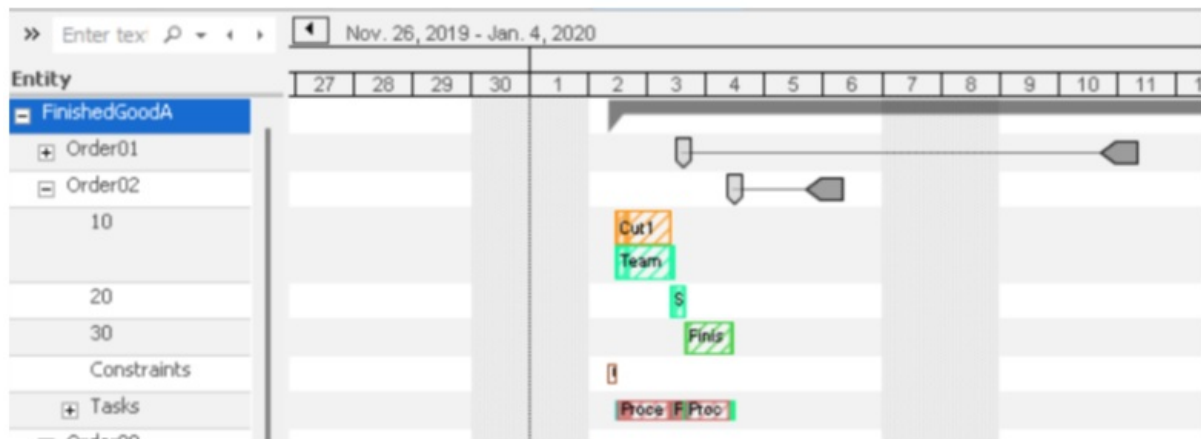
By firing this event, the "ParallelRouteController_TaskProcessName" process for the operation will complete and the ParallelRouteController will process the next Operation or exit the server and move to Ship if there are no more Operations to be processed.

Results From Scheduling

Here are the results of scheduling each type of order. These examples highlighted do not have any work-in-process (WIP) orders.

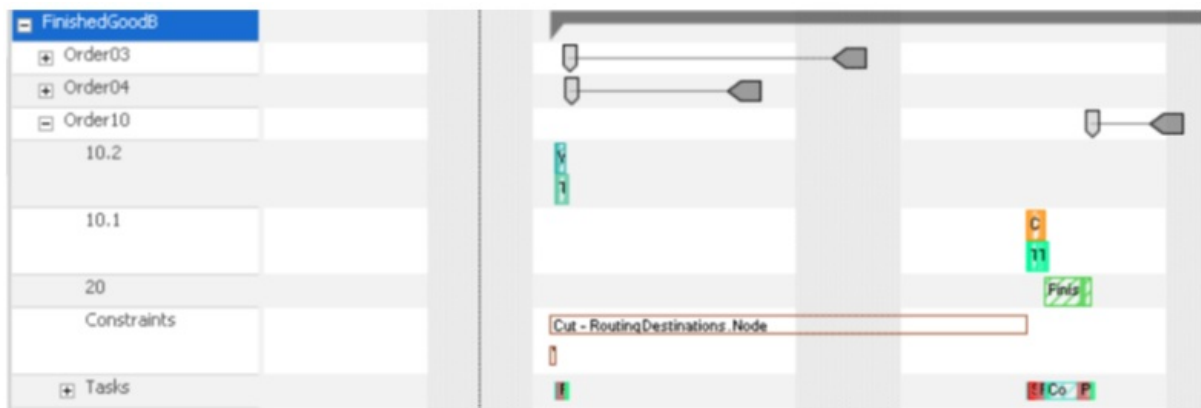
FinishedGoodA

All the operations are being processed in sequence.



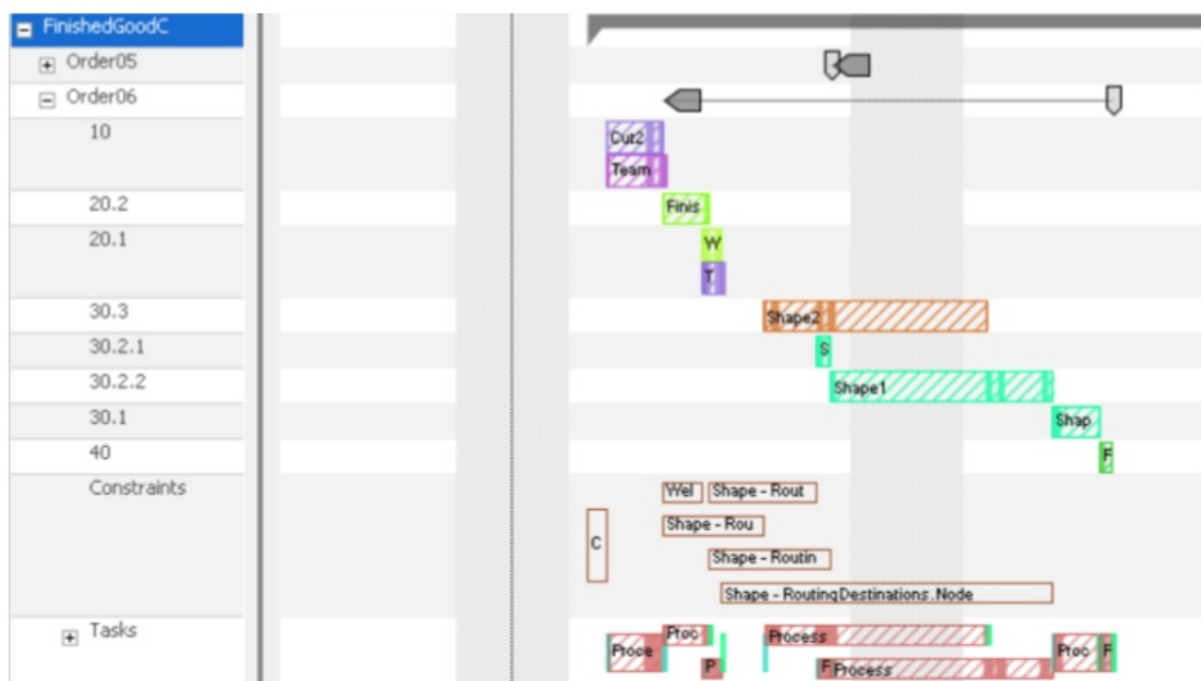
FinishedGoodB

First 2 operations need to be completed before the last operation can start.



FinishedGoodC

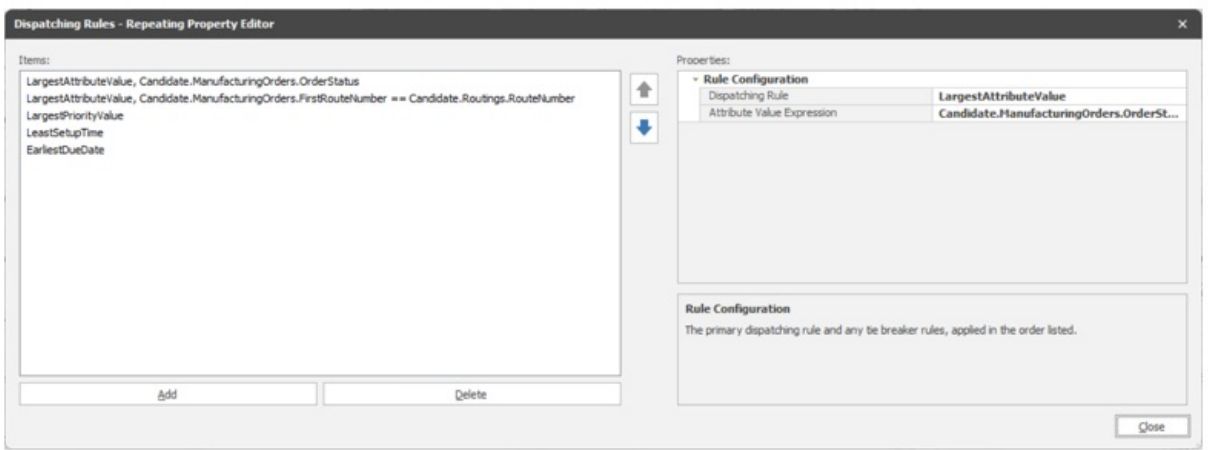
This includes work-in-process (WIP) orders. First operation must be completed before next set of operations can start. All previous operations need to be completed before the last operation can start.



From the Resource Plan Gantt, the schedule has been generated across resources. The WIP operations (yellow) have been schedule first. After the WIP operations, the operations are then schedule by priority, then least setup and lastly by earliest due date.



This schedule sequence is defined using the dispatching rules. The first rule makes sure the WIP ManufacturingOrders are scheduled first. Next, if the "ManufacturingOrder.FirstRouteNumber equals the Routing.RouteNumber" (i.e., the operations have WIP), the operations are scheduled. Third is Largest Priority, Fourth is LeastSetupTime and finally Earliest Due Date.



WarehouseExample - Example

General Description

This model represents a warehouse. The general process flow is as follows:

- Products are generated and processed at the finish machine.
- A forklift picks up the products for storage.
- A second forklift picks up the stored products from the storages to the shipping dock.
- Products are loaded into a truck for shipment.

* This example uses a subclassed server 'MyServer'.

Detailed Description

Initializing the Storage Areas

We initialize the Storage Areas with an Initial WIP. Each storage area has its Initial WIP defined in a table, Table1. To create these entities at the beginning of the run, we add a OnRunInitialized Process in the Process window. The process searches Table1, creating a separate token for each row. These tokens exit through the found branch and create the amount of Product defined in the 'InitialWIP' column. Then we assign a small processing time, signifying this product has already been stored for a period of time. And then we transfer to the storage areas.

Storing the Product

The warehouse contains 12 storage areas. The products are picked up by a forklift after being processed by the Finish machine, which is a Server. The forklift knows where to drop the products by setting PickUp node *Entity Destination Type* to 'Select From List' where the list includes the input nodes of the storages. The drop off is done randomly by setting the *Selection Goal* to 'Random'.

Storage Areas

As the storages share the same processing time, initial capacity and external view, we decide to model these storages as a subclassed server 'MyServer' to avoid repetitive value settings. The External View and the default values for the 'ProcessingTime' and 'InitialCapacity' properties were altered in the MyServer's Definitions tab, Properties panel, so that when placed in the model, it is prepopulated with the correct values and symbol.

Shipping

The products are picked up from the storages by a second forklift that delivers them to the shipping truck. The forklift is triggered through setting the storage area's output node's *Ride on Transporter* property to 'True'. The products are dropped off at a Server with zero processing time where they are loaded onto a conveyor to the shipping station that is modeled as a Sink.

Useful Tips

Note the use of polylines to define the path decorator of the forklifts circuits.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Frequently Asked Questions

- How can I graphically show the utilization of a server or resource?

You may utilize the ResourceState of the server or resource to display the state of the object either in a pie chart format or a status label. To display all the object states within a pie chart, select Status Pie from the animation ribbon, place the pie chart and select *Data Type* as 'ListState'. Select the object name from the *List State* property field pull down, such as 'Server1.ResourceState' or 'Resource1.ResourceState'. To show the percentage of time an object is in a specific state, you may wish to use a status label with the ResourceState within an expression. For example, the function 'ObjectName.ResourceState.PercentTime(StateValue)' will provide the percentage of time that an object is in a given state. The [ListStates](#) page provides the state values (between 1-7) for various objects. For example, you would use 'Server1.ResourceState.PercentTime(1)' in the Expression field of a status label to display the percentage of time that Server1 is processing (which is state value "1").

- How can I graphically show the utilization of dynamic object, such as a worker or vehicle?

Like static objects, you may utilize the ResourceState of the worker or vehicle to display the state of the object either in a pie chart format or a status label. To display all the object states within a pie chart, select Status Pie from the animation ribbon, place the pie chart and select *Data Type* as 'ListState'. Select the object name from the *List State* property field pull down, such as 'Worker1.ResourceState' or 'Vehicle1.ResourceState'. You must also add the specific unit that you would like to have displayed, for example, Worker1[1] or Worker1[2]. Thus, your *List State* should then be 'Worker1[1].ResourceState' to provide the statistics on the first unit of the worker1 objects (as the *Initial Number in System* property may be greater than 1). To show the percentage of time an object is in a specific state, you may wish to use a status label with the ResourceState within an expression. For example, the function 'ObjectName.ResourceState.PercentTime(StateValue)' will provide the percentage of time that an object is in a given state. The [ListStates](#) page provides the state values (between 1-7) for various objects. For example, you would use 'Worker1[1].ResourceState.PercentTime(1)' in the Expression field of a status label to display the percentage of time that Worker1[1] is busy (which is state value "1").

- Are there random number streams within Simio and if so, how do I use them?

The random numbers within a Simio model are generated from independent streams, and you can employ as many streams within your model as you desire. The initialization seeds for each stream are automatically set by Simio. By default all random samples are taken from internal stream number 0, however you can specify a different stream by appending the optional stream parameter for each distribution. For example Random.Uniform(3,5,1) returns a random sample from a uniform distribution between 3 and 5 using random stream number 1. See the [Simulation Replications](#) and [Distributions](#) pages for additional information.

- I built a model with a vehicle, but the vehicle does not move. What happened?

It is most likely that your vehicle has no path to move where it needs to go. For example, for a vehicle or worker to move from the entry point of a server to the exit point of the server, a path needs to be drawn between these two nodes. Also, check the *Initial Node (Home)* property of the vehicle. By default, there is nothing specified and its home location will be the first object that was placed in the model. By specifying a home location for a vehicle, you indicate where you would like the vehicle to be located at the start of the simulation run.

- Does Simio have examples that I can see for various modeling situations?

Yes, we have SimBits which are small examples that demonstrate a particular modeling situation or set of constructs. The [SimBits](#) page categorizes these models based on their functionality.

- I used the Add-On Processes to Create a new entity. However, I see the entity moving up in space when it shouldn't be. What happened?

When you use the Create step to generate a new entity, the entity should be transferred immediately to given a specific location (Transfer step). If the entity is first sent to some type of delay step (Delay, Wait, Execute, etc.), it will be seen at the 0,0,0 XYZ location and start moving at the specified movement speed. You may stop this movement by assigning ModelEntity.Movement.Rate = 0.

- Does Simio's API work with popular database like Access, or SQL?

Simio's API can be used to access databases such as Access, Oracle or SQL. You would create your own custom Step (and possibly an Element) that would connect to your database of choice, perform your selected tasks and then return information to Simio. Our custom Steps and Elements can be written in any .NET code. We provide a couple of Visual Studio templates to get you started writing a custom Step or Element. They can be installed with the tool

that is found under the Advanced folder in the Simio folder found in your Start Menu. See also the [Custom Simio Extensions](#) page for more information.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Modeling in Simio

In Simio, the user should think in terms of the physical objects that make up the system. The users places these objects in the Facility Window and they interact based on their internal logic. The objects that are placed in the Facility Window are user-defined and are stored in libraries. A Simio library contains object definitions that represent the physical components of a system. Simio has a [Standard Object Library](#) that includes 14 pre-built object definitions that can be used to model a wide range of systems.

Simio also provides the user with the ability to create their own object definitions. These can be created from any of the five classes of [Object Types](#). The user defined objects are stored in the [Project Library](#).

Simio allows a user to create their own Design Add-Ons (such as the Source, Server, Sink Add On available in the Project Home tab), User Selection Rules, Steps, Elements, and Data Table importers. For more information on integrating custom code into Simio, see [Custom Simio Extensions](#).

Modeling and [animation](#) are done as a single step in Simio. And Simio provides a number of tools for enhancing the [animation](#) and for importing and downloading [symbols](#) into the model.

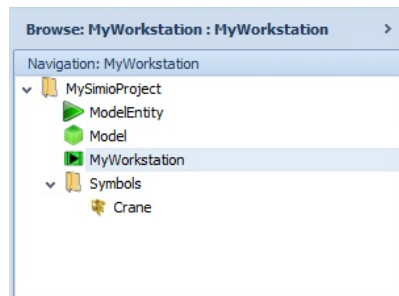
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Navigation

The Navigation window is found in the top right side of the interface, above the Properties window. It is used to navigate between different models within the project, navigate to the [Project window](#), to [Experiments](#), and to project [Symbols](#).

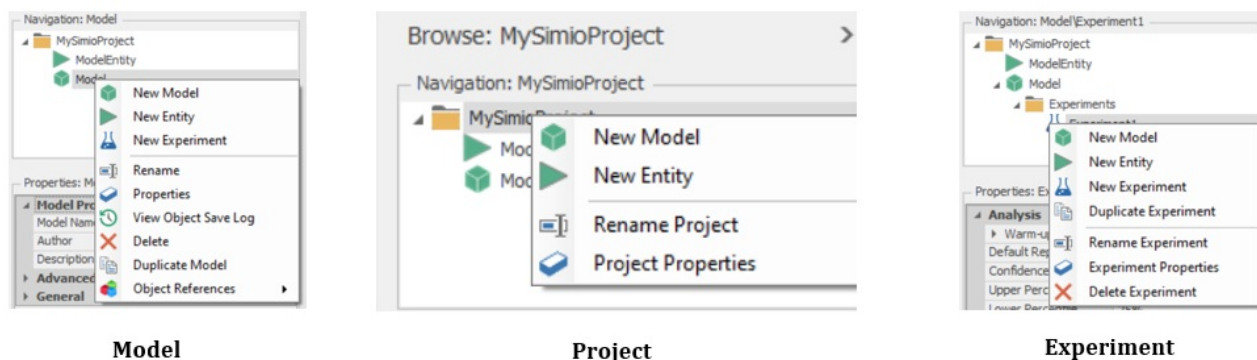
The Navigation Window



Navigating between the different models within the project is done through the Navigation window. When a user clicks on a model in this window, it becomes the active model. Therefore, the model windows in the main part of the interface are now associated with the model that is highlighted in the Navigation window. The type of windows that are available for a particular model depend on the object class of the active model. For example, for a model of object class Fixed, the Facility, Processes, Definitions, Data, and Results windows are shown. However, for a model of object class Entity, only Processes, Definitions, and Data are shown. This is because it does not make sense for an Entity object to have a Facility window or a Results window.

When a Model is selected in the Navigation window, a right click will give the user a choice to add a new fixed object class Model, add a new Entity model, add a new Experiment to this model, rename the selected model, view the model properties, delete the selected model, or duplicate the selected model. There is an additional option to view / modify the object references in a given model, as well as add / remove explicit object references. See [Versioning of Simio Objects](#) for more information on object references. When the Project is selected in the Navigation window, a right click will give the user a choice to add a new fixed object class Model, add a new Entity model, rename the project and view the project properties. When an Experiment is selected in the Navigation window, a right click will give the user a choice to add a new fixed object class Model, add a new Entity model, add a new experiment, duplicate the experiment, rename the experiment, view the experiment properties or delete the experiment.

The Navigation Window - Right Click Menus



Clicking View Object Save Log allows you to view the Saved Date Time, Product Version, Edition, and Save Reason for each object. A new entry will only be added if the Product Version or Edition is different from the last entry.

Within the Navigation window, users have the ability to re-order project items in the window via drag and drop. Moving a model can be done by simply highlighting the model, dragging it to a location in the list and dropping it. A small blue curved "insert here" arrow will appear when moving it to display where it will be located when dropped into a new location. For example, a user may wish to move the ModelEntity model below the various project related models.

Note that the various sections of the 'tree' are collapsible as well. When you click on one of the "v" symbols next to a model that has multiple experiments, for example, it will close the items and then contain a ">" next to it instead of a "v".

Users may re-arrange the order of *sibling* items (nodes) by dragging them around. That is, you can re-order items within the same parent node:

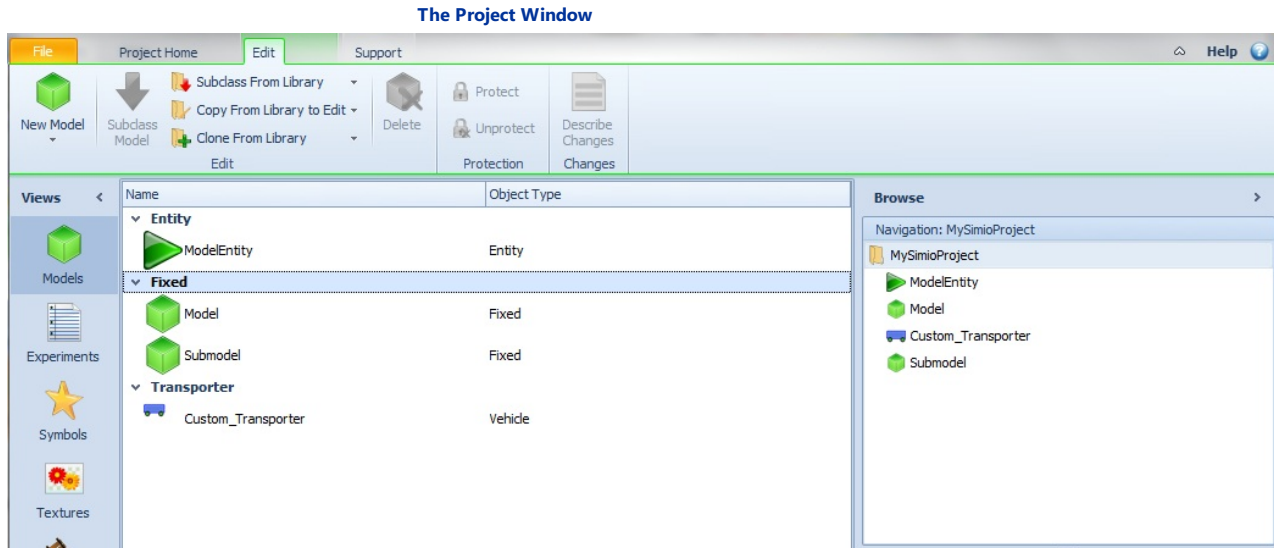
- Top-level models within the project
- Experiments within their parent model (actually, the "Experiments" node for that model)
- Baseline models within their parent model (similarly, the "Baselines" node for that model)
- Symbols within the "Symbols" node

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Project Window

The Project Window is accessed by clicking on the name of the project from within the Navigation window. It contains all of the models, experiments, symbols, textures, and path decorators that a user has in their project.



The Models Panel

The Models panel is selected by default when the Project window is first opened. All of the models that are contained within the project are displayed in this window and organized by their class type. To view the properties of each model, simply select the appropriate model and its properties are shown in the properties window in the bottom right of the interface. When the model is selected, a right click will allow the user to either delete the model or edit the model. If the user selects Edit, they are navigated away from the project window and the Facility window of that particular model is now displayed, ready for editing.

A model can be password [protected](#) by selecting the Protect icon from the ribbon when the model is highlighted.

New models can be added to a project in a few different ways. One option is to add a new model that does not contain any logic so that the user can customize it from scratch. This is done by selecting the appropriate object class type from the **New Model** icon in the Project Home ribbon. Another option for creating a new model is to subclass a model that exists within the current project. This is done by first selecting the project name (MySimioProject by default). Within the project view, highlight the model to subclass and select **Subclass Model** from the Edit ribbon. A third option is to subclass a model that exists in an attached library. There might be models that exist in the Standard Library that the user would like to subclass or models that exist in a library that the user loaded into this project. Select the **Subclass From Library** icon in the ribbon to subclass a model from an attached library. Note that the Subclass option is also available when in the Facility window by right clicking on the library object itself in the Libraries panel. Right clicking on an existing object within the Facility window also provides the option to **Create Object From This** which will create a subclassed object with default data. And finally, a user can simply copy a model into this project from an attached library by selecting the **Copy From Library** icon in the ribbon. This simply copies the model into this project and therefore the user cannot override the inherent logic of the model. See [Creating New Objects](#) for additional information on how to create a new model.

The Experiment Panel

This is where a user can add new experiments to their models, view the properties of an experiment, and rename, edit, duplicate and delete experiments. A new experiment is added to a model by selecting the appropriate model from the Add Experiment drop down of the ribbon. Notice that an experiment is at the model level and not at the project level. A model can contain more than one experiment. The properties of an experiment can be viewed in the Properties window by selecting the appropriate experiment. The experiment can be edited with a double click or a right click. An experiment can also be duplicated using the Duplicate Experiment option. When an experiment is duplicated, the data within the Controls section and the Results are duplicated. See the [Experiment](#) page for additional information on creating and running experiments.

The Symbols Panel

A user can view, rename, edit and delete the symbols that exist within the project from the Symbols panel. New symbols can also be imported, downloaded and created from this panel. Symbols exist at the project level and can be used in all the models that exist within the project. See the [Symbols](#) page for additional information on Symbols.

The Textures Panel

A user can view, rename and delete textures that exist within the project from the Textures panel. New textures can also be added to the project from this panel by selecting the New Texture icon in the ribbon. Textures exist at the project level and can be used in all the models that exist within the project.

Path Decorators Panel

Path Decorators can be viewed, renamed and deleted from this panel. A Path Decorator is added to a link from within the Facility Window, see the [Animating Links](#) page for more details.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Model Windows

Windows

The following windows can be accessed using the Model Window tabs.

- [The Facility Window](#)
- [The Processes Window](#)
- [The Definitions Window](#)
- [The Data Window](#)
- [The Results Window](#)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

The Facility Window

The Facility Window

The Facility Window defines the model in terms of animated objects. Objects can be placed into the Facility Window by either dragging them from the [Standard Library](#) or from the [Project Library](#).

Add an object to your model by dragging it from either the Standard Library (e.g. Source, Server, Sink) or from the Project Library (e.g. MyServer, MySubmodel). Add multiple objects for a model by double clicking on the model in the library, which places you in add-repeat mode. When in this mode each left click places a new object. You exit this mode by right-clicking or pressing the Esc key. Add links (e.g. Path, Conveyor) between nodes by either clicking on the link in the library and then the click on the starting node which places you in link drawing mode. Add link vertices by clicking at each location, and finish link drawing mode by clicking on the ending node. You can also enter link drawing mode by double clicking on the starting node.

The Facility Window is where the appearance of an object can be changed for the purposes of animation. The Symbols tab in the ribbon menu provides options for changing the appearance of symbols, [adding symbols](#) or [importing symbols](#). The [Drawing tab](#) allows the user to add static symbols, lines, rectangles, ellipses, etc. to the Facility Window. And the [View tab](#) allows the user to change the view of the Facility window from 2D to 3D. It also provides other options for changing how the user can view the Facility Window. The [Run tab](#) is where the user can start an interactive run of the model, add breakpoints, begin the trace, adjust the animation speed and specify the starting and ending times.

Status Labels, Status Plots, Status Pies, Floor Labels, Circular Gauges, Linear Gauges, Stacked Bars and Buttons can also be placed in the Facility window from the Animation tab. They can also be attached to objects in the Facility window. If they are attached to Entities or Transporters, they will travel with those dynamic instances. To learn more about these displays, see the [Console](#) page.

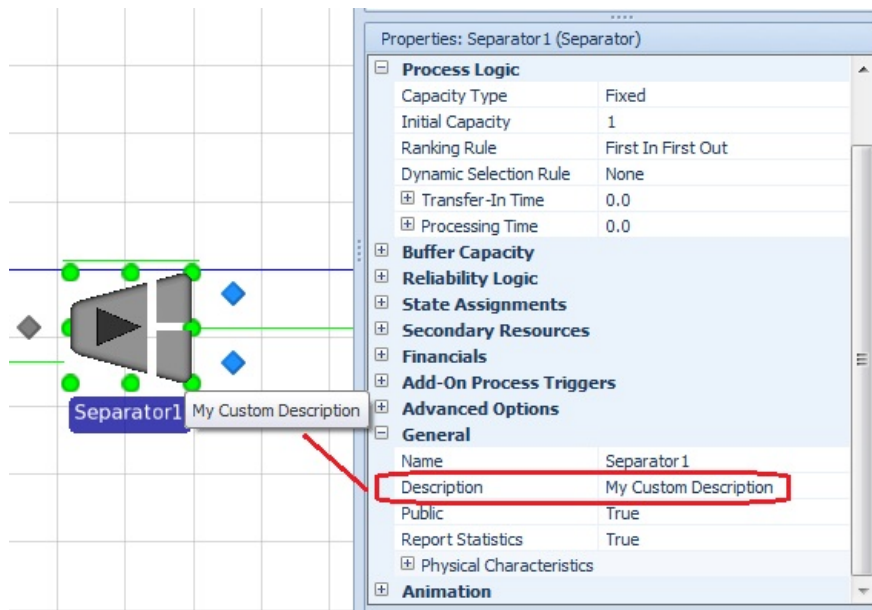
Each object (excluding Links) has a property called *Location*, which indicates where this object is located in the Facility Window in relation to the grid point (0,0). The *Location* property is the center point of a node and the bottom center of a fixed object. For example, if the value of Y is '0', the object is sitting on the floor.

The Grids, Labels, Axes, Arrows, Nodes and Queues can be visible or hidden in the Facility Window. This is controlled on the View tab. Additional information on changing the view can be found [here](#).

Multi-Select of objects for the purpose of moving, copying and/or deleting the objects is supported in the Facility window. Multi-select of objects is also useful for editing similar properties of objects in the Properties window. Objects from the Standard or Project libraries, as well as items from the Draw and Animation ribbon tabs, can be cut /copied / pasted within the Facility Window using Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste). Ctrl-D (Delete) is currently supported only within the Facility window. Please refer to the [User Interface](#) for more details.

The text of an object's *Description* property will appear within the Facility window when the mouse is hovered over the object, as shown in the image below.

Description Text on an Object



Model Level Properties

Listed below are the properties of a **Model**:

Property	Description
Model Name	The name of this model object. This is what is shown in the Navigation window and what is used if this model object is placed within a different model.
Author	The author of this object.
Description	Description text for this instance. The description text will be displayed in the Project window when the object model is to be selected for placement into a different model.
Object Type	The type of this object model as Fixed, Node, Link, Agent, Entity or Transporter.
Parent Class	The name of the parent class from which this object is subclassed.
Keywords	List of keywords, separated by commas, for this object definition.
Categories	List of categories, separated by commas, for this object definition.
Resource Object	This property indicates whether an object of this definition type is a resource object with capacity that can be seized and released.
Runnable	This property indicates whether an object can be run or not. For example, an object model defining a vehicle would not be runnable, while a model containing Source, Servers, Sink would be runnable.
Library Visibility	This property indicates if this object is shown when the project is attached as a library. 'Always' indicates the object will always be shown. 'Never' indicates the object will never be shown. 'Deprecated' indicates the object will not be shown unless the active model uses this object, or the user has explicitly enabled "Show Deprecated Objects" in the library view.
Table Imports are Undoable	If 'True', an explicit import into a data-bound table will be undoable. Set to 'False' if memory usage by the records of previous rows being held in memory for undo-ability becomes a concern.
Load Action	This property is visible when the <i>Runnable</i> property is 'True'. It indicates if the model should automatically enter a run mode immediately after loading. Options include None, Reset, Step, Run and FastForward.

Update Interval	Provides a model-level value that may be used in models or custom objects to trigger periodic user-specified actions. This value, while not used directly by Simio, is available at run-time via <code>Run.UpdateInterval</code> .
Units	Time units used for the model's <i>Update Interval</i> .
Check Base for Advanced Properties	Indicates if Simio should include this object's inherited properties when checking if it has any Advanced properties.
Exhaustive Constraint Checking	This options indicates whether constraints defined using Constraint Logic elements are exhaustively checked after any relevant state change in the system. Enabling this option will provide the most accurate detail in Trace and the Constraint Log, but may result in significantly longer run times.
Available in Express Edition (RPS Only)	If True, this object can be used to build models in the Express Edition of Simio.
Freeze Period Hours (RPS Only)	The value that will be returned by the <code>Run.FreezePeriod</code> function.
Concurrent Replication Limit (Risk Analysis - RPS Only)	Specifies the maximum number of replications that will be run simulataneously. The default value of 0 allows Simio to determine this value.
Parse Element Functions In Expressions (Compatability)	If 'True', element functions will be the last thing searched to get a match to an identifier in an expression. If False, element functions will be searched before states, expression functions, child elements and objects, external nodes, and schedules. If this value is changed, the model must be saved and reloaded in order for it to take effect across all expressions. This was implicitly 'False' in models built in Sprint 106 and prior sprints.
Parse Non-Entity Associated Objects for Property Reference (Compatability)	If 'True', then non-entity types are allowed for the <code>ObjectType.PropertyName</code> property reference syntax for all properties. The recommended value for this is 'True'. If this value is changed, the model must be saved and reloaded in order for it to take effect. This was implicitly 'False' in models built in Sprint 166 and prior sprints.
Allow non-standard element/object names (Compatability - RPS Only)	Simio RPS Edition allows the creation of elements and objects with non-standard names. While standard names contain only letters, digits and underscores, non-standard names can contain ANY characters. RPS Edition enables the entry of non-standard names into table columns that are set to automatically create element or object instances. This setting determines if the user may additionally enter non-standard names interactively, in this model's Facility or Elements windows.
Allow Trace (Protection - Professional / RPS Only)	If False, does not allow trace to be written for the entire model.
Allow Profile (Protection - Professional / RPS Only)	If False, does not allow runtime profiling for the entire model.

Allow Watch If False, disables watch window functionality for the model.
(Protection -
Professional /
RPS Only)

****NOTE**** - With Protection properties, if the functionality is disabled, the user is notified, as shown below. Additionally, if the model is password protected, users cannot set the values for the Protection properties, even if they have the proper license.

Watch		
Name		
Watch functionality has been disabled on the model.		
Profile - Profiling functionality has been disabled for the model	Trace - Trace functionality has been disabled for the model	Watch

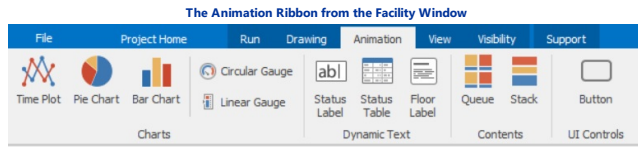
Note: Within our SimBits, we utilize the Description, Keywords and Categories properties to separate the various files, as seen when using the Support ribbon, Sample SimBit Solutions button that allows users to search the various SimBits files. Users may add models to the default SimBit directory (documents/Simio/SimBits) and any *.spfx model files will be indexed and categorized based on the Category and Keyword properties listed. Categories are used in both the Basic and Advanced Search, while Keywords are used in the Advanced Search. The Description text property will be displayed below the specific Category and is used for searching specific terms as well.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Status Labels, Plots, Gauges and Buttons

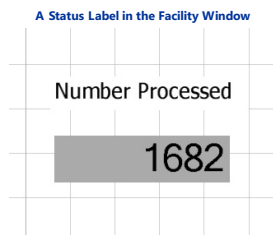
Status Labels, Status Plots, Status Pies, Floor Labels, Circular Gauges, Linear Gauges, Stacked Bars and Buttons can be placed in the [Facility Window](#). If an animation status object is attached to Entities or Transporters within the Facility window, it will travel with the dynamic instances. Many of the same animation items can be placed within the Definitions tab, [External Window](#) and [Console Window](#).



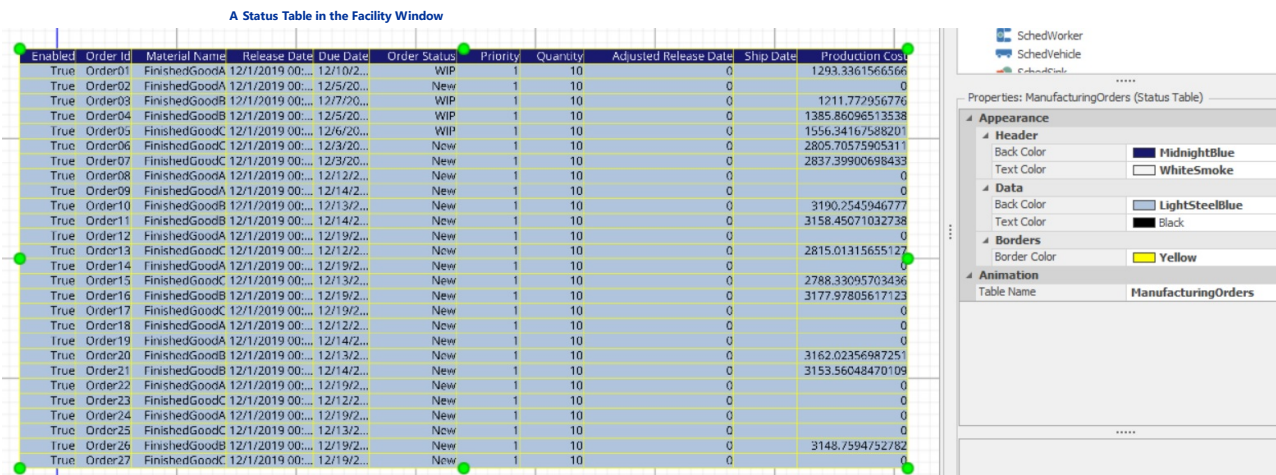
More information regarding the various object queues and animating queues can be found on the [Queues](#) page.

The user can add a **Status Label** for displaying static text or the value of an expression inside a text display rectangle. After selecting Status Label left click in the window to place the first vertex, and then drag and left click again to add the opposite vertex.

The following is an example of first using the Status Label for text and then using it to display an expression. In this case the expression shown is `Server1.Processing.NumberExited`.



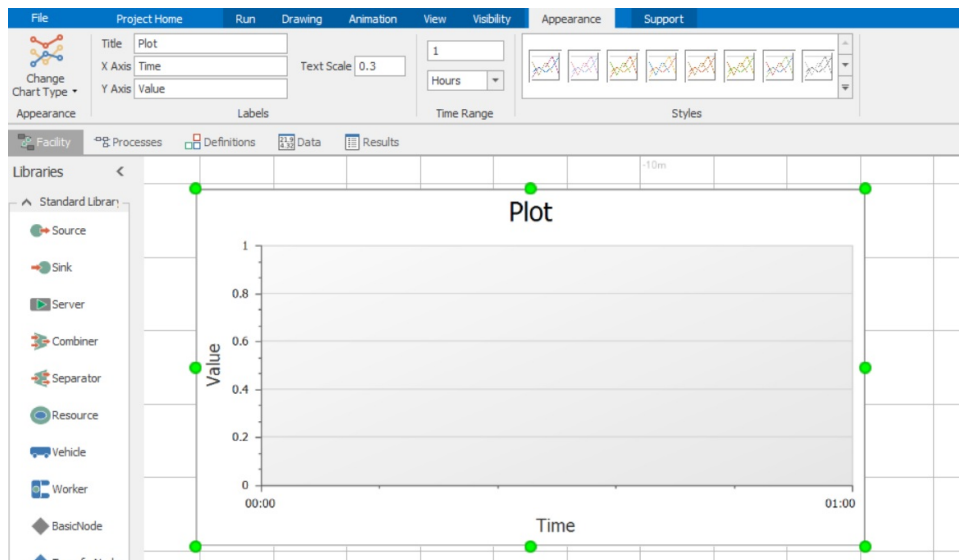
The **Status Table** adds a status table for displaying static text or the current values of expressions that have been defined in a separate specified source table. Click this button to enter drawing mode. Then position the cursor at the place in the Facility window where you want the top-left corner of the status table. Then left click, drag, and left click again to draw the status table to a desired size. Then use the Properties window to specify the name of the source table.



Note: A Status Table only displays the first 50 rows of data and does not display Output Tables. You can filter which table rows are displayed at a given time by using the Status Table's *Filter Expression*.

Time Plot

A Time Plot displays a specified value on the Y-Axis against time on the X-Axis.



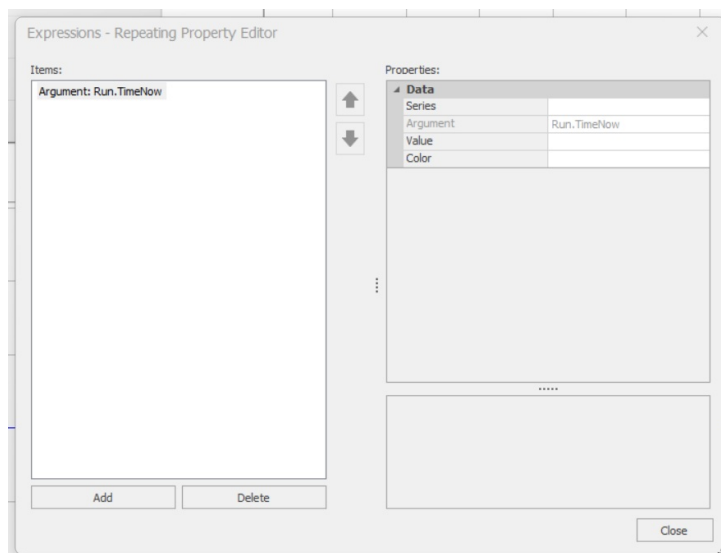
You can specify different Chart Types via the Change Chart Type dropdown. These options are: Point, Line, StackedLine, FullStackedLine, StepLine, Area, StackedArea, FullStackedArea, StepArea, StackedStepArea, FullStackedStepArea, or Legacy.

You can set the Title, X Axis, Y Axis and Text Scale in the Labels section. Time Range will display the given range in the Time Plot, once the range has been met, the Time Plot will adjust. Styles will choose some preselected color combinations for the Plot.

Time Plots can have a Data Source of either 'Expressions', a ListState, a Data Table, or an Output Table.

Expressions

Using a Data Source of 'Expressions' will give a Repeating Property Editor where you can set the Series, Value, and Color. Note: Argument will always be set to 'Run.TimeNow' for Time Plots.



Series - A Series represents a grouping of the related data points which are plotted on a chart. Each data point is plotted on the chart, based on its X value (the point's Argument) and one or more Y values (the Values(s) of the data point). Series data can be represented in different visual manners based on the cart type (Bar, Pie, Time Plot). The Series appears in the Legend for each chart type. For Bar Charts, the Series is represented by the location within the grouping of bars. For Pie Charts, the Series is represented by the individual pies per Series. For Time Plots, the Series is represented by the individual lines or areas.

Argument - The Argument is the data point's argument value. For Bar Charts, the Argument is represented by the grouping of bars on the X axis. For Pie Charts, the Argument is represented by slices of the pie. For Time Plots, the Argument is represented by the point in time on the X axis.

Value - The Value is the data point's data value. For Bar Charts, the Value is represented by the height of the bar. For Pie charts, the Value is represented by the percentage of the slice. For Time Plots, the Value is represented by the value of the Y axis.

Color - Allows you to set the color of your choice when drawing chart data. For Bar Charts, it is the color of the bar, for Pie Charts, it is the color of the slice, and for Time Plots, it is the color of the line/area.

Data Tables/Output Tables

Using a Data Source of TableName/OutputTableName will give four properties (Series Data Field, Argument Data Field, Value Data Field, and Color Data Field) that can be set up to reference columns from a table.

Series Data Field - A Series represents a grouping of the related data points which are plotted on a chart. Each data point is plotted on the chart, based on its X value (the point's Argument) and one or more Y values (the Values(s) of the data point). Series data can be represented in different visual manners based on the cart type (Bar, Pie, Time Plot). The Series appears in the Legend for each chart type. For Bar Charts, the Series is represented by the location within the grouping of bars. For Pie Charts, the Series is represented by the individual pies per Series. For Time Plots, the Series is represented by the individual lines or areas.

Argument Data Field - The Argument is the data point's argument value. For Bar Charts, the Argument is represented by the grouping of bars on the X axis. For Pie Charts, the Argument is represented by slices of the pie. For Time Plots, the Argument is represented by the point in time on the X axis.

Value Data Field - The Value is the data point's data value. For Bar Charts, the Value is represented by the height of the bar. For Pie charts, the Value is represented by the percentage of the slice. For Time Plots, the Value is represented by the value of the Y axis.

Color Data Field - Allows you to set the color of your choice when drawing chart data. For Bar Charts, it is the color of the bar, for Pie Charts, it is the color of the slice, and for Time Plots, it is the color of the line/area.

Summary Function - Summary Function used to summarize the chart data.

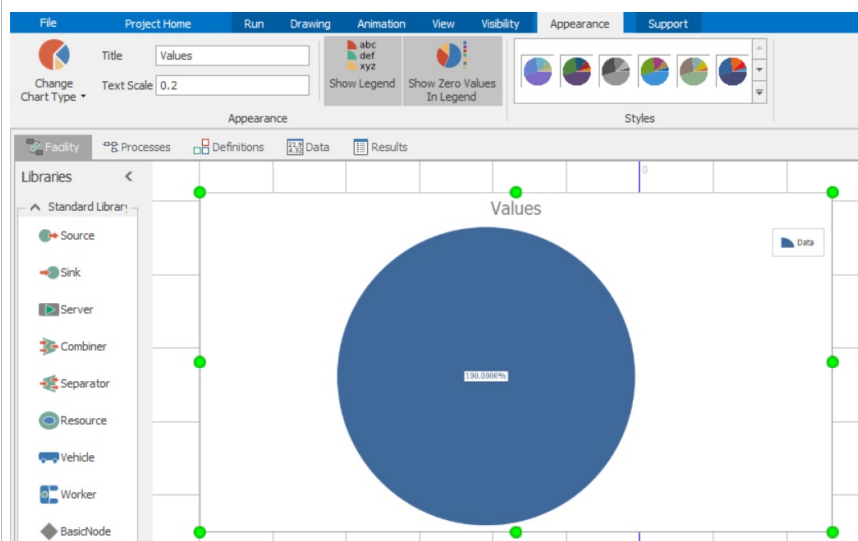
Aggregation Interval - Interval used to aggregate the chart data. When *Aggregation Interval* is set, the *Summary Function* value defaults to Average if the value is set to None.

List State

Using a *Data Source* of ListState will resulting in a datasource where the Arguments are the List State values, and Values are the Percent Time in each value. Note: Percent Time in a list state value will be 'reset' if statistics are cleared during the run.

Pie Chart

A Pie Chart displays Values of Arguments as a percentages with respect to each other in sets of Series.



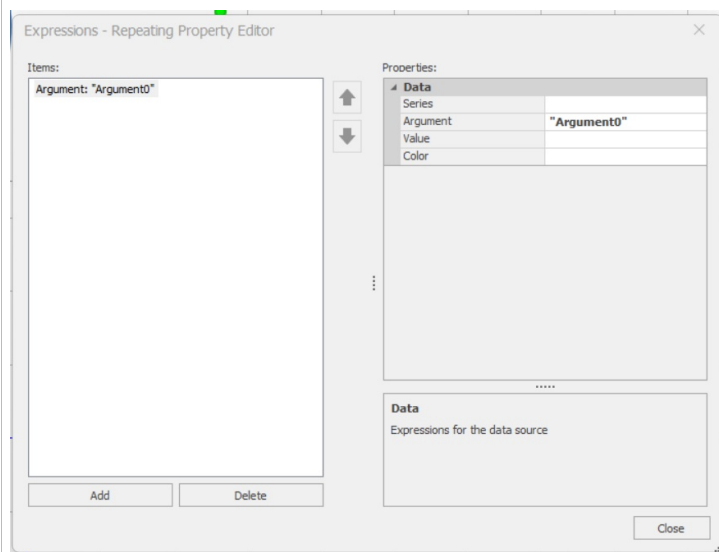
You can specify different Chart Types via the Change Chart Type dropdown. These options are: Pie, Donut, or NestedDonut.

You can set the Title and Text Scale in the Appearance section. You can also toggle the Legend options on and off here. Styles will choose some preselected color combinations for the Pie.

Pie Charts can have a *Data Source* of either 'Expressions', a ListState, a Data Table, or an Output Table.

Expressions

Using a *Data Source* of 'Expressions' will give a Repeating Property Editor where you can set the *Series*, *Argument*, *Value*, and *Color*.



Series - A Series represents a grouping of the related data points which are plotted on a chart. Each data point is plotted on the chart, based on its X value (the point's Argument) and one or more Y values (the Values(s) of the data point). Series data can be represented in different visual manners based on the cart type (Bar, Pie, Time Plot). The Series appears in the Legend for each chart type. For Bar Charts, the Series is represented by the location within the grouping of bars. For Pie Charts, the Series is represented by the individual pies per Series. For Time Plots, the Series is represented by the individual lines or areas.

Argument - The Argument is the data point's argument value. For Bar Charts, the Argument is represented by the grouping of bars on the X axis. For Pie Charts, the Argument is represented by slices of the pie. For Time Plots, the Argument is represented by the point in time on the X axis.

Value - The Value is the data point's data value. For Bar Charts, the Value is represented by the height of the bar. For Pie charts, the Value is represented by the percentage of the slice. For Time Plots, the Value is represented by the value of the Y axis.

Color - Allows you to set the color of your choice when drawing chart data. For Bar Charts, it is the color of the bar, for Pie Charts, it is the color of the slice, and for Time Plots, it is the color of the line/area.

Data Tables/Output Tables

Using a *Data Source* of TableName/OutputTableName will give four properties (*Series Data Field*, *Argument Data Field*, *Value Data Field*, and *Color Data Field*) that can be set up to reference columns from a table.

Series Data Field - A Series represents a grouping of the related data points which are plotted on a chart. Each data point is plotted on the chart, based on its X value (the point's Argument) and one or more Y values (the Values(s) of the data point). Series data can be represented in different visual manners based on the cart type (Bar, Pie, Time Plot). The Series appears in the Legend for each chart type. For Bar Charts, the Series is represented by the location within the grouping of bars. For Pie Charts, the Series is represented by the individual pies per Series. For Time Plots, the Series is represented by the individual lines or areas.

Argument Data Field - The Argument is the data point's argument value. For Bar Charts, the Argument is represented by the

grouping of bars on the X axis. For Pie Charts, the Argument is represented by slices of the pie. For Time Plots, the Argument is represented by the point in time on the X axis.

Value Data Field - The Value is the data point's data value. For Bar Charts, the Value is represented by the height of the bar. For Pie charts, the Value is represented by the percentage of the slice. For Time Plots, the Value is represented by the value of the Y axis.

Color Data Field - Allows you to set the color of your choice when drawing chart data. For Bar Charts, it is the color of the bar, for Pie Charts, it is the color of the slice, and for Time Plots, it is the color of the line/area.

Summary Function - Summary Function used to summarize the chart data.

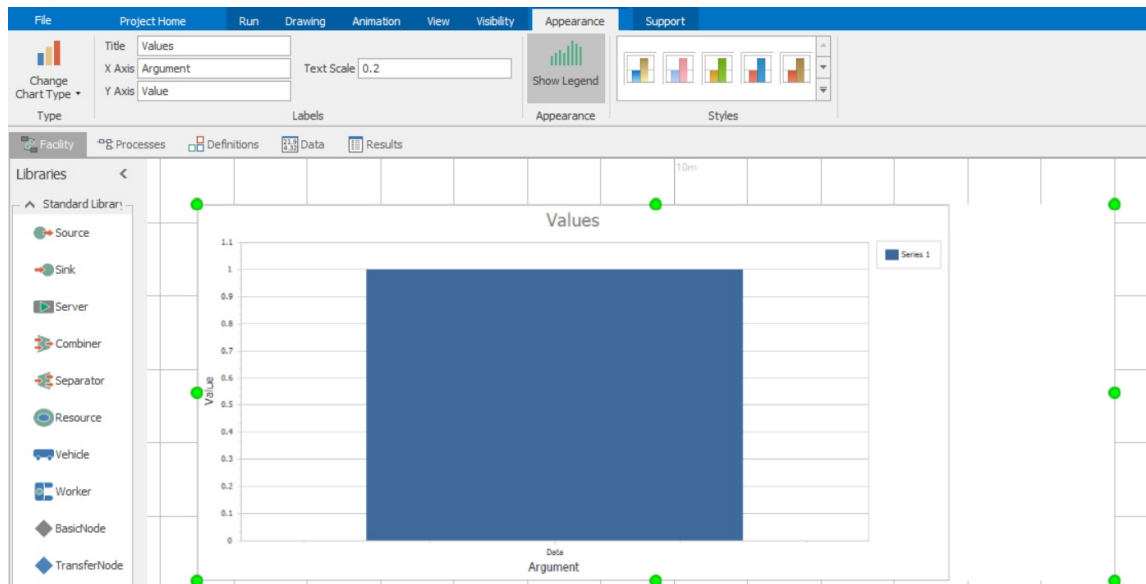
Aggregation Interval - Interval used to aggregate the chart data. When **Aggregation Interval** is set, the **Summary Function** value defaults to Average if the value is set to None.

List State

Using a **Data Source** of ListState will result in a datasource where the Arguments are the List State values, and Values are the Percent Time in each value. Note: Percent Time in a list state value will be 'reset' if statistics are cleared during the run.

Bar Chart

A Bar Chart displays Values of a Series against each Argument.



You can specify different Chart Types via the Change Chart Type dropdown. These options are: Bar, StackedBar, FullStackedBar.

You can set the Title, X Axis, Y Axis, and Text Scale in the Labels section. The Legend can be toggled on and off in the Appearance section. Styles will choose some preselected color combinations for the Pie.

Bar Charts can have a **Data Source** of either 'Expressions', a ListState, a Data Table, or an Output Table.

Note that when using an Output Table as the **Data Source** with a Bar Chart, bars with the same **Argument** will overlap in the default Bar Chart type. For a Stacked Bar type, bars with the same **Argument** are stacked and will show the last recorded value in the Output Table.

Expressions

Using a **Data Source** of 'Expressions' will give a Repeating Property Editor where you can set the **Series**, **Argument**, **Value**, and **Color**.

Series - A Series represents a grouping of the related data points which are plotted on a chart. Each data point is plotted on the chart, based on its X value (the point's Argument) and one or more Y values (the Values(s) of the data point). Series data can be represented in different visual manners based on the cart type (Bar, Pie, Time Plot). The Series appears in the Legend for each chart type. For Bar Charts, the Series is represented by the location within the grouping of bars. For Pie Charts, the Series is represented by the individual pies per Series. For Time Plots, the Series is represented by the individual lines or areas.

Argument - The Argument is the data point's argument value. For Bar Charts, the Argument is represented by the grouping of bars on the X axis. For Pie Charts, the Argument is represented by slices of the pie. For Time Plots, the Argument is represented by the point in time on the X axis.

Value - The Value is the data point's data value. For Bar Charts, the Value is represented by the height of the bar. For Pie charts, the Value is represented by the percentage of the slice. For Time Plots, the Value is represented by the value of the Y

axis.

Color - Allows you to set the color of your choice when drawing chart data. For Bar Charts, it is the color of the bar, for Pie Charts, it is the color of the slice, and for Time Plots, it is the color of the line/area.

Data Tables/Output Tables

Using a *Data Source* of *TableName/OutputTableName* will give four properties (*Series Data Field*, *Argument Data Field*, *Value Data Field*, and *Color Data Field*) that can be set up to reference columns from a table.

Series Data Field - A Series represents a grouping of the related data points which are plotted on a chart. Each data point is plotted on the chart, based on its X value (the point's Argument) and one or more Y values (the Value(s) of the data point). Series data can be represented in different visual manners based on the chart type (Bar, Pie, Time Plot). The Series appears in the Legend for each chart type. For Bar Charts, the Series is represented by the location within the grouping of bars. For Pie Charts, the Series is represented by the individual pies per Series. For Time Plots, the Series is represented by the individual lines or areas.

Argument Data Field - The Argument is the data point's argument value. For Bar Charts, the Argument is represented by the grouping of bars on the X axis. For Pie Charts, the Argument is represented by slices of the pie. For Time Plots, the Argument is represented by the point in time on the X axis.

Value Data Field - The Value is the data point's data value. For Bar Charts, the Value is represented by the height of the bar. For Pie charts, the Value is represented by the percentage of the slice. For Time Plots, the Value is represented by the value of the Y axis.

Color Data Field - Allows you to set the color of your choice when drawing chart data. For Bar Charts, it is the color of the bar, for Pie Charts, it is the color of the slice, and for Time Plots, it is the color of the line/area.

Summary Function - Summary Function used to summarize the chart data.

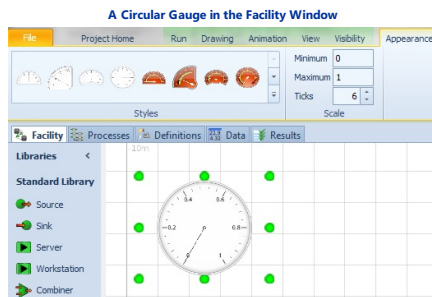
Aggregation Interval - Interval used to aggregate the chart data. When *Aggregation Interval* is set, the *Summary Function* value defaults to Average if the value is set to None.

List State

Using a *Data Source* of *ListState* will result in a datasource where the Arguments are the List State values, and Values are the Percent Time in each value. Note: Percent Time in a list state value will be 'reset' if statistics are cleared during the run.

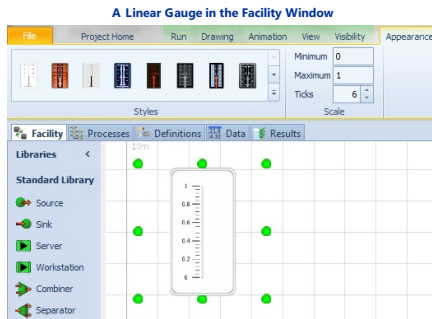
The **Circular Gauge** will display the value of an expression in a the style of a circular gauge. The Appearance ribbon (as seen above the circular gauge shown below) allows the user to change the Style and Scale of the gauge.

The following is an example of a circular gauge displaying the value of `Server1.Capacity.Allocated.Average`.



The **Linear Gauge** will display the value of an expression in a the style of a linear gauge. The Appearance ribbon (as seen above the linear gauge shown below) allows the user to change the Style and Scale of the gauge.

The following is an example of a linear gauge displaying the value of `Server1.NumberWaitingAllocation`.



The **Stacked Bar** adds a status stacked bar for displaying information in the simulation. After selecting the Stacked Bar, left click in the window to place the first vertex, then drag and left click again to add the opposite vertex.

The Stacked Bar can be rectangular or ellipse shaped. The scale type of the Stacked Bar can be specified as either Volume or Linear. If Volume is selected, the value of the expression will be used for the volume of each block. Alternatively, if Linear is selected, the value of the expression will be used for the height of each block. The large portion of this animation symbol is best viewed in 3D, as the expressions or queue contents are displayed on top of each other. The Floor Display options of Left and Right, however, allow the contents to be shown in 2D within a gauge type rectangle to the left or right of the main stacked bar.

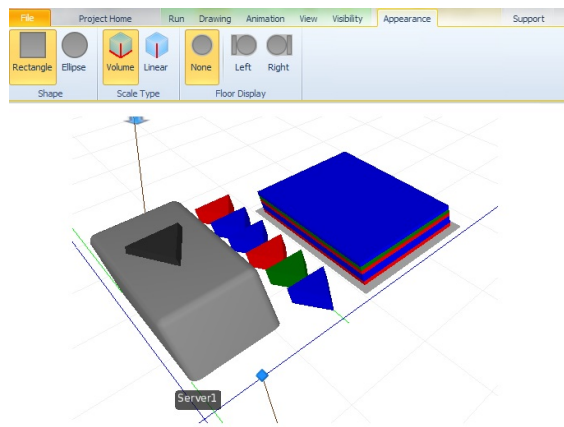
The types of stacked bars include Container, Queue, Expressions, StorageBin, and StorageArea. If the Type is Container, then simply indicate the Container element to animate, and the stacked bar will display Container contents by Volume. To visualize a Container by weight, the Queue Type stacked bar can be used. If the Container referenced has a non-infinite weight capacity, but an infinite volume capacity, the stacked bar will display Container contents by Weight. If the Type is Queue, then the options include Queue State, Queue Item Expression and Total Expression are available. Total Expression allows a user to specify a total space available. If Expressions is selected, then a repeat group of Expression Values is displayed, where users can specify an Expression and a Color. Like Queue, a Total Expression can also be specified.

If the Type is StorageBin or StorageArea, the Stack Bar will animate the contents of the specified storage bin or storage area. The color of each bar is defined by the Display Color property of the corresponding material, and the size of each bar is defined by the total volume of the material contained within the storage bin or storage area.

Any texture can be applied to a stacked bar (the middle blue part), and that material will be the one used to display the total volume available.

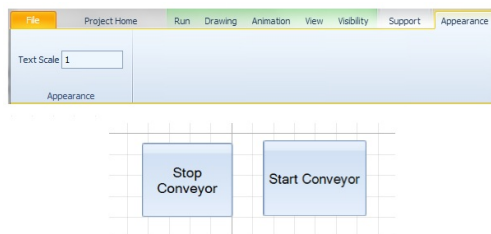
The following is an example of a stacked bar displaying the Server's processing contents queue, `Server1.Processing.Contents`, in terms of the `ModelEntity.Size`.

A Stacked Bar in the Facility Window



Adding a **Button** will provide the user the opportunity for firing an event within the model while it is running. After selecting Button left click in the window to place the first vertex, and then drag and left click again to add the opposite vertex. The text size of the button can be adjusted by specifying the *Text Scale* property for the Button on the Appearance ribbon. A button can be used during the run to trigger a Process, create a new Entity at a Source, etc. During a run, if the focus is not on the Facility Window when the user wants to click the button, the user will need to first click somewhere in the Facility Window to return focus to this window and then click on the button to fire the Event.

Buttons within the Facility Window



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

The Processes Window

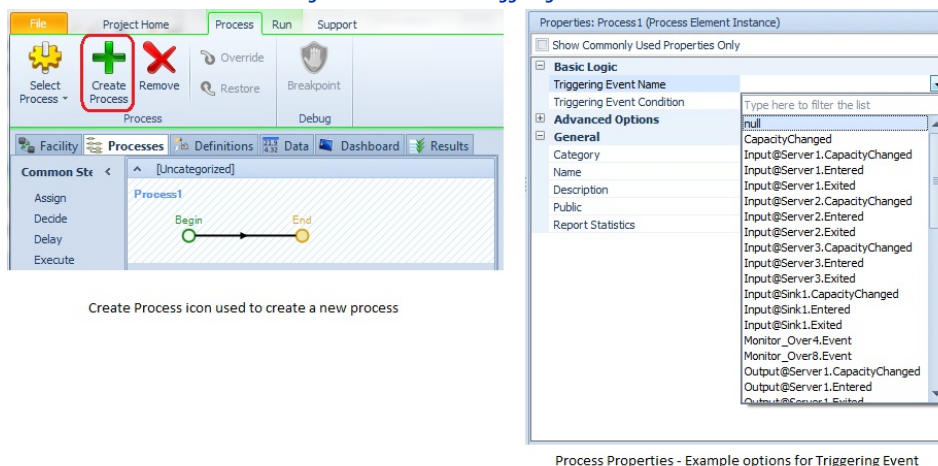
The Processes Window

Adding Processes and Steps

The Processes Window defines the model logic in terms of traditional process flow comprised of a sequence of steps that are executed by **tokens** and change the state of **elements**. Process logic is typically used for creating new models or for adding custom logic to the objects in your model through "add-on" processes.

The user can create a new process with the Create Process icon in the ribbon menu. The user can specify a triggering event (in the Process Properties Window) for these new processes. Whenever the specified event is fired (and given that the triggering event condition is met), this process is executed. Examples of an **Event** that might be used are Server1.CapacityChanged, Timer1.Event (which might be a user created Timer element), or Event1 (which might be a user defined Event that is fired by a **Fire Step**). Besides using a triggering event to execute a new process, another way for this process to be called is from an **Execute Step** located somewhere else in the project. Yet another way to call a process is to specify it as an add-on process in a Standard Library object. In either of these latter cases, you do not need to specify a triggering event.

Creating a New Process with a Triggering Event

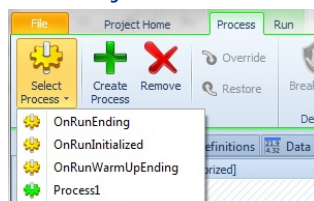


Create Process icon used to create a new process

Process Properties - Example options for Triggering Event

A user can also add one of the standard processes that are automatically executed by Simio by selecting one from the drop down under the Select Process icon. There is no need to define a triggering event for these processes because Simio executes them automatically. For example, the OnRunInitialized process is executed when this object/model is first initialized. The OnCapacityReleased is executed as soon as the capacity of this object/model has been released. The capacity related standard processes will only appear in the drop down if this object is a Resource object. (An object is a Resource object if its *Resource Object* property is set to 'True'.)

Adding a Standard Process



Steps are added to processes by clicking and dragging from either the **Common Steps** panel on the left or from the **All Steps(A-Z)** panel on the bottom left. All of the Common Steps are also listed in the All Steps panel. Any custom user created steps will appear in the **User Defined steps** panel on the bottom left.

Process Element Properties

Listed below are the properties of a **Process**:

Property	Description
Triggering Event Name	The name of an event that will trigger a new token to execute the steps in this process. Refer to the Subscribe and Unsubscribe steps to dynamically add or cancel triggering events for a process during a simulation run.
Triggering Event Condition	Optional condition to be evaluated whenever a triggering event occurs, and which must also be true to cause the execution of this process.
Token Class Name	The token class that all tokens executing this process will be created from.
Input Arguments	A list of input arguments that will be passed to the process if it is executed using an Execute step.
Name (Input Arguments)	A unique identifier for the input argument.

Description (Input Arguments)	Description text for the input argument.
State Variable Name (Input Arguments)	Name of the token state variable that will store the value for the input argument when a token begins executing the process.
Return Values	A list of values that will be returned by the process if it is executed using an Execute step.
Name (Return Values)	A unique identifier for the returned value.
Description (Return Values)	Description text for the returned value.
Expression (Return Values)	The expression that is evaluated to return the value.
Token Action On Associated Object Destroyed	The action taken by a token in this process if the token's associated object is destroyed. If the action is 'EndProcess', then the token will automatically end processing upon exiting its current step. Note that this property setting will be ignored by a token if it is directly responsible for the associated object being destroyed (e.g., the token is executing a Destroy step in the process). In that case, the token will continue processing regardless.
Token Action On Associated Object Transfer Requested	The action taken by a token in this process if the token's associated object requests a transfer to a new location. If the action is 'EndProcess', then the token will automatically end processing upon exiting its current step. Note that this property setting will be ignored if it is directly responsible for the transfer request (e.g., the token is executing a Transfer step in the process). In that case, the token will continue processing regardless.
Initially Enabled	Specifies whether the process is enabled when the system is initialized. Any attempt to execute a disabled process will be ignored by the simulation engine.
Allow Step Trace	Allows or suppresses trace messages from steps in this process. (Note: This applies ONLY to steps in this process; it does not propagate to steps in other processes that might be called from this one.)

Process Element Functions

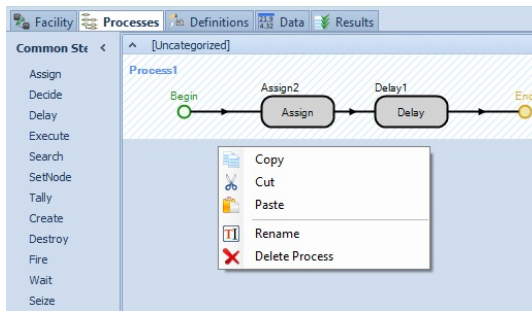
Listed below are the functions of a **Process**:

Function	Description
TokensInProcess	Provides functions for accessing the tokens that are currently executing the process.
TokensInProcess.NumberItems	Returns the number of tokens that are currently executing the process.
TokensInProcess.FirstItem	Returns a reference to the first token in the list of tokens that are currently executing the process.
TokensInProcess.LastItem	Returns a reference to the last token in the list of tokens that are currently executing the process.
TokensInProcess.IndexOfItem(token)	Returns the one-based index of a specified token in the list of tokens that are currently executing the process. If the token is not found then the value 0 is returned.
TokensInProcess.ItemAtIndex(index)	Returns a reference to the token at a specified index position in the list of tokens that are currently executing the process.
TokensInProcess.Contains(token)	Returns True (1) if the list of tokens that are currently executing the process contains the specified token. Otherwise, the value False (0) is returned.
TokensInProcess.NumberLinkedToObject(object)	Returns the number of tokens in process whose associated (primary) object reference is a specified object.

Navigating the Processes Window

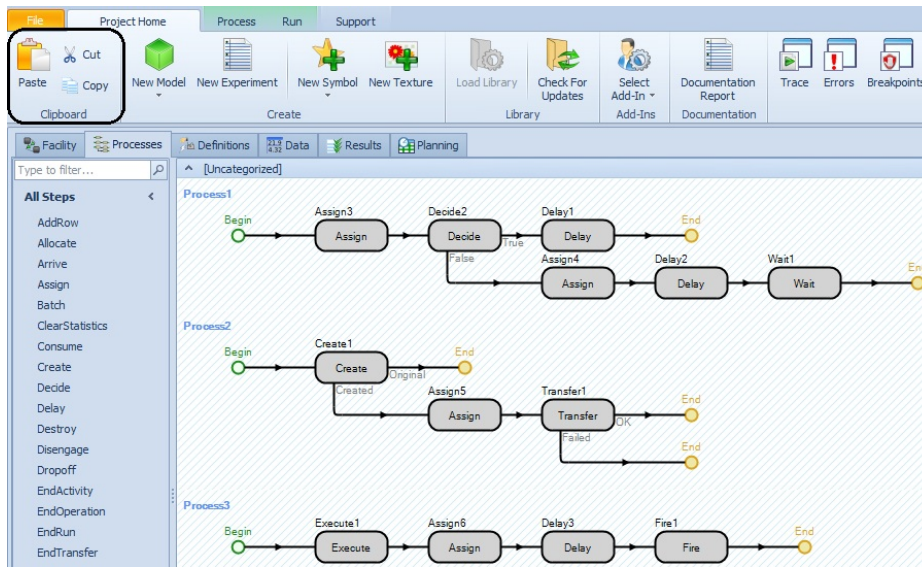
A user can **zoom in** on an individual process by clicking on the '+' key and **zoom out** by clicking on the '-' key. And the image of a Process can be **copied onto the clipboard**. This is done by selecting the process to be copied and either selecting the Copy icon from the Project Home Tab or by hitting CTRL-C. If pasting into a document, paste the bitmap. A right click within a Process, will produce a menu giving the user a choice to Copy, Cut, Paste, Rename or Delete the Process.

Right Clicking within a Process



Multiple entire processes can be selected for copy/paste into the same Processes window or a different model Processes window. This can be done by clicking the first process to copy to select it. Shift-click on another process will highlight the first selected, the last selected and all processes in between. Ctrl-click will keep the current selection and highlight the process that is selected as well, allowing for multiple processes to be highlighted that are not next to each other. When the processes are highlighted, the light blue hatching can be seen in the background of the process. From the keyboard, Ctrl-C / Ctrl-V / Ctrl-X can be used to then cut/copy/paste the multiple processes, or alternatively, the Cut/Copy/Paste buttons within the Project Home ribbon may be used. While the model is running, you can right-click on the Process and select "Go To Watch" and you will be taken to that process in the Watch Window

Shift-Select and Ctrl-Select Multiple Processes for Copy/Paste

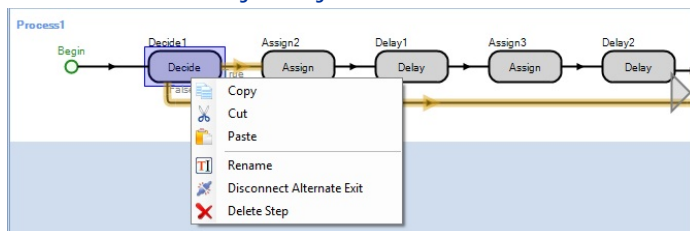


Multiple steps can be selected for cut/copy/paste/delete simultaneously as long as they are contiguous and have a single "starting step". However, if cutting or deleting, there is an additional requirement that only one step being cut/deleted can be connected to a step that hasn't been selected. When pasting multiple steps, Simio will keep the same formation as was cut/copied. When cutting or deleting, Simio will move and/or remove the remaining unselected steps to fill in the gap. Please ensure you double check your process logic while doing this.

A user can change where any segment from a step goes. To do this, select the step, then move the endpoints of its outgoing segments by clicking and dragging on the little circles on the end of the segment. A segment cannot be moved from a step if it leaves part of the process disconnected from the begin step.

Within the Processes window, there is an autoscroll capability which is useful when editing large processes. If you are moving a step from one part of the process to another, you can either use the '-' key to zoom out to view the process, or simply click on the step to move and Simio will autoscroll to the left or right as you move the step. Autoscrolling is also helpful in connecting the step's exit to another step that is several steps away. By selecting the step's segment circle and moving it to the left or right, the processes window will autoscroll in that direction. When working with larger processes, you are able to disconnect a segment by selecting the step and right-clicking, as seen below. The options for Disconnect Exit and/or Disconnect Alternate Exit are available if the step's exit can be disconnected without leaving part of the process disconnected from the Begin step.

Right Clicking to Disconnect an Exit



The [Properties Spreadsheet Window](#) is available for editing information within a Process or particular Step type in a grid-like format. Copy/paste may be used within the grid to easily change information in multiple columns for processes or steps. Additionally, multiple steps can be multi-selected by using Ctrl-click. This allows common properties to be edited within the Properties window on the right side of the Simio interface.

Properties Spreadsheet - 2 SetNode Objects					
Basic Logic		Advanced Options		General	
Instance Name	Process	Destination Type	Node Name	Node ID Number	Search Expression
SetNode2	GateCheckin1	Specific	Input@WaitingArea1		
SetNode2	GateCheckin2	Specific	Input@WaitingArea2		

The Process Run Ribbon

The Process Run Ribbon contains additional functionality to assist in Process debugging.

The Step button here will forward to the next simulation event. Typically this corresponds to a Token executing a Process Step.

The Current Step button will go to the Step in the Process View that was just executed.

Documenting Your Process

We encourage you to take advantage of the features added to make your processes and the trace they generate easier to use.

In particular, adding a Description to each process that describes what the process does will make it easier for you and others to understand its function. And customizing the Name for each step (Press F2) will not only make the process more self-documenting, but will also make trace easier to interpret. So, for example, instead of leaving the default 'Decide1' name on a Decide step, change it to something more meaningful like 'DecideIfAtDestination'.

Additionally, steps consist of a Color property, which can be found within the General section of properties. Changing the color of a step or group of steps from the default color to a unique color enables you to easily distinguish and quickly find various sections of steps in larger models. The model trace also reflects the step colors when turned on.

Server 1 AddOn Processes

Server1_BeforeProcessing

Begin

Seize Worker

Delay for SetupTime

Release Worker

Assign New Priority

End

Navigation: Model

MySimioProject

ModelEntity

Model

Properties: Assign New Priority (Assign Step Insta

Show Commonly Used Properties Only

Basic Logic

State Variable ...

New Value

Assignments (M...

Advanced Options

General

Name

Description

Color

Multiple Passed and Returned Values on Execute Step

Simio supports the ability to pass input arguments and return values on executed processes. This allows single shared processes to be executed within multiple objects. Processes have a repeat group where you can identify, for the token class, the Input Arguments and Return Values.

For token Input Arguments, the State Name Variable should always be a state on a custom token, where that custom token is specified as the Token Class Name of the parent process. Doing otherwise could cause modeling problems. Making it a token state makes sure that each token has its own unique value. Specifying a state variable here indicates that when the process is executed the value of this Input Parameter will be assigned to that state. Once the Input Arguments have been defined, they will appear anywhere their parent process is referenced (ex: in an Execute step, an Add-On Process, etc.). Note that the Input Arguments will pick up the unit type (if any) from the state referenced in the state variable name of the Input Argument. Refer to SimBit [MultipleInputArgumentsOnProcesses](#) for an example on how to use this feature.

For token Return Values, the expression here is evaluated in the context of the process and it's token. When the process is finished, the value of this expression is assigned back to a state variable specified. These Return Values will appear anywhere their parent process is referenced (ex: in an Execute step or an Add-On process).

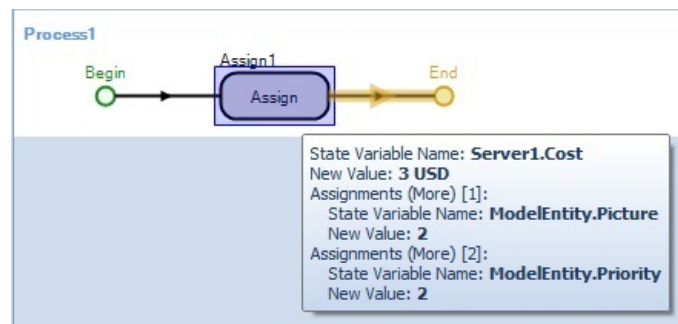
Steps

Steps are used to define logic within your simulation model. Each step performs an action such as seize, delay, decide, or wait. Steps are stateless, but may change the state of an element/token/entity/object. Steps are organized into groups within the Processes Window.

They include the [Common Steps](#), [All Steps](#) and [User-Defined Steps](#).

If you hover over a step within a process, you can view the information specified in the Properties and/or Repeat Group without having to open the details in the properties window.

Hovering over a Step that contains information in a Repeat Group property



Customizable "Common Steps" Panel

The Common Steps panel can be customized. If you're looking at a different steps panel (All Steps or User Defined), you can right-click on a step and add or remove it from Common Steps. You can also use drag and drop to re-arrange the steps on the Common Steps panel. There is also an option to restore the Common Steps to their default steps and order. This is a product-level feature (not per-project or per-model).

Tokens

Note: Tokens execute Steps in a process. A Step often acts on either the token's '**AssociatedObject**' or the '**ParentObject**'. The Parent object is the object that contains the process where the step is being used, which is often the overall Model. The Associated object is typically an object that the token is representing, such as an entity.

Exclusion Expression on Steps

All Simio Steps have a special property called *Exclusion Expression*. This property is used in some of the Steps that exist within the logic of the Standard Library objects, in order to increase run speed and eliminate unnecessary Steps from being evaluated during a run. The user can place an expression in this property of a Step in order to control whether or not that Step is evaluated during a run. If specified, the expression is evaluated at the start of the run to determine if it should be excluded. If this expression evaluates to 1, this step will be excluded and the tokens will flow directly to its primary exit point. If this expression evaluates to 2, the step has a secondary exit, the step will be excluded and the tokens will flow to the secondary exit. If the expression evaluates to any other value, it will remain active during the run.

Trace Level on Steps

All Simio Steps have a special property called *Trace Level*. This property indicates whether to enable or disable Trace messages for executing steps. 'Always' indicates that Trace for this step is enabled. This overrides the Allow Step Tracing setting on the process level. This functionality is similar to the Notify Step. 'Never' indicates that Trace for this step is disabled. 'Default' indicates that trace for this step is enabled based on the model Trace being enabled.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Common Steps

The Common Steps library includes some of the most commonly used steps including:

Assign step is used to assign a new value to a state variable.

Create step may be used to create new entity objects of a specified type, create copies of an existing entity, or to simply create new tokens that reference existing objects.

Decide step may be used to determine the flow of a token through process logic.

Delay step delays the arriving token in the step for the specified time duration.

Destroy step destroys either the parent object or the executing token's associated object.

EndTransfer step may be used to indicate that the entity object associated with the executing token has completed transfer into an object or station.

Execute step may be used to execute a specified process.

Find step may be used to search the value of an expression over a specified range of one or more index variables. The expression will typically involve array variables (vectors or multi-dimensional arrays) or indexing related functions.

Fire step may be used to fire an object event.

Move step may be used to request a move from one or more moveable resources that have been seized by either the parent object or object associated with the executing token. The executing token will be held in the Move step until the resources have arrived to the requested locations.

Release step releases capacity of one or more objects on behalf of the parent object or the object associated with the executing token.

Scan step may be used to hold a process token at the step until a specified condition is true.

Search step may be used to search a collection of objects.

Seize step may be used to seize capacity of one or more objects on behalf of the parent object or the object associated with the executing token.

SetNode step may be used to set the destination node of any entity object.

Tally step tallies an observation for each token arriving to this step.

Transfer step may be used to transfer the entity object associated with the executing token between objects and between free space and objects.

Wait step may be used to hold the arriving token in the step until a specified event occurs.

Customizable "Common Steps" Panel

The Common Steps panel can be customized. If you're looking at a different steps panel (All Steps or User Defined), you can right-click on a step and add or remove it from Common Steps. You can also use drag and drop to re-arrange the steps on the Common Steps panel. There is also an option to restore the Common Steps to their default steps and order. This is a product-level feature (not per-project or per-model).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

All Steps

The All Steps (A-Z) library includes all of the Simio steps. These steps provide the runtime logic for the simulation and include:

AddRow step may be used to add a new row to a specified output table. A reference to the added row is assigned to the executing token or object.

AddStock step may be used to add material stock to a storage bin.

AddStorageLocations step may be used to add a child storage location to a parent storage area.

Allocate step may be used to manually trigger an allocation attempt of the parent object's available resource capacity (if any) to other objects waiting for units of that capacity.

Arrive step may be used by an entity object to notify its accepted visit requests that the entity has 'arrived' to its current location.

Assign step is used to assign a new value to a state variable. The Set* steps are used to set the values of other items.

Batch step may be used in conjunction with the BatchLogic element to match multiple entities together, form those entities into a batch, and then attach the batched members to a parent entity.

Changeover step may be used in conjunction with a ChangeoverLogic element to model a sequence dependent setup at a specified resource.

ClearStatistics step may be used to clear model statistics.

Consume step consumes the specified quantity of a material - either at the top level only - or the exploded bill of material.

Create may be used to create new entity objects of a specified type, create copies of an existing entity, or to simply create new tokens that reference existing objects.

Decide step may be used to determine the flow of a token through process logic.

Delay step delays the arriving token in the step for the specified time duration.

Destroy step destroys either the parent object or the executing token's associated object.

Disengage step may be used to unlock an entity's location from a location on the parent link so that the link and entity may move independently.

Dropoff step may be used by a transporter object to drop off a riding entity into the transporter's current node.

EndRun step ends the simulation run.

EndTransfer step may be used to indicate that the entity object associated with the executing token has completed transfer into an object or station.

Engage step may be used to lock an entity's location to a location on the parent link object.

Execute step may be used to execute a specified process.

Fail step starts a downtime of type Failure Name.

Find step may be used to search the value of an expression over a specified range of one or more index variables. The expression will typically involve array variables (vectors or multi-dimensional arrays) or indexing related functions.

Fire step may be used to fire an object event.

Insert step may be used to insert an object into a specified queue.

Interrupt step may be used to interrupt process delays.

Move step may be used to request a move from one or more moveable resources that have been seized by either the parent object or object associated with the executing token. The executing token will be held in the Move step until the resources have arrived to the requested locations.

Notify step may be used to output a user defined trace or warning message.

NotifyConstraint step may be used to add a custom log entry to the model's constraint log.

Park step may be used to move an entity object into the parking station of a node.

Pickup step may be used by a transporter object to pick up an entity waiting in the ride pickup queue of the transporter's current node.

PlanVisit step may be used by a transporter object to search for and accept a pickup reservation request in the system.

Produce step produces the specified quantity of material - either at the top level only - or at the exploded bill of material..

Release step releases capacity of one or more objects on behalf of the parent object or the object associated with the executing token.

Remove step may be used to remove an object from a specified queue.

RemoveRows step may be used to remove all existing rows from an output table.

RemoveStock step may be used to remove material stock from a storage bin.

RemoveStorageLocations step may be used to remove a child storage location from a parent storage area.

Repair step ends a downtime of type Failure Name.

Reserve step allows the reservation of a specific resource capacity.

ReserveBins step may be used to reserve storage bin capacity for an entity's material stock requirements.

ReserveStock step may be used to reserve storage bin stock for an entity's material stock removal requirements.

RestartRun may be used to set the ending time of the simulation run to the current time. This will cause the run to end once all simulation events scheduled for the current time have been processed.

Resume step may be used to resume a process or to resume the movement of an object.

Ride step may be used to initiate a transporter ride request for an entity object at a node.

Route step may be used with a RoutingGroup element to route an entity object to a destination selected from a list of candidate nodes.

Scan step may be used to hold a process token at the step until a specified condition is true.

Search step may be used to search a collection of objects.

Seize step may be used to seize capacity of one or more objects on behalf of the parent object or the object associated with the executing token.

SelectDropoff step may be used to set the destination node of the parent transporter object to the dropoff location of a riding entity.

SelectVisit step may be used to set the destination node of the parent transporter object to a pickup location that was accepted using the PlanVisit step.

SetNetwork step may be used to set the network of links used by an entity to travel between node locations.

SetNode step may be used to set the destination node of any entity object.

SetRow step may be used to assign a data table row or sequence table row to a token or object.

SetWorkSchedule step may be used to assign a new work schedule to a resource object.

StartTasks step may be used to initiate a task sequence that is associated with an object in the model.

Stipulate step may be used to add dependencies and requirements that entities must satisfy before processing.

Subscribe step may be used to add a new triggering event for a process, indicating that the process is to be executed if the event occurs.

Suspend step may be used to suspend a process or to suspend the movement of an object.

Tally step tallies an observation for each token arriving to this step.

Transfer step may be used to transfer the entity object associated with the executing token between objects and between

free space and objects.

[Travel](#) step may be used to do a direct (straight-line) movement of an entity in free space to a specified location.

[UnBatch](#) step may be used to remove batch members from the parent entity object associated with the executing token.

[UnPark](#) step may be used to move an entity object out of the parking station of a node.

[UnReserve](#) step may be used to cancel any reservations for a resource made by a specified owner up to a desired number of reserved capacity units cancelled.

[UnSetRow](#) step may be used to remove table row references previously set for the executing token or a specified object or element.

[UnSubscribe](#) step may be used to cancel a triggering event for a process.

[VisitNode](#) step may be used within a node object's logic to initiate the OnVisitingNode process of the entity associated with the executing token.

[Wait](#) step may be used to hold the arriving token in the step until a specified event occurs.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

User Defined Steps

Simio provides an open architecture that allows users to add new steps and elements to the system. Steps and elements may be coded in any .NET language (Visual Basic, C#, J#, etc.). You would simply add your *.dll files to the UserExtensions folder of Simio. For additional information on creating your own custom Steps, see [Custom Simio Extensions](#) and **C:\Program Files\Simio LLC\Simio\Simio API Reference Guide.chm**. See [Simio Visual Studio Templates](#) for information on how to use the templates provided by Simio to get started with User Defined Steps.

Simio provides several sample User Defined steps including [CloseGate](#), [OpenGate](#), [PassThruGate](#), [Read](#), and [Write](#).

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

CloseGate

CloseGate

The CloseGate user defined step may be used to close a specific gate name.

The CloseGate step is used in conjunction with the OpenGate and PassThruGate user defined steps and the BinaryGate user defined element.

Listed below are the properties of **CloseGate**:

Property	Description
Gate	The name of the gate to close.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DbExecute

DbExecute

The DbExecute is a user defined step may be used to execute an SQL statement on a database.

The DbExecute step is used in conjunction with the [DbConnect](#) user defined element.

Listed below are the properties of **DbExecute**:

Property	Description
DbConnect	The name of the DbConnect element to reference.
SQL Statement	Statement used to execute database calls. It can be used to insert, update and delete data. Since the Db Write uses expressions instead of states, the position is to populate data into the SQL statement.(e.g. @1 is expression 1).
Items	The list of expressions. NOTE: Datetimes need to be converted into a string that is recognizable by the database. (e.g., String.FromDateTime(Entity.TimeCreated, "yyyy/MM/dd HH:mm:ss")

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DbQuery

DbQuery

The DbQuery is a user defined step may be used to query data from a database.

The DbQuery step is used in conjunction with the [DbConnect](#) user defined element.

Listed below are the properties of **DbQuery**:

Property	Description
DbConnect	The name of the DbConnect element to reference.
SQL Statement	<p>The SQL statement that is called to retrieve the data. The column names returned are mapped into the states in the same order. First column into first state, second column into second state, etc.. For the where statement, the state value used is based on position in the states repeating properties.</p> <p>In the example below, the 6th state (@6) (RowID) will be used to populate the value used by the where statement. The position is used instead of the state name to be consistent with the DbRead step: Select String1, Integer1, Real1, DateTime1, DateTime2 from TestReadWrite where ID = @6</p>
States	The states values to read the states into.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DbRead

DbRead

The DbRead is a user defined step may be used to read data from a database.

The DbRead step is used in conjunction with the [DbConnect](#) user defined element.

Listed below are the properties of **DbRead**:

Property	Description
DbConnect	The name of the DbConnect element to reference.
Table Name	The database table name from which the data will be read.
Columns	These will be used to select the data from database columns into states.
Where	The column names and values from which to select data.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DbWrite

DbWrite

The DbWrite is a user defined step may be used to write data to a database.

The DbWrite step is used in conjunction with the [DbConnect](#) user defined element.

Listed below are the properties of **DbWrite**:

Property	Description
DbConnect	The name of the DbConnect element to reference.
Table Name	The database table name where the data will be written.
Columns	These will be used to enter data from expressions into the database columns.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExcelRead

ExcelRead

The ExcelRead is a user defined step may be used to read values from an Excel worksheet.

The ExcelRead step is used in conjunction with the [ExcelConnect](#) user defined element.

Listed below are the properties of **ExcelRead**:

Property	Description
ExcelConnect	The name of the ExcelConnect element to reference.
Worksheet	The name of the worksheet to use.
Row	The row within the worksheet to read.
StartingColumn	The starting column within the worksheet to read.
States	The states within the model to populate data from the worksheet.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExcelWrite

ExcelWrite

The ExcelWrite is a user defined step may be used to write values to an Excel worksheet.

The ExcelWrite step is used in conjunction with the [ExcelConnect](#) user defined element.

NOTE: When running the ExcelWrite step, you must stop the model before you can look at the data. If you don't, you will get a file lock warning.

Listed below are the properties of **ExcelWrite**:

Property	Description
ExcelConnect	The name of the ExcelConnect element to reference.
Worksheet	The name of the worksheet to use within the Excel file.
Row	The row within the worksheet to write.
Starting Column	The starting column within the worksheet to write.
Items	The expression items to be written out.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

OpenGate

OpenGate

The OpenGate user defined step may be used to open a specific gate name.

The OpenGate step is used in conjunction with the CloseGate and PassThruGate user defined steps and the BinaryGate user defined element.

Listed below are the properties of **OpenGate**:

Property	Description
Gate	The name of the gate to open.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

PassThruGate

PassThruGate

The PassThruGate user defined step allows tokens through a specific gate name if it is opened.

The PassThruGate step is used in conjunction with the OpenGate and CloseGate user defined steps and the BinaryGate user defined element.

Listed below are the properties of **PassThruGate**:

Property	Description
Gate	The name of the gate to pass through. If the gate is currently closed, the token will wait here until the gate opens.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Read

Read

The Read user defined step may be used to read data into the model from a file.

The Read step is used in conjunction with the [File](#) user defined element. If the *Separator* property is left blank, one entire line in the file will be read into one State. In order to read in individual values, enter the appropriate character into the *Separator* property. For example, a .csv or .xls file should have a comma listed in the *Separator* property so that each individual cell is read into a State. Note that if you read values into a State, it must be numeric since States cannot hold non-numeric characters.

The Read step supports reading to and from Numeric states, as well as String states and DateTime states.

Listed below are the properties of **Read**:

Property	Description
File	The name of the file from which to read.
Separator	The character that is separating the numeric values in the file.
State	The state values to read the values into.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RelocateObject

RelocateObject

The RelocateObject user defined step may be used to instantaneously change the location of an object, including fixed objects and unconnected nodes, at runtime.

When relocating or moving a fixed object, such as a Server or Combiner, the *Include Associated Nodes* property set to 'True' will move the object location (and animation) as well as nodes that are attached to the object.

Note that attempting to change the location of a Link or a Node with attached Links will result in a runtime error.

Note that there is a LocationAt.Geographic function that can be used in conjunction with this step. To assign cartesian coordinates, the LocationAt.Geographic function is available for conversion from geographic coordinate to equivalent cartesian. For example, Entity.Movement.X = LocationAt.Geographic(latitude, longitude).X. See the [Functions in Simio - Automatic](#) page for more information on the function.

Listed below are the properties of **RelocateObject**:

Property	Description
Object	The name of the object to move.
X	The new X location of the object.
Y	The new Y location of the object.
Z	The new Z location of the object.
Include Associated Nodes	If true, when moving a fixed object, its associated nodes will be offset by the same amount.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Write

Write

The Write user defined step may be used to write data from the model to a file. To reference an item from the items specified in the Item Property when writing to a text file, use balanced brackets {}. {0} will reference the first item specified, {1} the second item, etc. For example, the following sentence can be specified in the format property: There are {0} parts of type {1} at time {2}. When writing to a .csv type file, the format field can be left empty since it uses a common-separated format by default.

The Write step is used in conjunction with the [File](#) user defined element. See the [WritingToAFile](#) SimBit for an example on how to use the File Element.

The Write step supports writing to and from Numeric states, as well as String states and DateTime states.

Write Step in Experiments

When a Model contains Write Steps in its Process Logic, and an Experiment is run, the output files will be merged for improved data portability, provided in addition to a file per Scenario per Replication.

To make the most use of the merged file created from Write steps via Experiments, consider adding Run.ScenarioName and Run.ReplicationNumber as Items so record entries can be associated with the corresponding information.

Listed below are the properties of **Write**:

Property	Description
File	The name of the file into which to write. File is defined with the File User Defined Element on the Elements tab of the Process Window.
Format	This provides the format of the string to write out. If none is provided, it will use a comma-separated format.
Items	The item(s), specified by this expression, that will result in a value to write out.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Steps Reference

This Steps Reference section has detailed information for all steps in Simio.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AddStock

AddStock Step

The AddStock step may be used to add material stock to a storage bin. The *Stock Addition Type* property in the Stock Additions repeat group determines the source of information for the stock addition.

If the *Stock Addition Type* is 'Specific', the material type, destination storage bin, quantity, and quant selection condition expression are specified within the step to determine how the material is added to the storage bin. If the *Stock Addition Type* is 'Reservation', the AddStock step will use the information from the specified Stock Reservation to add material to a storage bin. For more information about stock reservations, see [Stock Reservations](#).

Because the quantity of material in a storage bin is concurrently limited by its volume capacity, weight capacity, and total capacity, an AddStock step cannot add more stock to a storage bin than the storage bin can contain. If the available storage bin capacity is insufficient for the entirety of the stock addition, the addition will fail. Subsequent additions specified by the Stock Additions repeat group will still be attempted.

For more information about modeling material storage and retrieval, see [Material Storage – Discussion and Examples](#).

Listed below are the properties of the **AddStock** step:

Property Name	Description
Stock Additions	The material stock to add.
Save Stock Quant Reference	Optional state variable to save a reference to each created or updated stock quant.
Stock Additions.Stock Addition Type	The stock addition type. Specific – Add a specified quantity of material to a specified storage bin. Reservation – Add material stock using a specified stock putaway reservation.
Stock Additions.Storage Bin Name	The name of the storage bin to add material stock to.
Stock Additions.Material Name	The name of the material to add.
Stock Additions.Quantity	The quantity of material to add.
Stock Additions.Stock Quant Selection Condition	Optional condition evaluated for each candidate stock quant in the first-in-first-out list of stock quants currently present in the storage bin. If left unspecified then the first stock quant in the list that is the same material will be selected to increase its quantity in stock. If no existing stock quant is selected then a new quant will be created and added to the storage bin. In the expression, use the syntax <code>Candidate.StockQuant.[Attribute]</code> to reference an attribute of the candidate stock quant (e.g., <code>Candidate.StockQuant.QuantityAvailable</code>).
Stock Additions.Stock Reservation Reference	The stock putaway reservation reference.
Stock Additions.Skip Stock Addition	Optional condition indicating whether to skip the material stock addition.

Condition

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AddStorageLocations

AddStorageLocations

The AddStorageLocations step may be used to add a child storage location to a parent storage area. Use the RemoveStorageLocations step to remove a child storage location.

Listed below are the properties of **AddStorageLocations**

Property	Description
AddStorageLocations.StorageLocationAdditions	The child storage locations to add.
AddStorageLocations.ParentStorageAreaName	The name of the parent storage area.
AddStorageLocations.ChildStorageLocationType	The type of child storage location to add.
AddStorageLocations.ChildStorageAreaName	The name of the child storage area to add.
AddStorageLocations.ChildStorageBinName	The name of the child storage bin to add.
AddStorageLocations.SkipStorageLocationAdditionIf	Optional condition indicating whether to skip the child storage location addition.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Allocate

Allocate

The Allocate step may be used to manually trigger an allocation attempt of the resource's available capacity (if any) to other objects waiting for units of that capacity. The *Cancel All Reservations First* property, if set to 'True', will cancel any active reservations (done through the [Release](#) step, if any) before the allocation is started.

The Allocate step will force the designated resource to step through its allocation queue using its designated Ranking Rule and Dynamic Selection Rule, giving each waiting seize request in turn an opportunity to re-evaluate its seize. This may result in the designated resource being allocated, or re-evaluation of the seize request may result in a different resource being allocated, or possibly no resource allocation at all.

Listed below are the properties of **Allocate**:

Property	Description
Resource Type	The resource object whose capacity units are to be allocated. May be specified as either the object associated with the executing token, the parent object, or as a specific resource reference.
Resource Object	The specific resource object whose capacity units are to be allocated.
Cancel All Reservations First	Indicates whether to first cancel all active reservations (if any) for the resource before attempting to allocate its capacity.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

AddRow

AddRow

The AddRow step and associated Output table can only be used with Simio Professional and RPS Editions.

The AddRow step may be used to add a new row to a specified Output table. This new row will be appended to the end of the Output table. The *Add Row Condition* may be specified to evaluate a conditional expression before adding a row. A reference to the added row is assigned to the executing token or object.

The AddRow step is used only with Output tables (not standard data tables). To assign the data within the row, use an Assign step, where you assign the *State Variable Name* of 'OutputTableName.ColumnName' to a *New Value* based on the type of column specified (i.e., real, integer, string, object reference, etc.).

The [RemoveRows](#) step may be used to remove all rows or a single row from an output table.

For an example of using the AddRow step, please refer to the SimBit [UsingAddRowandOutputTable](#).

Listed below are the properties of **AddRow**:

Property	Description
Table Name	The name of the sequence or data table.
Key Column Value	The unique key value for this row. This property should only be used with RPS licensing.
Object Type	Indicates the item for which a reference to the newly created table row will be assigned. It may be a reference to the parent object, to the executing token, to the object associated with the executing token, or to a specific object or element.
Object or Element	The name of the object or element to be assigned a reference to the newly added table row.
Add Row Condition	Optional condition that, if specified, must evaluate to true to perform the add row action.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Arrive

Arrive

The Arrive step may be used to have an entity object notify its accepted visit requests that the entity has 'arrived' to its current location.

Within the Standard Library, the Vehicle and Worker objects both use the Arrive step in their OnVisitingNode process logic (logic related to be used as a moveable resource for processing tasks).

The step's current usage is for modeling a moveable resource entity. It may be used by a resource entity to explicitly notify a token waiting at a Seize or Move step that the entity resource has officially 'arrived' at the requested destination node location and thus the token responsible for the resource move request may now be released from the Seize or Move step to execute any subsequent process logic.

Listed below are the properties of **Arrive**:

Property	Description
Entity Type	The entity object that is notifying accepted visit requests of its arrival. May be specified as either the object associated with the executing token, the parent object or as a specific object reference.
Entity Object	The specific entity object that is notifying accepted visit requests of its arrival.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Assign

Assign

The Assign step may be used to assign a new value to a state variable.

The state specified can be owned by the token, visiting entity, parent object, or member element.

The Assignments (More) section can be used for conditional assignments using the *Skip Assignment If* property.

For examples of using the Assign step, please refer to the SimBits [LearningCurveWithFunction](#), [LeveledArrivals](#), [LogicBasedOnEntityState](#), [RecordDistanceTraveled](#) and [SimpleTank](#).

Other steps, such as the Set* steps, are used for assigning the values of other items. The [SetRow](#) step can be used to assign a table name and row to an object or token. The [SetNode](#) step may be used to assign the destination node of an entity. The [SetNetwork](#) step is used to assign a network of links to an entity for travel.

Listed below are the properties of **Assign**:

Property	Description
State Variable Name	Name of the state variable that will be assigned a new value.
New Value	The new value to assign.
Assignments (More)	A repeating set of assignments.
Assignments (More).State Variable Name	Name of the state variable that will be assigned a new value.
Assignments (More).New Value	The new value to assign.
Assignments (More).Skip Assignment If	Optional condition indicating whether to skip the state variable assignment.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Batch

Batch

The Batch step may be used in conjunction with the BatchLogic element to group multiple entities together and then attach the batch members to a parent entity. Use the UnBatch step to later remove batch members from a parent entity.

Listed below are the properties of **Batch**:

Property	Description
BatchLogic Name	The name of the BatchLogic element used to execute the batching logic.
Batch Queue Type	Indicates which queue of the Batch Logic element to insert the entity.
Entity Type	The entity that executing the batching logic. May be specified as either the object associated with the executing token, the parent object or a s specific object reference.
Entity Object	The specific entity object that is executing the batching logic.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Changeover

Changeover

The Changeover step may be used in conjunction with a [ChangeoverLogic](#) element to model a sequence dependent setup at a specified resource.

The executing token will be held in the Changeover step for the total changeover time. Any suspension of the Changeover step's parent process will extend the token's delay time accordingly.

The *Delay Multiplier* property allows for dynamic adjustments to the *Changeover Time*. If the process that is executing the Changeover step is suspended, then the base changeover time remaining is saved. When the process is resumed, the *Delay Multiplier* expression property is re-evaluated and the base changeover time remaining multiplied by that number to schedule the end of the changeover back on the simulation event calendar.

For example, suppose the changeover time is calculated as '1' hour and the *Delay Multiplier* expression returns the value of '2' (based on model entity state or table value). This results in a changeover time of 2 hours. If the process is suspended after 0.5 hours have elapsed and the changeover thus 25% completed, then the base changeover time remaining is 0.75 hours. When the process is resumed, if the *Delay Multiplier* expression property now returns the value '1.5' then the end of the changeover is scheduled back on the calendar to occur at $\text{TimeNow} + (0.75 \times 1.5 =) 1.125$ hours. If the *Delay Multiplier* expression property returns a negative value, then a runtime error is thrown.

Listed below are the properties of **Changeover**:

Property	Description
Resource Name	The name of the resource object to which the changeover applies.
Changeover Logic Name	The name of the ChangeoverLogic element used to determine any sequence dependent setup times.
Entity Type	The operation entity used to determine any sequence dependent setup times, by comparing its attributes to those of the previous entity. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Delay Multiplier	Increases or decreases the total setup time due to the changeover by some factor. Specify 2, for example, to mean that the delay is twice as long. Note that this expression is automatically reevaluated if the process is suspended and resumed, potentially adjusting the remaining delay time.
Entity Object	The specific operation entity object used to determine any sequence dependent setup times, by comparing its attributes to those of the previous entity.
Save Setup Time	Optional discrete state variable to store the total setup time required due to the changeover.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ClearStatistics

ClearStatistics

The ClearStatistics step may be used to clear model statistics. Users can easily reset either all model statistics or only the statistics for specific objects or elements.

State variables have a property named *Auto Reset When Statistics Cleared* that will indicate whether or not the particular state variable will be automatically reset to its initial value if the statistics are cleared by this step.

Possible uses for this step include wanting to clear all statistics during an interactive run based on some triggering event or 'warmup' time, or wanting to collect and report periodic statistics (e.g., some logic that every hour writes out some object or element statistics and then clears them).

Listed below are the properties of **ClearStatistics**:

Property	Description
Statistics Type	Indicates whether to clear all model statistics or only the statistics for specific objects or elements.
Objects or Elements	The repeating list of specific objects or elements whose statistics are to be cleared.
Objects or Elements.Object or Element	The specific object or element whose statistics are to be cleared.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Consume

Consume

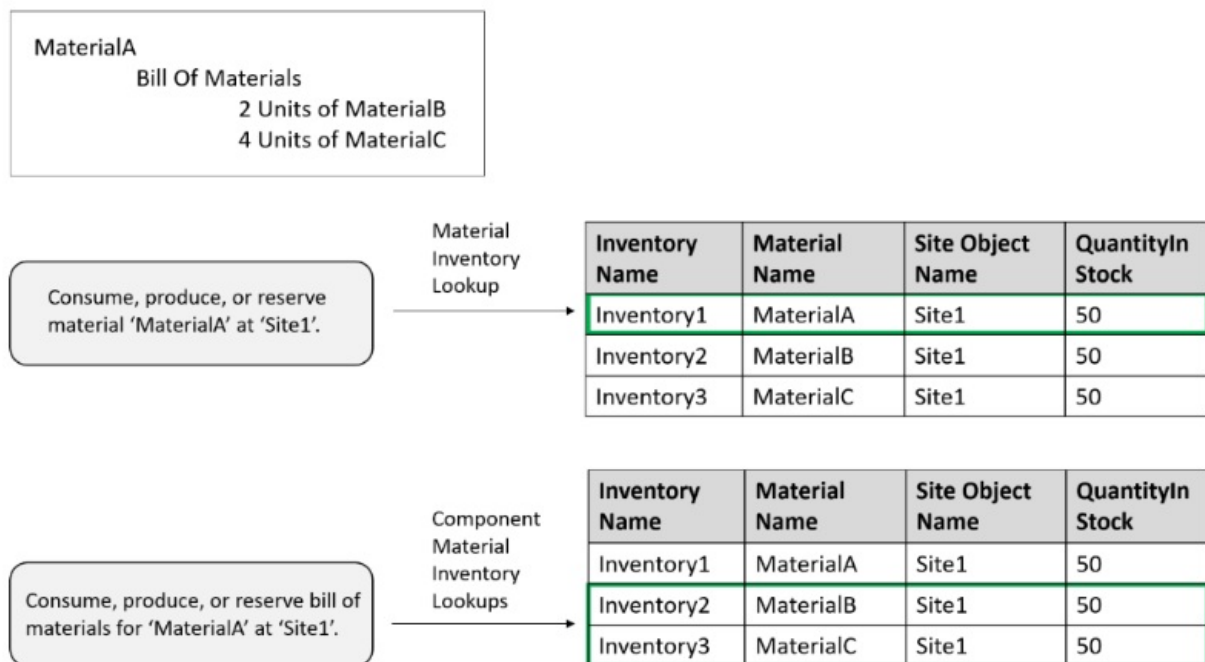
The Consume step may be used to consume stock on behalf of a specified owner. If the *Must Simultaneously Consume* property is false then partial BOM allocation is allowed. A *Consume Materials (More)* repeat group is provided that allows the definition of repeating material consumption data. Alternatively, a data table may be mapped to that repeat group.

For the single material type, the specified quantity of this material is directly consumed. For the bill of material type, the material listed in the "bill of materials" is consumed by the token, where quantity is the quantity in the bill times the top level quantity (typically 1). A token waits at this step until all quantities have been allocated. Allocations are made incrementally as material becomes available, if the *Must Simultaneously Consume* property is set to 'False' (default behavior). If the *Must Simultaneously Consume* property is 'True', the full quantity of all materials must be available before any are allocated. Only the first waiting entity in the Material queue is eligible for allocation.

If the *Lot ID* property is specified, then only quantities of the material assigned to that specific lot identifier may be consumed. The *Lot ID* property may be specified as a string literal enclosed in double quotes (e.g., "Lot1"), the name of a string state variable or string column in a table, or any other expression that returns a string value.

If the *Lot ID* property is not specified, then only quantities of the material that are not pegged (i.e., not assigned to any lot) may be consumed.

If consuming a single material that is location-based inventory, then Simio will look up the Inventory element holding the specified material at the specified inventory site. If consuming a bill of materials, then Simio will look up the Inventory elements holding the component materials at the specified inventory site.



If the *Inventory Site Type* property value is 'None' but a site object reference is required because the material is location-based inventory, or if an inventory reference for a material-site object pair cannot be found, then a runtime error will occur.

The Consume step works in conjunction with the Produce step. If the quantity of all the bill of materials in the Consume step is not available, the token waits at the Consume step. Elsewhere in the model, the Produce step should be used to increase a material's quantity and once that is done, the Consume step will use the material produced.

The *Owner Object* property provides a context when determining which object will be allocated the consumed material.

A process token created to execute an *On Consumed Process* will have the following characteristics:

- Its associated object reference will be set to the object that consumed the material.

- It will be initialized with the same table row references as the original token executing the Consume step.
- Its [material order detail](#) reference will provide the detail of the material consumption.

If specified, the *On Consumed Process* will be executed any time a quantity of material is consumed; either a full or partial allocation case.

Property Name	Valid Entry	Switch Condition	Description
Consumption Type	Material, BillOfMaterials, BillOfMaterialsGroup	None	The type of material stock consumption. Material – A single material. BillOfMaterials – A list of component materials. BillOfMaterialsGroup – Alternative lists of component materials.
Material Name	Material Reference	Consumption Type == Material	The name of the material.
BOM Name	Bill Of Materials Reference	Consumption Type == BillOfMaterials	The name of the bill of materials.
BOM Group Name	Bill Of Materials Group Reference	Consumption Type == BillOfMaterialsGroup	The name of the bill of materials group.
Quantity	Real	None	The quantity of the material or parent assembly.
Lot ID	String	None	Optional string value indicating the lot identifier of the material or parent assembly.

Listed below are the properties of **Consume**:

Property	Description
Consume Materials (More)	A repeating set of additional materials to consume.
Consume Materials (More).Consumption Type	Indicates whether to consume a single material or bill of materials.
Consume Materials (More).Material Name	The name of the material which is to be either specifically consumed or whose bill of materials is to be consumed.
Consume Materials (More).Inventory Site Type	Indicates the fixed object that is the inventory location. Applies only to material elements whose <i>Location Based Inventory</i> property is set to 'True'.
Consume Materials (More).Site Object Name	The name of the fixed object that is the inventory location.
Consume Materials (More).Quantity	The quantity to be consumed.
Consume Materials (More).Lot ID	Optional string value indicating the lot identifier of the consumed material.
Consume Materials (More).Record Material Cost	Indicates whether costs of the consumed material will be recorded (charged) to the cost of the owner object.
Consume Materials	Optional process to be executed when material is consumed. The created token's

(More).On Consumed Process	material order detail reference will provide the detail of the material consumption. The process will trigger for the item specified by the Consume Materials (More) properties. Each On Consumed Process is specified per item listed for consumption.
Consume Materials (More).Skip Consumption If	Optional condition indicating whether to skip the material consumption.
Must Simultaneously Consume	Indicates whether all of the required materials must be available before any can be consumed.
Owner Type	The object that is the owner of the material consumption request. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Owner Object	The specific owner object that the material consumption is in behalf of.
Record Material Costs	Indicates whether the cost of the consumed material will be recorded (charged) to the cost of the owner object.
On Consumed Process	Optional process to be executed when material is consumed. The created token's material order detail reference will provide the detail of the material consumption. This process will only trigger for the item specified by the Material Properties at the top level of the Consume step's properties. Each On Consumed Process is specified per item listed for consumption.
Immediately Try Consume	Indicates whether to immediately try consuming the required materials before the execution of any other simulation logic in the system and, if successful, skip the material or inventory allocation queues. Setting this property to False will first insert the material consumption request into the material or inventory allocation queues. An evaluation of those queues will then be scheduled on the simulation's current event calendar as a late priority event.
Exclusion Expression	If specified, an expression that will be evaluated at the start of a simulation run to determine if this step should be excluded from the run. If this expression evaluates to 1, this step will be excluded, and tokens will flow directly to its primary exit. If this expression evaluates to 2, and this step has a secondary exit, the step will be excluded, and tokens will flow directly to its secondary exit. If the expression evaluates to any other value (typically 0), the step will remain active in the process.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Create

Create

The Create step may be used to create new entity objects of a specified type, create copies of an existing entity, or to simply create new tokens that reference existing objects. New tokens associated with the created or referenced objects will exit the 'Created' exit point of the step.

If the *Create Type* is 'NewObject' and *Entity Type* property is not specified, no object will be created. If the *Create Type* is 'CopyParentObject' or 'CopyAssociatedObject' and the *Entity Type* is not specified, the newly created object will be of the same type as the original.

If the *Create Type* is 'NewToken' and the *Token Associated Object* and/or *Token Context Object* are not specified, the references of the original token executing the Create step will be assumed.

If the *Create Type* is 'NewObject' and *Entity Type* property is specified, a newly created object of the specified type will be created. The object will have its default values for all states and properties. If the *Create Type* is 'CopyParentObject' or 'CopyAssociatedObject' and *Entity Type* is specified, much information from the original object is copied to the newly created object. This includes all table references assigned to the original, all the original object's state values, and the original's assigned network and destination. If the original is following a sequence table, the current sequence table index is copied as well. Note that if the entity type of the copy is different than the original's type, then only the state values of common inherited object states can be copied. Also, the size state values will not be copied and thus a copy of a different type will be sized according to the default size for that different type.

When an object is created using *Create Type* 'NewObject', it is placed into Free Space within the Facility Window, at the object instance location. When an object is created by *Create Type*= 'CopyParentObject' or 'CopyAssociatedObject', it is initially located in Free Space at the same location as the original object that was copied. Therefore, the user may need to use a Transfer step directly following a Create step, in order to place the new object in a specific location, such as on a Link or into a Station or Node.

Within the Advanced Options section of properties, the *Save Created Entity Reference* may be used to optionally save the name of a state variable that stores a reference to the entity created by the step (will be the last entity if multiple were created). Note that the assigned state variable may be a state on the original token executing the Create step or simply a model state variable.

There are two exits from the Create step, including the Original and the Created exits. The order in which the tokens exit the step is based on the *Token Wait Action* property value. By default, the property value is 'NewTokensExitFirst' and a token for the newly created entity (or token) will first exit from the 'Created' exit and proceed until a delay type step occurs for the token or its associated entity. Then the original token will exit the 'Original' exit.

For examples of using the Create step, please refer to the SimBits [CONWIP](#) and [DynamicallyCreatingVehicles](#).

Listed below are the properties of **Create**:

Property	Description
Create Type	The creation method. If creating copies of an existing entity, then the original entity object to copy may be specified as either the object associated with the executing token, the parent object, or as a specific object reference. 'NewObject' creates new objects of the specified instance type. 'CopyParentObject' creates a copy of the parent object. 'CopyAssociatedObject' creates a copy of the object associated with the token executing the Create step. 'CopySpecificObject' creates a copy of a specified source entity object. 'NewToken' creates a new token that may reference existing objects.
Original Entity Object	Specified if the <i>Create Type</i> is 'CopySpecificObject'. The specific original entity objects to copy.
Entity Type	Specified if the <i>Create Type</i> is anything except 'New Token'. The type of entity objects to create. If left unspecified and creating copies, then the same type as the original entity being copied will be assumed.
Token Associated	Specified if the <i>Create Type</i> is 'New Token'. The primary object reference associated

Object	with the new token(s). If left unspecified, then the associated object reference of the original token executing the Create step will be assumed.
Token Context Object	Specified if the <i>Create Type</i> is 'New Token'. The secondary object reference associated with the new token(s). If left unspecified, then the context object reference of the original token executing the Create step will be assumed.
Number To Create	The number of entity objects or tokens to create. May be specified as an expression truncated to integer.
Save Created Entity Reference	Specified if the <i>Create Type</i> is anything except 'New Token'. Optional entity reference state variable to save a reference to the created entity. Note that if the number of entities created is greater than one, then the saved reference will be to the last created entity.
Token Wait Option	<p>The wait action to be taken by the token arriving to the Create step.</p> <p>If the action is 'NoWait', then the execution of all new tokens associated with the created or referenced objects will be scheduled on the simulation's current event calendar as an early priority event. The original token will then immediately exit the step.</p> <p>If the action is 'NewTokensExitFirst', then all new tokens associated with the created or referenced objects will exit the step first before the original token exits.</p> <p>If the action is 'WaitUntilNewTokenProcessingCompleted', then the original token will wait until the processing of all new tokens associated with the created or referenced objects has been completed before exiting the step.</p> <p>If the action is 'NoWait', then the execution of all new tokens associated with created entities will be scheduled on the simulation's current event calendar as an early priority event. The original token will then immediately exit the step.</p> <p>If the action is 'NewTokensExitFirst', then all new tokens associated with created entities will exit the step first before the original token exits.</p> <p>If the action is 'WaitUntilNewTokenProcessingCompleted', then the original token will wait until the processing of all new tokens associated with created entities has been completed before exiting the step.</p>
Stock Requirements	Material stock putaway or removal requirements for each created entity.
Stock Requirements.ID Number	Option ID number for the stock requirement.
Stock Requirements.Material Name	The name of the material to putaway or remove.
Stock Requirements.Quantity	The quantity to putaway or remove.
Stock Requirements.Skip Stock Requirement Condition	Optional condition indicating whether to skip the stock requirement.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Decide

Decide

The Decide step may be used to determine the flow of a token through process logic. The Decide Step evaluates an Expression so any properties that are referenced in the Decide step must be expression properties.

The arriving token exits either the 'True' or 'False' exit point based on the specified probability or condition.

For examples of using the Decide step, please refer to the SimBits [SequentialProcessingByBatchSpecifiedInTable](#) and [MoveableOperator](#).

Listed below are the properties of **Decide**:

Property	Description
Decide Type	Indicates whether the decision is based on a probability or condition.
Expression	The probability or condition specified as an expression. If a probability, then enter the chance of selecting 'True' as a value between 0 and 1 (including values of 0 or 1). If a condition, then enter the logical condition.
Random Number Stream	The random number stream to be used by the probability-based decision type.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Delay

Delay

The Delay step delays the arriving token in the step for the specified time duration.

The *Delay Multiplier* property allows for dynamic adjustments to the *Delay Time*. If the process that is executing the Delay step is suspended, then the base delay time remaining is saved. When the process is resumed, the *Delay Multiplier* expression property is re-evaluated and the base delay time remaining multiplied by that number to schedule the end of the delay back on the simulation event calendar.

For example, suppose the *Delay Time* is specified as '1' hour and the *Delay Multiplier* expression returns the value of '2' (based on model entity state or table value). This results in a delay time of 2 hours. If the process is suspended after 0.5 hours have elapsed and the delay thus 25% completed, then the base delay time remaining is 0.75 hours. When the process is resumed, if the *Delay Multiplier* expression property now returns the value '1.5' then the end of the delay is scheduled back on the calendar to occur at $\text{TimeNow} + (0.75 \times 1.5 =) 1.125$ hours. If the *Delay Multiplier* expression property returns a negative value, then a runtime error is thrown.

NOTE: Simio will produce an error if a delay time expression returns a negative value. This can occur if you use an unbounded distribution that might produce negative values, such as a Normal Distribution, within the *Delay Time* of the Delay step. See [Distributions](#) for more information.

Listed below are the properties of **Delay**:

Property	Description
Delay Time	The duration of the delay. This property may be specified with any expression, including using a random sample from a distribution. A special case is to use the keyword 'Math.Epsilon', which causes the token or entity to be delayed to the end of the <i>current</i> time advance -- still executing at the same simulated time but after all normal events have been processed.
Delay Multiplier	Increases or decreases the delay time by some factor. Specify 2, for example, to mean that the delay is twice as long. Note that this expression is automatically reevaluated if the process is suspended and resumed, potentially adjusting the remaining delay time.
Interruptible	Indicates whether token delays at this step can be interrupted using the Interrupt step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Destroy

Destroy

The Destroy step destroys either the parent object, the executing token's associated object or the object specified as an object reference.

Only object types supporting dynamic creation may be destroyed. All processes are immediately terminated for the destroyed object. All schedule events (e.g. end of step delays) are cancelled from the calendar. If the parent is destroyed then the token does not exit the step.

Listed below are the properties of **Destroy**:

Property	Description
Destroy Type	The object to be destroyed. Specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object to be destroyed.
Record Entity Statistics	Indicates whether or not to record time in system and accrued cost statistics for the entity being destroyed.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Disengage

Disengage

The Disengage step may be used to unlock an entity's location from a location on the parent link so that the link and entity may move independently. The Engage step is used to engage an entity.

Once disengaged an entity is free to move across the link at its own speed, and the link is free to move at its own speed without being affected by the movement (or stoppage) of the entity.

Note

This Step can only be used from within a Link object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Dropoff

Dropoff

The Dropoff step may be used to attempt a transporter drop-off of a riding entity into a node.

Use the Pickup step to pick up an entity waiting to ride.

NOTE: Simio will produce an error if a delay time expression returns a negative value. This can occur if you use an unbounded distribution that might produce negative values, such as a Normal Distribution, within the *Unload Time* of the Dropoff step. See [Distributions](#) for more information.

Listed below are the properties of **Dropoff**:

Property	Description
Node Name	The name of the node that is the dropoff location. If left unspecified, and the transporter is currently at a node, then defaults to that node.
Unload Time	A delay required for unloading a rider from the transporter.
Transporter Type	The transporter object that is to perform the drop-off. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Transporter Object	Visible if <i>Transporter Type</i> is 'SpecificObject'. The specific transporter object that is to perform the drop-off.
Fail If Blocked	Indicates whether to cancel the dropoff attempt if the location where the entity is to be physically dropped into is considered blocked. If the dropoff attempt is cancelled, then the arriving token will immediately exit the 'Failed' exit point of the Dropoff step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EndRun

EndRun

The EndRun step ends the simulation run.

This step ends the current simulation run. (Note that the OnRunEnding process will not be triggered by this step during an interactive run, but instead the user must press the Stop or Reset button to trigger that process.)

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

EndTransfer

EndTransfer

The EndTransfer step may be used to indicate that the parent object, object associated with the executing token, or specific entity object reference has completed transfer into an object or station.

Note: An EndTransfer step is not required when transferring into a Node, as the EndTransfer step is already in the Node's logic. However, when transferring an entity into a station, that transfer has to be ended (Simio has to complete the transfer) otherwise the entity is still in a "transferring" state.

Listed below are the properties of **EndTransfer**:

Property	Description
Entity Type	The entity object that is ending the transfer. Specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object whose transfer is to be ended.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Engage

Engage

The Engage step may be used to lock an entity's location to a location on the parent link object. The method used to align the entity on the link is determined by the link's EntityAlignment property. The Disengage step is used to disengage an entity.

Once engaged, an entity remains engaged until it is disengaged by the Disengage step. The entity engages the link using the link's EntityAlignment property setting of 'AnyLocation' or 'CellLocation'. If 'CellLocation', then the entity can only engage the link at fixed cell locations along the link's length. The entity will wait for the next cell to arrive before engaging the link. If 'AnyLocation', then the entity can engage the link at any location without waiting. When engaged both the entity and moveable link location are co-aligned and must move in unison. When an engaged entity arrives to the end of a link it forces the link to stop to remain co-aligned at the engaged position.

Note

This Step can only be used from within a Link object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Execute

Execute

The Execute step may be used to execute a specified process.

The owner of the process can be the parent object or the associated entity. The arriving token either exits immediately or waits on the process to finish based on the *Token Wait Action*.

If the *Token Wait Action* is 'None (Continue)', the execution of the specified process is scheduled on the simulation calendar as an early current event. The original token then immediately exits the step. For example, suppose a token passes through multiple Execute steps (with action 'None (Continue)') in sequence. Once the original token stopped running, you would see the execution of the specified processes occur in the same sequence that the Execute steps were hit (FIFO order).

If the *Token Wait Action* is 'WaitUntilProcessCompleted', the specified process is immediately executed. If that process is completed with no delays, then the original token can immediately exit the Execute step and continues its processing. Otherwise, if the specified process did have delays, then the original token is held at the Execute step until that other process is completed. For example, suppose the OnRunEnding interface process has an Execute step with action 'WaitUntilProcessCompleted'. As long as the specified process runs with no delays, you would see the full logic of that specified process executed as part of the 'OnRunEnding' call.

If the *Token Wait Action* is 'WaitUntilProcessCompleted', there is an additional property available, *CopyTableRowReferencesFromCreatedToken*. If set to 'True', all Table Row References from the created Token will replace the references of the Token that is waiting at the Execute step. If set to 'False', the Token waiting at the Execute step will keep its Table Row References regardless of the created Token's references.

Listed below are the properties of **Execute**:

Property	Description
Process Name	The name of the process to execute.
Token Wait Action	The action to take while the process is executing.
Copy Table Row References From Created Token	Indicates whether to copy the created Token's Table Row References to the waiting Token after the created Token has finished processing.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Fail

Fail

The Fail step starts a downtime of type Failure Name.

Listed below are the properties of **Fail**:

Property	Description
Failure Name	The type of failure to start.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Find

Find

The Find step may be used to search the value of an expression over a specified range of one or more index variables. The expression will typically involve array variables (vectors or multi-dimensional arrays) or functions that index into collections.

Search types supported by the Find step include finding the index variable value(s) that satisfy the specified expression as a logical condition, as well as searches that minimize or maximize the expression. The arriving token exits either the 'Found' or 'NotFound' exit point of the step based on whether or not a successful result was found by the search.

The starting index of an index variable can be smaller or larger than the ending index, corresponding to a forward or backward search.

If the *Search Type* is 'Condition', then the token will exit the 'Found' exit point of the Find step with the index variable(s) set to the first value(s) within the search range that satisfy the conditional search expression. Additionally, the value 'True (1)' will be stored into the ReturnValue state of the executing token.

If the condition is not satisfied by a result within the search range, then the token will exit the 'NotFound' exit point of the Find step with the index variable(s) set to 0 and the value 'False (0)' stored into the ReturnValue state of the executing token.

If the *Search Type* is specified as 'MinimizeExpression' or 'MaximizeExpression', then the token will exit the 'Found' exit point of the Find step with the index variable(s) set to the value(s) within the search range that minimized or maximized the expression. Additionally, the minimum or maximum expression value found will be stored into the ReturnValue state of the executing token.

If multiple index variables are specified, then the search range will be conducted as a nested indexed search in the order that the index variables are listed. For example, if two index variables 'J' and 'K' are specified with search ranges 3-4 and 4-6 respectively, then the search expression will be evaluated using the following index value order:

J=3,K=4

J=3,K=5

J=3,K=6

J=4,K=4

J=4,K=5

J=4,K=6

For examples of using the Find step, please refer to the SimBits [FindA MinimumStateValue](#) or [VehicleFleetManagement](#).

Listed below are the properties of **Find**:

Property	Description
Index Variable Name	The name of the state variable used to hold the index value.
Starting Index	The integer start limit of the index range to be searched.
Ending Index	The integer end limit of the index range to be searched.
Indexed Variables (More)	Additional index variables. If multiple index variables are specified, then the search range will be conducted as a nested indexed search in the order that the index variables are listed.
Search	The objective of the search.

Type	<p>If the search type is 'Condition', then the search will stop on the first index result within the search range that satisfies the logical condition specified in the Search Expression property.</p> <p>If the search type is 'MinimizeExpression' or 'MaximizeExpression', then the search will find the index result within the search range that minimizes or maximizes the expression specified in the Search Expression property.</p>
Search Expression	<p>The search condition or expression that uses the specified index variables.</p> <p>If the search type is 'Condition' and an index result satisfying the search condition is found, then the value 'True' will be stored into the ReturnValue state of the executing token. Otherwise the value 'False' will be stored.</p> <p>If the search type is 'MinimizeExpression' or 'MaximizeExpression', then the minimum or maximum expression value found will be stored into the ReturnValue state of the executing token.</p>

Copyright 2006-2025, Simio LLC. All Rights Reserved.

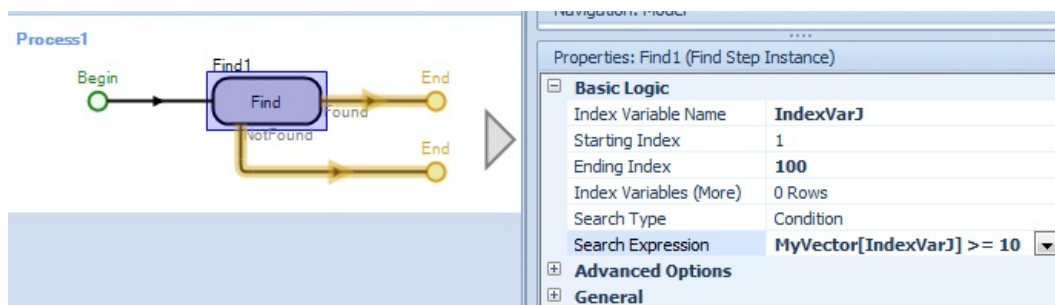
Send comments on this topic to [Support](#)

Find - Discussion and Examples

Example 1 - Conditional Search of a Vector Variable

In this example, a user would like to find a value within the 'MyVector' array that has a value greater or equal to 10.

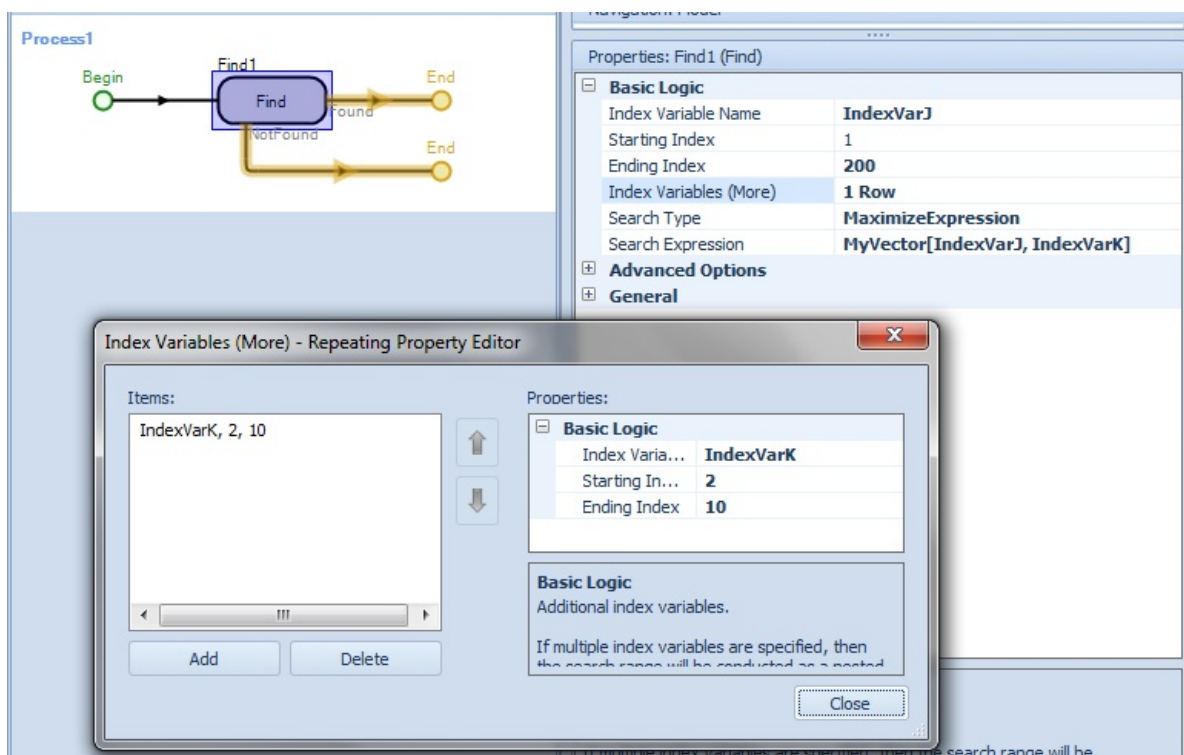
The token will exit the 'Found' exit point of the Find step with the variable 'IndexVarJ' set to the first value where the logical condition 'MyVector[IndexVarJ] >= 10' evaluates to true. The index range of the search will be from IndexVarJ = 1 to 100. If the logical condition is not satisfied by any index value within the range (none have the value greater or equal to 10), then the token will exit the 'Not Found' exit point of the Find step.



Example 2 - Searching a 2D Matrix Variable for Maximum Value

In this example, the user is trying to find the maximum value within a subset of rows and columns in a matrices.

The columns 2 through 10 in the matrix 'MyMatrix' will be searched row by row and the token will exit the Find step with the variables 'IndexVarJ' and 'IndexVarK' set to the row and column in the matrix that contains the maximum value.



Fire

Fire

The Fire step may be used to fire an object event. It can be used in conjunction with the Wait step.

For examples of using the Fire step, please refer to the SimBits [VehicleFinishesAndParksWhenOffShift](#) and [EntityStopsOnLink](#).

Listed below are the properties of **Fire**:

Property	Description
Event Name	The name of the event to fire.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Insert

Insert

The Insert step may be used to insert an object into a specified queue. Refer to the [Storage](#) element to define a custom queue in the model.

Listed below are the properties of **Insert**:

Property	Description
Queue State Name	The name of the queue state into which the object will be inserted.
Object Type	The object to be inserted into the queue. May be specified as the object associated with the executing token, the parent object or as a specific object reference.
Object	Visible if <i>Object Type</i> is 'SpecificObject'. The specific object to be inserted into the queue.
Rank Placement	If specified, this overrides the queue state's ranking rule. If the queue's ranking rule is FirstInFirstOut or LastInFirstOut, then this expression is interpreted as a specific rank index to place the inserted object. If the queue's ranking is SmallestValueFirst or LargestValueFirst, then this expression is interpreted as the inserted object's ranking value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Interrupt

Interrupt

The Interrupt step may be used to interrupt process delays. A token delaying at a process' Delay step is considered a candidate for interruption if the 'Interruptible' property of the Delay step evaluates to 'True'. To add logic for handling interrupted items (e.g., re-routing logic), the Interrupt step can create new tokens that are associated with the interrupted items. These tokens will exit the 'Interrupted' exit point of the step.

The Interrupt step may be used to model resource preemption scenarios, allowing model logic to interrupt a process delay activity that is using a resource, in order to release the resource for a higher priority activity. This approach allows very flexible preemption capability with precise control over which entities are preempted and how the preempted entities are subsequently processed.

Objects within the Standard Library that may be interrupted with the Interrupt step include the Server, Combiner and Separator. Within each of these objects, the processing time delay is 'interruptible'. To interrupt a token at a Server, Combiner or Separator, simply specify `ObjectName.OnEnteredProcessing` as the *Process Name* property of the Interrupt step.

When a token enters the Interrupt step, it will interrupt the specified process and then the token will exit the "Original" exit of the step and continue processing. Once this original token completes its steps (or begins a delay type step), a token associated with the interrupted entity is created and will leave the "Interrupted" exit of the step and continue any specified steps.

For examples of using the Interrupt step, please refer to the SimBits [InterruptibleOperator](#), [InterruptingAcrossMultipleServers](#), [InterruptingServerWithMultipleCapacity](#) and [WorkerUsesWorkSchedule_InterruptWorkingOffShift](#).

Listed below are the properties of **Interrupt**:

Property	Description
Process Name	The name of the process with interruptible delay activity.
Selection Rule	The rule used to select which process delay(s) to interrupt from one or more candidates.
Interrupted Process Action	Indicates how an interrupted process delay is to be handled once the new token created by the Interrupt step to handle the interruption occurrence ends its processing. If the action is 'EndProcess', then the processing of the interrupted token will be ended. If the action is 'ResumeDelay', then the interrupted token will resume delaying at the Delay step for its remaining delay time. If the action is 'EndDelay', then the interrupted token will end its delay and exit the Delay step.
Process Names(More)	The names of the additional processes with interruptible delay activity.
Limit	The maximum number of process delay activities to interrupt.
Save Number Interrupted	Optional discrete state variable to store the number of process delay activities that were interrupted. This state variable assignment will be applied to the original token executing the Interrupt step.
Save Remaining Time	Optional discrete state variable to store an interrupted process delay's remaining time (in hours). This state variable assignment will be applied to each token exiting the 'Interrupted' exit point of the Interrupt step.
Value Expression	The expression used to get the value for each candidate. In the expression, use the syntax <code>Candidate.[Class].[Attribute]</code> to reference an attribute of the candidates (e.g.,

Candidate.Entity.Priority).*

Filter Expression	Optional condition evaluated for each candidate that must be true for the candidate to be selected. In the expression, use the syntax Candidate.[Class].[Attribute] to reference an attribute of the candidates (e.g., Candidate.Entity.Priority).*
----------------------	---

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Move

Move

The Move step may be used to request a move from one or more moveable resources that have been seized by a specified owner. The executing token is held at the Move step until all of the requested movements are completed.

A list of objects for moving may be specified by using the Definitions / Lists and referencing the 'ListName'. Alternatively, a list of objects may be specified within a table column and referencing the 'TableName.ColumnName' as the *Resource List Name* property.

This step may be used in conjunction with the Seize step and the Release step.

Listed below are the properties of **Move**:

Property	Description
Resource Move Requests	The resources that will be requested to move.
Resource Move Requests.Resource Type	The method for specifying the resource object(s) that will be requested to move.
Resource Move Requests.Resource Name	The name of the resource object that will be requested to move.
Resource Move Requests.Resource List Name	The name of the object list from which one or more resource objects will be requested to move.
Resource Move Requests.Destination Node	The name of the specific node location that the resource(s) will be requested to move to. If not specified, and the owner object requesting the movement is an entity object at a node, then this property defaults to that owner entity's current node.
Resource Move Requests.Number Of Resources	The number of resource objects that will be requested to move.
Resource Move Requests.Move Priority	The priority of the move request if a PlanVisit or SelectVisit step is used by a resource to choose a visit destination from multiple candidates. If not specified, and the owner object requesting the movement is an entity object, then this property defaults to that entity's Priority state value.
Resource Move Requests.Move Request Order	The order in which to request moves from resources that have been seized by the owner object.
Resource Move Requests.Selection Condition	Optional condition evaluated for each seized resource that must evaluate to true for the object to be moved. In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the seized resource objects (e.g. Candidate.Object.Capacity).
Resource Move Requests.Skip Move If	Optional condition indicating whether to skip the resource move(s).

Owner Type	The object that owns the seized resources to be moved. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Owner Object	The specific object that has seized the resources to be moved.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Notify

Notify

The Notify step may be used to output a user defined trace, warning message or error, depending on the *Notification Type*.

The Warning Level in the Run ribbon is used to determine where warning messages are displayed.

A warning notification from the same “warning source” is only shown once per timeslice. For example, a warning from the Notify at time zero will only be shown once. Also note that the identification for the “warning source” from Notify steps is the process and step name. Therefore, if two Notify steps have the same name in the same process, and both gives Warnings in the same timeslice, only the first will be shown.

Note: If a Notify contains a random expression, it will be evaulated without randomness

Listed below are the properties of **Notify**:

Property	Description
Notification Type	The context of the outputted message. If specified as 'Trace', and the model trace is enabled, then the message will be displayed in the trace window. If specified as 'Warning', then the message will be handled according to the model's warning settings.If specified as 'Error', the model will terminate with the provided message.
Message Heading	Optional text used as the heading for the outputted message. May be specified as either a string expression or as a literal string of characters enclosed in double quotes.
Message Content	Optional text used as the content for the outputted message. May be specified either as a string expression or as a literal string of characters enclosed in double quotes.
Notify Condition	Optional condition that, if specified, must evaluate to true for the message to be outputted to the user.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

NotifyConstraint

NotifyConstraint

The NotifyConstraint step may be used to add a custom log entry to the model's constraint log. To enable constraint logging during the simulation run, select Enable Interactive Logging in Run, Advanced Options. These entries, by extension, will then also be seen in the Entity Gantt chart.

When a NotifyConstraint step is used, with *Notification Type* set to 'StartConstraint', Simio creates a new entry in the Constraint Log, setting the StartTime column to TimeNow. Then later, when a NotifyConstraint step is executed with *Notification Type* set to 'EndConstraint', with matching values for the other properties, Simio 'closes' that entry in the log by setting the EndTime column to the current TimeNow.

Note that the Entity Id, Entity, Facility Location and Station columns come from the specified constrained entity, while the Constraint Item Id, Constraint Item and Constraint Description columns are populated from the step's Object or Entity property.

Simio 'matches' the StartConstraint and EndConstraint steps based on the property values. As long as Constraint Type, Object or Element, and Entity Type (along with Entity Object, if appropriate) have the same values, it is considered a match.

Simio requires that the StartConstraint is encountered before a matching EndConstraint step. If not, a runtime warning is reported if an EndConstraint is executed before a matching StartConstraint (that is, if the model is trying to close a log entry that doesn't yet exist). Similarly, if a StartConstraint is executed for some combination of property values, and then another StartConstraint with those same values (that is, we've opened up a log entry, and before closing it we try to open another for the same values), a warning will be reported.

The resulting entries in the Constraint Log should populate the Owner Gantt view, just like the built-in constraint types.

Listed below are the properties of **NotifyConstraint**:

Property	Description
Notification Type	Indicates the type of notification sent to the model's constraint log. If specified as 'StartConstraint', then a new constraint log entry will be created. If specified as 'EndConstraint', then the end timestamp will be logged for the active constraint log entry whose fields match the same entity, constraint type and constraining object or element.
Constraint Type	The value of the Constraint Type field in the constraint log entry. May be specified as an expression that evaluates to a string. (Note: literal strings must be in quotes.)
Object or Element	The constraining object or element. Determines the values of the Constraint Item Id, Constraint Item, and Constraint Description fields in the constraint log entry.
Entity Type	The entity object that is being constrained. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Park

Park

The Park step may be used to move an entity object into the parking station of a node.

The *Token Wait Action* property indicates whether the token is going to wait in the step until the transferring entity 'Transferring' or 'Transferred' event is fired.

Listed below are the properties of **Park**:

Property	Description
Node Name	Name of the node owning the parking station where the entity will be parked. Defaults to the entity's current node or, if not at a node, then the first node found in the system.
Entity Type	The entity object to be parked. Specified as either the object associated with the executing token, the parent object or as a specific object reference.
Entity Object	Visible if <i>Entity Type</i> is 'Specific Object'. The specific entity object to be parked.
Token Wait Action	The wait action to be taken by the token arriving to the Park step. Options include WaitUntilTransferred and WaitUntilTransferring.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Pickup

Pickup

The Pickup step may be used to attempt a transporter pick-up of an entity waiting in the ride pickup queue of a node.

The waiting entities are examined in the order of the pickup queue. Note that both the entity and the transporter have to "Accept" the selected entity pickup. If the entity/transporter accept the pickup, the Pickup step removes the entity from the RidePickupQueue of the node and transfers it into the ride station of the transporter using the standard transfer mechanism (i.e., the EndTransfer step completes the transfer). The token is held in the Pickup step until the entity transfer is completed. If no TransferIn process handler is provided for the ride station on the transporter then a zero time transfer is provided by default. The token exits one of two exit points depending on the result of the pickup attempt. It exits the primary exit point if a pickup was successful. It exits the secondary exit if the pickup attempt failed. Entities that are picked up by a transporter may be later dropped off using the Dropoff step.

NOTE: Simio will produce an error if a delay time expression returns a negative value. This can occur if you use an unbounded distribution that might produce negative values, such as a Normal Distribution, within the *Load Time* of the Pickup step. See [Distributions](#) for more information.

For an example of using the Pickup step, please refer to the SimBit [VehicleDoingSimultaneousLoading](#).

Listed below are the properties of **Pickup**:

Property	Description
Node Name	The name of the node that is the pickup location. If left unspecified, and the transporter is currently at a node, then defaults to that node.
Selection Goal	The goal used to rank pickup preference when selecting from multiple candidates.
Load Time	A delay required for loading a rider onto the transporter.
Transporter Type	The transporter object that is to perform the pick-up. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Transporter Object	Visible if <i>Transporter Type</i> is 'SpecificObject'. The specific transporter object that is to perform the pick-up.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

PlanVisit

PlanVisit

The PlanVisit step may be used to have an entity object search for and accept a not yet reserved visit request in the system. Possible visit requests may include pickup requests initiated at Ride steps (applicable to transporter entities) as well as move requests initiated at Seize or Move steps (applicable to resource enabled entities). The SelectVisit step can be later used to have an entity actually set its current destination node to an accepted visit location.

The trip is selected based on the selection goal specified. The goal can be SmallestDistance, LargestDistance, SmallestPriority, LargestPriority or FirstInQueue. When selecting a trip, the entities in the global visit request queue are examined in order based on the selection goal. A candidate entity is not selected unless accepted by the entity. Once an entity is selected, this same process is done for the transporter. Hence both the candidate entity and the transporter have to agree to the trip before the reservation is booked.

For an example of using the PlanVisit step, please refer to the SimBit [VehicleFleetManagement](#).

Listed below are the properties of **PlanVisit**:

Property	Description
Selection Goal	The goal used to rank pickup destination preference when selecting from multiple candidates.
Entity Type	The entity object that is searching the system for a visit request. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object that is searching the system for a visit request.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

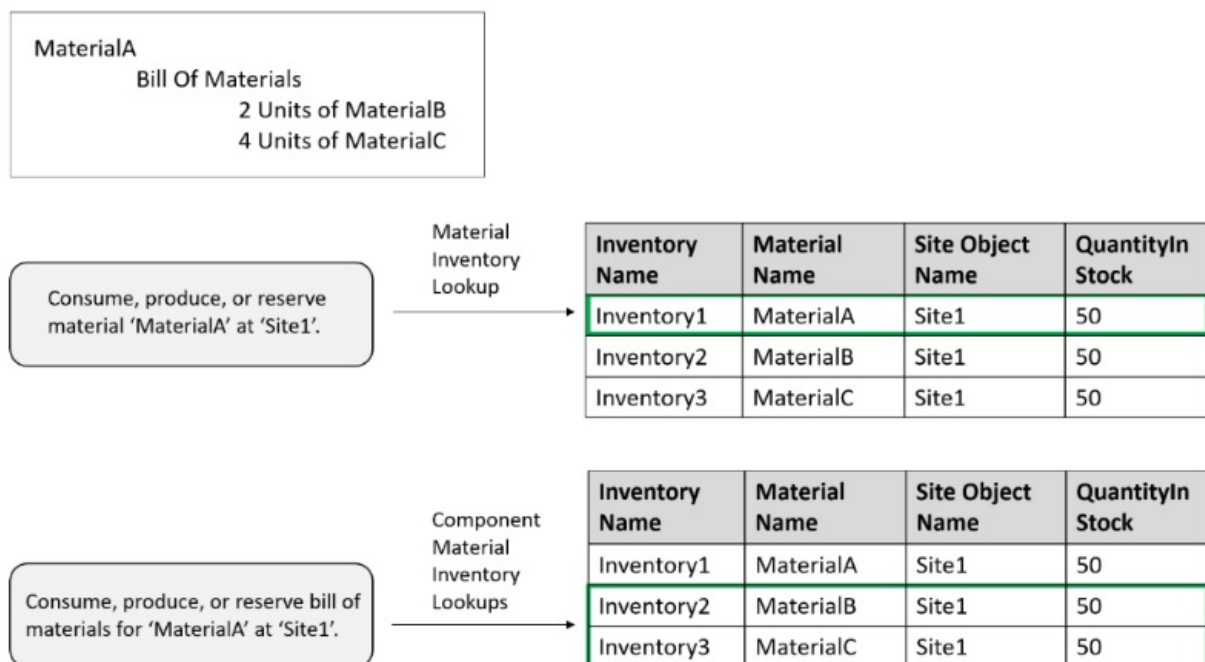
Produce

Produce

The Produce step may be used to produce stock on behalf of a specified owner. Will produce all valid alternative BOMs or alternative materials given the specified validity constraints. A *Produce Materials (More)* repeat group is provided that allows the definition of repeating material production data. Alternatively, a data table may be mapped to that repeat group.

The *Lot ID* property may be specified as a string literal enclosed in double quotes (e.g., "Lot1"), the name of a string state variable or string column in a table, or any other expression that returns a string value.

If producing a single material that is location-based inventory, then Simio will look up the Inventory element holding the specified material at the specified inventory site. If producing a bill of materials, then Simio will look up the Inventory elements holding the component materials at the specified inventory site.



If the *Inventory Site Type* property value is 'None' but a site object reference is required because the material is location-based inventory, or if an inventory reference for a material-site object pair cannot be found, then a runtime error will occur.

NOTE: Simio will produce an error if a delay time expression returns a negative value. This can occur if you use an unbounded distribution that might produce negative values, such as a Normal Distribution, within the *Production Delay Time* of the Produce step. See [Distributions](#) for more information.

If the *Record As Order Received If* condition property returns 'True' and the material is location-based inventory, then the inventory's quantity on order is automatically decremented by either the produced quantity or to zero, the latter if the produced quantity is greater than the quantity on order.

For examples of using the Produce step, please refer to the SimBits [ScheduledMaterialArrivals](#) and [WorkstationWithMaterialConsumptionAndReplenish](#).

Listed below are the properties of **Produce**:

Property	Description
Production Type	The type of material stock production. Material – A single material. BillOfMaterials – A list of component materials. BillOfMaterialsGroup – Alternative lists of component materials.

Material Name	The name of the material.
BOM Name	The name of the bill of materials.
BOM Group Name	The name of the bill of materials group.
Inventory Site Type	Indicates the fixed object that is the inventory location. Applies only to material elements whose <i>Location Based Inventory</i> property is set to 'True'.
Site Object Name	The name of the fixed object that is the inventory location.
Quantity	The quantity of the material or parent assembly.
Lot ID	Optional string value indicating the lot identifier of the material or parent assembly.
Produce Materials (More)	A repeating set of additional materials to produce.
Produce Materials (More).Production Type	Indicates whether to produce a single material or a bill of materials.
Produce Materials (More).Material Name	The material which is to be either specifically produced or whose bill of materials is to be produced.
Produce Materials (More).Inventory Site Type	Indicates the fixed object that is the inventory location. Applies only to material elements whose <i>Location Based Inventory</i> property is set to 'True'.
Produce Materials (More).Quantity	The quantity to be produced.
Produce Materials (More).Lot ID	Optional string value indicating the lot identifier of the produced material.
Produce Materials (More).Production Delay Time	An optional delay time until the material is actually added to the system. The executing token always immediately exits the Produce step, but the material production does not occur until after this time duration has elapsed.
Produce Materials (More).Skip Production If	Optional condition indicating whether to skip the material production.
Owner Type	The object that is the owner of the material production request. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Owner Object	The specific owner object that the material production is in behalf of.
Production Delay Time	An optional delay time until the material is actually added to the system. The executing token always immediately exits the Produce step, but the material production does not occur until after this time duration has elapsed.
Record As Order Received If	Indicates whether to record the produced material as a received order.
On Produced Process	Optional process to be executed when material is produced.

Immediately Try Allocate	Indicates whether to immediately try allocating the produced material to waiting material consumption requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the material or inventory allocation queues will be scheduled on the simulation's current event calendar as a late priority event.
Exclusion Expression	If specified, an expression that will be evaluated at the start of a simulation run to determine if this step should be excluded from the run. If this expression evaluates to 1, this step will be excluded, and tokens will flow directly to its primary exit. If this expression evaluates to 2, and this step has a secondary exit, the step will be excluded, and tokens will flow directly to its secondary exit. If the expression evaluates to any other value (typically 0), the step will remain active in the process.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Release

Release

The Release step may be used to release capacity of one or more resources that have been seized by a specified owner. Use the Seize step to seize resources.

A list of resource candidates to release may be specified by using the Definitions / Lists and referencing the 'ListName'. Alternatively, a list of resource candidates may be specified within a table column and referencing the 'TableName.ColumnName' as the *Resource List Name* property.

This step may be used in conjunction with the [Seize](#) step.

The Release step has an option to allow an owner object to keep the released resource capacity 'reserved' for possible later reuse. Reserved resource capacity is capacity that although unallocated has been set aside for a specific future owner thus preventing its use by anyone else. Whenever resource capacity becomes available, that capacity cannot be allocated to a seize request waiting in the resource's allocation queue unless that there is either sufficient unreserved capacity available or the owner of the seize request has sufficient reservation(s). For example, if all of a resource's available capacity has been reserved, a seize request without a reservation will have to wait in the allocation queue until either active reservations for the desired resource have been cancelled or sufficient unreserved capacity has been released or scheduled. Note that seize requests with reservations will be allowed to seize a resource even if ranked in that resource's allocation queue behind requests with no reservations.

A resource reservation will be automatically cancelled if one of the following events occurs - the *Reservation Timeout* property is used and the specified timeout period expires OR the reservation owner is an entity that is destroyed. Note that if the *Reservation Timeout* is specified as the value 'Math.EPSILON' then the resource reservation will be automatically cancelled at the end of the current simulation time step if there has been no immediate re-seize of the resource by the same owner object. When a reservation is cancelled, the *OnCapacityReservationCancelled* process for that particular resource is executed.

The [Allocate](#) step provides an option to cancel all active reservations for a resource (regardless of who has made the reservations) before attempting to allocate the resource's capacity to any waiting seize requests.

The *Immediately Try Allocate* property can be set to 'False' to allow all resources to be released at a given time period before any resource allocations are made to entities waiting in an allocation queue(s).

For examples of using the Release step, please refer to the SimBits [OverflowWIP](#), [ResourcesWithWorkSchedules](#), [SeizingVehicle](#) and [SelectingResourceFromList](#).

Listed below are the properties of **Release**:

Property	Description
Releases	The resources to be released.
Releases.Resource Type	The method for specifying the resource object(s) to release.
Releases.Resource Name	The name of the resource object to release.
Releases.Resource List Name	The name of the object list from which to select the resource object(s) to release.
Releases.Release Quantity Type	Indicates the type of resource quantity to release. If 'Specific', then the exact quantities entered in the <i>Number of Objects</i> and <i>Units Per Object</i> are released. If 'All', then all currently seized resources of the specified resource type are released.
Releases.Number Of Resources	The number of individual resource objects to release capacity units of.
Releases.Units Per	The number of capacity units to release per resource object.

Resource

Releases.Release Order	The order in which to release objects that have been seized by the owner object.
Releases.Selection Condition	An optional condition evaluated for each seized resource that must be true for the object to be eligible to release. In the condition, use the keyword 'Candidate' to reference an object in the collection of candidates (e.g. Candidate.Object.ID)
Releases.Keep Reserved If	<p>Optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects will be unable to seize the reserved capacity unless the reservation is cancelled.</p> <p>In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released (e.g., Candidate.Object.Capacity).</p> <p>Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p> <p>Leaving this property blank (no condition) is equivalent to entering False.</p>
Releases.Reservation Timeout	<p>If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation.</p> <p>In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released (e.g., Candidate.Object.Capacity).</p>
Releases.On Released Process	Optional process that is executed by a token associated with a released object after the release occurs.
Releases.SkipReleaseIf	Optional condition indicating whether to skip the release requirement.
Owner Type	The object that owns the seized resources to be released. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Owner Object	The specific object that has seized the resource capacity units to be released.
Seized Resources Filter Type	Filter type applied to the owner object's list of currently seized resources before attempting the resource releases. If 'Token', filters to only include seized resource items that are linked to a given token's execution.
Filter Token Reference	Expression returning a reference to the process token to which the seized resource items must be linked. If the expression returns 'Nothing' then no filtering occurs.
Use Strict List Referencing	If a resource release requirement is 'FromList', indicates whether the release can include only resources that have been seized from the same specified list. The default setting is 'False' and if left unchanged then the list membership will be sufficient to include a resource.
Immediately Try Allocate	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource allocation queues will be scheduled on the simulation's current event calendar as a late priority event.

Remove

Remove

The Remove step may be used to remove an object from a specified queue. There are two exits from the Remove step, including the Original and the Removed exits. A order in which the original token and the token for the removed object exit the step is based on the *Token Wait Option* property value. By default, the value is 'NewTokenExitsFirst' and the token for the removed entity will first exit from the 'Removed' exit and proceed until a delay type step occurs for the token or its associated entity. Then the original token will exit the 'Original' exit.

When the object is removed from a queue, its associated Token waiting to enact certain logic can be destroyed. Removing an object from an Allocation queue will cancel and destroy the Token waiting at the corresponding Seize step. Removing an entity from a Batch Queue will cancel and destroy the Token at the Batch step. Other Queue removals might destroy a Token at an associated Transfer step. If you would like the process to continue from one of these destroyed Tokens, consider using an Execute step to retrigger the logic.

The [Interrupt Step](#) is different, yet somewhat similar. It can be used to Interrupt a processing delay. So if you want to cancel an entity's request for a Resource's capacity or a entity's Transfer request, use a Remove Step to remove an entity from a queue. But if you want to cancel an entity that is currently within a process delay, use an Interrupt Step to cancel the delay.

Removing from a Storage Queue

The user can create their own queue within Simio and place entities in this queue (Insert Step) and remove the entities from the queue (Remove Step). Refer to the [Storage](#) element to define a custom queue in the model.

Removing from a Resource Allocation Queue

Removing an item from a Resource's Allocation Queue is very powerful and allows for what is commonly referred to as "**Reneging**". This cancels a Seize attempt. If an entity has already requested capacity of any resource type object and is therefore waiting in the Resource's Allocation Queue (i.e. Server1.AllocationQueue), the entity can be removed from the queue and therefore their request for that object's capacity is cancelled. An example of use for this feature might be when a customer has waited too long in line and decides to leave the facility without being serviced. The customer had already requested capacity of the "server" and was waiting in line for the server, but the Remove Step can cancel their request.

Removing from a Station's Entry Queue

Removing an item from a Station's EntryQueue or a Link object's EntryQueue, cancels the Transfer step transfer request. In other words, the entity no longer waits to enter a station. The token executing the Transfer step ends its processing (i.e., is destroyed) upon the canceling action.

Removing from a Link Object's Entry Queue

Removing an entity from a Link object's EntryQueue cancels the Transfer request for this entity to wait to enter the Link.

Removing from a Batch Queue

Removing an entity from a BatchLogic element's ParentQueue cancels a Batch Step activity where the entity is waiting to collect a batch. Similarly, removing an entity from a BatchLogic element's MemberQueue cancels a Batch Step activity involving a member waiting to be batched.

Removing from a RoutingGroup RouteRequest Queue

Removing an entity from a RoutingGroup element's RouteRequestQueue cancels a Route Step waiting to route this entity to a set of Blocked node destinations.

Removing from a Network VisitRequest Queue

Removing an entity from a Network (such as Global) VisitRequestQueue cancels a waiting entity ride request that has not yet been accepted by a transporter.

Removing from a Material Allocation Queue

Removing an entity from a Material AllocationQueue cancels the entity request for consumption of that particular material.

Note

Simio does not support removing from the following types of queues: Node RidePickupQueue, Entity

Listed below are the properties of **Remove**:

Property	Description
Queue State Name	The name of the queue state from which the object will be removed.
Object Type	The object to be removed from the queue. May be specified as the object associated with the executing token, the parent object, or as a specific object reference.
Object	Visible if <i>Object Type</i> is 'SpecificObject'. The specific object to be removed from the queue.
Token Wait Option	<p>The wait action to be taken by the token arriving to the Remove step.</p> <p>If the action is 'NoWait', then the execution of the new token associated with the removed object will be scheduled on the simulation's current event calendar as an early priority event. The original token will then immediately exit the step.</p> <p>If the action is 'NewTokenExitsFirst', then the new token associated with the removed object will exit the step first before the original token exits.</p> <p>If the action is 'WaitUntilNewTokenProcessingCompleted', then the original token will wait until the processing of the new token associated with the removed object has been completed before exiting the step.</p>

RemoveRows

RemoveRows

The RemoveRows step may be used to remove all existing rows or a single active row from an output table. Any table references held by tokens or objects become invalid.

When using the RemoveRows step to remove a specific single row from a table, it is important that the active row for the *Object Type* specified be set. This can be done by using the [SetRow](#) step followed by a RemoveRows step.

When objects reference an output table, they reference a row number. If a row is removed, the table rows update accordingly, but the objects still point to the same row number, so the values may be different. If the row number no longer exists, then a run time error is thrown.

Listed below are the properties of **RemoveRows**:

Property	Description
Table Name	The name of the output table from which to remove all rows.
Removal Type	Indicates whether to remove all rows from the specified table or only a specific row.
Object Type	The object or element that has a reference to the table row to be removed. May be specified as either the executing token, the object associated with the executing token, the parent object or as a specific object or element reference.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RemoveStock

RemoveStock Step

The RemoveStock step may be used to remove material stock from a storage bin. The *Stock Removal Type* property in the Stock Removals repeat group determines the source of information for the stock removal.

If the *Stock Removal Type* is 'Specific', the material type, storage bin, quantity, and quant selection condition expression are defined within the repeat group row to determine how the material is removed from the storage bin. If the *Stock Removal Type* is 'Reservation', the RemoveStock step will use the information from the specified stock reservation to remove material from the storage bin. For more information about stock reservations, see [Stock Reservations](#).

If there is less stock in the storage bin than the RemoveStock step attempts to remove, the removal will fail, though other removals specified by the Stock Removals repeat group will still be attempted.

For more information about modeling material storage and retrieval, see [Material Storage – Discussion and Examples](#).

Listed below are the properties of the **RemoveStock** step:

Property Name	Description
Stock Removals	The material stock to remove.
Save Stock Quant Reference	Optional state variable to save a reference to each removed or updated stock quant.
Stock Removals.Stock Removal Type	The stock removal type. Specific – Remove a specified quantity of material from a specified storage bin. Reservation – Remove material stock using a specified stock removal reservation.
Stock Removals.Storage Bin Name	The name of the storage bin to remove material stock from.
Stock Removals.Material Name	The name of the material to remove.
Stock Removals.Quantity	The quantity of material to remove.
Stock Removals.Stock Quant Selection Condition	Optional condition evaluated for each candidate stock quant in the first-in-first-out list of stock quants currently present in the storage bin. If left unspecified then the first stock quant in the list that is the same material will be selected to decrease its quantity in stock. In the expression, use the syntax <code>Candidate.StockQuant.[Attribute]</code> to reference an attribute of the candidate stock quant (e.g., <code>Candidate.StockQuant.QuantityAvailable</code>).
Stock Removals.Stock Reservation Reference	The stock removal reservation reference.
Stock Removals.Skip Stock Removal Condition	Optional condition indicating whether to skip the material stock removal.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RemoveStorageLocations

RemoveStorageLocations

The RemoveStorageLocations step may be used to remove a child storage location from a parent storage area. Use the AddStorageLocations step to add a child storage location.

Listed below are the properties of **RemoveStorageLocations**

Property	Description
RemoveStorageLocations.StorageLocationRemovals	The child storage locations to remove.
RemoveStorageLocations.ParentStorageAreaName	The name of the parent storage area.
RemoveStorageLocations.ChildStorageLocationType	The type of child storage location to remove.
RemoveStorageLocations.ChildStorageAreaName	The name of the child storage area to remove.
RemoveStorageLocations.ChildStorageBinName	The name of the child storage bin to remove.
RemoveStorageLocations.SkipStorageLocationRemovalIf	Optional condition indicating whether to skip the child storage location removal.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Repair

Repair

The Repair step ends a downtime of type Failure Name.

Listed below are the properties of **Repair**:

Property	Description
Failure Name	The type of failure to end.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

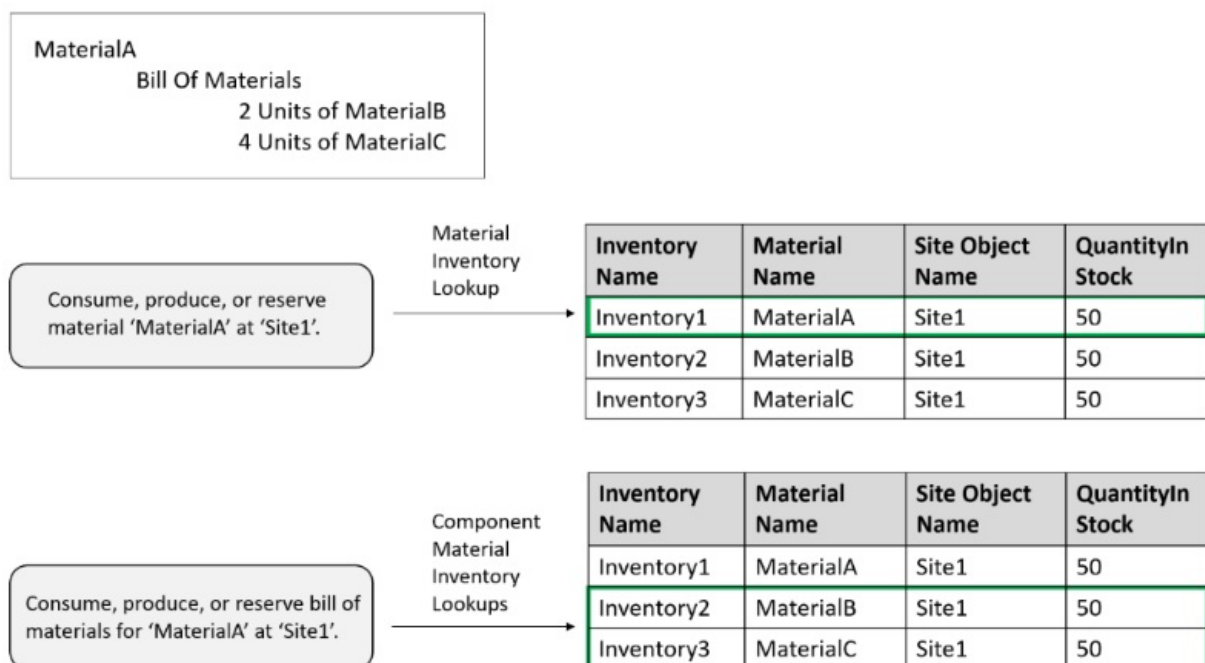
Reserve

The Reserve step may be used to reserve stock for a specified owner. Will reserve all valid alternative BOMs or alternative materials given the specified validity constraints. Other objects will be unable to use a reserved item unless the reservation is cancelled. Use the [UnReserve](#) step to cancel a resource or material reservation.

Resource is used in this context to refer to Resources, Workers and Vehicles in the Standard Library. Reserved resource capacity is capacity that, although unallocated, has been set aside for a specific future owner thus preventing its use by anyone else without a reservation.

The *Lot ID* property may be specified as a string literal enclosed in double quotes (e.g., "Lot1"), the name of a string state variable or string column in a table, or any other expression that returns a string value.

If *Consumption Type* for materials is a single 'Material' that is location-based inventory, then Simio will look up the Inventory element holding the specified material at the specified inventory site. If *Consumption Type* is a 'Bill of Materials', then Simio will look up the Inventory elements holding the component materials at the specified inventory site.



If the *Inventory Site Type* property value is 'None' but a site object reference is required because the material is location-based inventory, or if an inventory reference for a material-site object pair cannot be found, then a runtime error will occur.

A *Reservations (More)* repeat group allows a user to define repeating reservations either through entry into the repeat group or by using a data table mapped to that repeat group.

If the name of the resource to reserve is a member of a population (i.e., worker or vehicle), the *Resource Name* can be specified as either the specific resource within the population, such as 'Truck[1]' or 'Truck[2]' or may alternatively be specified by simply the resource name, such as 'Truck'. If the population of that resource is greater than 1, keep in mind that reserving 'Truck' would always reserve the first member of the population, Truck[1] unless explicitly specified.

Many functions, such as *Capacity.Reserved*, *Capacity.ReservedTo(owner)* and *ReservationOwners.** (see the [Functions, States and Events for All Intelligent Objects](#) page) are updated when a resource is reserved or unreserved. The capacity reserved of a given resource by various entities can be greater than the actual capacity of the resource. Allocating the resource to the entity then decreases the *Capacity.Reserved* and *Capacity.ReservedTo(owner)* function values. Keep in mind that the function *ReservedTransporter* for a given entity object is NOT related to the Reserve step. If the entity object currently has made a reservation to ride on a transporter (i.e., by indicating *Ride On Transporter* 'True' in a TransferNode), then this function returns a reference to that transporter once it is allocated. Otherwise, the Nothing keyword is returned. This value is set to the reserved transporter until the entity is actually loaded and has moved into the transporter's RideStation.

See also the [Reservations](#) page for more details.

Listed below are the properties of **Reserve**:

Property	Description
Reservation Type	The type of material stock consumption. Material – A single material. BillOfMaterials – A list of component materials. BillOfMaterialsGroup – Alternative lists of component materials.
Resource Name	The name of the resource object to reserve.
Number Capacity Units	The number of capacity units to reserve (for <i>Reservation Type</i> 'Resource').
Consumption Type	Indicates whether the material to reserve is a single material or bill of materials.
Material Name	The name of the material.
Inventory Site Type	Indicates the fixed object that is the inventory location. Applies only to material elements whose <i>Location Based Inventory</i> property is set to 'True'.
Site Object Name	The name of the fixed object that is the inventory location.
BOM Name	The name of the bill of materials.
BOM Group Name	The name of the bill of materials group.
Quantity	The quantity of the material or parent assembly.
Lot ID	Optional string value indicating the lot identifier of the material or parent assembly.
Reservation Timeout	An optional wait time before automatically cancelling the reservation.
Reservations (More)	A repeating set of additional reservations.
Reservations (More).Reservation Type	Indicates whether to reserve a resource, a single material or a bill of materials.
Reservations (More).Resource Name	The name of the resource object to reserve.
Reservations (More).Material Name	The name of the material which is to be either specifically reserved or whose bill of materials is to be reserved.
Reservations (More).Number Capacity Units	The number of capacity units to reserve (for <i>Reservation Type</i> 'Resource').
Reservations (More).Quantity	The quantity to reserve (for <i>Reservation Type</i> 'Material' or 'BillOfMaterial').
Reservations (More).Lot ID	Optional string value indicating the lot identifier of the reserved material.
Reservations (More).Reservation Timeout	An optional wait time before automatically cancelling the reservation.

Owner Type	The object that will own the reservation. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Owner Object	The specific object that will own the reservation.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ReserveBins

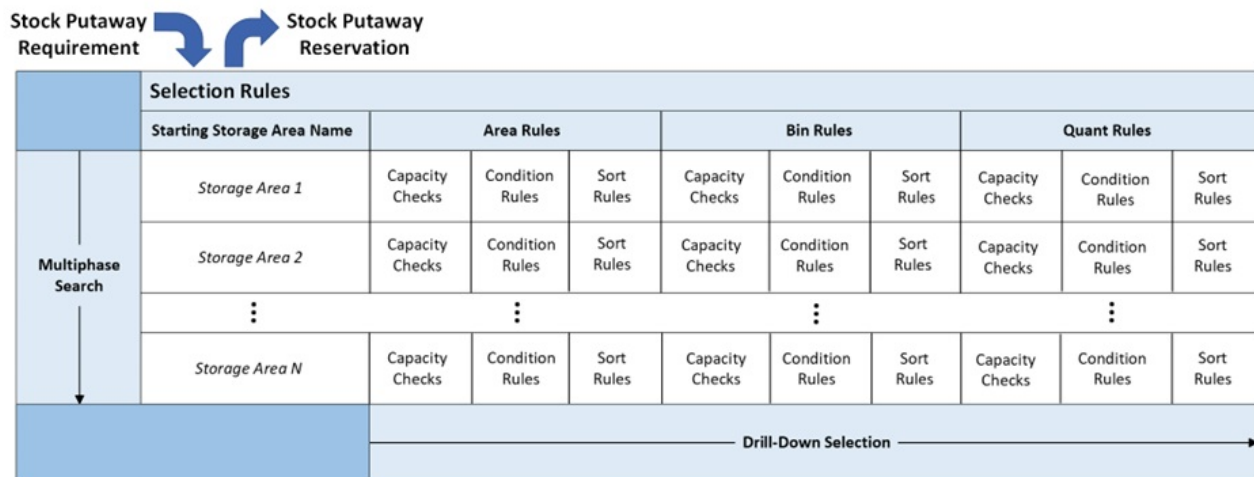
ReserveBins Step

The ReserveBins step may be used to reserve storage bin capacity for an entity's material stock requirements. Using the Selection Rules repeat group and the stock requirements, the ReserveBins step will attempt to select a storage bin and reserve a portion of its capacity. Once storage bin capacity is reserved, an associated stock reservation will be created, and a new token will depart from the Reserved exit. The created reservation in conjunction with the AddStock step can be used to add material to the reserved storage bin capacity, completing the reservation. See the [AddStock](#) step for more information.

The original token will evaluate each requirement but will not necessarily wait until all requirements have been completed before exiting the step. Instead, the ReserveBins step will attempt to create reservations to satisfy each requirement, starting from the first requirement, until no more reservations can be made. The executing token will then depart the step from the Original exit.

Selection Rules

The general approach used to model selection rules in a ReserveBins step is illustrated in the figure below. An entity's unreserved stock requirements are considered separately (one-by-one) in the order listed on the entity. In a selection rule expression, the syntax `StockRequirement.[Attribute]` may be used to reference an attribute of the stock putaway requirement currently being evaluated. For example, `StockRequirement.Material`.



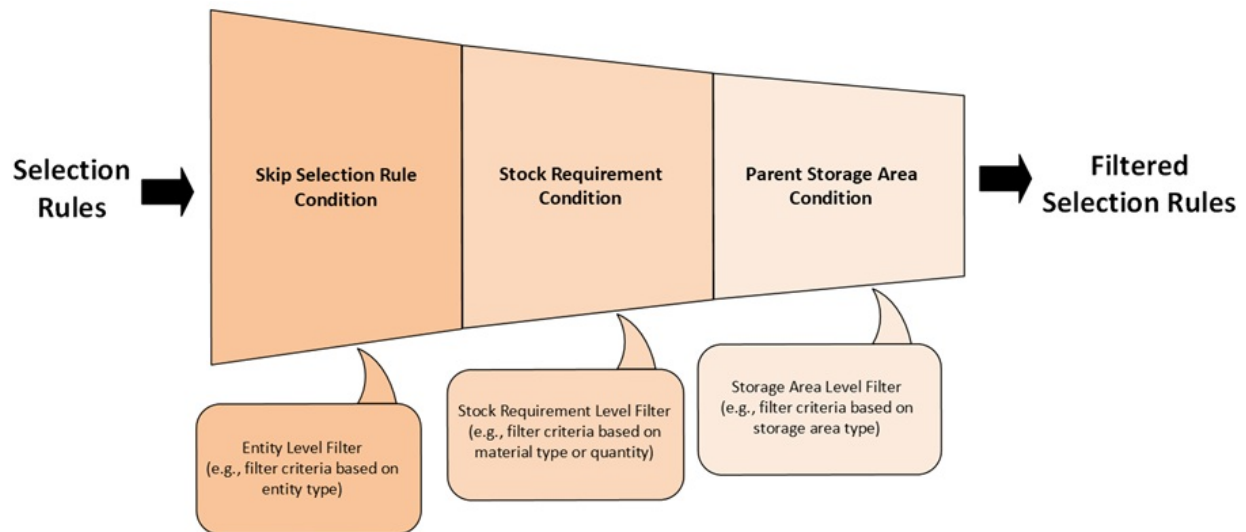
The selection rules may be configured as a multiphase search strategy with each selection rule group looking only in a specified storage area. For example, first search in 'Area1'. If that search is unsuccessful in creating a stock putaway reservation then search in 'Area2', and so forth.

Drill-down selection within a parent storage area first considers any storage area levels, then the storage bin level, then the stock quant level if adding to existing stock. Hierarchy levels in the drill-down may also be skipped. See the following Selection Rule Behavior section for more information.

The system automatically performs capacity checks for each storage area, storage bin, or stock quant candidate. Split putaway allocations are allowed, so one requirement can be partially completed. This also means one requirement could create multiple putaway reservations. Custom conditions and sorting rules to evaluate storage area, storage bin, or stock quant candidates may be specified and sorting can be single or multiple level sorts.

Filtering Selection Rules

The ReserveBins step supports three types of selection rule filters, as shown below. All selection conditions are optional and allow rules to conditionally be ignored or skipped.



The *Skip Selection Rule If Condition* property is an optional condition indicating whether to always skip a selection rule. It is evaluated once per step execution and is useful as an entity-level filter. For example, given an entity with a Boolean state variable named "InternalOrder", the expression "MyEntity.InternalOrder == True" would cause the step to only use the corresponding selection rule when the entity associated with the stock requirements had its InternalOrder state set to 'False'.

The *Stock Requirement Condition* is an optional condition indicating whether a selection rule applies to a particular stock requirement. It is evaluated when resolving a stock requirement's selection rules and is useful as a stock requirement-level filter. For example, given a material element called "Material1", the expression "StockRequirement.Material == Material1" would cause the corresponding rule to only be applied if the material associated with the current stock requirement is Material1.

The *Parent Storage Area Condition* is an optional condition indicating whether a selection rule applies when searching within a particular parent storage area. It is evaluated whenever checking a rule and is useful as a storage area level filter. For example, given a storage area element named "Area1", the expression "Candidate.StorageArea == Area1" would cause the corresponding rule to only be applied if the candidate storage area is Area1.

Selection Rule Behavior

The behavior of a selection rule group, i.e. a subset of selection rules with the same *Starting Storage Area Name*, depends on what rules are defined. The table below describes the selection rule behavior of each possible selection rule configuration.

The selection rule configuration might change based on the *Parent Storage Area Condition* when resolving the rules for each selected area. If a selection rule becomes applicable or ceases to be applicable when the selection rules are resolved for the new parent storage area, the search behavior could change. For example, if rules are evaluated as in Case 1 and then the *Parent Storage Area Condition* of the defined area rules cause them to no longer apply, the selection rule behavior would switch from Case 1 to Case 2 in the middle of the search.

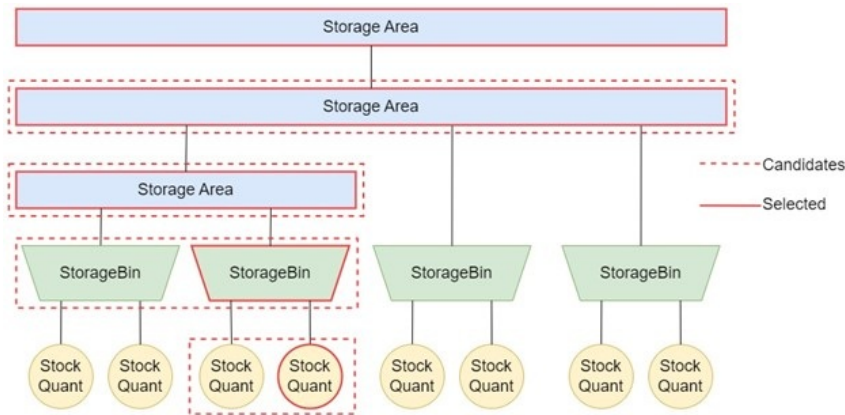
Case	Applicable Selection Rules			Selection Candidates		
	Area Rules?	Bin Rules?	Quant Rules?	Area Candidates	Bin Candidates	Quant Candidates
1	Yes	Yes	Yes	Child Areas	Child Bins	Selected Bin Quants
2	No	Yes	Yes	N/A	Descendant Bins	Selected Bin Quants
3	Yes	No	Yes	Child Areas	N/A	Child Bin Quants
4	No	No	Yes	N/A	N/A	Area Quants
5	Yes	Yes	No	Child Areas	Child Bins	N/A
6	No	Yes	No	N/A	Descendant Bins	N/A
7	Yes	No	No	Child Areas	N/A	N/A
8	No	No	No	N/A	N/A	N/A

The terms "Child" and "Descendant" are distinct. A child indicates that the object is directly contained by the parent, whereas a descendant object is either directly or indirectly contained by the parent. Thus, a child is always a descendant, but a descendant is not always a child. For instance, if StorageBin1 is in StorageArea2, StorageBin1 is both a descendant and child of StorageArea2. If StorageArea2 is in StorageArea1, StorageBin1 is only a descendant of StorageArea1, not a

child.

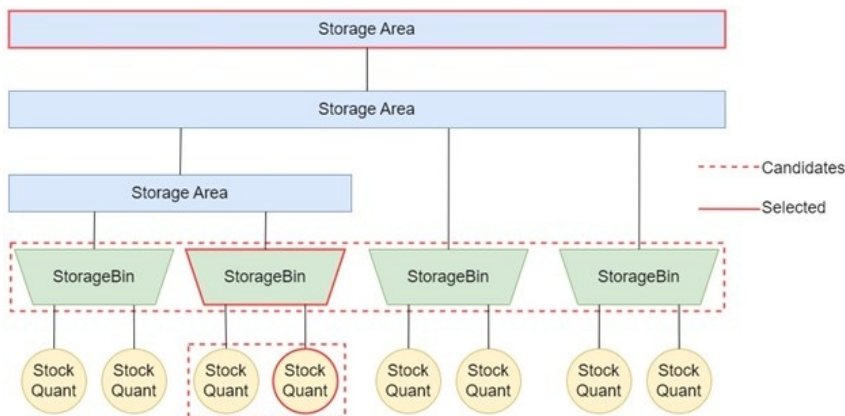
Case 1 – Area rules, bin rules, and quant rules

If all rule types are present, the ReserveBins step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and all rule types are still applicable, the step will use the bin rules to select one of the selected storage area's child bins. Once a bin has been selected, the step will use the quant rules to select one of the selected storage bin's child quants. If no quant is selected, a new quant will be created.



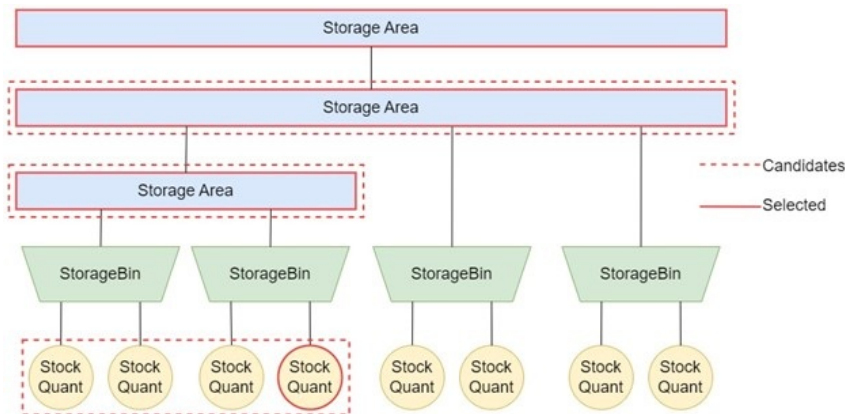
Case 2 – Bin rules and quant rules

If only bin and quant rules are defined, the ReserveBins step will begin the search by selecting a descendant bin in the starting storage area. Once a bin has been selected, the step will use the quant rules to select one of the selected storage bin's quants. If no quant is selected, a new quant will be created.



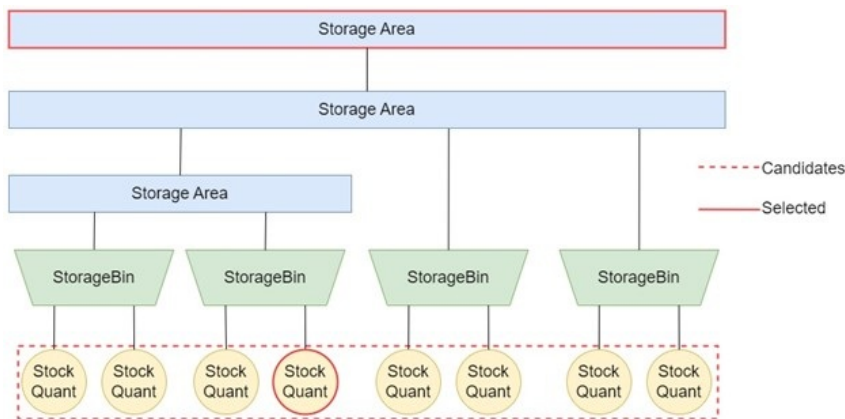
Case 3 – Area rules and quant rules

If only area and quant rules are defined, the ReserveBins step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and only area and quant rules are still applicable, the step will select a quant from the selected storage area's child bins.



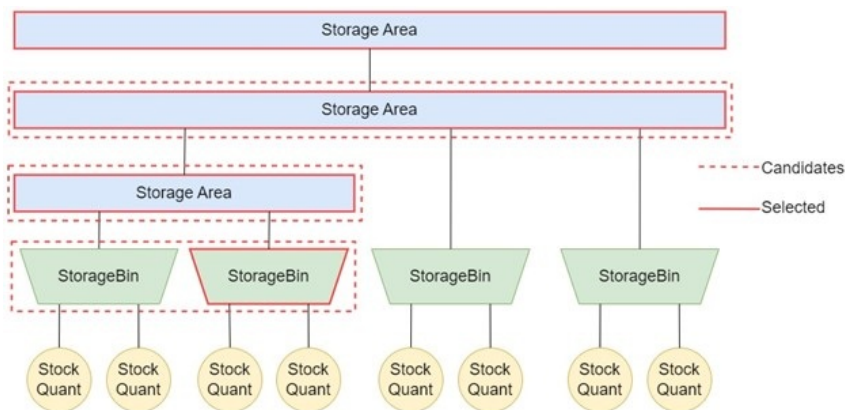
Case 4 – Quant rules only

If only quant rules are defined, the ReserveBins step will attempt to select a quant from all descendant quants of the starting storage area. If no quant is selected, no reservation is created, and the step will begin evaluating the next group of selection rules.



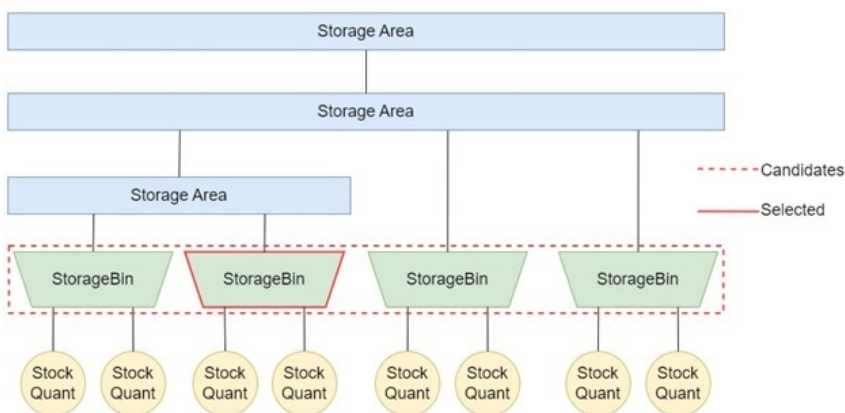
Case 5 – Area rules and bin rules

If only area and bin rules are defined, the ReserveBins step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and only area and bin rules are still applicable, the step will select a storage bin from the selected storage area's child bins. If a storage bin is selected, the step will create a new quant.



Case 6 – Bin rules only

If only bin rules are defined, the ReserveBins step will attempt to select a storage bin from all descendant storage bins of the starting storage area. If a storage bin is selected, the step will create a new quant.



Case 7 – Area rules only

If only area rules are defined, the ReserveBins step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and there are no applicable bin or quant rules, the search will end. However, it is possible that other rules become applicable when the selection rules are resolved due to the Parent Storage Area Condition. Depending on the applicable selection rules, the behavior will change to be consistent with

the new selection rule configuration.

Case 8 – No rules

The ReserveBins step will do nothing because there are no rules defined. Defining a new rule in the group will change interpretation of the rules. Adding a bin rule would cause the group of rules to be executed as shown in Case 6. Adding a quant rule would cause the group of rules to be executed as shown in Case 4.

Listed below are the properties of the **ReserveBins** step:

Property Name	Description
Selection Rules	The selection rules used to reserve storage bin capacity for the specified entity's material stock putaway requirements.
Save Stock Reservation Reference	Optional state variable to save a reference to each created stock putaway reservation.
Entity Type	The entity containing the material stock putaway requirements. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference. Refer to the Create step to add material stock putaway or removal requirements to a created entity.
Entity Object	The specific entity containing the material stock putaway requirements.
Selection Rules.Starting Storage Area Name	The name of the storage area to start the search. If multiple starting storage area names are specified in the selection rules' list then those storage areas will be searched in the order listed.
Selection Rules.Rule Type	The selection rule type. Area - The rule is used to select a child storage area of the parent storage area. Bin - The rule is used to select a descendant storage bin of the parent storage area. Quant - The rule is used to select a stock quant from the first-in-first-out list of stock quants currently present in the parent storage area or the selected storage bin. Drill-down selection within a parent storage area first considers any storage area levels, then the storage bin level, then the stock quant level. At least one Area, Bin, or Quant rule must be defined otherwise that hierarchy level is skipped in the drill-down. For example, if only Quant rules are defined (no Area or Bin rules), then the selection will drill down to the stock quants currently present in the parent storage area. As opposed to first selecting a child storage area or descendant storage bin.
Selection Rules.Sub Rule Type	The selection sub-rule type. If multiple condition rules are specified then those rules are treated as AND conditions. If multiple sort rules are specified then those rules are treated as multi-level sorts in the order listed. If no sort rules are specified then the implicit rule used is select the first candidate storage area, storage bin, or stock quant found that is eligible. If a storage bin is selected but not an existing stock quant, then a new quant will be created and added to the storage bin.
Selection Rules.Selection Condition	The condition evaluated for each candidate storage area, storage bin, or stock quant that must be true for the candidate to be selected. In the expression, use the syntax Candidate.[Class].[Attribute] to reference an attribute of the candidates (e.g., Candidate.StorageArea.CurrentTotalCapacityAvailable, Candidate.StorageBin.CurrentTotalCapacityAvailable, or Candidate.StockQuant.QuantityAvailable).
Selection Rules.Sort Value Expression	The expression evaluated for each candidate storage area, storage bin, or stock quant to get its numeric value used for sorting. In the expression, use the syntax Candidate.[Class].[Attribute] to reference an attribute of the candidates (e.g., Candidate.StorageArea.CurrentTotalCapacityAvailable, Candidate.StorageBin.CurrentTotalCapacityAvailable, or Candidate.StockQuant.QuantityAvailable).
Selection Rules.Sort Order	The sort order. Ascending – Sorts values in ascending (smallest to largest) order. Descending – Sorts values in descending (largest to smallest) order.
Selection Rules.Skip Selection Rule	Optional condition indicating whether to skip the selection rule.

If

Selection Rules.Stock Requirement Condition	Optional condition indicating whether the selection rule applies to a particular stock putaway requirement. In the expression, use the syntax <code>Stock.Requirement.[Attribute]</code> to reference an attribute of the stock requirement (e.g., <code>Stock.Requirement.Material</code>).
---	---

Selection Rules.Parent Storage Area Condition	Optional condition indicating whether the selection rule applies when searching within a particular parent storage area. In the expression, use the syntax <code>Candidate.StorageArea.[Attribute]</code> to reference an attribute of the parent storage area (e.g., <code>Candidate.StorageArea.CurrentTotalCapacityAvailable</code>).
---	---

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ReserveStock

ReserveStock Step

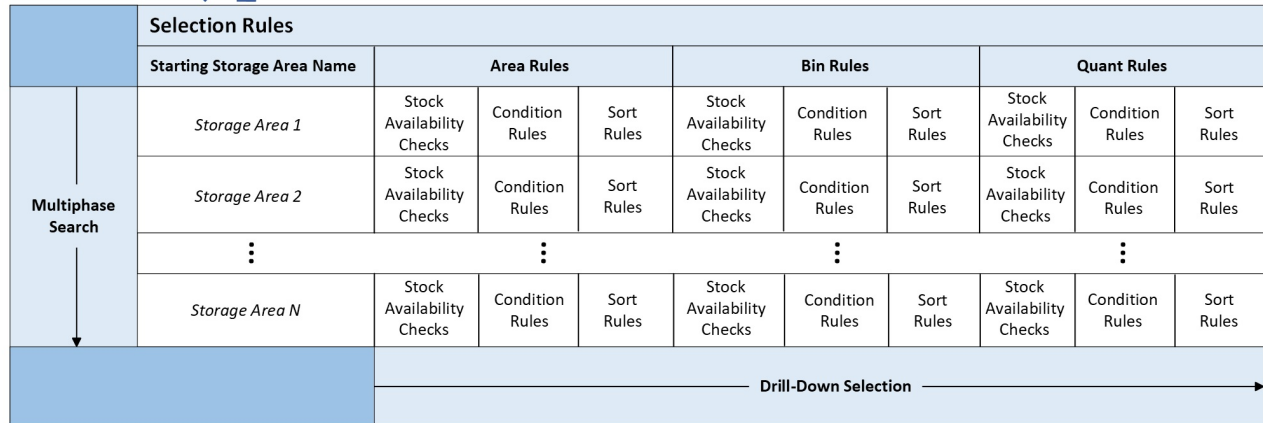
The ReserveStock step may be used to reserve storage bin stock for an entity's material stock removal requirements. Using the Selection Rules repeat group and the removal requirements, the ReserveStock step will attempt to select and reserve available stock in a for removal. Once the stock is reserved, an associated stock reservation will be created, and a new token will depart from the Reserved exit. ReserveStock step is used in conjunction with the RemoveStock step to remove material from the quantity reserved. See the [RemoveStock](#) step page for more information.

The original token will evaluate each requirement but will not necessarily wait until all requirements have been completed before exiting the step. Instead, the ReserveStock step will attempt to create reservations to satisfy each requirement, starting from the first requirement, until no more reservations can be made. The executing token will then depart the step from the Original exit.

Selection Rules

The general approach used to model selection rules at a ReserveStock step is illustrated in the figure below. **At least one quant rule must be defined for the ReserveStock step to find existing stock.** An entity's unreserved stock removal requirements are considered separately (one-by-one) in the order listed on the entity. In a rule expression, the syntax "StockRequirement[Attribute]" may be used to reference an attribute of the stock removal requirement currently being evaluated. For example, "StockRequirement.Material".

Stock Removal Requirement → Stock Removal Reservation



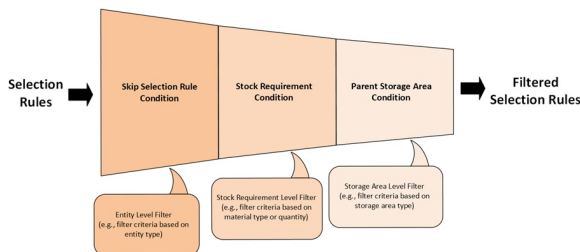
The selection rules may be configured as a multiphase search strategy with each search phase looking only in a specified storage area. For example, first search in 'Area1'. If that search is unsuccessful creating a stock removal reservation, then search in 'Area2', and so forth.

Drill-down selection within a parent storage area first considers any storage area levels, then the storage bin level, then the stock quant level. Hierarchy levels in the drill-down may also be skipped. See the following Selection Rule Behavior section for more information.

The system automatically performs a stock availability check for each storage area, storage bin, or stock quant candidate, meaning that a storage area or storage bin will not be selected if it has no stock matching the requirement. Split removal allocations are allowed, so one requirement can be partially completed. This also means one requirement could create multiple removal reservations. Custom conditions and sorting rules to evaluate storage area, storage bin, or stock quant candidates may be specified and sorting can be single or multiple level sorts.

Filtering Selection Rules

The ReserveStock step supports three types of selection rule filters, as shown below. All selection conditions are optional and allow rules to conditionally be ignored or skipped.



The *Skip Selection Rule If Condition* property is an optional condition indicating whether to always skip a selection rule. It is evaluated once per step execution and is useful as an entity-level filter. For example, given an entity with a Boolean state variable named "InternalOrder", the expression "MyEntity.InternalOrder == True" would cause the step to only use the corresponding selection rule when the entity associated with the stock requirements had its InternalOrder state set to 'False'.

The *Stock Requirement Condition* is an optional condition indicating whether a selection rule applies to a particular stock requirement. It is evaluated whenever resolving a stock requirement's selection rules and is useful as a stock requirement-level filter. For example, given a material element called "Material1", the expression "StockRequirement.Material == Material1" would cause the corresponding rule to only be applied if the material associated with the current stock requirement is Material1.

The *Parent Storage Area Condition* is an optional condition indicating whether a selection rule applies when searching within a particular parent storage area. It is evaluated whenever checking a rule and is useful as a storage area level filter. For example, given a storage area element named "Area1", the expression "Candidate.StorageArea == Area1" would cause the corresponding rule to only be applied if the candidate storage area is Area1.

Selection Rule Behavior

The behavior of a selection rule group, i.e. a subset of selection rules with the same *Starting Storage Area Name*, depends on what rules are defined. The table below describes the selection rule behavior of each possible selection rule configuration.

The selection rule configuration might change based on the Parent Storage Area Condition when resolving the rules for each selected area. If a selection rule becomes applicable or ceases to be applicable when the selection rules are resolved for the new parent storage area, the search behavior could change. For example, if rules are evaluated as in Case 1 and then the *Parent Storage Area Condition* of the defined area rules cause them to no longer apply, the selection rule behavior would switch from Case 1 to Case 2 in the middle of the search.

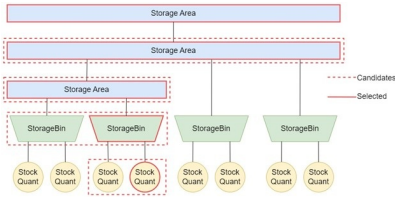
Case	Applicable Selection Rules			Selection Candidates		
	Area Rules?	Bin Rules?	Quant Rules?	Area Candidates	Bin Candidates	Quant Candidates
1	Yes	Yes	Yes	Child Areas	Child Bins	Selected Bin Quants
2	No	Yes	Yes	N/A	Descendant Bins	Selected Bin Quants
3	Yes	No	Yes	Child Areas	N/A	Child Bin Quants
4	No	No	Yes	N/A	N/A	Area Quants
5	Yes	Yes	No	Child Areas	Child Bins	N/A
6	No	Yes	No	N/A	Descendant Bins	N/A
7	Yes	No	No	Child Areas	N/A	N/A
8	No	No	No	N/A	N/A	N/A

The terms "Child" and "Descendant" are distinct. A child indicates that the object is directly contained by the parent, whereas a descendant object is either directly or indirectly contained by the parent. Thus, a child is always a descendant.

but a descendant is not always a child. For instance, if StorageBin1 is in StorageArea2, StorageBin1 is both a descendant and child of StorageArea2. If StorageArea2 is in StorageArea1, StorageBin1 is only a descendant of StorageArea1, not a child.

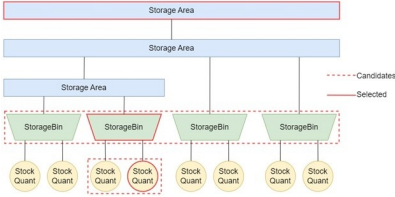
Case 1 – Area rules, bin rules, and quant rules

If all rule types are present, the ReserveStock step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and all rule types are still applicable, the step will use the bin rules to select one of the selected storage area's child bins. Once a bin is selected, the step will use the quant rules to select one of the selected storage bin's child quants. If no quant is selected, no reservation is created, and the step will begin evaluating the next group of selection rules.



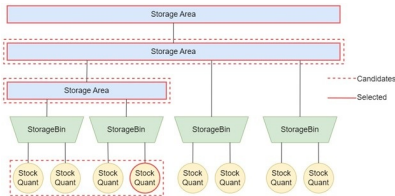
Case 2 – Bin rules and quant rules

If only bin and quant rules are defined, the ReserveStock step will begin the search by selecting a descendant bin in the starting storage area. Once a bin has been selected, the step will use the quant rules to select one of the selected storage bin's quants. If no quant is selected, no reservation is created, and the step will begin evaluating the next group of selection rules.



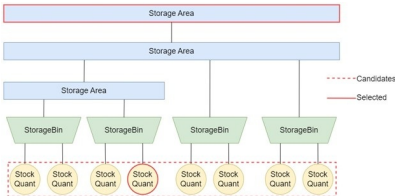
Case 3 – Area rules and quant rules

If only area and quant rules are defined, the ReserveStock step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and only area and quant rules are still applicable, the step will select a quant from the selected storage area's child bins. If no quant is selected, no reservation is created, and the step will begin evaluating the next group of selection rules.



Case 4 – Quant rules only

If only quant rules are defined, the ReserveStock step will attempt to select a quant from all descendant quants of the starting storage area. If no quant is selected, no reservation is created, and the step will begin evaluating the next group of selection rules.



Case 5 – Area rules and bin rules

If only area and bin rules are defined, the ReserveStock step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and there are no applicable bin or quant rules, the search will end. However, it is possible that other rules become applicable or cease to be applicable when the selection rules are resolved due to the Parent Storage Area Condition. Depending on the applicable selection rules, the behavior will change to be consistent with the new selection rule configuration.

Case 6 – Bin rules only

If only bin rules are applicable, the ReserveStock step will attempt to select a storage bin from all descendant storage bins of the starting storage area. After a bin is selected, because there are no applicable quant rules, the search will end.

Case 7 – Area rules only

If only area rules are defined, the ReserveStock step will begin the search by selecting a child area of the starting storage area. If an area is selected, the step will continue selecting a child area of each selected area, resolving the rules in the group each time a new area is selected. If no other areas can be selected and there are no applicable bin or quant rules, the search will end. However, it is possible that other rules become applicable when the selection rules are resolved due to the Parent Storage Area Condition. Depending on the applicable selection rules, the behavior will change to be consistent with the new selection rule configuration.

Case 8 – No rules

The ReserveStock step will do nothing because there are no rules defined. Defining a new quant rule in the group will change this behavior to the behavior described in Case 4.

Listed below are the properties of the **ReserveStock** step:

Property Name	Description
Selection Rules	The selection rules used to reserve storage bin capacity for the specified entity's material stock putaway requirements.
Save Stock Reservation Reference	Optional state variable to save a reference to each created stock putaway reservation.
Entity Type	The entity containing the material stock putaway requirements. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference. Refer to the Create step to add material stock putaway or removal requirements to a created entity.
Entity Object	The specific entity containing the material stock putaway requirements.
Selection Rules.Starting Storage Area Name	The name of the storage area to start the search. If multiple starting storage area names are specified in the selection rules' list then those storage areas will be searched in the order listed.
Selection Rules.Rule Type	The selection rule type. Area - The rule is used to select a child storage area of the parent storage area. Bin - The rule is used to select a descendant storage bin of the parent storage area. Quant - The rule is used to select a stock quant from the first-in-first-out list of stock quants currently present in the parent storage area or the selected storage bin. Drill-down selection within a parent storage area first considers any storage area levels, then the storage bin level, then the stock quant level. At least one Area, Bin, or Quant rule must be defined otherwise that

	hierarchy level is skipped in the drill-down. For example, if only Quant rules are defined (no Area or Bin rules), then the selection will drill down to the stock quants currently present in the parent storage area. As opposed to first selecting a child storage area or descendant storage bin.
Selection Rules.Sub Rule Type	The selection sub-rule type. If multiple condition rules are specified then those rules are treated as AND conditions. If multiple sort rules are specified then those rules are treated as multi-level sorts in the order listed. If no sort rules are specified then the implicit rule used is select the first candidate storage area, storage bin, or stock quant found that is eligible.
Selection Rules.Selection Condition	The condition evaluated for each candidate storage area, storage bin, or stock quant that must be true for the candidate to be selected. In the expression, use the syntax Candidate.[Class].[Attribute] to reference an attribute of the candidates (e.g., Candidate.StorageArea.CurrentTotalCapacityAvailable, Candidate.StorageBin.CurrentTotalCapacityAvailable, or Candidate.StockQuant.QuantityAvailable).
Selection Rules.Sort Value Expression	The expression evaluated for each candidate storage area, storage bin, or stock quant to get its numeric value used for sorting. In the expression, use the syntax Candidate.[Class].[Attribute] to reference an attribute of the candidates (e.g., Candidate.StorageArea.CurrentTotalCapacityAvailable, Candidate.StorageBin.CurrentTotalCapacityAvailable, or Candidate.StockQuant.QuantityAvailable).
Selection Rules.Sort Order	The sort order. Ascending – Sorts values in ascending (smallest to largest) order. Descending – Sorts values in descending (largest to smallest) order.
Selection Rules.Skip Selection Rule If	Optional condition indicating whether to skip the selection rule.
Selection Rules.Stock Requirement Condition	Optional condition indicating whether the selection rule applies to a particular stock putaway requirement. In the expression, use the syntax StockRequirement.[Attribute] to reference an attribute of the stock requirement (e.g., StockRequirement.Material).
Selection Rules.Parent Storage Area Condition	Optional condition indicating whether the selection rule applies when searching within a particular parent storage area. In the expression, use the syntax Candidate.StorageArea.[Attribute] to reference an attribute of the parent storage area (e.g., Candidate.StorageArea.CurrentTotalCapacityAvailable).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RestartRun

The RestartRun step may be used to set the ending time of the simulation run to the current time. This will cause the run to end once all simulation events scheduled for the current time have been processed. Interactive mode will require the next run to be started manually, by selecting the Stop button and then selecting the Run button (without Reset). Otherwise, a new run will be automatically started, treated as a restarted run for the same scenario replication instance if running in planning or experiment mode.

If in interactive mode, using the Reset button to re-initialize the model to its starting conditions will cancel a restart command and set the run count (Run.RunNumber) back to 1.

Note: While in Interactive Mode, if anything in the model changes between passes, press Reset to start over at the first pass.

While this step is available in all editions of Simio, this is an RPS only feature so models with the RestartRun step in it will not run in any versions except RPS.

There are no properties of **RestartRun**.

See also the *Keep Values Between Runs* property within [State Columns](#) in Data Tables, as well as [Output Tables](#).

See the [Single-Pass vs Multi-Pass Simulation Approach](#) page for more discussion.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Resume

Resume

The Resume step may be used to resume model behavior that has been suspended using the Suspend step. The Resume step can resume a process, the movement of an object, a flow regulator's capability or the batching of a Batch Logic element.

If the *Resume Type* is 'Process', the owner of the process can be the parent object or the associated object of the executing token. If the *Token Match Condition* for a Resume step is specified, then only the tokens currently in process that satisfy the condition will have their execution resumed. It is common to use 'Candidate.ModelEntity == Entity' and/or '(Candidate.Token.AssociatedObject == Token.ContextObject)' as part of an expression in the *Token Match Condition* property to be sure to correctly match a potential candidate to resume with the correct token/entity entering the step.

If the *Resume Type* is 'ObjectMovement', the movement of the parent or associated object or specific object is resumed. The object must be either a link or an agent. If the *Resume Type* is 'RegulatorFlow', the name of the flow regulator to resume flow capability is specified. If the *Resume Type* is 'BatchLogic', the name of the Batch Logic element is required.

A *Resume Actions (More)* repeat group allows a user to define multiple resumed actions either through entry into the repeat group or by using a data table mapped to that repeat group.

For examples of using the Resume step, please refer to the SimBits [EntityStopsOnLink](#), [VehicleFinishesAndParksWhenOffShift](#) and [VehicleStopsWhenServerOffShift](#).

Listed below are the properties of **Resume**:

Property	Description
Resume Type	The type of behavior or logic to resume. May be Process, ObjectMovement or RegulatorFlow.
Process Name	The name of the process to resume.
Token Match Condition	Optional match condition used to filter the tokens executing the process. Only tokens currently in process that satisfy this condition will be resumed. In the expression, use the syntax Candidate.[TokenClass].[Attribute] or Candidate.[TokenAssociatedObjectClass].[Attribute] to reference an attribute of either the candidate process tokens themselves or the objects associated with those process tokens (e.g., Candidate.Token.TimeInProcess or Candidate.Entity.TimeInSystem).
Object Type	The object whose movement is to be resumed. May be specified as either the object associated with the executing token, the parent object or as a specific object reference.
Flow Regulator Name	The name of the regulator whose flow capability is to be resumed.
Batch Logic Name	The name of the Batch Logic element to resume.
Resume Actions(More)	A repeating set of resumed actions.
Resume Actions(More).Resume Type	The type of behavior or logic to resume. May be Process, ObjectMovement or RegulatorFlow.
Resume Actions(More).Process Name	The name of the process to resume.
Resume	Optional match condition used to filter the tokens executing the process. Only tokens

Actions(More).Token Match Condition	currently in process that satisfy this condition will be resumed. In the expression, use the syntax Candidate.[TokenClass].[Attribute] or Candidate.[TokenAssociatedObjectClass].[Attribute] to reference an attribute of either the candidate process tokens themselves or the objects associated with those process tokens (e.g., Candidate.Token.TimeInProcess or Candidate.Entity.TimeInSystem).
Resume Actions(More).Object Type	The object whose movement is to be resumed. May be specified as either the object associated with the executing token, the parent object or as a specific object reference.
Resume Actions(More).Flow Regulator Name	The name of the regulator whose flow capability is to be resumed.
Resume Actions(More).Skip Resume If	Optional condition indicating whether to skip the resume action.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Ride

Ride

The Ride step may be used to initiate a transporter ride request for an entity object at a node.

When executing a Ride step, if the rider entity attempts to select a transporter from a group of candidates (e.g., from a list or from a population of some specific transporter type), the preference will be to select a transporter resource that the entity has reserved. For example, if an entity is attempting to seize a transporter from a list that contains candidates 'Worker1', 'Worker2', and 'Worker3', and the entity has an active resource reservation for 'Worker2', then the entity will have a preference to select 'Worker2' irrespective of the Ride step's specified reservation method and selection goal.

Additionally, the Ride step provides a *Token Wait Action* property in its advanced options. This property controls whether the token executing the Ride step immediately continues to the next process step, or whether the token waits until the rider entity starts or has finished transferring onto a selected transporter.

Listed below are the properties of **Ride**:

Property	Description
TransporterType	The transporter type that may be selected to ride on.
Transporter Name	The name of the specific transporter type that the entity may ride on.
Transporter List Name	The name of the list of transporter types that the entity may ride on.
Reservation Method	The method used to select and reserve a transporter object to ride on. 'ReserveClosest' will find the transporter that is closest on a Network or physically closest (if no network) to the TransferNode. If there are two that are of equal distance, the node's <i>Selection Goal</i> property will be used to determine which transporter to select. 'ReserveBest' simply looks to the <i>Selection Goal</i> and <i>Selection Criteria</i> properties and finds the Transporter that best fits the goal, regardless of the vehicle's current location. The 'FirstAvailableAtLocation' will cause the entities to wait for a Transporter to arrive to the node. This method should only be used when the vehicle has a <i>Routing Type</i> of 'Fixed Route' or when the user specifically moves the vehicle to an entity location. Otherwise, this method doesn't request the vehicle to move to the entity location.
Selection Goal	The goal used to rank transporter preference when multiple candidates are available.
Selection Expression	The expression criteria, evaluated for each candidate transporter object, used with a Smallest Value or Largest Value selection goal. Use the keyword Candidate at the beginning of the expression.
Selection Condition	An optional condition evaluated for each candidate transporter object that must be true for the transporter to be eligible to ride on. Use the keyword Candidate at the beginning of the expression.
Token Wait Action	The wait action to be taken by the token arriving to the Ride step.

Route

Route

The Route step may be used with a [RoutingGroup](#) element to route an entity object to a destination selected from a list of candidate nodes.

It is also possible to reserve materials and resources for consumption at an upstream node prior to routing to the destination node. This is done through the *Route Constraint Logic* repeat group and related [Constraint Logic](#) element.

The Required Materials repeat group on the Route step is now deprecated and no longer displayed in new models, replaced by the new *Route Constraint Logic* repeat group. The *Route Constraint Logic* repeat group allows a user to easily separate different constraint types into multiple [Constraint Logic](#) element references and thus tables (e.g., separate tables defining the material and secondary resource availability requirements for a Route step).

Note that models built prior to Sprint 203 may contain the Required Materials repeating property editor and related properties, which can be found within the Deprecated Properties area. To edit deprecated properties, go to File, Settings and under GUI area, set the *Display Deprecated Properties in the Properties Window* to 'True'.

Listed below are the properties of **Route**:

Property	Description
Routing Group Name	The name of the routing group from which a destination node will be selected.
Selection Goal	The goal used to select a destination node from the routing group.
Source Node	Optional node whose location will be used for the 'distance to' calculation instead of the routing entity, used with SmallestDistance or LargestDistance selection goal.
Value Expression	The expression criteria, evaluated for each candidate destination, used with a Smallest Value or Largest Value selection goal. Use the keyword Candidate at the beginning of the expression.
Blocked Destination Rule	The rule used to manage destination selection if there are destinations in the list of candidate nodes that are considered to be blocked. If the rule is 'Select Any', then whether or not destinations are considered blocked will be ignored. If the rule is 'Select Available Only', then only a destination that is not currently blocked may be assigned, and if all destinations are blocked, then the token will be held at the Route step until a destination becomes available. If the rule is 'Prefer Available', then a destination that is not currently blocked will be preferred, but if all destinations are blocked, then the best destination per the selection goal will still be immediately assigned.
Entity Type	The entity object to be routed. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific object to be routed.
Selection Condition	An optional condition evaluated for each candidate destination that must be true for the destination node to be selected. Use the keyword Candidate at the beginning of the expression.
Route Constraint Logic	Constraint logic used to enforce additional constraints on the entity's route request.
Route	The name of the Constraint Logic element used to enforce additional constraints on the

Constraint Logic.Constraint Logic Name	entity's route request.
Random Number Stream	The random number stream to be used if the destination <i>Selection Goal</i> is 'Random'.
Required Materials	Additional material availability constraints enforced on the entity's route request. NOTE: This repeat group has been deprecated. Use the <i>Route Constraint Logic</i> property instead. To edit the deprecated repeat group, go to File, Settings under the GUI area and set the <i>Display Deprecated Properties</i> in the Properties Window value to 'True'.

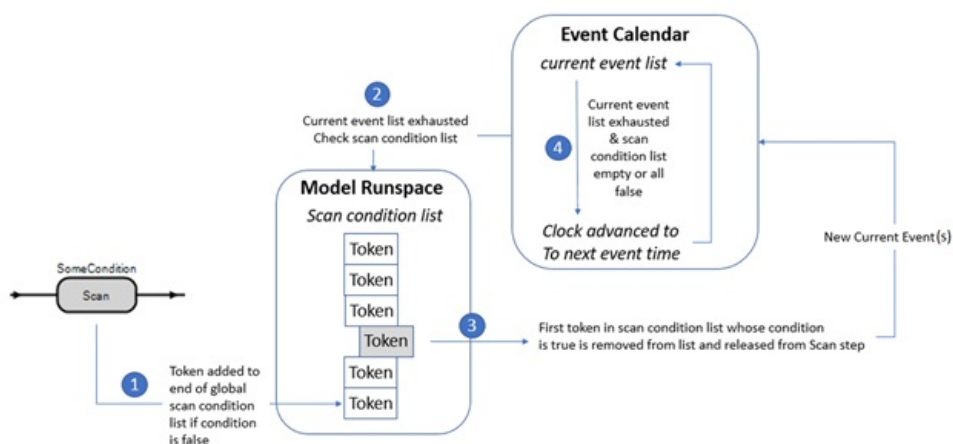
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Scan

The Scan step may be used to hold a process token at the step until a specified condition is true. The logical condition may be specified as any arbitrary expression.

1. When a token arrives at a Scan step, the scan condition is evaluated. If the condition is true, then the token is permitted to exit the step without delay. Otherwise, the token is added to a global scan condition list and is held at the Scan step until the condition is detected to be true.
2. If there are any tokens waiting on the global scan condition list, Simio will check that list as the last operation before advancing the simulation clock to the time of the next event in the system. The scan condition list is always checked in FIFO order.
3. The first token found in the scan condition list whose condition is true will be removed from the list and released from the Scan step where it has been waiting. The token is then allowed to move through process logic until it is either delayed or destroyed. Once all current events on the simulation's event calendar have again been exhausted, the scan condition list is rechecked. Note that this one-at-a-time release mechanism allows a released token to potentially change state values that will prevent other tokens waiting at Scan steps from being released.
4. The one-at-a-time release of tokens from Scan steps continues until either the scan condition list is empty or all conditions evaluate to false.



Conditions Involving the Clock

Note that if the logical condition specified for the Scan step involves the simulation clock (e.g., the condition `DateTime.Hour(Run.TimeNow) == 17` to hold an arriving token until 5:00 pm) then the polled waiting approach illustrated above will not detect such a condition as true at exactly 5:00 pm unless there is a system event scheduled to occur at that time. For example, by including a Timer element in the model that is firing an event every hour.

Momentary State Changes

Momentary state changes occurring in zero time will not be detected by a Scan step. For example, suppose a token is waiting at a Scan step for the condition `Server1.ResourceState == 0` (wait until Server1 is idle) and, at some point in time, that server is released but then immediately seized again by an entity waiting in the server's allocation queue. Which thus causes the server's resource state value to change from busy to idle and then immediately back to busy again. Since the checking of a model's scan condition list does not happen until after all events for the current time have been processed, that momentary change of the server's resource state to idle will go unnoticed and the token will simply continue waiting at the Scan step.

Effects Of Interactive Animation

When running interactively (with animation) the animation frame updates are themselves scheduled on the calendar. Thus, due to the way the Scan step operates, scan condition evaluations may occur more frequently (every animation frame update) than they would otherwise.

Additional information on the Scan step can be found in the [Scan - Discussion and Examples](#) page.

For an example of using the Scan step, please refer to the SimBit [BatchingProcessUsingScanOrWaitStepToControlBatchSize](#)

Listed below are the properties of **Scan**:

Property	Description
Condition	Logical condition that must be true for a token to pass through the step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

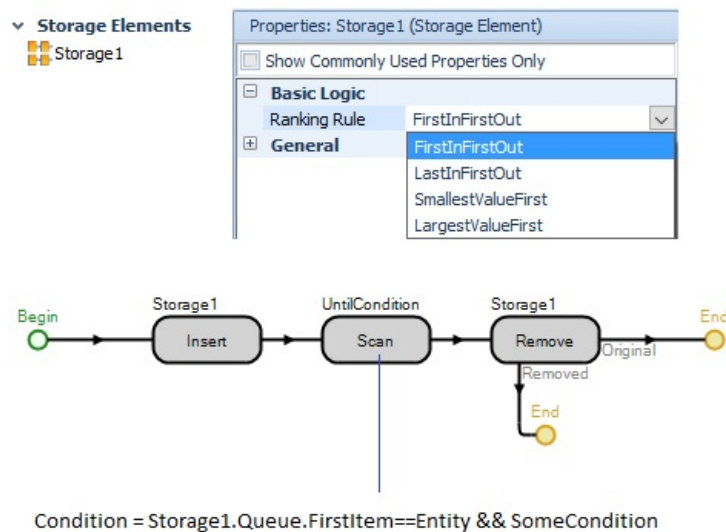
Send comments on this topic to [Support](#)

Scan - Discussion and Examples

Using a Ranked Queue

The figure below illustrates an example of how to release tokens from a Scan step using a ranked queue that is different from the default FIFO behavior. Right before the Scan step, use an Insert step to insert the token's associated entity into a specified Storage element's queue. Then, in the scan condition, include the constraint that the token's associated entity must be at the front of the queue. Finally, when a token has been released from the Scan step, use a Remove step to remove the token's associated entity from the queue.

Releasing Tokens from Scan Step Using Ranked Queue



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Search

Search

The Search step may be used to search a collection of objects or table rows. New tokens associated with any found items will exit the 'Found' exit point of the step.

The *Collection Type* for searching may be an entity population, a queue state, rows within a table, lists of objects/nodes/transporters, resources that have been seized, resource or reservation owners for a resource, network or node links, or object instance.

Prior to Simio sprint 166, the 'Forward' and 'Backward' *Search Type* property values would implicitly sum the *Search Expression*. Any Search steps added to a model prior to Sprint 166 convert the 'Forward' to 'ForwardSumExpression' and 'Backward' to 'BackwardSumExpression' automatically for compatibility in case the sum was used in the model.

The *Match Condition* property is commonly used to filter the searched items based on a particular condition. If searching objects, as in an entity population or queue contents, the Candidate keyword should be used (e.g., `Candidate.ModelEntity.Priority == 3`).

A *Search Expression* may be specified that is evaluated in the context of each item found by the search. This expression property is available when *Search Type* is 'MinimizeExpression', 'MaximizeExpression', 'ForwardSumExpression' or 'BackwardSumExpression'. The sum of the expression for the found items will be stored into the *Save Search Expression Sum* property value, which defaults to the *ReturnValue* state of the original executing token. The 'Token.ReturnValue' or state variable referenced in the *Save Search Expression Sum* property can then be referenced later on in the process.

The Search step may be used to create new tokens that are associated with, or point to, the items found by the search. These tokens will exit the 'Found' exit point of the step. If the search is done on a *Collection Type* such as 'EntityPopulation', 'QueueState' or 'SeizedResources' where there is an associated object that can be found, the 'Found' token will be associated with that found object (thus the associated object's states are accessible for reviewing, changing, etc.). However, if the *Collection Type* of the Search step is 'TableRows' for example, then the 'Found' token has a pointer to the associated row in the table (and not to any particular object). Note that if searching table rows, the table row references of the original token are copied to any new tokens (starting in Search steps added in Sprint 166+).

There are two exits from the Search step labeled 'Original' and 'Found'. The *Token Wait Action* property determines the order in which the tokens exit the Search step. By default ('NewTokensExitFirst'), the tokens for the found entity(s) will exit the 'Found' exit and proceed until a delay type step occurs for the token or its associated entity. Then the original token will exit from the 'Original' exit.

The original token always continues out of the Original segment, regardless of whether or not the Search step finds anything. All steps that exist in the Original segment leaving the Search step will always be executed. A new token will only be created in the Found segment if something is found in the Search step. So if something was found by the Search step, all steps that exist in the Found segment will be executed. But, if nothing was found by the Search step, then none of the steps that exist in that Found segment will be executed (because a token was never created for that segment).

Note: The token exiting the Original branch has the sum of the Search Expressions and the token(s) on the Found branch have the individual Search Expression value.

Model trace provided for the Search step during the simulation run includes the number of items and exact items found by the search. Additionally, if the *Save Index Found* and *Save Number Found* options are used, the trace includes the state variable assignment information.

For examples of using the Search step, please refer to the SimBits [CONWIP](#), [SearchTables](#), [RoutingWithoutPaths](#), [VehicleFinishesAndParksWhenOffShift](#), [VehicleFleetManagement](#), and [VehicleStopsWhenServerOffShift](#).

Additional information on the Search step can be found in the [Search - Discussion and Examples](#) page.

Listed below are the properties of **Search**:

Property	Description
Collection Type	The type of collection to be searched. Note that a special collection type 'ObjectInstance' is available to search any individual object in the system as a single item list. An object instance search is typically useful for creating a new process token that is associated within an already known object.

Entity Type	The entity population to be searched.
Object Instance Name	The name of the object instance to be searched.
Object List Name	The name of the object list to be searched.
Owner Type	The object that owns the seized resources to be searched. May be specified as either the object associated with the executing token, the parent object or as a specific object reference.
Owner Object	The specific owner object whose list of seized resource objects is to be searched.
Queue State Name	The name of the queue state to be searched. Examples of a public Queue State are: Input@Server1.RidePickupQueue, Server1.Processing.Contents, Vehicle.VisitRequestQueue
Table Name	The name of the table to be searched.
Search Related Rows Only	If the searched table has primary key/foreign key relationships, then this property indicates whether or not to include only related rows in the search. If this property is set to False, then the search will include all rows in the table regardless of any table relationships.
Node List Name	The name of the node list to be searched.
Transporter List Name	The name of the transporter list to be searched.
Node Name	The name of the node whose collection of drawn inbound and outbound links is to be searched.
Network Name	The name of the network whose collection of link members is to be searched.
Resource Type	The resource object whose list of current owners is to be searched. Specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Resource Object	The specific resource object whose list of current owners is to be searched.
Search Type	<p>The direction or objective of the search.</p> <p>If the search type is 'Forward' or 'Backward', then the collection will be searched for the items in either forward or backward order, respectively.</p> <p>If the search type is 'MinimizeExpression' or 'MaximizeExpression', then the collection will be searched for the items that minimize or maximize the sum of the specified <i>Search Expression</i>.</p> <p>If the search type is 'ForwardSumExpression' or 'BackwardSumExpression', then the collection will be searched in the forward or backward direction while also summing the specified <i>Search Expression</i>. Please refer to paragraph 3 above for additional information.</p>
Match Condition	<p>Optional match condition used to filter the items searched. Only items that satisfy this condition will be selected.</p> <p>If searching a collection of objects, then in the expression use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate objects (e.g., Candidate.Entity.Priority).</p>
Search Expression	Expression to be evaluated for each item in the searched collection and summed. If searching a collection of objects, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute

of the candidate objects (e.g., Candidate.Entity.Priority).

Save Search Expression Sum	Optional discrete state variable to save the sum of the specified <i>Search Expression</i> for the item(s) found by the search.
Limit	The maximum number of items to find.
Starting Index	The one-based starting index of the search. If defaulted, then the starting index is the first item in the collection if forward search or the last item in the collection if a backward search.
Ending Index	The one-based ending index of the search. If defaulted, then the ending index is the last item in the collection if a forward search or the first item in the collection if a backward search.
Save Index Found	Optional discrete state variable to save the one-based index of the last item found in the collection by the search.
Save Number Found	Optional discrete state variable to save the total number of items in the collection found by the search.
Copy Over Table Row References	Indicates whether to copy table row references over from the original token executing the Search step to new tokens associated with the found items.
Set Table Row Conflict Resolution	<p>The conflict resolution type when setting the table row reference for a new token associated with a found row in the searched table. Applies only if the <i>Copy Over Table Row References</i> property is set to True (due to potential for conflicts with the table row references copied over from the original token executing the Search step).</p> <p>Abort - The table row references copied over from the original token will be kept and no reference set to the found row in the searched table.</p> <p>Replace - The table row references copied over from the original token will be discarded, replaced by a reference to the found row in the searched table.</p>
Token Wait Option	<p>The wait action to be taken by the token arriving to the Search step.</p> <p>If the action is 'NoWait', then the execution of all new tokens associated with found items will be scheduled on the simulation's current event calendar as an early priority event. The original token will then immediately exit the step.</p> <p>If the action is 'NewTokensExitFirst', then all new tokens associated with found items will exit the step first before the original token exits.</p> <p>If the action is 'WaitUntilNewTokenProcessingCompleted', then the original token will wait until the processing of all new tokens associated with found items has been completed before exiting the step.</p>

Copyright 2006-2025, Simio LLC. All Rights Reserved.

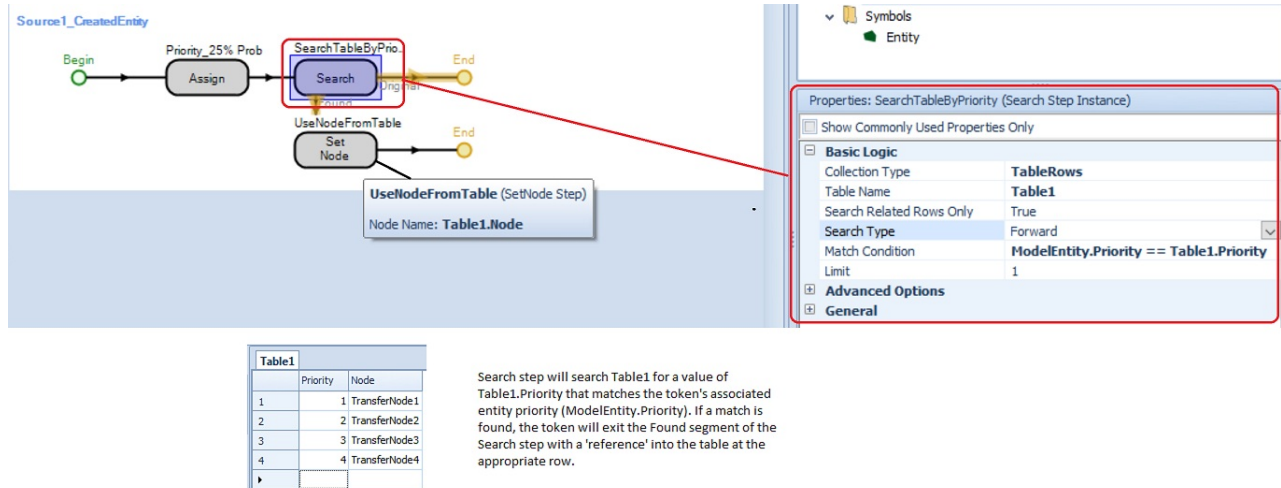
Send comments on this topic to [Support](#)

Search - Discussion and Examples

Example 1 - Using Search step to search a Table for a particular value

In this example, a user would like to search through a Simio Data Table to see which row matches a certain criteria.

Assume an entity has a discrete state on it called Priority and assume a Data Table contains a column named Priority. Each entity has a value of either 1, 2, 3 or 4 in its Priority. The column of Priority in the Data Table contains four rows, which consist of Priority as 1, 2, 3, 4. The Data Table also contains a column that is a Node Object Reference property (named Node), which contains the node where a particular entity should go. A token that is associated with an entity searches a table, using a Search step, to find the row where the value of ModelEntity.Priority equals the value in the table column Table1.Priority. In the Found segment of the Search step, a SetNode step will set the destination node of the entity to Table1.Node.



This above example can be found in the [SearchTables](#) simbit. Additionally, the three scheduling type examples within the Examples folder utilize the Search step for performing various tasks.

Example 2 - Using Search step to search a Queue for an entity with certain characteristics

In this example, we are searching for a particular entity within a queue state named 'WaitingArea.Queue'. The match condition is specified as 'ModelEntity.TimeCreated == Candidate.Entity.TimeCreated'. The term *ModelEntity.TimeCreated* refers to the entity's associated token that has entered the Search step. The term *Candidate.Entity.TimeCreated* refers to the collection of entities that are located within the WaitingArea queue. Any time a collection of items are searched, the keyword 'Candidate' should be used in the condition to explicitly reference a candidate entity within the collection.

The state variable WhichOne will store the index value into the queue of the entity (if any) that is found. This Search step can be found within the [UsingAStorageQueue](#) simbit.

Properties: SearchWaitingArea (Search Step Instance)

Show Commonly Used Properties Only

Basic Logic

Collection Type	QueueState
Queue State Name	WaitingArea.Queue
Search Type	Forward
Match Condition	ModelEntity.TimeCreated == Candidate.Entity.TimeCreated
Limit	1

Advanced Options

Starting Index	
Ending Index	
Save Index Found	WhichOne
Save Number Found	
Token Wait Action	NewTokensExitFirst
Exclusion Expression	

General

Example 3 - Searching a Table with Related Rows

In this example, an entity has an explicit row reference to a Data Table with a Key. The user would like to use a Search step to find rows in the associated Foreign Key Table.

Consider the following Key Table. The entity executing this process has an explicit row set to 2. So the Key is 'B'.

Key Table	
	Name
1	A
2	B
3	C

The Search step's default setting for *Search Related Rows Only* is 'True'. If we Searched the Table2 with a Foreign Key relating back to KeyTable, the Search step will search 4 related rows and will find 4 rows.

Key Table		Table2	
	Name	IP1	IP2
1	A	1	1
2	A	2	2
3	A	3	3
4	B	4	1
5	B	5	2
6	B	6	3
7	B	7	4
8	C	8	1
9	C	9	2

Similar results could be obtained without a Foreign Key Column. Table3 is the same as Table2 but without a key relationship. To find the subsection of rows for Name 'B', the Search step could use a Match Condition of 'Table3.Name == KeyTable.Name'. The Search step would look through all 9 rows, but only find the 4 where 'B' is present. Note that it will take longer to Search all the rows and then rule out ones that do not fulfill the Search's *Match Condition* than it would to only search the related rows if there was a Foreign Key relationship.

Key Table		Table2	Table3
	Name	IP1	IP2
1	A	1	1
2	A	2	2
3	A	3	3
4	B	4	1
5	B	5	2
6	B	6	3
7	B	7	4
8	C	8	1
9	C	9	2

Consider Table2 again with the Foreign Key column. If we change the Search step's setting for *Search Related Rows Only* to 'False' and use a *Match Condition* of 'Table3.Name == KeyTable.Name', we would expect to find 4 rows but instead we find 9 rows. The behavior changes because of the Foreign Key. When a Search step is looking through a Data Table, each row in that Data Table is a candidate item. The Token iterating through the list of Search items is going to get an explicit row reference each time it checks a new item in the list. When the Token gets an explicit row reference to Table2, the Foreign Key also connects to an implicit row reference back to the Key in the KeyTable. Since *Search Related Rows Only* is 'False', the whole Table2 is Searched, so all the Foreign Keys are included. When row 1 in Table2 is Searched, the Token will have a reference to Table2[1] and an implicit reference to KeyTable[1]. Since both of these Names are 'A', the *Match Condition* passes. All 9 rows in Table2 are Searched and all 9 pass the Match Condition.

For more information on table row resolution for process steps, check out the page "Tokens and Processes - Discussion and Examples". To better understand the row references as a model is running, check out information on the "Trace Table Row Resolution" page.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Seize

Seize

The Seize step may be used to seize capacity of one or more resources for a specified owner. The executing token is held at the Seize step until all of the seize requirements are completed. Use the Release step to release resources.

Any object may potentially represent a resource with a capacity. Resource capacity must be seized or released in whole units. The resource objects to be seized can be a specific object or selected from a resource object list. In the case of a specific object, it can be a static object, such as a Standard Library 'Resource', or selected from a population of dynamic objects, such as a Standard Library 'Worker' or 'Vehicle'. When selecting from an object list or a population and the number of resources required is less than the number available, the resource selection rule is used to select between the available objects. A resource object list may contain both dynamic objects and static objects. In the case of populations all members within the population are eligible for selection. Resource capacity may be released later by either a specific name or by an object list name. In the case of a list name, the resources that are owned from the specified list are eligible to be released. They are released based on order which is specified as FirstSeized or LastSeized.

The resource may be allocated to either the parent object, associated object or specific object. If it is seized by the parent object then it must be released by the parent object. If it is seized by the associated object or a specified object, then it may be carried by the associated object or specified object across multiple objects.

A list of resource candidates may be specified by using the Definitions / Lists and referencing the 'ListName'. Alternatively, a list of resource candidates may be specified within a table column and referenced by the 'TableName.ColumnName' as the *Resource List Name* property.

The Seize step may be used in conjunction with the Release step.

When an entity first attempts to seize a resource, the following conditions must be true for the entity to be eligible to complete the seize:

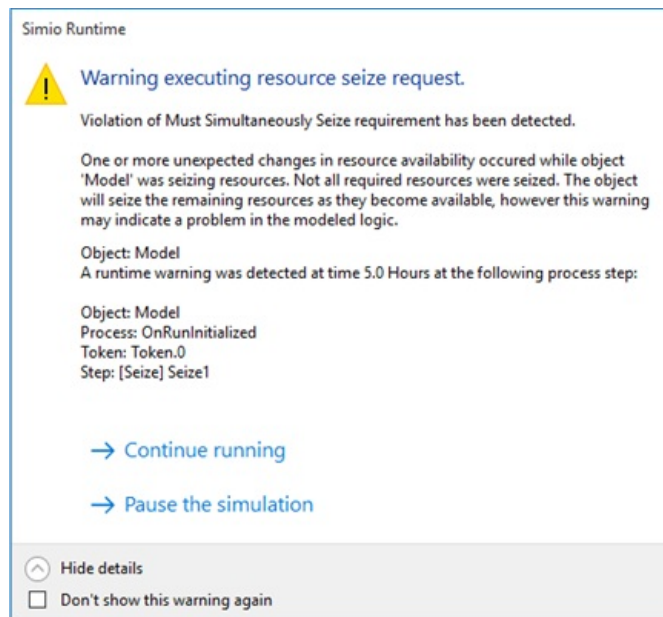
1. Sufficient capacity units are available. If capacity is reserved and there are not enough available unreserved units, then a reservation will be required for the object attempting the seize.
2. The *Selection Condition* of the Seize step is satisfied if it is specified.
3. The resource's *OnEvaluatingSeizeRequest* decision process returns True.
4. Any other scheduling constraints (such as those specified using a Stipulate step) are satisfied.

If the entity is not eligible for the seize(s), then it is placed in the AllocationQueue of the resource (assuming the entity was unable to seize some other resource if seizing from a list).

When a resource's capacity is freed or it's available capacity is increased via a schedule, the resource's allocation queue is now searched using the exact same set of eligibility conditions that are described above for an initial seize attempt. The first eligible seize request(s) found in the queue are allowed to seize the resource.

Whenever an event triggers the evaluation of a Seize request, if simultaneous seizing is required then all resource requirements will first be checked to verify that enough resources are presently available to fully satisfy the request. If that is the case, then all required resources will be seized. Otherwise, no seizures will occur.

Note that if the *Must Simultaneously Seize* requirement is violated while seizing resources due to one or more unexpected changes in resource availability, resulting in only some of the required resources being seized, then the warning below will be displayed to the user:



When a seized resource object's capacity has changed, the *On Capacity Changed Process* for each of the resource's owners (if any were specified at the Seize step(s)) will be first executed in seize order. Then last the resource object's *OnCapacityChanged* process will be executed.

When a new token is created to execute the *On Capacity Changed Process* that was specified at a Seize step:

- The token's associated object will be the resource.
- The token's context object will be the resource owner.
- The token will have a reference to the same active Task (if applicable) as the token that executed the Seize step.
- The token will have the same table references as the token that executed the Seize step.

If *Resource Type* is 'FromList' and the *Object List Name* is specified as a table column, then the *Resource Efficiency* property may be specified as another column in the same table, thereby specifying a different efficiency value for each candidate resource. If the *Resource Efficiency* expression property returns a negative value, then a runtime error is thrown.

The *Seize Constraint Logic* repeat group allows a user to easily separate different constraint types into multiple [Constraint Logic](#) element references and thus tables (e.g., separate tables defining the material and secondary resource availability requirements for a seize step).

For examples of using the Seize step, please refer to the SimBits [OverflowWIP](#), [ResourcesWithWorkSchedules](#), [SeizingVehicle](#) and [SelectingResourceFromList](#).

Listed below are the properties of **Seize**:

Property	Description
Resource Seizes	The resources to be seized.
Resource Seizes.Resource Type	The method for specifying the resource object(s) to seize.
Resource Seizes.Resource Name	The name of the resource object to seize.
Resource Seizes.Resource List Name	The name of the object list from which to select the resource object(s) to seize.
Resource Seizes.Selection Goal	The goal for selecting resource objects to seize.

Resource Seizes.Value Expression	The expression used to get the value for each candidate resource. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., Candidate.Object.Capacity.Remaining).
Resource Seizes.Request Move	Indicates whether a move to a specified location will be requested from the seized resource(s). The executing token will be held in the Seize step until the resources have arrived to the location.
Resource Seizes.Destination Node	The name of the specific node location that the seized resource(s) will be requested to move to. If not specified, and the owner object performing the seize is an entity object at a node, then this property defaults to that owner entity's current node.
Resource Seizes.Move Priority	The priority of the move request if a PlanVisit step or SelectVisit step is used by the resource to choose a visit destination from multiple candidates. If not specified, and the owner object requesting the movement is an entity object, then this property defaults to that entity's Priority state value.
Resource Seizes.Quantity Type	Indicates the type of resource quantity to seize. If 'Specific', it is the exact quantities entered in the <i>Number Of Objects</i> and <i>Units Per Object</i> properties. If 'Deficient', this is derived by subtracting the already seized resources of the specified resource type from the quantities entered in the <i>Number Of Objects</i> and <i>Units Per Object</i> properties. For example, if the <i>Number Of Objects</i> property is specified as '5' and the number of already seized resources that meet the requirement is 3, then the deficient number of resource objects to seize is calculated to be 2.
Resource Seizes.Number Of Resources	The number of individual resource objects to seize capacity units of.
Resource Seizes.Units Per Resource	The number of capacity units to seize per resource object.
Resource Seizes.Selection Condition	An optional condition evaluated for each candidate object that must be true for the object to be eligible for seizing. In the condition, use the keyword Candidate to reference an object in the collection of candidates (e.g., Candidate.Object.Capacity.Remaining)
Resource Seizes.Resource Efficiency	Optional value that can alter the rate at which work is performed using the seized resource(s), expressed as a fraction. The actual work duration is the planned work duration divided by the efficiency. Hence, an efficiency value greater than 1 shortens the time and a value less than 1 lengthens the time. In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource objects (e.g., Candidate.Object.Capacity). Note that the SeizedResources.AggregateEfficiency function of an object may be used to calculate and return an aggregate efficiency value for the list of resources currently seized by that object. For example, to alter a delay time accordingly in the model.
Resource Seizes.Record Resource Cost	Indicates whether the cost of the seized resource(s) will be recorded (charged) to the cost of the owner object.
Resource Seizes.Per-Use Cost Accrual	This property indicates when the one-time 'cost per use' for a seized resource will be charged, or accrued, to the cost of the owner object. Note that, if the owner is an entity object, then the 'AtNextStation' accrual method will charge the cost to the entity the next time it enters a station location.
Resource Seizes.Random Number Stream	The random number stream to be used if the resource selection goal is 'Random'.

Resource Seizes.On Seized Process	Optional process that is executed by a token associated with a seized object after the seize occurs.
Resource Seizes.On Capacity Changed Process	Optional process that is executed if the capacity of a seized resource has changed.
Resource Seizes.Skip Seize If	Optional condition indicating whether to skip the resource seize(s).
Must Simultaneously Seize	If multiple resources are required, indicates whether all must be available before any can be seized.
Owner Type	The object that will own the seized resources. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Owner Object	The specific object that will own the seized resource capacity units.
Seized Resources Filter Type	Filter type applied to the owner object's list of currently seized resources before attempting the new resource seizes. If 'Token', filters to only include seized resource items that are linked to a given token's execution.
Filter Token Reference	Expression returning a reference to the process token to which the seized resource items must be linked. If the expression returns 'Nothing' then no filtering occurs.
Use Strict List Referencing	If a resource seize requirement is 'FromList' and its quantity type 'Deficient', indicates whether the deficient quantity calculation counts only already seized resources that have been seized from the same specified list. The default setting is 'False' and if left unchanged then the list membership will be sufficient to count an already seized resource.
Seize Constraint Logic	Constraint logic used to enforce additional constraints on the seize request.
Seize Constraint Logic.Constraint Logic Name	The name of the Constraint Logic element used to enforce additional constraints on the seize request.
Immediately Try Seize	Indicates whether to immediately try seizing the resource requirements before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of those queues will then be scheduled on the simulation's current event calendar as a late priority event.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SelectDropoff

SelectDropoff

The SelectDropoff step may be used to set the destination node of a transporter object to the drop-off location of a riding entity.

If a destination is selected the token exits the primary exit point, otherwise it exits the failed exit point.

Listed below are the properties of **SelectDropoff**:

Property	Description
Selection Goal	The goal used to rank dropoff destination preference when selecting from multiple candidates.
Transporter Type	The transporter object that is selecting a drop-off location. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Transporter Object	Visible if <i>Transporter Type</i> is 'SpecificObject'. The specific transporter object that is selecting a drop-off location.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SelectVisit

SelectVisit

The SelectVisit step may be used to set the destination node of an entity object to the location of an accepted visit request. Possible visit requests may include pickup requests initiated at Ride steps (applicable to transporter entities) as well as move requests initiated at Seize or Move steps (applicable to resource enabled entities). To have an entity first search for and accept a not yet reserved visit request in the system, use the PlanVisit step.

If a destination is selected the token exits the primary exit point, otherwise it exits the failed exit point.

Listed below are the properties of **SelectVisit**:

Property	Description
Selection Goal	The goal used to rank visit preference when selecting a visit from multiple candidates.
Entity Type	The entity object that is searching the system for a visit request. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object that is searching the system for a visit request.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SetNetwork

SetNetwork

The SetNetwork step may be used to set the network of links used by an entity to travel between node locations.

Listed below are the properties of **SetNetwork**:

Property	Description
New Network Name	The new network value to set. 'No Network (Free Space)' is the default.
Entity Type	The entity object whose network value is to be set. May be specified as the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object whose network value is to be set.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SetNode

SetNode

The SetNode step may be used to set the destination node of any entity object.

For an example of using the SetNode step, please refer to the SimBit [SearchTables](#).

Listed below are the properties of **SetNode**:

Property	Description
Destination Type	The method used to assign the destination node. 'By Sequence' requires that the entity object has been assigned a sequence table. The destination node value will be set to the next destination in the sequence.
Node Name	The name of the specific node object that is the destination.
Node ID Number	The integer ID number of the node object that is the destination.
Match Condition	The first node found that satisfies this optional match condition will be selected as the destination, starting the search in the facility window where the entity is currently located. In the expression, use the syntax <code>Candidate.[NodeClass].[Attribute]</code> to reference an attribute of the candidate destination nodes (e.g., <code>Candidate.Node.RidePickupQueue</code>).
Entity Type	The entity object whose destination node value is to be set. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object whose destination node value is to be set.
Route Constraint Logic	Constraint logic used to enforce additional constraints on the entity's route request.
Auto Clear If Visit Different Node	Indicates whether to automatically clear the assigned destination node if for any reason the entity visits a different node beforehand.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SetRow

SetRow

The SetRow step (formerly named SetTable) may be used to assign a data table or sequence table to a token or object, or to set or reset the current associated row. The current row index into the table is specified by the RowIndex property. A token or an object can be bound to more than one table at a time. To do this, simply use two SetRow steps to reference each table of interest. The *Set Row Condition* may be specified to evaluate a conditional expression before setting a row. A token or object can have all existing table references cleared before setting a new row in the table, if desired.

For more information, please refer to the [SetRow - Discussion and Examples](#) page.

For examples of using the SetRow step, please refer to the SimBits [EntityFollowsSequenceWithTable](#), [EntityFollowsSequenceWithRelationalTables](#), [LeveledArrivals](#), [SelectEntityTypeFromTable](#) and [UsingRelationalTables](#).

Listed below are the properties of **SetRow**:

Property	Description
Table Name	The name of the table that contains the row to reference.
Row Number	The one-based index number of the row in the table. If left unspecified then defaults to either the first row or before the first row if a sequence table.
Object Type	The object or element for which to set the specified table row reference. May be specified as either the executing token, the object associated with the executing token, the parent object, or as a specific object or element reference.
Object or Element	The name of the object or element to be assigned the table reference.
Set Row Condition	Optional condition that, if specified, must evaluate to true to set the table row reference for the specified object or element.
Clear All Previous Table References First	If True, every table reference of the specified object will be removed before setting the row for the specified table.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

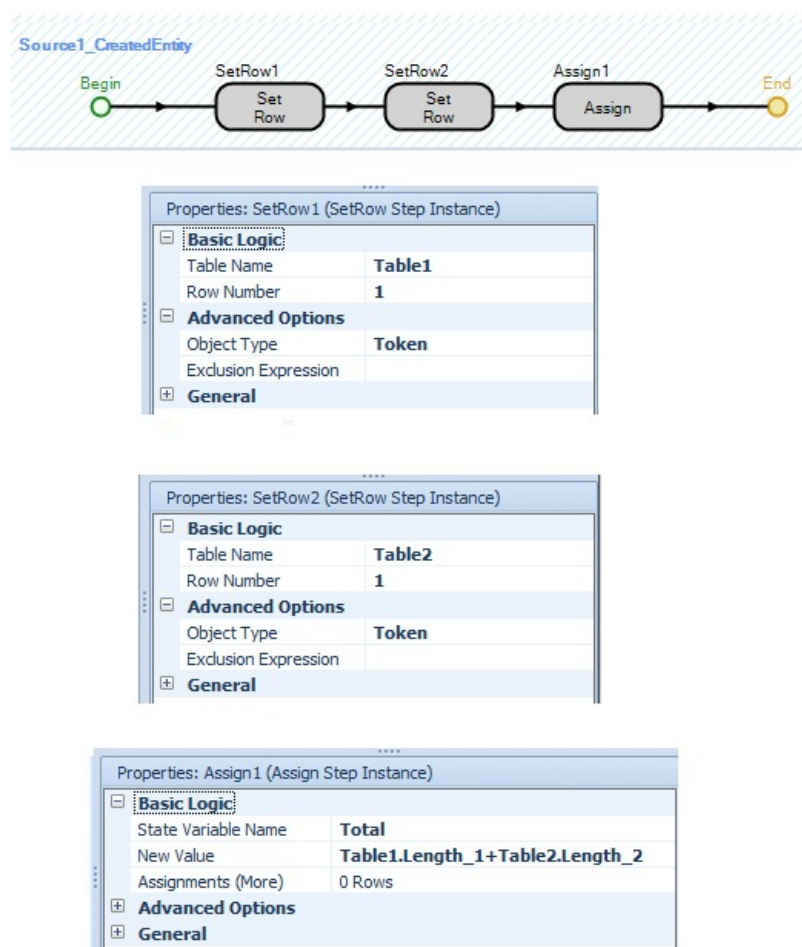
SetRow - Discussion and Examples

Example

Setting a Token to Two Different Tables

- In this example, a user would like to set a Token to point to two different tables and then reference properties from both tables in an Assign step.

The first SetRow step sets the token to point to Row 1 of Table1. The second SetRow step sets the same token to point to Row 2 of Table2. The Assign step references properties from both of the tables in its *New Value* expression.



Row Reference Resolution and Relational Data with the SetRow Step

If a SetRow Step uses a Table Reference, the *Row Number* property will resolve its value following the standard row resolution hierarchy (See [Tokens and Processes - Discussion and Examples](#)).

Generally, using the Object Type property of a SetRow Step can help achieve a desired behavior by specifying the item that receives the row reference. Here, "item" refers to an Object, Element, or Token. When there are existing references on the item specified via *Object Type*, the candidate rows are restricted by the existing relationship.

Given two tables, A and B, where B is a child table of A: When setting a row in B with the SetRow Step, where the item has an existing reference in A, the parent table, the step's specified item's (i.e., *Object Type*) existing references controls the selection of child rows.

Consider the following example: a ModelEntity triggers a Process. A Token executes the process as a delegate of the ModelEntity, and, in this case, the ModelEntity is the Token's Associated Object. In this scenario, a SetRow Step is about to set a row in a child table, a table with a Foreign Key. By this point in the Process, both the Token and the ModelEntity have different explicit row references to the Parent Table, a table with a Key, which occurred out of scope. The SetRow Step's *Object Type* by default is the 'AssociatedObject'. As such, the Token's parent table row reference, although higher in the row resolution hierarchy, will be ignored since the *Object Type* property points to the ModelEntity. As the SetRow Step is executed, pointing to the ModelEntity, the ModelEntity's Key reference will be used to determine the subset of rows, Potential Rows, in the Child Table. The *Object Type* property can be changed to a 'Token' or 'SpecificObjectOrElement' if a different item should receive the specific table row reference.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

SetWorkSchedule

SetWorkSchedule

The SetWorkSchedule step may be used to assign a new work schedule to a resource object. The function CurrentWorkSchedule.Name can be used with a resource object to determine the currently active work schedule. Additionally, the work schedule functions shown at the bottom of the [Work Schedules](#) page can be used with the CurrentWorkSchedule of a resource object (i.e., Worker1[1].CurrentWorkSchedule.Value(TimeNow)).

For more information about work schedules, please refer to the [Schedules](#) page.

Listed below are the properties of **SetWorkSchedule**:

Property	Description
Resource Type	The resource object to assign the new work schedule. May be specified as either the object associated with the executing token, the parent object, or as a specified object reference.
New Work Schedule Name	The name of the new work schedule to assign.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

StartTasks

StartTasks

The StartTasks step is used to initiate a task sequence that is associated with an object in the model. Refer to the [Task Sequence](#) element to define a task sequence.

Listed below are the properties of **StartTasks**:

Property	Description
Task Sequence Name	The name of the task sequence to start.
Object Type	The object that the task sequence execution will be associated with. May be specified as either the object associated with the executing token, the parent object or as a specific object reference.
Object	The specific object that the task sequence execution will be associated with.
Random Number Stream	The random number stream to be used if there is probabilistic branching in the task sequence.
Token Wait Action	The wait action to be taken by the token arriving to the StartTasks step. If 'None', then the token will immediately exit the step after starting the specified task sequence.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Stipulate

Stipulate

The Stipulate step may be used to add dependencies and requirements that entities must satisfy before processing.

For examples of using the Stipulate step, please refer to the SimBits [UsingTheStipulateStepForNodeLists](#) and [UsingTheStipulateStepForResourceAllocation](#).

Listed below are the properties of **Stipulate**:

Property	Description
Stipulation Type	The kind of stipulation being applied to the <i>Candidate Entity</i> . 'NodeListRoutingDependency' will prevent the entity from routing through a specific node list until the <i>Entity to Follow</i> has routed through first. 'ResourceAllocationDependency' will prevent the entity from seizing a specific resource (or seizing from a specific resource list) until the <i>Entity to Follow</i> has seized the resource (or seized from the resource list) first. 'NodeListRequirement' will force the entity to select a specific node from a <i>Node List</i> , overriding whatever node selection rule would normally be used. 'ResourceListRequirement' will force the entity to select a specific resource from a resource or list, overriding whatever resource selection rule would normally be used.
Candidate Entity	The name of the entity affected by this stipulation.
Node List	The name of the node list used by this dependency or requirement.
Required Node	Identifies the specific node in the node list that the <i>Candidate Entity</i> must select.
Resource or List	The name of the resource or resource list for this resource allocation dependency. The <i>Candidate Entity</i> will not proceed until the <i>Entity to Follow</i> has seized the specified resource, or seized from the specified resource list, <i>Prior Usage Count</i> times.
Resource List	The name of the resource list used by this requirement.
Required Resource	Identifies the specific resource in the resource list that the <i>Candidate Entity</i> must seize.
Entity to Follow	The name of the entity that must proceed through the specified resource, resource list, or node list, before the candidate entity may proceed.
Prior Usage Count	Specifies the number of times the <i>Entity to Follow</i> must seize the resource, seize from the resource list, or be routed through the node list, before the <i>Candidate Entity</i> may proceed. The default value is '1'.
Routing Group	An optional reference to the routing group that must be used by the candidate entity for this node stipulation to apply.
Seize Location	An optional reference to an object the owner making the seize must be located at for this resource stipulation to apply.

Send comments on this topic to [Support](#)

Subscribe

Subscribe

The Subscribe step may be used to add a new triggering event for a process, indicating that the process is to be executed if the event occurs. Multiple events may be subscribed to within a single Subscribe step by using the Events (More) repeating group of properties.

The Event Condition property allows a condition to be evaluated when the event occurs. This condition must be evaluated to True to trigger the process that is specified.

An example of its use would be in the scenario where a Server seizes a Resource before it starts processing and the Resource goes on and off shift during the run. If you want the Server to stop processing when the Resource goes off shift, the Subscribe step can be used to trigger a process that will suspend the Server when the event *Resource.CapacityChanged* occurs and the new capacity is 0. In this example, the process that will suspend the Server would use a Suspend Step to suspend the *OnEnteredProcessing* process of the Server. Since this process is not a public process in the Standard Library Server object, the user would need to subclass a Server object, override the *OnEnteredProcess* process and change the *Public* property to 'True'. After these changes, that process will be available in the Suspend Step.

Also, the Workstation (Deprecated) object takes advantage of these steps to suspend or resume the processing of the current job if any secondary resources go 'off-shift' or 'on-shift' respectively.

For examples of using the Subscribe step, please refer to the SimBits [EntityStopsOnLink](#) and [VehicleFleetManagement](#).

Listed below are the properties of **Subscribe**:

Property	Description
Event Name	The name of the event to which to subscribe.
Process Name	The name of the process that will be executed whenever the event occurs.
Event Condition	Optional condition to be evaluated whenever the event occurs, and which must be also true to trigger execution of the specified process.
Events (More)	Additional events to which to subscribe.
Events (More).Event Name	The name of the event to which to subscribe.
Events (More).Process Name	The name of the process that will be executed whenever the event occurs.
Events (More).Event Condition	Optional condition to be evaluated whenever the event occurs, and which must be also true to trigger execution of the specified process.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Suspend

Suspend

The Suspend step may be used to suspend various types of model behavior. The Resume step may then be later used to remove the suspension. The Suspend step can suspend a process, the movement of an object, a flow regulator's flow capability or the processing of a Batch Logic element batch.

If the *Suspend Type* is 'Process', the owner of the process can be the parent object or the associated object of the executing token. If the Tokens in process are in the delay time of a Delay or Changeover step, the delay time will be suspended. For other steps, the Token will complete the action for the step it is in process at when the suspend happens and then will not be able to execute the next step until resumed. If a process is suspended that is not yet started, when it is "started" it will be immediately suspended. If the *Token Match Condition* for a Suspend is specified, then only the tokens currently in process that satisfy the condition will have their execution suspended. It is common to use 'Candidate.ModelEntity == Entity' and/or '(Candidate.Token.AssociatedObject == Token.ContextObject)' as part of an expression in the *Token Match Condition* property to be sure to correctly match a potential candidate to suspend with the correct token/entity entering the step.

If the *Suspend Type* is 'ObjectMovement', the movement of the parent or associated object or specific object is suspended. The object must be either a link or an agent. If the *Suspend Type* is 'RegulatorFlow', the name of the flow regulator to suspend is specified. If the *Suspend Type* is 'BatchLogic', the name of the Batch Logic element to suspend is required.

A *Suspend Actions (More)* repeat group allows a user to define multiple suspended actions either through entry into the repeat group or by using a data table mapped to that repeat group.

For examples of using the Suspend step, please refer to the SimBits [EntityStopsOnLink](#), [VehicleFinishesAndParksWhenOffShift](#) and [VehicleStopsWhenServerOffShift](#).

Listed below are the properties of **Suspend**:

Property	Description
Suspend Type	The type of behavior or logic to suspend. May be Process, ObjectMovement, RegulatorFlow or BatchLogic.
Process Name	The name of the process to suspend.
Token Match Condition	Optional match condition used to filter the tokens executing the process. Only tokens currently in process that satisfy this condition will be suspended. In the expression, use the syntax Candidate.[TokenClass].[Attribute] or Candidate.[TokenAssociatedObjectClass].[Attribute] to reference an attribute of either the candidate process tokens themselves or the objects associated with those process tokens (e.g., Candidate.Token.TimeInProcess or Candidate.Entity.TimeInSystem).
Object Type	The object whose movement is to be suspended. May be specified as either the object associated with the executing token, the parent object or as a specific object reference.
Flow Regulator Name	The name of the regulator whose flow capability is to be suspended.
Batch Logic Name	The name of the Batch Logic element to suspend.
Suspend Actions(More)	A repeating set of suspended actions.
Suspend Actions(More).Suspend Type	The type of behavior or logic to suspend. May be Process, ObjectMovement or RegulatorFlow.
Suspend	The name of the process to suspend.

Actions(More).Process
Name

Suspend Actions(More).Token Match Condition	Optional match condition used to filter the tokens executing the process. Only tokens currently in process that satisfy this condition will be suspended. In the expression, use the syntax <code>Candidate.[TokenClass].[Attribute]</code> or <code>Candidate.[TokenAssociatedObjectClass].[Attribute]</code> to reference an attribute of either the candidate process tokens themselves or the objects associated with those process tokens (e.g., <code>Candidate.Token.TimeInProcess</code> or <code>Candidate.Entity.TimeInSystem</code>).
---	--

Suspend Actions(More).Object Type	The object whose movement is to be suspended. May be specified as either the object associated with the executing token, the parent object or as a specific object reference.
---	---

Suspend Actions(More).Flow Regulator Name	The name of the regulator whose flow capability is to be suspended.
---	---

Suspend Actions(More).Skip Suspend If	Optional condition indicating whether to skip the suspend action.
---	---

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Tally

Tally

The Tally step tallies an observation for each token arriving to this step.

The recorded value can be specified as an expression or can be the time between arrivals to the Tally step. The values are recorded by a TallyStatistic element.

For examples of using the Tally step, please refer to the SimBits [TallyStatisticsInTable](#), [SourceServerSinkApproaches](#) and [HourlyStatistic](#).

Listed below are the properties of **Tally**:

Property	Description
Tally Statistic Name	The name of the tally statistic for which to record an observation value.
Value Type	The type of observation value to record. The value type 'Expression' records the value of the specified expression. If the value type is 'TimeBetween', then the time since the last token arrival to this or any other Tally step referencing the tally statistic is recorded.
Value Expression	The expression returning the value to be recorded.
Tallies (More)	A repeating set of tallies.
Number Of Observations	The number of observation value(s) to be recorded.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Transfer

Transfer

The Transfer step may be used to transfer an entity object between objects and between different types of locations, such as free space and objects.

If transferring an entity to a node, the Transfer step automatically sets the entity's destination to that node if the entity does not already have a destination set.

A user can use a Transfer step to transfer an object between object locations without the entity having to use the external nodes to enter and exit objects. So, for example, you can immediately transfer an entity from inside object A to a node outside object B using the Transfer step.

Whenever attempting a transfer from 'CurrentNode' to 'OutboundLink', if the entity's CurrentTravelMode state is 'Free Space Only' or the entity has no network assigned then the transfer attempt will fail and the token exits the 'Failed' exit point of the Transfer step. Whenever attempting a transfer from 'CurrentNode' to 'FreeSpace', if the entity has an assigned network and its CurrentTravelMode is 'Network Only' then the transfer attempt will fail and the token exits the 'Failed' exit point of the Transfer step. Whenever attempting a transfer from 'CurrentNode' to 'OutboundLink', if there is only 1 possible outbound link from the node that is a member of the entity's currently assigned network, and the *Outbound Link Rule* is 'Shortest Path' but there is no followable network path to the entity's next destination, then:

- If the entity's CurrentTravelMode state is 'Network Only', then the Transfer step will go ahead and still transfer the entity onto the sole outbound link. This behaviour allows for the possibility that a followable network path emerges after the entity moves through the next downstream object.
- If the entity's CurrentTravelMode state is 'Network If Possible', then the transfer attempt will fail and the token exits the 'Failed' exit point of the Transfer step. This behavior allows the entity to then do an immediate transfer from 'CurrentNode' to 'FreeSpace' in order to begin traveling to its next destination in free space.

Note

It is only *safe* at this point to Transfer an entity into or out of an object through the external nodes since most objects make the assumption that an entity is going to enter or exit through nodes. For example, external logic might immediately Transfer an entity out of a Server object and transfer it over to some other location outside the logic. However, if that happens, the tokens running inside the Server will not realize that the entity was transferred out and it will assume the entity is still inside the Server. It will continue trying to process the entity and an error will occur.

For examples of using the Transfer step, please refer to the SimBits [CONWIP](#), [DynamicallyCreatingVehicles](#), and [FreeSpaceMovement](#).

Listed below are the properties of **Transfer**:

Property	Description
From	The location type that the entity object is to be transferred from. Options include FreeSpace, CurrentStation, CurrentNode, EndOfLink and CurrentContainer.
To	The location type that the entity object is to be transferred to. Options include FreeSpace, Station, Node, ParentExternalNode, AssociatedObject, OutboundLink and Container.
Facility Name	The object whose facility free space the entity is to be transferred into. If unspecified, then defaults to the most immediate parent object of the Transfer step that supports a facility view. This property is visible when <i>To</i> is 'FreeSpace'.
Station Name	The name of the station into which the entity is to be transferred, available when <i>To</i> is 'Station'.
Node Name	The node from which the entity is to be transferred. This property is visible when <i>To</i> is 'Node'.

External Node Name	The external node of the parent object from which the entity is to be transferred. This property is visible when <i>To</i> is 'ParentExternalNode'.
Outbound Link Preference	<p>The preference used by the entity to select an outbound link from its current node to its next destination.</p> <p>If the preference is 'Default', then the outbound link preference settings for the entity's current node will be used.</p> <p>If the preference is 'Any', then the specified <i>Outbound Link Rule</i> will be strictly adhered to and blocked outbound links may cause the entity to wait.</p> <p>If the preference is 'Available', then the specified <i>Outbound Link Rule</i> will be automatically adjusted to minimize waiting time by first applying the rule only to outbound links that are immediately available for entry. If no outbound links are immediately available, then the preference will revert to 'Any' selection.</p> <p>If the preference is 'Specific', then the entity will be transferred to the specified <i>Outbound Link Name</i>.</p>
Outbound Link Rule	<p>The rule used by the entity to select an outbound link from its current node to its next destination. Note that if the rule is set to 'Shortest Path', and the entity does not have an assigned destination, then the rule will revert to 'By Link Weight' selection.</p>
Outbound Link Name	The name of the specific outbound link that will be selected by the entity.
Container Name	The name of the container into which the entity is to be transferred. This property is visible when <i>To</i> is 'Container'.
Path Planner Name	The name of the Path Planner element used by the entity to find and reserve a deadlock-free shortest network path from its current node to its destination node. NOTE: If the Flow Regulator Name property is specified then the Path Planner Name property is ignored. If path planning is used by an entity then the Outbound Link Preference and Outbound Link Rule properties are ignored.
Flow Regulator Name	The regulator used if transferring the entity's mass and volume as a continuous flow into the new location. This property is visible when <i>To</i> is 'AssociatedObject', 'OutboundLink' or 'Container'.
Entity Type	The entity object to be transferred. Specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object that is to be transferred.
Random Number Stream	The random number stream to be used if the entity is transferring from current node to outbound link and is probabilistically selecting the outbound link using the node's 'By Link Weight' outbound link rule.
Fail If Blocked	Indicates whether to cancel the transfer attempt if the location where the entity is to be physically transferred into is considered blocked. If the transfer attempt is cancelled, then the arriving token will immediately exit the 'Failed' exit point of the Transfer step.
Token Wait Action	The wait action to be taken by the token arriving to the Transfer step. Options include WaitUntilTransferred and WaitUntilTransferring.

Send comments on this topic to [Support](#)

Transfer - Discussion and Examples

Example 1 - Current Node to Outbound Link Using Various Outbound Link Preferences

If you wanted to model a train moving on a track network, then the leading locomotive at a node in the track network might execute a Transfer step from current node to outbound link using an *Outbound Link Preference* of 'Default'.

Each following 'RailCar' object in a train, to simply select from a node the same outbound link that was selected by the car or locomotive immediately in front of them, might execute a Transfer step using *Outbound Link Preference* of 'Specific' and *Outbound Link Name* specified as 'RailCar.TrainPredecessor.TrailingLink', where 'TrainPredecessor' would be an entity reference state variable on a railroad car that is pointing to the predecessor car or locomotive that the railcar is directly connected to and immediately following.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Travel

Travel

The Travel step may be used to do a direct (straight-line) movement of an entity in free space to a specified location. The executing token may be held at the Travel step until the travel movement is completed (*Token Wait Option* set to 'WaitUntilTravelCompleted') or it may continue through its steps (*Token Wait Option* set to 'None (Continue)') while the travel occurs.

The default *Steering Behavior* is 'Direct to Destination' where the travel destination can be specified as a specific fixed (non-moveable) object, as absolute coordinates, or as relative coordinates from the entity's current location. Additionally, a desired maximum movement rate and ending movement rate for the travel may be specified.

A *Steering Behavior* of 'Follow Network Path' requires a starting and ending node. The entity will then follow the shortest path between them in free space, attempting to stay within the boundaries of the links, defined by each link's Width (or 1 meter if no Width is defined). The *Path Width* can also be varied so that the entity can directly specify a radius that will override the link's width.

The free space travel movement will be automatically cancelled if another Travel step is executed, the entity is transferred out of free space, or if any of the entity's movement parameters are changed by unrelated model logic (e.g., an Assign step is used to deviate the entity's movement direction or rate from the active travel operation's assigned parameters).

The Suspend and Resume steps may be used to suspend and resume the free-space travel movement.

Entity objects (such as ModelEntity, Worker, and Vehicle) have an OnEnteredFreeSpace standard process that is executed if the entity transfers from a non-free space location into free space.

For an example of using the Transfer step, please refer to the SimBit [FreeSpaceMovement](#).

Listed below are the properties of **Travel**:

Property	Description
Steering Behavior	The behavior determining how the specified entities travel. Options include None, Direct to Destination, and Follow Network Path.
Destination Type	The method used to specify the destination that the entity is to travel to. The entity will do a direct (straight-line) movement in free space to the specified location.
Destination Object	The specified object to travel to (when the <i>Destination Type</i> is 'SpecificObject').
X Location	The x coordinate location to travel to (when the <i>Destination Type</i> is 'AbsoluteCoordinates').
Y Location	The y coordinate location to travel to (when the <i>Destination Type</i> is 'AbsoluteCoordinates').
Z Location	The z coordinate location to travel to (when the <i>Destination Type</i> is 'AbsoluteCoordinates').
X Offset	The distance to travel in the x direction from the entity's current location (when the <i>Destination Type</i> is 'RelativeCoordinates').
Y Offset	The distance to travel in the y direction from the entity's current location (when the <i>Destination Type</i> is 'RelativeCoordinates').
Z Offset	The distance to travel in the z direction from the entity's current location (when the <i>Destination Type</i> is 'RelativeCoordinates').
Maximum Movement Rate	The desired maximum movement rate at which the entity is to travel.

Ending Movement Rate	The desired movement rate of the entity at the end of the travel.
Acceleration	The rate at which the entity will gain speed going from a slower movement rate to a faster movement rate.
Deceleration	The rate at which the entity will lose speed going from a faster movement rate to a slower movement rate.
Auto Orient Towards Destination	Indicates whether to automatically orient the entity such that its front is facing the travel destination. If this property is specified as False, then the entity will stay in its current orientation when initializing the travel movement.
Starting Node	The starting node of the path when using Follow Network Path type of Steering Behavior.
Ending Node	The ending node of the path when using Follow Network Path type of Steering Behavior.
Path Width	The value to use as the width of the path for Follow Network Path type of Steering Behavior.
Update Time Interval	The value that determines how often the entity will check if it is following the path and update its direction to do so.
Entity Type	The entity object that is to perform the travel operation. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Entity Object	The specific entity object that is to perform the travel operation.
Token Wait Option	The wait action to be taken by the token arriving at the Travel step. If 'None', then the token will immediately exit the step after initiating the travel operation.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UnBatch

UnBatch

The UnBatch step may be used to remove batch members from a parent entity object. A new token associated with an unbatched member will exit the 'Member' exit point of the step. The *Token Wait Action* will determine the order in which the tokens exit the Unbatch step. By default, the property value is 'NewTokensExitFirst' and a token for each member will first exit from the 'Member' exit and proceed until a delay type step occurs for the token or its associated entity. Then the original token will exit the 'Parent' exit.

The *Match Condition* property may be used to unbatch only certain entities from the batch based on a condition specified. For example, a *Match Condition* may be specified as 'Candidate.ModelEntity.Is.RedPart' where RedPart is the name of the entity. When unbatched from the parent, only those entities of type RedPart will be unbatched and any other members of the batch (BluePart, GreenPart, etc.) will remain with the batch. The *Removal Order* can be used to determine the order of searching for the *Desired Split Quantity* number of members to unbatch.

When a member entity is unbatched from a parent entity, it is placed into free-space at the same XYZ location of the parent entity. Currently, Simio only supports UnBatch into free space, not directly into a node. Free-space has no possible content capacity constraints so the members are unbatched there. Then, a Transfer step should be used to transfer the member entity into the station or node desired. If the Transfer step is not used, then by default, when an entity enters free space, it simply starts traveling at its desired in whatever current direction it happens to be pointing in. However, the ModelEntity's OnEnteredFreeSpace logic assigns the movement rate of the entity to '0.0' if the entity can't determine a free space travel movement to some particular node.

The UnBatch step is used internally within the Separate object in the Standard Library for unbatching. The member entity token enters a Transfer step after the UnBatch step to move the entity into the member output buffer station.

For an example of the Unbatch step, see the [CustomUnbatching](#) or [UsingAStorageQueue](#) SimBit.

Listed below are the properties of **UnBatch**:

Property	Description
Desired Quantity	The desired number of batch members to remove (unbatch) from the parent entity.
Removal Order	The order in which to search for and remove batch members from the parent entity.
Match Condition	Optional match condition used to filter the batch member entities. Only members that match the criteria given by the expression will be removed from the batch. In the condition, use the keyword 'Candidate' to explicitly reference a candidate entity in the batch members being searched (e.g., Candidate.Entity.Priority > 0).
Parent Entity Type	The parent entity object that is carrying the batch members. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Parent Entity Object	The specific parent entity object that is carrying the batch members.
On Entered Free Space Process	Indicates whether or not to execute the built-in OnEnteredFreeSpace process for each unbatched member entity after it is removed from the parent entity's batch members queue.
Token Wait Action	The wait action to be taken by the token arriving to the UnBatch step. If the action is 'NoWait', then the execution of all new tokens associated with unbatched members will be scheduled on the simulation's current event calendar as an early priority event. The original token will then immediately exit the step. If the action is 'NewTokensExitFirst', then all new tokens associated with unbatched members will exit the step first before the original token exits.

If the action is 'WaitUntilNewTokenProcessingCompleted', then the original token will wait until the processing of all new tokens associated with unbatched members has been completed before exiting the step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UnPark

UnPark

The UnPark step may be used to transfer an entity object out of a node's parking station and into the node.

The *Token Wait Action* property indicates whether the token is going to wait in the step until the transferring entity 'Transferring' or 'Transferred' event is fired.

Listed below are the properties of **UnPark**:

Property	Description
Entity Type	The entity object to be unparked. Specified as either the object associated with the executing token, the parent object or as a specific object reference.
Entity Object	Visible if <i>Entity Type</i> is 'Specific Object'. The specific entity object to be unparked.
Token Wait Action	The wait action to be taken by the token arriving to the UnPark step. Options include WaitUntilTransferred and WaitUntilTransferring.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

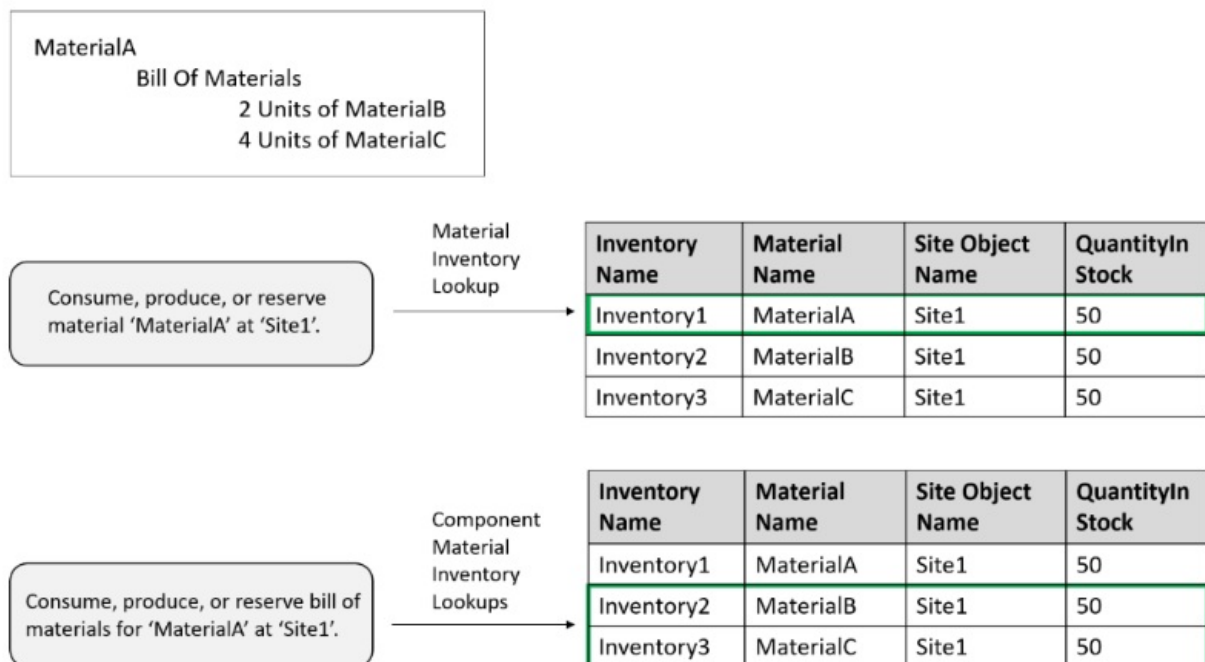
Send comments on this topic to [Support](#)

UnReserve

The UnReserve step may be used to cancel stock reservations up to a specified maximum assembly quantity. Will unreserve any alternative BOMs or alternative materials with reservations (validity constraints do not apply). Use the [Reserve](#) step to create a new resource or material reservation.

The *Lot ID* property may be specified as a string literal enclosed in double quotes (e.g., "Lot1"), the name of a string state variable or string column in a table, or any other expression that returns a string value.

If *Consumption Type* is unreserving a single 'Material' that is location-based inventory, then Simio will look up the Inventory element holding the specified material at the specified inventory site. If *Consumption Type* is unreserving a 'Bill of Materials', then Simio will look up the Inventory elements holding the component materials at the specified inventory site.



If the *Inventory Site Type* property value is 'None' but a site object reference is required because the material is location-based inventory, or if an inventory reference for a material-site object pair cannot be found, then a runtime error will occur.

If the name of the resource to unreserve is a member of a population (i.e., worker or vehicle), the *Resource Name* can be specified as either the specific resource within the population, such as 'Truck[1]' or 'Truck[2]' or may alternatively be specified by simply the resource name, such as 'Truck'. If the population of that resource is greater than 1, keep in mind that unreserving 'Truck' would always unreserve the first member of the population (numerically) that has been reserved. For example, let's say an entity has reserved Truck[3] and Truck[1]. Then, if the entity tries to unreserve 'Truck', if Truck[1] still has an outstanding reservation, it will first be unreserved, otherwise Truck[3] will be unreserved.

The functions *Capacity.Reserved* and *Capacity.ReservedTo(owner)* will be updated when a resource is reserved or unreserved. The capacity reserved of a given resource by various entities can be greater than the actual capacity of the resource. Allocating the resource to the entity then decreases the *Capacity.Reserved* and *Capacity.ReservedTo(owner)* function values. Keep in mind that the function *ReservedTransporter* for a given entity object is NOT related to the Reserve step. If the entity object currently has made a reservation to ride on a transporter (i.e., by indicating *Ride On Transporter* 'True' in a TransferNode), then this function returns a reference to that transporter once it is allocated. Otherwise, the Nothing keyword is returned. This function is set to the reserved transporter until the entity is actually loaded and has moved into the transporter's RideStation.

See the [Reservations](#) page for more details.

Listed below are the properties of **UnReserve**:

Property	Description
----------	-------------

Reservation Type	The type of material stock consumption. Material – A single material. BillOfMaterials – A list of component materials. BillOfMaterialsGroup – Alternative lists of component materials.
Resource Name	The name of the resource object for which to cancel the reservation(s).
Number Capacity Units	The desired number of capacity units of the resource for which to cancel the reservation(s). Any existing reservations made by the owner object will be cancelled up to this maximum. (for <i>Reservation Type</i> 'Resource').
Consumption Type	Indicates whether the material to unreserve is a single material or bill of materials.
Material Name	The name of the material.
Inventory Site Type	Indicates the fixed object that is the inventory location. Applies only to material elements whose <i>Location Based Inventory</i> property is set to 'True'.
Site Object Name	The name of the fixed object that is the inventory location.
BOM Name	The name of the bill of materials.
BOM Group Name	The name of the bill of materials group.
Quantity	The quantity of the material or parent assembly.
Lot ID	Optional string value indicating the lot identifier of the material or parent assembly.
Reservations (More)	A repeating set of additional reservations to cancel.
Reservations (More).Reservation Type	Indicates whether to cancel the reservation(s) for a resource, a single material or a bill of materials.
Reservations (More).Resource Name	The name of the resource object for which to cancel the reservation(s).
vReservations (More).Material Name	The material which is to be either specifically unreserved or whose bill of materials is to be unreserved.
Reservations (More).Number Capacity Units	The desired number of capacity units of the resource for which to cancel the reservation(s). Any existing reservations made by the owner object will be cancelled up to this maximum. (for <i>Reservation Type</i> 'Resource').
Reservations (More).Quantity	The desired quantity of the material for which to cancel the reservation(s). Any existing reservations made by the owner object will be cancelled up to this maximum (for <i>Reservation Type</i> 'Material' or 'BillOfMaterial').
Reservations (More).Lot ID	Optional string value indicating the lot identifier of the unreserved material.
Owner Type	The object that owns the reservation(s) to be cancelled. May be specified as either the object associated with the executing token, the parent object, or as a specific object reference.
Owner Object	The specific object that owns the reservation(s) to be cancelled.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

UnSetRow

UnSetRow

The UnSetRow step may be used to remove table row references previously set for the executing token or a specified object or element. Use the SetRow step to set a table row reference.

Listed below are the properties of **UnSetRow**:

Property	Description
Table Type	Indicates the tables to remove the row references to. 'All' will remove all table row references previously set for the specified object or element.
Table Names	The names of the tables to remove the row references to.
Object Type	The object for which to remove the specified table row references. May be specified as either the executing token, the object associated with the executing token, the parent object, or as a specific object or element reference.
Unset Row Condition	Optional condition that, if specified, must evaluate to true to remove any table row references for the specified object or element.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Unsubscribe

Unsubscribe

The Unsubscribe step may be used to cancel a triggering event for a process. This is used in conjunction with the Subscribe step. See the [Subscribe step](#) for an example of use. Multiple events and processes may be specified within a single Unsubscribe step by using the Events (More) repeating properties.

Listed below are the properties of **Unsubscribe**:

Property	Description
Event Name	The name of the event from which to unsubscribe.
Process Name	The name of the process that will not longer be executed whenever the event occurs.
Events (More)	Additional events from which to unsubscribe.
Events (More).Event Name	The name of the event from which to unsubscribe.
Events (More).Process Name	The name of the process that will not longer be executed whenever the event occurs.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

VisitNode

VisitNode

The VisitNode step may be used within a node object's logic to initiate the OnVisitingNode process of the entity associated with the executing token. The token waits until the entity's OnVisitingNode process is completed before exiting the step.

Note

This Step can only be used from within a Node object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Wait

Wait

The Wait step may be used to hold the arriving token in the step until one or more specified events occur.

The *Token Wait Action* determines whether the arriving token is waiting for all of the specified events to occur ('WaitUntilAllEvents') or just a single event to occur ('WaitUntilAnyEvent').

The tokens are released from the Wait step in order that they arrived to the step (that is, First In First Out). When multiple tokens are waiting at the Wait step, all tokens are released when the specified event is fired.

The Wait step can be used in conjunction with the Fire step.

For examples of using the Wait step, please refer to the SimBits [EntityStopsOnLink](#), [BatchingProcessUsingScanOrWaitStepToControlBatchSize](#), [VehicleVisitsServiceCenter](#), and [VehicleFinishesAndParksWhenOffShift](#).

Listed below are the properties of **Wait**:

Property	Description
Event Name	The name of the event to wait for.
Event Condition	Optional condition to be evaluated on an occurrence of the event, and which must be true in order for the wait for the event to be considered ended.
Events (More)	A repeating set of additional events to wait for.
Assignments (More).Event Name	The name of the event to wait for.
Assignments (More).Event Condition	Optional condition to be evaluated on an occurrence of the event, and which must be true in order for the wait for the event to be considered ended.
Token Wait Action	The wait action to be taken by the token arriving to the Wait step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

StartActivity - DEPRECATED

StartActivity - DEPRECATED

The StartActivity step starts the next activity in a reserved operation.

NOTE: It is recommended that the [TaskSequence](#) element and [StartTasks](#) step be used instead of the deprecated StartOperation / StartActivity / EndActivity / EndOperation steps. To view these steps within the All Steps panel in the Processes window, go to Application Settings and change the *Display Deprecated Steps In Processes Window Panels* property to 'True'. This may also require closing and re-opening Simio if the Processes window had already been opened.

Listed below are the properties of **StartActivity**:

Property	Description
Operation	The name of the operation to reserve.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EndActivity - DEPRECATED

EndActivity - DEPRECATED

The EndActivity step ends the current activity within an operation.

NOTE: It is recommended that the [TaskSequence](#) element and [StartTasks](#) step be used instead of the deprecated StartOperation / StartActivity / EndActivity / EndOperation steps. To view these steps within the All Steps panel in the Processes window, go to Application Settings and change the *Display Deprecated Steps In Processes Window Panels* property to 'True'. This may also require closing and re-opening Simio if the Processes window had already been opened.

Listed below are the properties of **EndActivity**:

Property	Description
Operation	The name of the operation to reserve.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

StartOperation - DEPRECATED

StartOperation - DEPRECATED

The StartOperation step starts an operation.

An operation cannot start until capacity is available and a time slot is available. The required time slot is the best case time based on the parent work schedule - this is extended by the buffer time to account for delays caused by secondary resources, materials, travel, etc. If capacity is available and the best case time slot is available it seizes the parent resource and starts the operation. If the actual operation time exceeds the available time slot the future operation is slipped to accommodate this operation.

NOTE: It is recommended that the [TaskSequence](#) element and [StartTasks](#) step be used instead of the deprecated StartOperation / StartActivity / EndActivity / EndOperation steps. To view these steps within the All Steps panel in the Processes window, go to Application Settings and change the *Display Deprecated Steps In Processes Window Panels* property to 'True'. This may also require closing and re-opening Simio if the Processes window had already been opened.

Listed below are the properties of **StartOperation**:

Property	Description
Operation	The name of the operation to start.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

EndOperation - DEPRECATED

EndOperation

The EndOperation step ends the operation.

This step releases the parent resource and shifts any conflicting operation spans into the future.

NOTE: It is recommended that the [TaskSequence](#) element and [StartTasks](#) step be used instead of the deprecated StartOperation / StartActivity / EndActivity / EndOperation steps. To view these steps within the All Steps panel in the Processes window, go to Application Settings and change the *Display Deprecated Steps In Processes Window Panels* property to 'True'. This may also require closing and re-opening Simio if the Processes window had already been opened.

Listed below are the properties of **EndOperation**:

Property	Description
Operation	The name of the operation to reserve.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

The Definitions Window

The Definitions Window

The Definitions window displays a panel of buttons where the user can define the following:

Elements represent things in a process that change state over time.

Properties are model input parameters that do not change during the simulation run. Although the definition cannot change, the value returned (e.g. an evaluated expression or random variable) may change.

States are dynamically changing values that can be thought of as output responses that change throughout the execution of the object logic.

Events are logic triggers used within the Processes window.

Functions are queries of calculated values that can be called by another object. To query for the value of a function, use it in an expression.

Lists are used to define a collection of strings, objects, nodes, or transporters.

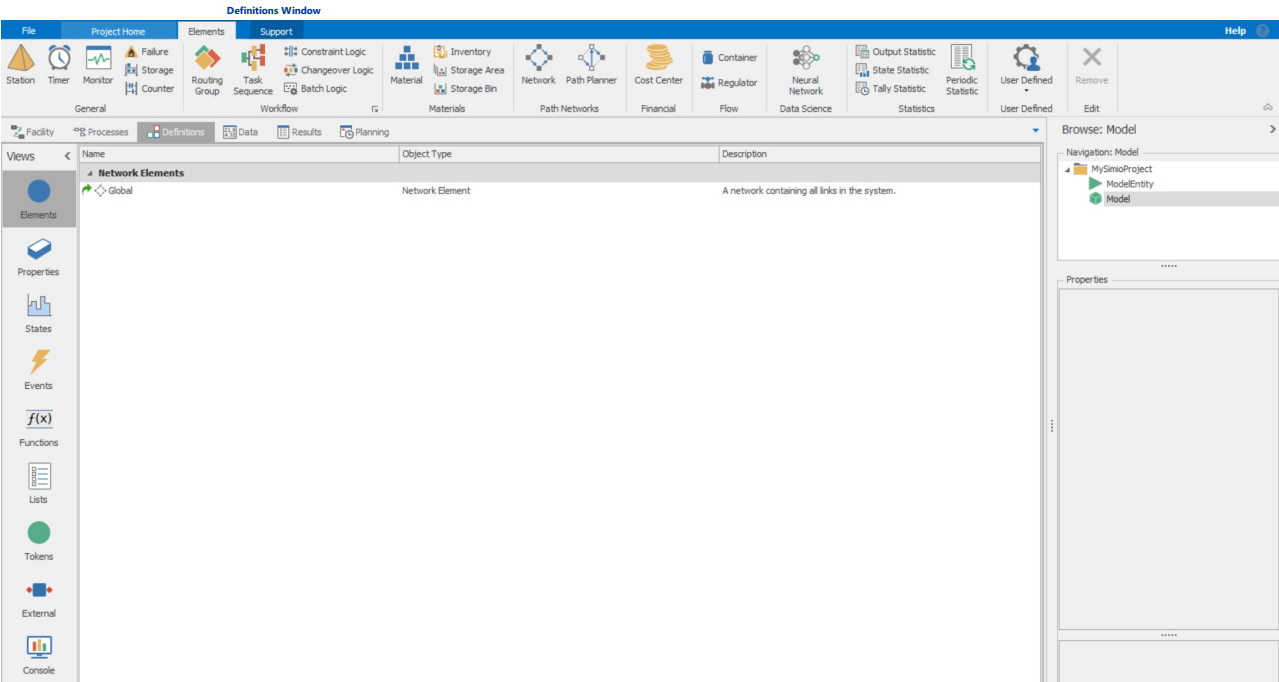
Tokens execute the steps in a process flow within the Processes window.

External defines the graphical representation of a model that is instantiated as an object into another model.

Console allows you to define run time status and control mechanisms for your model.

The Definitions tab also allows the user to view the Simio properties, states and events that are automatically generated with each new model.

Properties, states and events are either inherited or explicitly added to your model. You can add/edit/delete your own, but in general you cannot edit or delete inherited properties, states and events. The one exception is that you can hide or change the display name, category, and description of an inherited property. Inherited properties, states and events have a green arrow to their left.



Note

The collapsable feature of the inherited properties, states, and events is only available with the Microsoft Vista O/S or higher. Earlier versions of the O/S will not see the inherited properties, states and events collapsed.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Elements

Elements represent things in a process that change state over time. Elements are added to a list and then referenced by one or more process steps in the Processes window. The elements within Simio include the following:

BatchLogic element may be used in conjunction with the Batch step to match multiple entities together, form those entities into a batch, and then attach the batched members to a parent entity.

BillOfMaterials element may be used to define a comprehensive list of the component materials required to manufacture a product. Multiple BOMs that describe alternative lists of component materials to manufacture one product may be grouped together using the Bill Of Materials Group element.

BillOfMaterialsGroup element may be used to group together BOMs that describe alternative lists of component materials to manufacture one product. Refer to the BillOf Materials element to define a single BOM.

ChangeoverLogic element may be used in conjunction with the Changeover step to model sequence dependent setups at one or more associated resources.

ConstraintLogic element may be used to enforce material availability, resource availability or other types of constraint conditions on an applicable process step (e.g., a Seize or Route step).

Container element may be used to define a volume or weight capacity constrained location for holding entities representing quantities of fluids or other mass. Use the Transfer step to initiate transfers of entities into or out of a container.

CostCenter element may be used to define an identifiable area of responsibility where costs are allocated or incurred. Cost centers may be linked in a hierarchical structure for cost accounting and reporting purposes, whereby costs are automatically rolled up through the cost center hierarchy.

Failure element may be used to define a failure mode. Failure downtime occurrences are started and ended using the Fail and Repair steps.

Inventory element may be used to define a storage bucket for holding stock of a specific material. A *Site Object Name* property is provided to designate a fixed object in the Facility window as the inventory's physical location.

Material can be consumed and produced by tokens.

Monitor element may be used to monitor a state variable and fire an event whenever there is a discrete change or threshold crossing of the state value.

Network is a list of links to which an entity may be assigned to follow via the shortest path to its destination.

Neural Network element may be used to integrate a neural network regression model into simulation logic.

OutputStatistic element defines an expression that can be optionally recorded at the end of each replication of a simulation.

Regulator element may be used to regulate flow rates into or out of a location. Use the Assign step to change regulator maximum flow rates as well as perform more complex regulator flow actions involving wait conditions.

RoutingGroup element is used with a Route step to route an entity object to a destination selected from a list of candidate nodes.

StateStatistic element may be used to record time-persistent statistics on a state variable.

Station element may be used to define a capacity constrained location within an object where one or more visiting entity objects can reside. The Transfer step is used to transfer entities into and out of a station.

Storage element may be used to define a custom queue state for temporarily holding one or more objects. The Insert and Remove steps are used to insert and remove objects from the storage element's queue.

TallyStatistic element tallies observational statistics that are recorded using the Tally step.

TaskSequence element may be used to more easily define and execute structured sequences of processing tasks. The **StartTasks** step is used to start a task sequence that is associated with an object in the system.

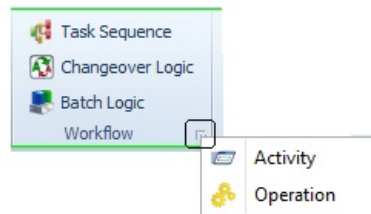
Timer element fires a stream of events according to a specified IntervalType.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for all Elements. Please refer to the [User Interface](#) for

more details.

Please refer to the [Table-Based Elements \(Auto-Create\)](#) page for more information about automatically creating elements from a data table.

NOTE: There are two elements that were removed from the main Elements panel as of August 2016 (Sprint 143), as noted below. These elements can still be found by using the pressing the small arrow on the bottom right of the Workflow section of the Elements ribbon.



[Operation](#) is a sequence of activities that are performed by an object over time.

[Activity](#) is performed by an operation that is owned by an object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Elements Reference

This Elements Reference section has detailed information for all elements in Simio.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

BatchLogic

BatchLogic

The BatchLogic element may be used in conjunction with the Batch step to group multiple entities together, and then attach the batched members to a parent entity.

There is a *Must Simultaneously Batch* Boolean property provided by the BatchLogic element to indicate whether the full target number(s) of member entities must be available before any can be collected and attached to a parent entity. If this property is 'False', then batch members will be collected from the BatchLogic's MemberQueue as soon as they become available. The parent entity eventually gets released once the target batch quantity or quantities are reached. If this property is 'True', then no batch members can be collected by a parent entity until its full target batch quantity or quantities are available. In that case, the collection of all batch members and release of the parent entity is always going to happen as a single event (at a single point in time). Note that one possible result if forcing simultaneous batching is parent entities may jump over other parent entities in the parent queue if requirements differ between parents.

The *Batch Quantity* and *Matching Rule* properties can be parent entity dependent (presumably coming from data table). For example, there can be one parent entity that needs to collect '10' *Batch Quantity* using *Matching Rule* 'Any Entity', while another parent entity at the same BatchLogic element needs to collect some batch quantities using *Matching Rule* 'Match Members And Parent' and so forth. Note that the *Must Simultaneously Batch* option can also be parent-entity dependent data.

The *Release Batch Early Triggers* defined for a parent entity can be either time based (trigger fires when specified wait duration expires) or event based (trigger fires when a specified event occurs). By default, the release decision when a trigger has fired is 'Always', but may alternatively be 'Conditional' or 'Probabilistic'. A parent entity that is being released early attempts to collect as many batch members as possible up to its target batch quantity(ies) - possibly no batch members are collected. The parent entity's associated queue item is then removed from the BatchLogic element's parent queue and its associated process token exits the Batch step. * Note that the data for a BatchLogic element or Combiner's *Release Batch Early Triggers* repeat group may be defined in a data table, with the trigger definitions thus potentially differing between parent entities.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For examples of using the BatchLogic element, please refer to the SimBits [UsingAStorageQueue](#) and [CombinerNode](#) or any of the SimBits under the Combining and Separating topic, as many of those use the Combiner, which includes a BatchLogic element.

Listed below are the properties of **BatchLogic**:

Property	Description
Report Statistics	Specifies if statistics are to be automatically reported for this element.
Batch Quantity	The target number of member entities to collect and attach to a parent entity. May be specified as an expression truncated to an integer.
Matching Rule	The matching rule used to group member and parent entities together. If the matching rule is 'AnyEntity', then any entity can be grouped with any other entity. If the matching rule is 'MatchMembers', then member entities must be grouped together using a specified match expression. If the matching rule is 'MatchMembersAndParent', then the parent entity must also match the member entities to attach a group to the parent.
Parent Match Expression	The expression whose value for a parent entity must be the same as the match expression for the member entities to attach those members to the parent.
Member Match Expression	The expression whose value must be the same between member entities to group them together.
Batch	Additional member entities to collect and attach to a parent entity.

Quantities
(More)

Batch Quantity (Batch Quantities)	The target number of member entities to collect and attach to a parent entity. May be specified as an expression truncated to an integer.
Matching Rule (Batch Quantities)	The matching rule used to group member and parent entities together. If the matching rule is 'AnyEntity', then any entity can be grouped with any other entity. If the matching rule is 'MatchMembers', then member entities must be grouped together using a specified match expression. If the matching rule is 'MatchMembersAndParent', then the parent entity must also match the member entities to attach a group to the parent.
Parent Match Expression (Batch Quantities)	The expression whose value for a parent entity must be the same as the match expression for the member entities to attach those members to the parent.
Member Match Expression (Batch Quantities)	The expression whose value must be the same between member entities to group them together.
Parent Ranking Rule	The rule used to rank the queue of parent entities waiting to collect batch members.
Parent Ranking Expression	The expression used with a SmallestValueFirst or LargestValueFirst parent ranking rule.
Member Ranking Rule	The rule used to rank the queue of member entities waiting to be batched.
Member Ranking Expression	The expression used with a SmallestValueFirst or LargestValueFirst member ranking rule.
Must Simultaneously Batch	Indicates whether the full target number(s) of member entities must be available before any can be collected and attached to a parent entity. If this property is set to False, then batch members will be collected as soon as they arrive to the BatchLogic member's queue.
Release Batch Early Triggers	Optional time or event-driven triggers that can cause a batch to be released early before collecting the full target number(s) of member entities.
Trigger Type (Release Batch Early Triggers)	The type of trigger. A time based trigger can release a batch early once a wait duration expires. An event based trigger can release a batch early whenever a specified event occurs.
Wait Duration (Release Batch Early Triggers)	The wait duration until deciding whether to release a batch early.
Triggering Event Name (Release Batch Early Triggers)	The name of the event whose occurrence will trigger a decision whether to release a batch early.
Release Decision Type (Release Batch	Optional time or event-driven triggers that can cause a batch to be released early before collecting the full target number(s) of member entities.

Early Triggers)

Release Condition or Probability (Release Batch Early Triggers)	The release condition or probability specified as an expression. If a probability, then enter the chance of releasing the batch early as a value between 0.0 (0%) and 1.0 (100%).
---	---

Listed below are the states of **BatchLogic**:

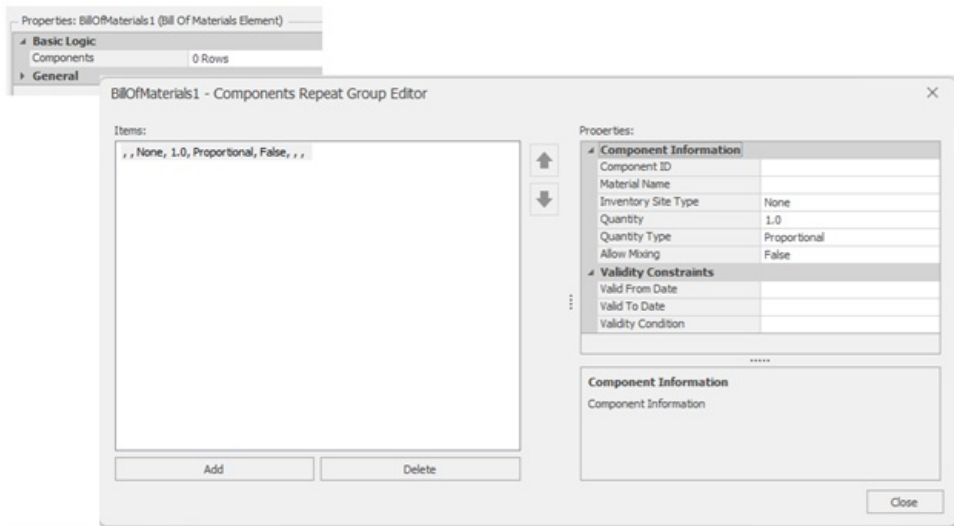
State	Description
ParentQueue	The queue of entities waiting to collect a batch of members.
MemberQueue	The queue of entities waiting to be added as a member to a batch.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

BillOfMaterials

The Bill Of Materials Element may be used to define a comprehensive list of the component materials required to manufacture a product.



Properties

Property Name	Valid Entry	Switch Condition	Description
Components	Repeat Group	None	The list of component materials. NOTE: When executing a stock availability check, alternative materials for the same Component ID will be automatically sorted to prefer first any alternatives with existing stock reservations and then to prefer no mixing. Otherwise, the preference will be in the order listed. If a validity constraint is violated for an alternative material then that alternative's allocation will be restricted to only reserved stock.

Components Repeat Group Properties

Property Name	Valid Entry	Switch Condition	Description
Component ID	String	None	Optional identifier used to group together alternative materials for the same component. This field may be left unspecified to indicate a component material with no alternatives.
Material Name	Material Reference	None	The name of the component material.
Inventory Site Type	None, ParentObject, AssociatedObject, SpecificObject	None	Indicates the fixed object that is the inventory location. Applies only to material elements whose Location Based Inventory property is set to True.
Site Object Name	Fixed Object Reference	Inventory Site Type == SpecificObject	The name of the fixed object that is the inventory location.
Quantity	Real	None	The quantity of the component material.

Quantity Type	Fixed, Proportional	None	The quantity type of the component material. Proportional – The quantity of the component material is proportional to the total quantity of the parent assembly. For example, if the Quantity property is specified as 2 and the parent assembly quantity is 3 then the quantity of the component material is 6 (2 x 3 = 6). Fixed – The quantity of the component material is independent of the total quantity of the parent assembly. For example, if the Quantity property is specified as 2 then the quantity of the component material is always 2 regardless of the parent assembly quantity.
Allow Mixing	True, False	None	Indicates whether the component material can be mixed with other alternatives for the same Component ID to satisfy the component quantity requirement. Using this option requires that the Allow Partial Allocation Condition expression on the associated Inventory or Material element evaluates to true. Please refer to the documentation for examples of component material mixing.
Valid From Date	Date Time	None	Optional start date and time that the component material is considered a valid alternative for the Component ID.
Valid To Date	Date Time	None	Optional end date and time that the component material is considered a valid alternative for the Component ID.
Validity Condition	Expression	None	Optional condition that indicates whether the component material is considered a valid alternative for the Component ID. NOTE: This condition will be evaluated in the context of the owner object referencing the Bill Of Materials element. In the expression, use the syntax [Class].[Attribute] to reference an attribute of that object (e.g., Entity.Priority).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Bill Of Materials - Discussion And Examples

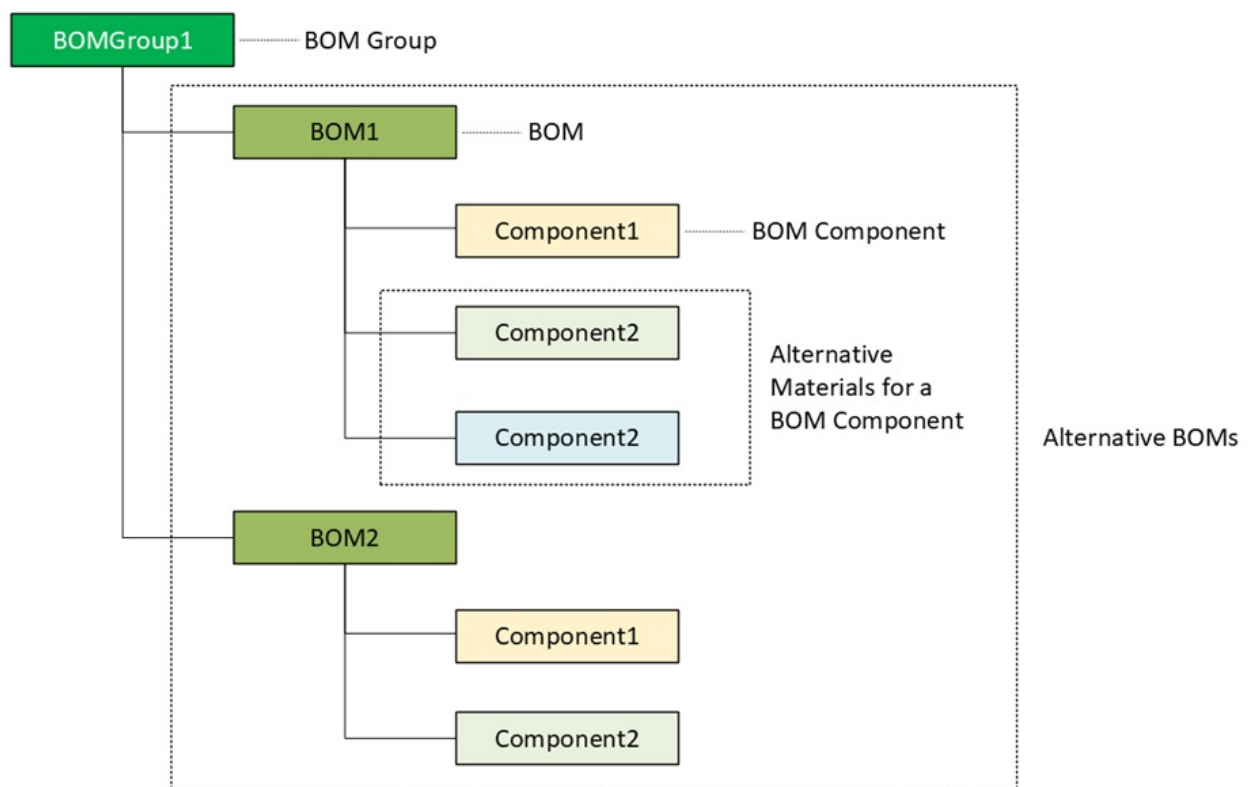
Introduction

A bill of materials (BOM) is a list of the component materials and quantities needed to manufacture a product.

This document describes a new framework for modeling a bill of materials in Simio that consists of the following elements:

- **Bill Of Materials Element** - May be used to define a single BOM with possible alternative materials to satisfy a BOM component.
- **Bill Of Materials Group Element** - May be used to group multiple BOMs together as a set of alternative BOMs to satisfy an assembly.

An overview of the bill of materials framework is shown below.



Key features of the bill of materials framework include:

- Component materials may be allocated from global inventories or from specific inventory sites.
- Component material quantities may be either fixed or proportional to the total quantity of the parent assembly.
- Alternative materials may be defined to satisfy a BOM component, listed in a preferred order and with an option of whether or not to allow mixing of alternatives to satisfy the component quantity requirement.
- Date range or condition constraints may be specified to restrict when an alternative material is considered valid for use in production.
- Multiple BOMs may be grouped together as a set of alternative BOMs to satisfy an assembly, listed in a preferred order.
- Date range or condition constraints may be specified to restrict when an alternative BOM is considered valid for use in production.

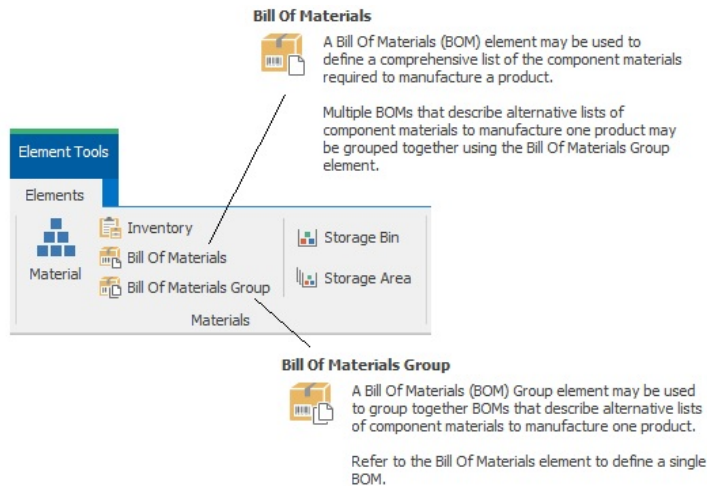
Modeling Overview

Adding Bill of Materials Elements to a Model

To add a **Bill Of Materials** or **Bill Of Materials Group** element to a model, go to the Definitions > Elements > Materials

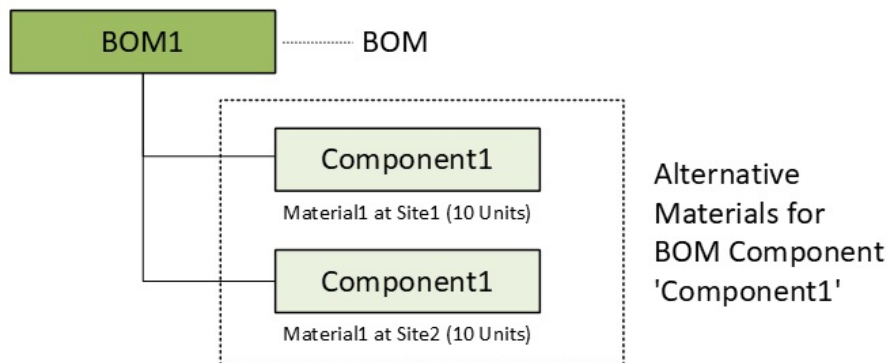
group in the ribbon as shown below.

These elements may also be auto created from data tables like any other element.



Defining Alternative Materials for a BOM Component

The Bill Of Materials element allows alternative materials to be defined to satisfy a BOM component as illustrated by the example below.



For example, 10 units of Material1 from either Site1 or Site2 may be used to satisfy the BOM component identified as Component1. The preference is to first source from Site1 and then if necessary from secondary source Site2.

BOM Component Quantity Types

The Bill Of Materials element supports two primary methods for defining the quantity required to satisfy a BOM component:

Proportional – The quantity of the component material is proportional to the total quantity of the parent assembly. For example, if the *Quantity* property is specified as 2 and the parent assembly quantity is 3 then the quantity of the component material is 6 ($2 \times 3 = 6$).

Fixed – The quantity of the component material is independent of the total quantity of the parent assembly. For example, if the *Quantity* property is specified as 2 then the quantity of the component material is always 2 regardless of the parent assembly quantity.

Mixing Alternative Materials

The Bill Of Materials element provides an *Allow Mixing* option that indicates whether a component material can be mixed with other alternatives for the same *Component ID* to satisfy the component quantity requirement.

For example, referring back to the example above, perhaps material from both Site1 and Site2 can be mixed together to source the total 10 units required if insufficient stock at Site1.

The figure below provides examples that demonstrate the use of the *Allow Mixing* option when alternative materials are defined for a BOM component.

$$\sum \left(\frac{\text{Quantity Allocated}}{\text{Quantity Required}} \right) = 1.0 \text{ for a set of alternative BOM component materials that can be mixed}$$

Example 1:

Component1, Material1, Quantity: 100, **Allow Mixing: False**
 Component1, Material2, Quantity: 100, **Allow Mixing: False**

If 40 of Material1 is available and 150 of Material2 is available then will allocate 100 of Material2 because mixing is not allowed and Material2 is the alternative BOM component with sufficient quantity available.

Example 2:

Component1, Material1, Quantity: 100, **Allow Mixing: False**
 Component1, Material2, Quantity: 200, **Allow Mixing: False**

If 40 of Material1 is available and 150 of Material2 is available then will not be able to allocate any material. Mixing is not allowed and no alternative BOM component has sufficient quantity available.

Example 3:

Component1, Material1, Quantity: 100, **Allow Mixing: True**
 Component1, Material2, Quantity: 100, **Allow Mixing: True**

If 40 of Material1 is available and 150 of Material2 is available then will allocate 40 of Material1 and 60 of Material2 because mixing allowed.

$$40/100 (\text{Material1}) + 60/100 (\text{Material2}) = 1.0$$

Example 4:

Component1, Material1, Quantity: 100, **Allow Mixing: True**
 Component1, Material2, Quantity: 200, **Allow Mixing: True**

If 40 of Material1 is available and 150 of Material2 is available then will allocate 40 of Material1 and 120 of Material2 because mixing allowed.

$$40/100 (\text{Material1}) + 120/200 (\text{Material2}) = 1.0$$

Note that using the *Allow Mixing* option for an alternative material requires that the *Allow Partial Allocation Condition* expression on the associated Inventory or Material element evaluates to true.

Alternative Material Validity Constraints

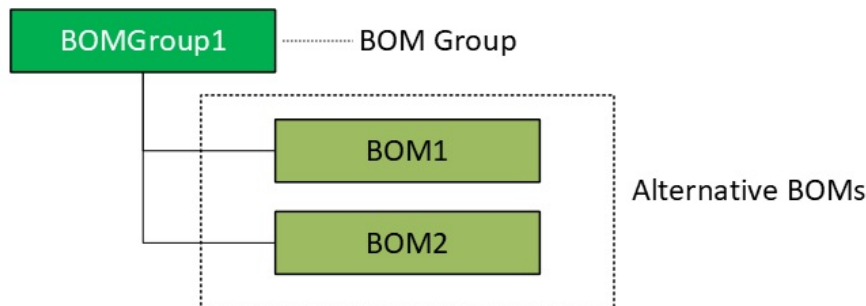
Date range or condition constraints may be specified to restrict when an alternative material for a BOM component is considered valid for use in production.

For example, referring back to the example above, perhaps the alternative material for Component1 from Site1 has a date range constraint whereby that alternative is considered valid only during a specified period.

Or, for example, perhaps the alternative material for Component1 from Site2 has a condition constraint whereby that alternative is considered valid only if the assembly lot size is greater than or equal to a specified minimum value.

Defining Alternative BOMs

The Bill Of Materials Group element may be used to define a group of alternative BOMs to satisfy an assembly as illustrated by the example below.



For example, a group of alternative BOMs might be defined to allow for product variations using different combinations of materials or quantities based on factors such as production date, lot size, or customer requirements.

Alternative BOM Validity Constraints

Date range or condition constraints may be specified to restrict when an alternative BOM is considered valid for use in production.

For example, referring back to the example above, perhaps the alternative BOM1 in BOMGroup1 has a date range constraint whereby that alternative BOM is considered valid only during a specified period.

Or, for example, perhaps the alternative BOM2 in BOMGroup1 has a condition constraint whereby that alternative BOM is considered valid only if the assembly lot size is greater than or equal to a specified minimum value.

Stock Availability Checks

When executing a stock availability check for an assembly referencing a BOM group, the alternative BOMs will be automatically sorted to prefer first any alternatives with existing stock reservations. Otherwise, the preference will be in the order listed.

If a validity date range or condition constraint is violated for an alternative BOM then that alternative's allocation will be restricted to only reserved stock.

When executing a stock availability check for a BOM component, alternative materials for the same *Component ID* will be automatically sorted to prefer first any alternatives with existing stock reservations and then to prefer no mixing. Otherwise, the preference will be in the order listed.

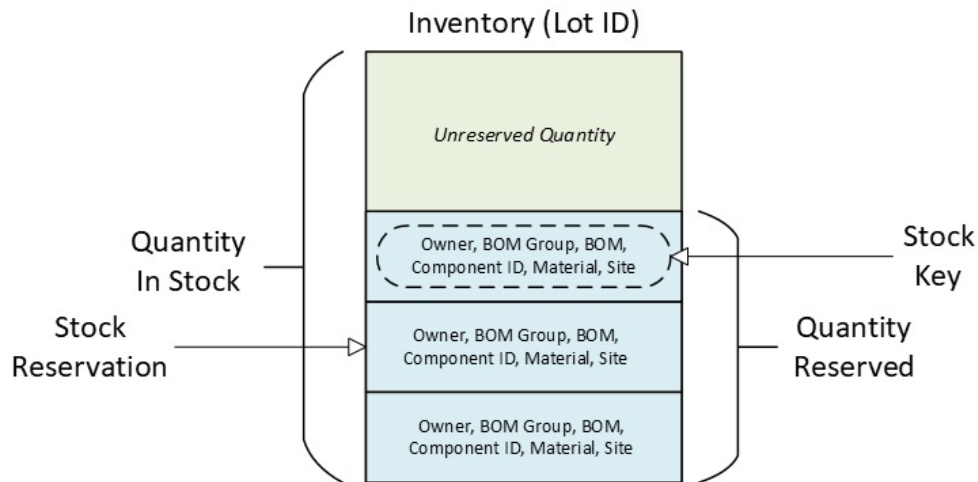
If a validity date range or condition constraint is violated for an alternative material then that alternative's allocation will be restricted to only reserved stock.

Note that if a stock consumption request's *Must Simultaneously Consume* property is false then the first alternative with any

available stock will be selected even if the stock is only partially available.

The figure below illustrates the inventory quantity available calculation for a specific stock key which is an internal key used by the simulation engine to uniquely identify and differentiate material stock quantities for allocation purposes.

For example, if the inventory of Material1 at Site1 currently has 20 unreserved units and 30 units are currently reserved for stock key = {Order1, BOMGroup1, BOM1, Component1, Material1, Site1} then the current quantity available for an operation with that same stock key is 50 units.



$$Quantity\ Available_{Stock\ Key} = Quantity\ In\ Stock - (Quantity\ Reserved - Quantity\ Reserved_{Stock\ Key})$$

$$Quantity\ Available_{Stock\ Key} = Unreserved\ Quantity + Quantity\ Reserved_{Stock\ Key}$$

Referencing BOMs or BOM Groups in Steps or Elements

The following steps and elements support references to BOMs or BOM Groups:

Consume Step

May be used to consume stock on behalf of a specified owner. If the *Must Simultaneously Consume* property is false then partial BOM allocation is allowed.

Produce Step

May be used to produce stock on behalf of a specified owner. Will produce all valid alternative BOMs or alternative materials given the specified validity constraints.

Reserve Step

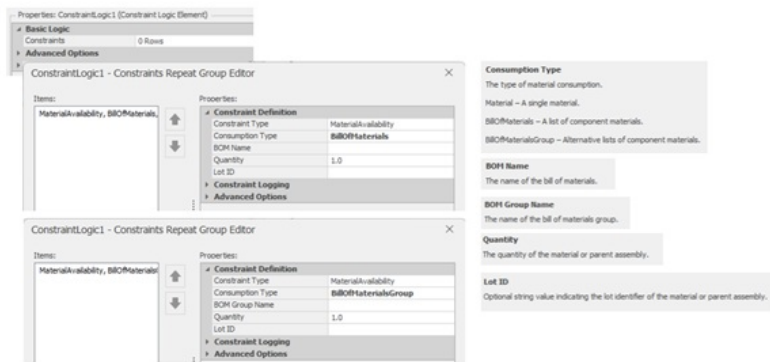
May be used to reserve stock for a specified owner. Will reserve all valid alternative BOMs or alternative materials given the specified validity constraints.

UnReserve Step

May be used to cancel stock reservations up to a specified maximum assembly quantity. Will unreserve any alternative BOMs or alternative materials with reservations (validity constraints do not apply).

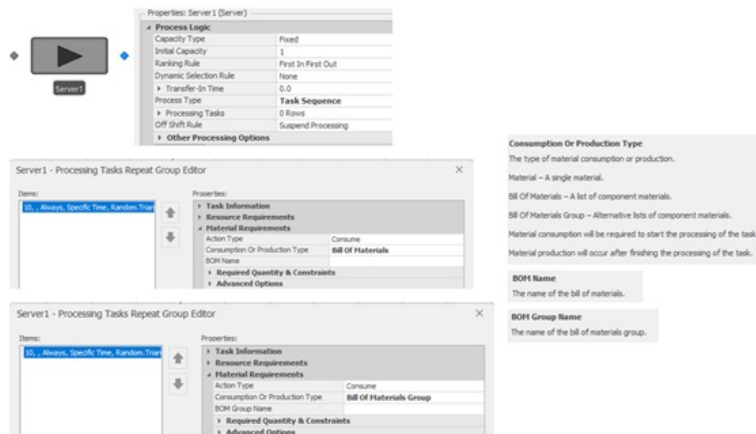
Constraint Logic Element

May be used to enforce material availability constraints on a constrainable process step (e.g., a Seize step). See the figure below. Creates stock reservations when all materials available.



Referencing BOMs or BOM Groups in the Standard Library

A processing task in a Server, Combiner, or Separator object may reference a BOM or BOM Group for material requirements as shown below.

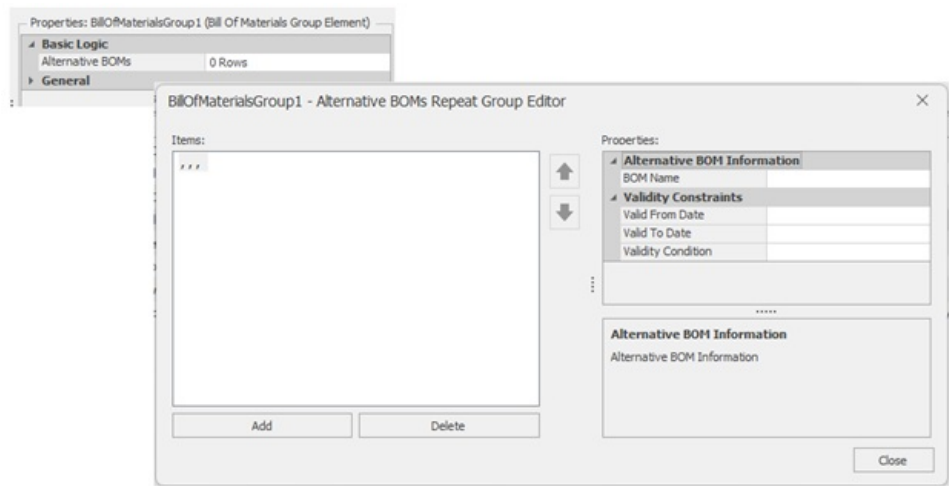


Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Bill Of Materials Group

Bill Of Materials Group Element may be used to group together BOMs that describe alternative lists of component materials to manufacture one product.



Properties

Property Name	Valid Entry	Switch Condition	Description
Alternative BOMs	Repeat Group	None	The alternative bill of materials lists. NOTE: When executing a stock availability check, the alternative BOMs will be automatically sorted to prefer first any alternatives with existing stock reservations. Otherwise, the preference will be in the order listed. If a validity constraint is violated for an alternative BOM then that alternative's allocation will be restricted to only reserved stock.

Alternative BOMs Repeat Group Properties

Property Name	Valid Entry	Switch Condition	Description
BOM Name	Bill Of Materials Reference	None	The name of the alternative bill of materials list.
Valid From Date	Date Time	None	Optional start date and time that the alternative bill of materials list is considered valid.
Valid To Date	Date Time	None	Optional end date and time that the alternative bill of materials list is considered valid.
Validity Condition	Expression	None	Optional condition that indicates whether the alternative bill of materials list is considered valid. NOTE: This condition will be evaluated in the context of the owner object referencing the Bill Of Materials Group element. In the expression, use the syntax [Class].[Attribute] to reference an attribute of that object (e.g., Entity.Priority).

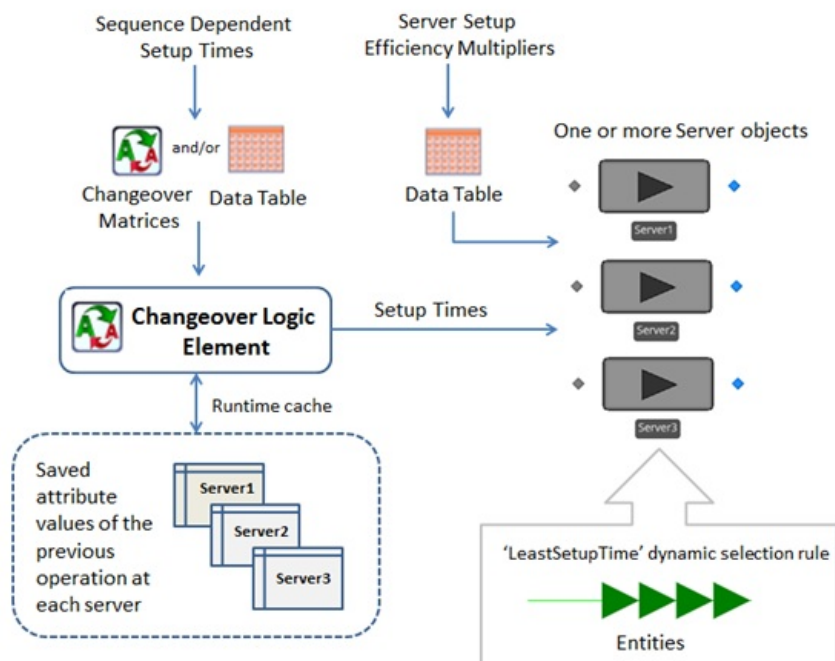
ChangeoverLogic

ChangeoverLogic

The ChangeoverLogic element may be used in conjunction with the [Changeover](#) step and [Changeover](#) matrices to model sequence dependent setups for a single resource or group of resources.

Sequence dependent setup time data is defined in changeover matrices and/or data tables. That setup time data is then referenced by a ChangeoverLogic element which is being used by either a single server or a group of servers. If there are multiple servers using the same changeover logic, then those individual servers may have different setup efficiency multipliers where each setup is possibly performed slower or faster at a particular server than the average time.

During the model run, entities representing jobs might be selected for processing at the servers using the 'LeastSetupTime' dispatching rule. The ChangeoverLogic element takes care of caching the attribute values of the previous operation at each server in order to determine and return the next sequence dependent setup time.



The *Operation Attribute* may return a numeric, string, or element reference expression result. It is evaluated in the context of the resource or operation entity. However:

- If the *From Value Type* or *To Value Type* properties are specified as 'Range', then the *Operation Attribute* expression must return a numeric value.
- If a changeover matrix is being used, then the *Operation Attribute* expression must return a non-negative (zero-based) integer index into the specified changeover matrix's enumerated list of possible from/to setup statuses.

The total number of changeovers and total changeover time incurred at resources due to using a particular ChangeoverLogic element will be automatically included in the results.

Average						Drop Column Fields Here
Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	ChangeoverLogic1	Changeovers	ChangeoverTime	Total (Hours)	0.5000
				NumberChangeovers	Total	1.0000

Per the Overall Equipment Effectiveness (OEE) definition, the resource utilization calculation setting of the 'Setup' and 'OffShiftSetup' resource states are included in allocated capacity statistics but not considered in utilized capacity statistics. If usage cost rates are specified for an object that incurs setup, the usage cost rates are not incurred during setup times.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For examples of using the **ChangeoverLogic** element, please refer to the [SimBits ServersUsingTaskSequenceWithSequenceDependentSetups](#) and [ServersUsingTaskSequenceWithDataTables_SequenceDependentSetups](#).

Listed below are the properties of **ChangeoverLogic**:

Property	Description
Setup Transitions	Repeat group for the possible setup transitions at resources using the changeover logic that will incur a setup time.
Setup Transitions.Operation Attribute	<p>The attribute that will be compared between the previous and next operation to decide the setup time. Specified as an expression evaluated in the context of the resource or operation entity.</p> <p>Note: If using a changeover matrix to determine the setup times, then the values returned by this property must be integer indexes (zero-based) into the specified changeover matrix's enumerated list of possible from/to setup statuses.</p> <p>For example, if the attribute value was '2' for the previous entity and now is '4' for the next entity, then the setup time is the duration entered in the 3rd row, 5th column of the specified changeover matrix.</p> <p>Typically, when the changeover matrix is used, the operation attribute expression will be the name of a list property that is defined either on the operation entity itself or in table data referenced by the entity, and which is based on the same enumerated string list that the changeover matrix is using. For further guidance, please refer to Simio's examples or SimBits that demonstrate changeover matrix usage.</p>
Setup Transitions.Use Changeover Matrix	Indicates whether setup times based on changes in the operation attribute are defined using a changeover matrix.
Setup Transitions.Changeover Matrix Name	The name of the changeover matrix that defines the sequence dependent setup times.
Setup Transitions.From Value Type	The method used to indicate the attribute value of the previous operation.
Setup Transitions.Value	The attribute value of the previous operation.
Setup Transitions.Range Start	The start of the range containing the attribute value of the previous operation.
Setup Transitions.Range End	The end of the range containing the attribute value of the previous operation.
Setup Transitions.To Value Type	The method used to indicate the attribute value of the next operation.
Setup Transitions.Value	The attribute value of the next operation.
Setup Transitions.Range Start	The start of the range containing the attribute value of the next operation.
Setup Transitions.Range End	The end of the range containing the attribute value of the next operation.
Setup Transitions.Setup Time	The required setup time. May be specified as a random distribution or as an expression evaluated in the context of the operation entities.
Setup Efficiency Multiplier	Increases or decreases the total time taken for a setup by some factor. May be specified as an expression evaluated in the context of the associated resource, useful if

multiple resources are using the same changeover logic but have different efficiencies.

Assume Concurrent Setups If	Optional condition that is evaluated at the beginning of each changeover. If the condition is true, then the total time taken for an operation's setup will be the time of the longest setup incurred by the specified setup transitions; otherwise, all of the individual setup times will be added together. May be specified as an expression evaluated in the context of the associated resource, useful if a resource can conditionally perform multiple setups for the next operation concurrently.
-----------------------------	---

Listed below are the functions of **ChangeoverLogic**:

Function	Description
ExpectedSetupTime(resource, entity)	Returns a deterministic estimation of the setup time (in hours) if the changeover logic were to be used for the specified operation entity at the specified resource.
TotalChangeoverTime	Returns the total changeover time (in hours) incurred at resources due to using the changeover logic.
TotalNumberChangeovers	Returns the total number of changeovers that have occurred at resources due to using the changeover logic.

NOTE: The ExpectedSetupTime function may be used in the *Expected Setup Time Expression* property of a Server, Combiner, or Separator to support the use of the 'LeastSetupTime' standard dispatching rule (Dynamic Selection Rule).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

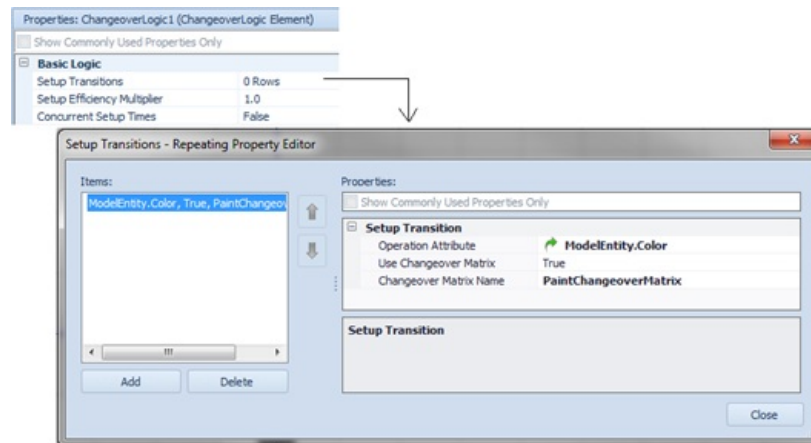
Send comments on this topic to [Support](#)

ChangeoverLogic - Discussion And Examples

Changeover Logic - Setup Transition Examples

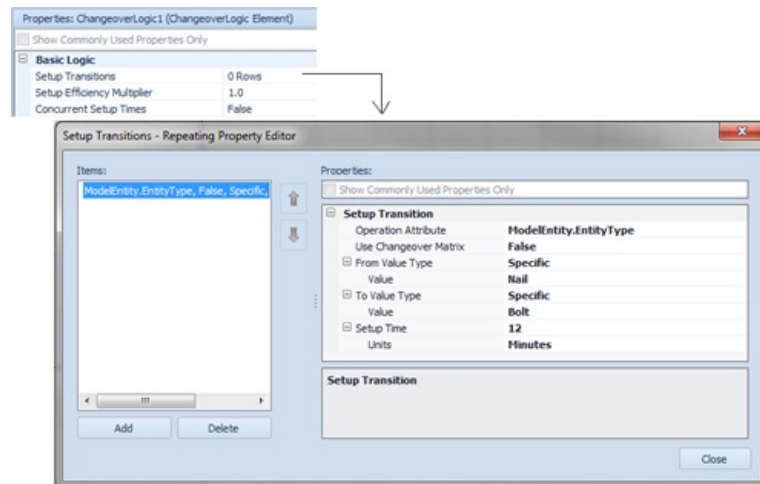
Example 1

The changeover logic has a setup time based on changes in the *Operation Attribute* 'ModelEntity.Color'. The sequence dependent setup times are defined in changeover matrix 'PaintChangeoverMatrix'.



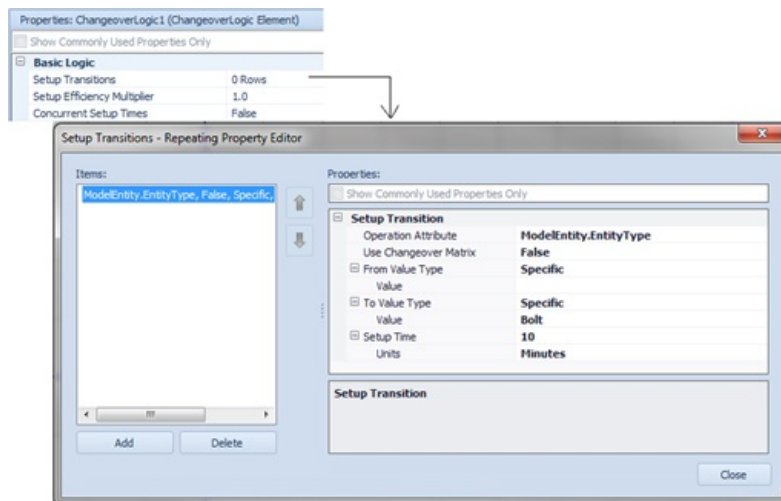
Example 2

The changeover logic has a setup time based on changes in the *Operation Attribute* 'ModelEntity.EntityType'. A setup time of 12 minutes is incurred if the entity type of the previous operation was 'Nail' and the entity type of the next operation is 'Bolt'.



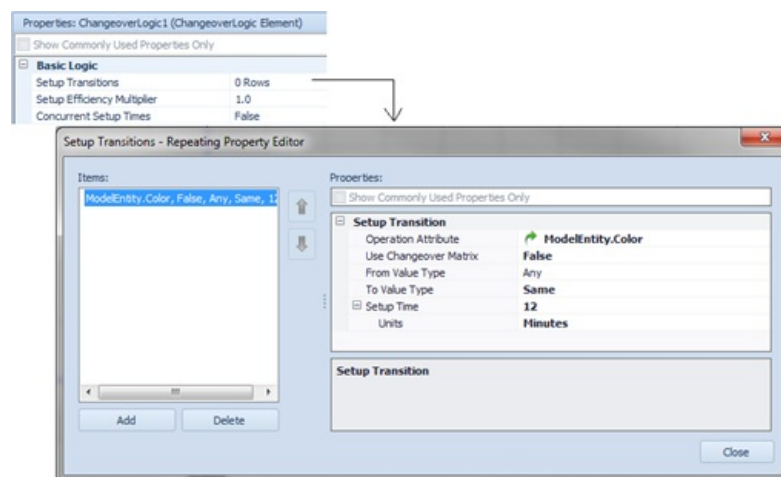
Example 3

The changeover logic has a setup time based on changes in the *Operation Attribute* 'ModelEntity.EntityType'. A setup time of 10 minutes is incurred if the operation is the first operation at the resource (no previous operation value) and the entity type is 'Bolt'.



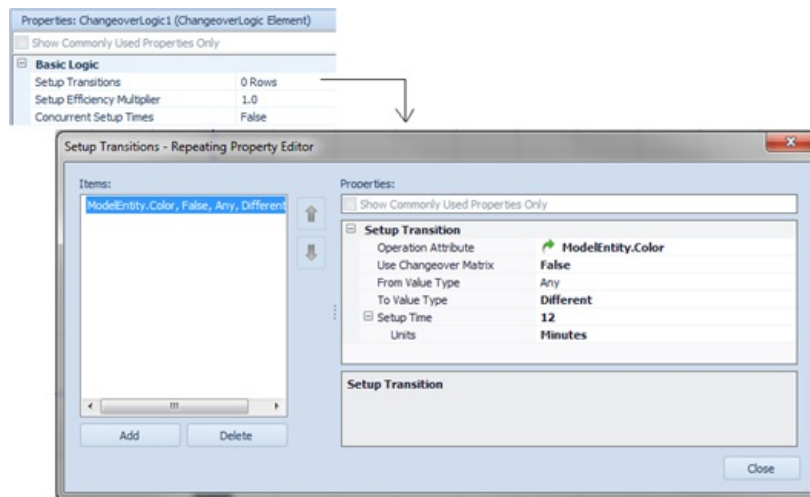
Example 4

The changeover logic has a setup time based on changes in the *Operation Attribute* 'ModelEntity.Color'. A setup time of 12 minutes is incurred if the color of the next operation is the same as the color of the previous operation.



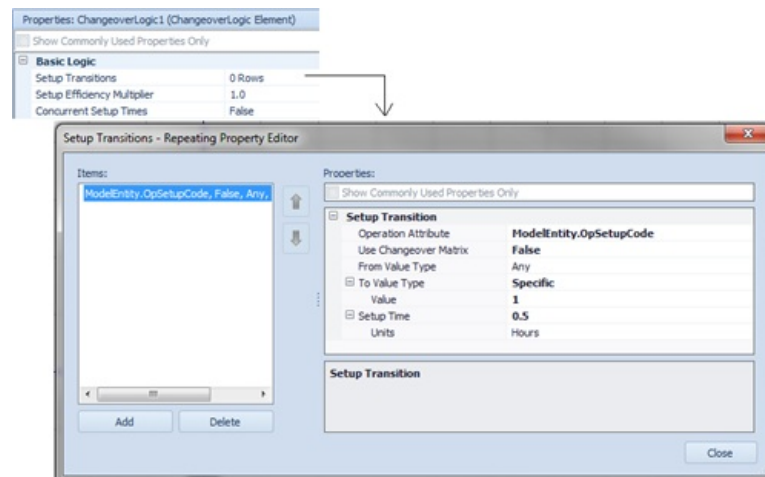
Example 5

The changeover logic has a setup time based on changes in the *Operation Attribute* 'ModelEntity.Color'. A setup time of 12 minutes is incurred if the color of the next operation is different than the color of the previous operation.



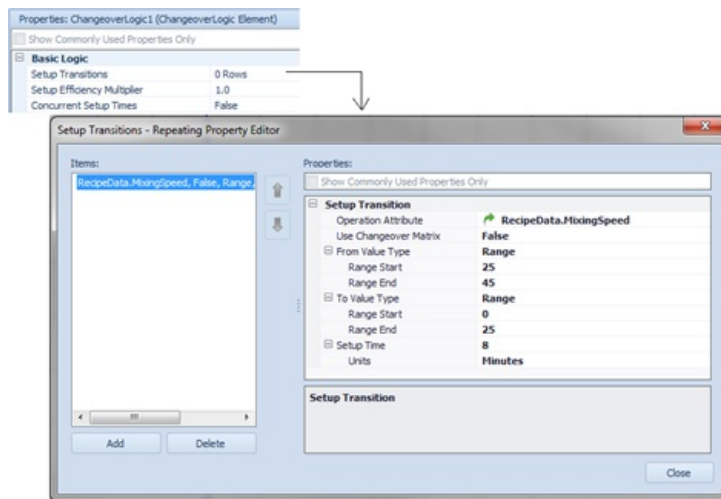
Example 6

The changeover logic has a setup time based on changes in the *Operation Attribute* 'ModelEntity.OpSetupCode'. A setup time of 0.5 hours is always incurred if the setup code of the next operation is '1', regardless of the setup code of the previous operation.



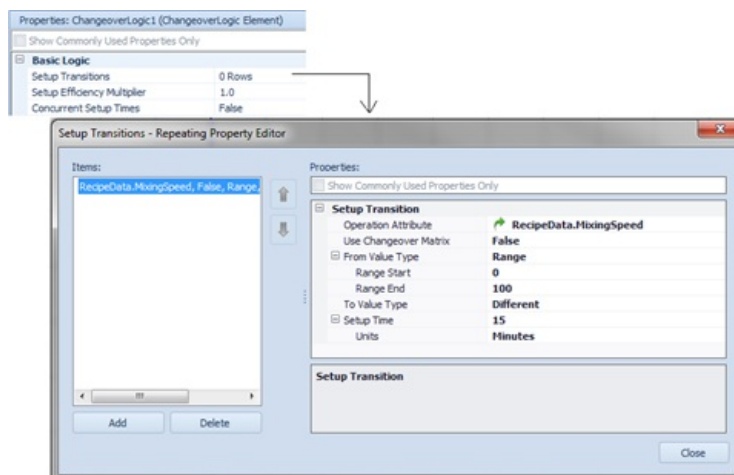
Example 7

The changeover logic has a setup time based on changes in the *Operation Attribute* 'RecipeData.MixingSpeed'. A setup time of 8 minutes is incurred if the mixing speed of the previous operation was in the range 25 to 45 and the mixing speed of the next operation is in the range 0 to 25.



Example 8

The changeover logic has a setup time based on changes in the *Operation Attribute* 'RecipeData.MixingSpeed'. A setup time of 15 minutes is incurred if the mixing speed of the previous operation was in the range 0 to 100 and the mixing speed of the next operation is not in that range.



Example 9

The setup transitions for the changeover logic are defined in a data table.

Changeover Logic Types Setup Transitions

Type

1 PaintChangeoverLogic

Setup Transitions

	Changeover Logic Type	Operation Attribute	From Value	To Value	SetupTime (Minutes)
1	PaintChangeoverLogic	ModelEntity.Color		Red	200
2	PaintChangeoverLogic	ModelEntity.Color		White	100
3	PaintChangeoverLogic	ModelEntity.Color		Blue	100
4	PaintChangeoverLogic	ModelEntity.Color	Red	White	50
5	PaintChangeoverLogic	ModelEntity.Color	Red	Blue	50
6	PaintChangeoverLogic	ModelEntity.Color	White	Red	45
7	PaintChangeoverLogic	ModelEntity.Color	White	Blue	45
8	PaintChangeoverLogic	ModelEntity.Color	Blue	Red	50
9	PaintChangeoverLogic	ModelEntity.Color	Blue	White	30

Initial setup times if first operation

Properties: ChangeoverLogic1 (ChangeoverLogic Element)

Show Commonly Used Properties Only

Basic Logic

Setup Transitions SetupTransitions

Setup Efficiency Multiplier 1.0

Concurrent Setup Times False

Setup Transitions - Repeating Property Editor

Items:

SetupTransitions.OperationAttribute, False

Properties:

Show Commonly Used Properties Only

Setup Transition

Operation Attribute SetupTransitions.OperationAttribute

Use Changeover Matrix False

From Value Type Specific

Value SetupTransitions.FromValue

To Value Type Specific

Value SetupTransitions.ToValue

Setup Time SetupTransitions.SetupTime

Setup Transition

Add Delete Close

Copyright 2006-2025, Simio LLC. All Rights Reserved.

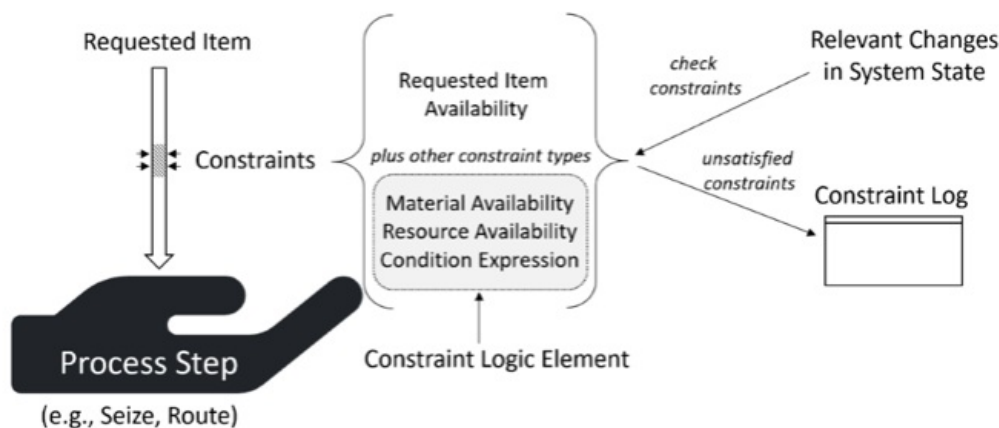
Send comments on this topic to [Support](#)

ConstraintLogic

ConstraintLogic

The Constraint Logic element may be used to enforce material availability, resource availability, or other types of constraint conditions on an applicable process step (e.g., a Seize or Route step). Whereby for example, as shown in Figure 1, a resource cannot be seized or a destination node assigned unless that requested item is available plus the additional constraint logic satisfied. As of Sprint 209, the Constraint element is only available for *Constraint Type* of 'MaterialAvailability', 'ResourceAvailability', and 'Condition Expression'.

The Constraint Logic Element sets up automatic triggers to reevaluate Route Request Queues and Allocations Queues for Route and Seize steps using the Constraint Element. This occurs if a material listed in the Constraint Logic is produced or when a resource listed in the Constraint Logic becomes available.



If the *Constraint Type* is 'MaterialAvailability', then the quantity of the material that is required will be automatically reserved when a resource is successfully seized (Seize step) or when the entity is successfully assigned a destination node (Route step). This is done to ensure that other seize or route requests waiting for the same material do not violate any material availability constraints. See the [Reserve](#) and [Unreserve](#) steps for more information.

If the *Constraint Type* is 'ResourceAvailability', then that resource requirement will be automatically reserved using the specified selection goal when a resource is successfully seized (Seize step) or when the entity is successfully assigned a destination node (Route step). This is done to ensure that other seize or route requests waiting for the same resources do not violate any resource availability constraints.

If the *Constraint Type* is 'ConditionExpression', then a Condition Expression Constraint will be automatically checked whenever the Constraint Logic is checked for the related queue item waiting in the resource's Allocation Queue or Routing Group's Route Request Queue. The *Condition Expression* is evaluated in the context of the constrained entity executing the Seize or Route.

Note: *Exhaustive Constraint Checking*, a property located in a model's Advanced Options Property Pane, provides the option to check Constraint Logic elements after any relevant state change in the system. Setting this property to 'True' will provide the most accurate detail in Trace and the Constraint Log, but may result in longer run times. Setting this property to 'False' will stop the checking when a Constraint is met.

If there are multiple constraints listed at a location, when *Exhaustive Constraint Checking* is 'True', each constraint in the list is checked. This will provide the most in-depth information about constraint availability. Each constraint will be logged in the Constraint Log. If *Exhaustive Constraint Checking* is 'False', the constraint list is checked in order and the first one that fails, the checking stops. So, you will not have information about the other constraints in the list. The tradeoff is check all constraints so more information can be collected which might result in slower run time, or stop checking the constraints right away once one has failed so the model can run faster.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For examples of using the ConstraintLogic element, please refer to the all of the Scheduling related examples within our Support ribbon.

Listed below are the properties of **ConstraintLogic** 'MaterialAvailability':

Property	Description
Constraints	Repeat group for the constraints enforced when referencing the Constraint Logic element.
Constraints.Constraint Type	The constraint type.
Constraints.Consumption Type	The type of material stock consumption. Material – A single material. BillOfMaterials – A list of component materials. BillOfMaterialsGroup – Alternative lists of component materials.
Constraints.Material Name	The name of the material.
Constraints.Inventory Site Type	Indicates the fixed object that is the inventory location. Applies only to material elements whose <i>Location Based Inventory</i> property is set to 'True'.
Constraints.Site Object Name	The name of the fixed object that is the inventory location.
Constraints.BOM Name	The name of the bill of materials.
Constraints.BOM Group Name	The name of the bill of materials group.
Constraints.Quantity	The quantity of the material or parent assembly.
Constraints.Lot ID	Optional string value indicating the lot identifier of the material or parent assembly.
Constraints.Requested Item Condition	Optional condition evaluated for a requested item that must be true for the constraint to be applicable. If route constraint logic, then the requested item will be a destination node. In the expression, use the syntax Candidate.[NodeClass].[Attribute] to reference an attribute of the candidate destination. If seize constraint logic, then the requested item will be a resource object. In the expression, use the syntax Candidate.[ResourceClass].[Attribute] to reference an attribute of the candidate resource. Note: The Request Item Condition value is cached the first time the Entity and the Request Item are evaluated together for a Request.
Constraints.Constraint Scope	Determines the scope of the constraint when it is applied to a process step (e.g., Seize or Route step). Request - Use this scope if the constraint applies irrespective of the items being requested at the process step. The constraint will be checked at most once whenever the eligibility of the request itself is being checked. RequestedItem - Use the Requested Item Condition to identify which requested item(s) the constraint applies to. The constraint will be checked whenever the request is checking the requested item's availability.
Constraints.Skip Constraint (If) Condition	Optional condition indicating whether to skip the constraint entirely regardless of the requested item.

Listed below are the properties of **ConstraintLogic** 'ResourceAvailability':

Property	Description
Constraints	Repeat group for the constraints enforced when referencing the Constraint Logic element.

Constraints.Constraint Type	The constraint type.
Constraints.Resource Type	The method for specifying the resource object(s) required.
Constraints.Resource Name	The name of the resource object required.
Constraints.Resource List Name	The name of the object list from which to select the resource object(s) required.
Constraints.Selection Goal	The rule used to select a resource object from multiple candidates.
Constraints.Value Expression	The expression used to get the value for each candidate resource. In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).
Constraints.Number of Resources	The number of individual resource objects required.
Constraints.Units per Resource	The number of capacity units required per resource object.
Constraints.Selection Condition	Optional condition evaluated for each candidate resource that must be true for the resource to be selected. In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).
Constraints.Requested Item Condition	Optional condition evaluated for a requested item that must be true for the constraint to be applicable. If route constraint logic, then the requested item will be a destination node. In the expression, use the syntax <code>Candidate.[NodeClass].[Attribute]</code> to reference an attribute of the candidate destination. If seize constraint logic, then the requested item will be a resource object. In the expression, use the syntax <code>Candidate.[ResourceClass].[Attribute]</code> to reference an attribute of the candidate resource.
Constraints.Constraint Scope	Determines the scope of the constraint when it is applied to a process step (e.g., Seize or Route step). Request - Use this scope if the constraint applies irrespective of the items being requested at the process step. The constraint will be checked at most once whenever the eligibility of the request itself is being checked. RequestedItem - Use the Requested Item Condition to identify which requested item(s) the constraint applies to. The constraint will be checked whenever the request is checking the requested item's availability.
Constraints.Requested Item Condition	Optional condition evaluated for a requested item that must be true for the constraint to be applicable. This condition is evaluated once per requested item for a request, at the time the requested item's availability is first checked. If route constraint logic, then the requested item will be a destination node. In the expression, use the syntax <code>Candidate.[NodeClass].[Attribute]</code> to reference an attribute of the candidate destination. If seize constraint logic, then the requested item will be a resource object. In the expression, use the syntax <code>Candidate.[ResourceClass].[Attribute]</code> to reference an attribute of the candidate resource.
Constraints.Skip Constraint (If) Condition	Optional condition indicating whether to skip the constraint entirely, excluding it from the constraint logic. This condition is evaluated once per request, at the start of execution of the constrained process step (e.g., the Seize or Route step).

Listed below are the properties of **ConstraintLogic** 'ConditionExpression':

Property	Description
Constraints	Repeat group for the constraints enforced when referencing the Constraint Logic element.
Constraints.Constraint Type	The constraint type.
Constraints.Condition Expression	The condition expression that must evaluate to true.
Constraints.Monitoring Event Name	Optional name of an Event whose occurrence will trigger a check of the condition expression
Constraints.Constraint Type	Optional custom value written to the Constraint Type field in the Constraint Log. May be specified as an expression that evaluates to a string.
Constraints.Constraint Item	Optional custom value written to the Constraint Item field in the Constraint Log. May be specified as an expression that evaluates to a string.
Constraints.Constraint Description	Optional custom value written to the Constraint Description field in the Constraint Log. May be specified as an expression that evaluates to a string.
Constraints.Constraint Scope	Determines the scope of the constraint when it is applied to a process step (e.g., Seize or Route step). Request - Use this scope if the constraint applies irrespective of the items being requested at the process step. The constraint will be checked at most once whenever the eligibility of the request itself is being checked. RequestedItem - Use the Requested Item Condition to identify which requested item(s) the constraint applies to. The constraint will be checked whenever the request is checking the requested item's availability.
Constraints.Requested Item Condition	Optional condition evaluated for a requested item that must be true for the constraint to be applicable. If route constraint logic, then the requested item will be a destination node. In the expression, use the syntax Candidate.[NodeClass].[Attribute] to reference an attribute of the candidate destination. If seize constraint logic, then the requested item will be a resource object. In the expression, use the syntax Candidate.[ResourceClass].[Attribute] to reference an attribute of the candidate resource.
Constraints.Skip Constraint If	Optional condition indicating whether to skip the constraint entirely, excluding it from the constraint logic. The condition is evaluated once per request, at the start of execution of the constrained process step (e.g., the Seize or Route step).

Owner Type: The object that is the owner of the constraint logic. All constraints will be evaluated in this specified owner context. May be specified as either the object associated with the constrained process step, the parent object, or as a specific object reference.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Container

Container

The Container element may be used to define a volume or weight capacity constrained location for holding entities representing quantities of fluids or other mass. Use the [Transfer](#) step to initiate transfers of entities into or out of a container.

The states listed below for the flow into and out of Containers are read-only. These may be monitored to detect when a target volume or weight quantity has been transferred (e.g., new monitor threshold = $\text{ContainerName.CurrentVolumeFlowIn} + \text{SomeVolumeQuantity}$). The rate values of the state variables might be monitored to detect flow rate changes (e.g., you could monitor $\text{ContainerX.CurrentVolumeFlowIn.Rate}$ for changes).

Statistics are generated automatically for the flow in and flow out of a container based on the flow states listed below.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For examples of using the Container element, please refer to the Flow Library objects including FlowSource, FlowSink, and Tank.

Listed below are the properties of **Container**:

Property	Description
Initial Volume Capacity	The initial maximum contents volume for this container.
Initial Weight Capacity	The initial maximum contents weight for this container.
Contents Ranking Rule	The static rule used to rank the entities representing quantities of fluids or other mass that are located in this container's 'Contents' queue.
On Full Process	Optional process that is executed when the container has become full per its current maximum volume or weight capacities.
On Empty Process	Optional process that is executed when the container has become empty.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Listed below are the States of **Container**:

State	Description
CurrentVolumeCapacity	State used to get or set the current volume capacity of this container.
CurrentWeightCapacity	State to get or set the current weight capacity of this container.
CurrentVolumeFlowIn	State to get the current total volume flowed into this container. The 'rate' from this state is also available.
CurrentVolumeFlowOut	State to get the current total volume flowed out of this container. The 'rate' from this state is also available.
CurrentWeightFlowIn	State to get the current total weight flowed into this container. The 'rate' from this state is also available.

CurrentWeightFlowOut	State to get the current total weight flowed out of this container. The 'rate' from this state is also available.
----------------------	---

Listed below are the Events of **Container**:

Event	Description
Empty	The Empty event provides notification that a container is empty.
Full	The Full event provides notification that a container is full.

Listed below are the Functions of **Container**:

Function	Description
TimeOfNextFullEvent	If inflow is filling the container, returns the simulation time (in hours) that the container will become full. If the container is not being filled then NaN is returned.
TimeUntilNextFullEvent	If inflow is filling the container, returns the time duration remaining (in hours) until the container will become full. If the container is not being filled then NaN is returned.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

CostCenter

CostCenter

The CostCenter may be used to define an identifiable area of responsibility where costs are allocated or incurred. Cost centers may be linked in a hierarchical structure for cost accounting and reporting purposes, whereby costs are automatically rolled up through the cost center hierarchy.

Costs can be manually assigned to a Cost Center element by using an Assign Step. The Cost is updated by assigning a value to CostCenter.Cost and the Rate is updated by assigning a value to CostCenter.Cost.Rate

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **CostCenter**:

Property	Description
Parent Cost Center	The name of a cost center to roll up this element's cost center into. If a Parent Cost Center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this cost center.
Initial Cost	The initial value of this element's Cost state, which stores the total cost that has been allocated to the cost center.
Initial Cost Rate	The initial value of this element's Cost.Rate parameter, which indicates a cost per unit time that is to be accrued to the cost center.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Counter

Overview

A Counter element may be used to track attribute-based counts of entities in the system.

Properties

A Counter element provides the following properties:

Property Name	Valid Entry	Switch Condition	Description
Unit Type	Unspecified, Time, TravelRate, Length, Currency, CurrencyPerTimeUnit, Volume, Weight, VolumeFlowRate, WeightFlowRate, TravelAcceleration, Area, AreaRate		The unit type of the values to count.
Counter Type	NumberWaiting, NumberEnroute	None	The counter type.
Queue State Name	Queue state name	Counter Type == NumberWaiting	The name of the queue state for which to count the current number of entities waiting.
Node Name	Node object reference	Counter Type == NumberEnroute	The name of the node for which to count the current number of entities en route.
Key Expression	Expression	None	The key expression used to count based on an entity attribute.
Value Expression	Expression	None	Increments/decrements the count by the value specified.

Functions

A Counter element provides the following functions:

Function Name	Description
CountForKey(key)	Returns the count for a specified key.
AverageCountForKey(key)	Returns the time-weighted average count for a specified key.
MaximumCountForKey(key)	Returns the maximum count for a specified key.
MinimumCountForKey(key)	Returns the minimum count for a specified key.

Examples

Property Name	Value
Name	Counter1
Counter Type	NumberWaiting
Queue State Name	Server1.InputBuffer.Contents

Key Expression	Entity.EntityType
----------------	-------------------

Will store and provide access to the current number of entities waiting in the queue Server1.InputBuffer.Contents by attribute Entity.EntityType.

If the possible entity types are defined as TypeA and TypeB then:

Counter1.CountForKey(TypeA) will return the current number of entities of TypeA waiting in the queue.

Counter1.CountForKey(TypeB) will return the current number of entities of TypeB waiting in the queue.

Property Name	Value
Name	Counter1
Counter Type	NumberEnroute
Node Name	Input@Server1
Key Expression	Entity.EntityType

Will store and provide access to the current number of entities en route to destination node Input@Server1 by attribute Entity.EntityType.

If the possible entity types are defined as TypeA and TypeB then:

Counter1.CountForKey(TypeA) will return the current number of entities of TypeA en route to the node.

Counter1.CountForKey(TypeB) will return the current number of entities of TypeB en route to the node.

Property Name	Value
Name	Counter1
Counter Type	NumberWaiting
Queue State Name	Server1.InputBuffer.Contents
Key Expression	Orders.Color

Will store and provide access to the current number of entities waiting in the queue Server1.InputBuffer.Contents by attribute Orders.Color which is a data table property.

Counter1.CountForKey(Color.Red) will return the current number of entities of Color.Red waiting in the queue.

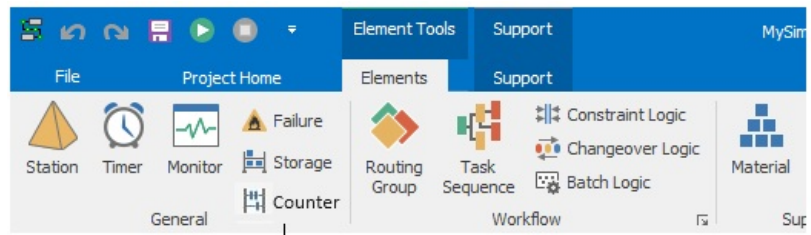
Counter1.CountForKey(Color.Blue) will return the current number of entities of Color.Blue waiting in the queue.

Remarks

If *Counter Type* is NumberWaiting, then the *Key Expression* is evaluated once for an entity when inserted into the specified queue state to increment the count for the specified key. The count for the same key is decremented when the entity is removed from the queue.

If *Counter Type* is NumberEnroute, then the *Key Expression* is evaluated once for an entity when its destination is set to the specified node to increment the count for the specified key. The count for the same key is decremented when the entity's destination is no longer set to the node or if the entity is waiting to be assigned a new destination by a Route step.

A button in the Elements Ribbon tab will be provided to add a new Counter element to a model. The button will be located in the General ribbon group.



Button to add a Counter element

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Failure

Failure

The Failure element may be used to define a failure mode. Failure downtime occurrences are started and ended using the Fail and Repair steps.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the Properties of **Failure**:

Property	Description
Report Statistics	Specifies if statistics are to be automatically reported for this element.
On Failed Process	Optional process that is executed when a failure downtime occurrence begins.
On Repaired Process	Optional process that is executed when a failure downtime occurrence ends.

Listed below are the Functions associated with a **Failure**:

Function	Description
Active	Returns the value of 1 if the Failure is currently active (object is in a failed state) and returns a value of 0 if the Failure is not currently active (object is not failed)
AverageDowntime	Returns the average amount of time that this Failure was active (object in failed state)
LastFailureStartTime	Returns the start time of the last failure.
MaximumDowntime	Returns the maximum amount of time that this Failure was active (object in failed state)
MinimumDowntime	Returns the minimum amount of time that this Failure was active (object in failed state)
NumberOccurrences	Returns the number of times that this Failure has become active (object in failed state)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

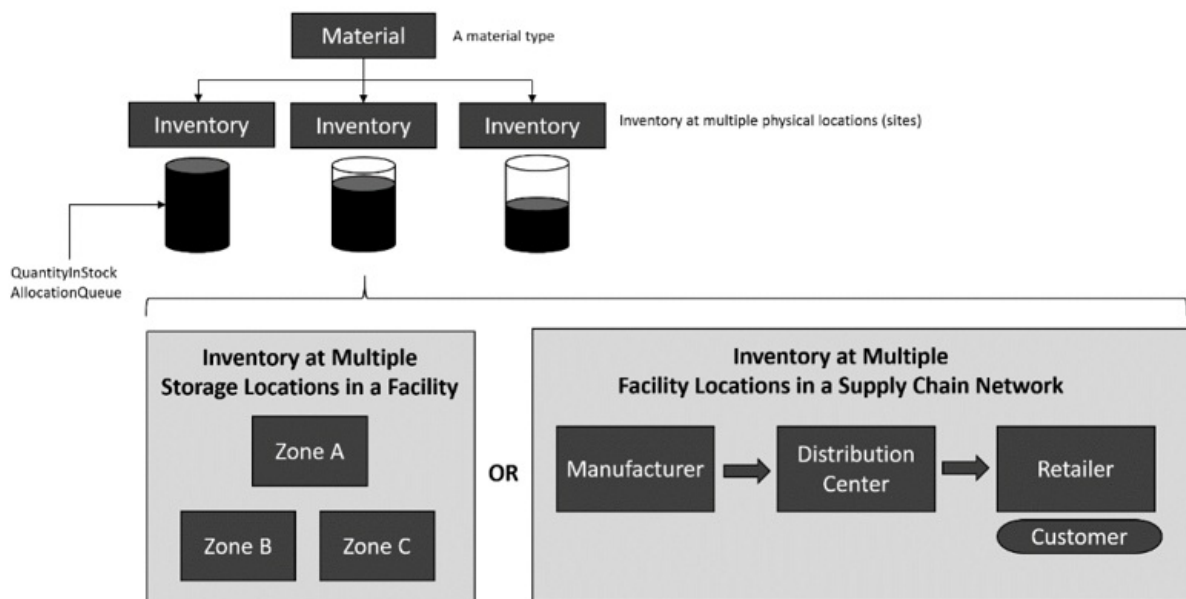
Send comments on this topic to [Support](#)

Inventory

Inventory

The Inventory element may be used to define a storage bucket for holding stock of a specified material. A Site Object Name property is provided to designate a fixed object in the Facility window as the inventory's physical location. This element makes it easier for users to model material stock that is stored at multiple physical locations in the system, with each inventory having its own stock level and allocation queue.

Inventory Overview



The specified *Material Name* must be a [Material](#) element whose *Location Based Inventory* property is set to 'True'. Additionally, the *Material Name* - *Site Object Name* pair of each Inventory element must be unique. In both the prior cases, an error will be displayed at run initialization.

New material consumption requests are always inserted into the inventory's allocation queue and a late priority current event then scheduled to try allocation.

If an inventory's *Review Period* is specified as 'Timer', then the *Review Timer Name* will typically be a fixed-interval or repeating schedule (arrival table) timer to trigger periodic inventory reviews.

When an inventory review is triggered, a late priority event will be scheduled on the simulation's current events calendar to check the inventory's replenishment policy.

A *Replenishment Policy* property type specifies a reference to an inventory replenishment policy. Several replenishment policy examples are installed with Simio: 'Min/Max' policy, 'Reorder Point/Reorder Qty' policy, 'Order-Up-To' policy, and a 'Custom Reorder Condition' policy. Refer to the [Inventory Replenishment Policies](#) installed with Simio for more information.

A process token created to execute an inventory's *On Replenishment Order Process* will have the following characteristics:

- Its associated object reference will be set to the inventory's site object.
- It will be initialized with the same table row references as the Inventory element.
- Its [material order detail](#) reference will provide the detail of the replenishment request.

Whenever an inventory's *On Replenishment Order Process* is initiated by Simio, the inventory's quantity on order will be automatically incremented by the calculated replenishment order size. If further adjustments are necessary, the inventory's *QuantityOnOrder* state variable may be assigned using an Assign step.

The *Allow Partial Allocation If* property may be specified as a conditional expression that is evaluated in the context of the entity requesting the material. If the specified expression contains a data table reference, then that table value may be a

row reference that has been set for either the Inventory element, the process token that is executing the Consume step, or the entity (in that search priority order).

Refer to the [Inventory and Material Backorders](#) and [Measuring Service Level](#) sections in help for additional details.

See the [InventoryAndMaterials](#) SimBit and [MultiEchelonSupplyChain](#) example for examples of the Inventory and Material elements.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **Inventory**:

Property	Description
Material Name	The name of the material held in the inventory.
Site Object Name	The name of the fixed object that is the inventory's physical location.
Initial Quantity	The quantity of material present in the inventory at the beginning of the simulation run.
Review Period	The frequency of inventory review to determine whether a replenishment order is required. If the review period is 'Continuous', then an inventory review is triggered once at the start of the simulation and then whenever the inventory's position decreases. If the review period is 'Timer', then an inventory review is triggered whenever the specified timer fires its event.
Review Timer Name	The name of the timer used to trigger the inventory reviews.
Replenishment Policy	The replenishment policy checked at the time of an inventory review to determine whether a replenishment order is required and, if so, the recommended order size.
Reorder Condition	The condition that, if true, signals the need to place a replenishment order.
Reorder Quantity	The fixed reorder quantity when placing a replenishment order.
Red Zone Size	The expression used to calculate and update the buffer red zone size. The red zone is the safety embedded in the buffer.
Yellow Zone Size	The expression used to calculate and update the buffer yellow zone size. The top of the yellow zone is the minimum threshold for the net flow position that signals the need to place a replenishment order.
Green Zone Size	The expression used to calculate and update the buffer green zone size. The top of the green zone is the target level to return the net flow position to if at or below the top of the yellow zone.
Qualified Spike Demand	The expression used to calculate and update the qualified spike demand value for calculating the net flow position.
Reorder Point	The minimum threshold for the inventory position that signals the need to place a replenishment order.
Order-Up-To-Level	The target level to return the inventory position to if at or below the reorder point.
Log Inventory	Indicates whether inventory review events are to be automatically logged. Go to Results ->

Reviews	Logs -> Inventory Review Log to view the logged data.
On Replenishment Order Process	<p>The name of the process that will be executed to handle an inventory replenishment order.</p> <p>Note that whenever this process is triggered by the replenishment policy, the inventory's quantity on order will have been automatically increased by the recommended order size.</p> <p>NOTE: The created token's material order detail reference will provide the detail of the replenishment request. For example, the function <code>Token.MaterialOrderDetail.Quantity</code> may be used to get the recommended order size.</p>
Allocation Ranking Rule	The ranking rule used to allocate material from the inventory to competing material consumption requests.
Allocation Ranking Expression	The expression used with a <code>SmallestValueFirst</code> or <code>LargestValueFirst</code> ranking rule.
Assume Infinite Availability If	<p>Optional condition that is evaluated whenever an entity is attempting to consume material from the inventory. If the condition is true, then the entity will be allowed to immediately consume its full required quantity of the material regardless of the actual quantity in stock. Note that this may result in negative inventory levels.</p> <p>If specified, this condition overrides the <code>Assume Infinite Availability Condition</code> on the associated Material element.</p>
Allow Partial Allocation If	<p>Optional condition that is evaluated whenever an entity is attempting to consume material from the inventory. If the condition is true, then the entity will be allowed to immediately consume only a portion of its requested quantity if the full quantity is not available.</p> <p>If specified, this condition overrides the <i>Allow Partial Allocation If</i> condition on the associated Material element.</p>
Allow Backorder Policy	<p>Indicates whether an entity attempting to consume material from the inventory can be backordered if its full requested quantity is not immediately satisfied from inventory on hand.</p> <p>If the backordering of an entity is rejected, then its outstanding requested quantity will be automatically cancelled and the balked demand observation recorded in the statistics.</p>
Balk Condition or Probability	The balk at backorder condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
On Balked At Backorder Process	<p>The name of the process that will be executed to handle an entity that has balked at being backordered.</p> <p>NOTE: The created token's material order detail reference will provide the detail of the balked material consumption request. For example, the function <code>Token.MaterialOrderDetail.Quantity</code> may be used to get the demand quantity cancelled due to the balked request.</p>
Backorder Max Wait Time	<p>An optional limit on the total time that a backordered entity will wait for its full requested quantity of material from the inventory.</p> <p>If the max wait time for a backordered entity is reached, then its outstanding requested quantity will be automatically cancelled and the reneged demand observation recorded in the statistics.</p>
On Reneged Backorder Process	<p>The name of the process that will be executed to handle an entity whose backorder has been cancelled.</p> <p>NOTE: The created token's material order detail reference will provide the detail of the reneged material consumption request. For example, the function <code>Token.MaterialOrderDetail.Quantity</code> may be used to get the demand quantity cancelled due to the reneged request.</p>

Listed below are the states of **Inventory**:

State	Description
QuantityInStock	The current quantity of material present in the inventory.

QuantityOnOrder The current quantity of material that has been ordered to replenish the inventory but has not yet been received. This state variable may be assigned using an Assign step.

AllocationQueue Accesses the inventory's queue of material consumption requests.

Listed below are the functions of **Inventory**:

Function	Description
InventoryPosition	Returns the inventory's quantity in stock plus quantity on order minus quantity backordered minus quantity reserved (excluding reserved units already backordered).
AverageQuantityInStock	Returns the average quantity of material present in the inventory during the simulation run.
MinimumQuantityInStock	Returns the minimum quantity of material present in the inventory during the simulation run.
MaximumQuantityInStock	Returns the maximum quantity of material present in the inventory during the simulation run.
QuantityConsumed	Returns the total quantity of material removed from the inventory during the simulation run.
QuantityProduced	Returns the total quantity of material added to the inventory during the simulation run.
QuantityReserved	Returns the current quantity of material from the inventory that is reserved.
QuantityReservedTo (owner)	Returns the current quantity of material from the inventory that is reserved for use by a specified owner object.
QuantityBackordered	Returns the current quantity of material required to satisfy all material consumption requests waiting in the inventory's allocation queue.
AverageQuantityBackordered	Returns the average quantity backordered during the simulation run.
MinimumQuantityBackordered	Returns the minimum quantity backordered during the simulation run.
MaximumQuantityBackordered	Returns the maximum quantity backordered during the simulation run.
Demand	Provides functions for accessing additional statistics on the inventory's demand.
Demand.NumberOrders	Returns the total number of material consumption requests (i.e., orders) received during the simulation run.
Demand.QuantityDemanded	Returns the total quantity that was demanded to satisfy the material consumption requests received during the simulation run.
Demand.NumberBalked	Returns the total number of material consumption requests that balked at being backordered in the inventory's allocation queue.
Demand.QuantityBalked	Returns the total demand quantity cancelled due to balked material consumption requests.
Demand.NumberReneged	Returns the total number of material consumption requests that abandoned waiting as a backorder in the inventory's allocation queue.

Demand.QuantityReneged	Returns the total demand quantity cancelled due to reneged material consumption requests.
Demand.OrderFillRate	Returns the percentage of material consumption requests (i.e., orders) whose full quantity demanded was immediately satisfied from inventory on hand (no balking or backordering).
Demand.QuantityFillRate	Returns the percentage of the total quantity demanded that was immediately satisfied from inventory on hand (no balking or backordering).
Stockout	Provides functions for accessing time-dependent statistics on the inventory's stockout frequency.
Stockout.TotalTime	Returns the total time that no material was present in the inventory.
Stockout.PercentTime	Returns the percent time that no material was present in the inventory.
Stockout.NumberOccurrences	Returns the number of occurrences where no material was present in the inventory.
Stockout.AverageTime	Returns the average time of the occurrences where no material was present in the inventory.
UsageCostCharged	Returns the total cost that has been charged to consumers of material from that inventory.

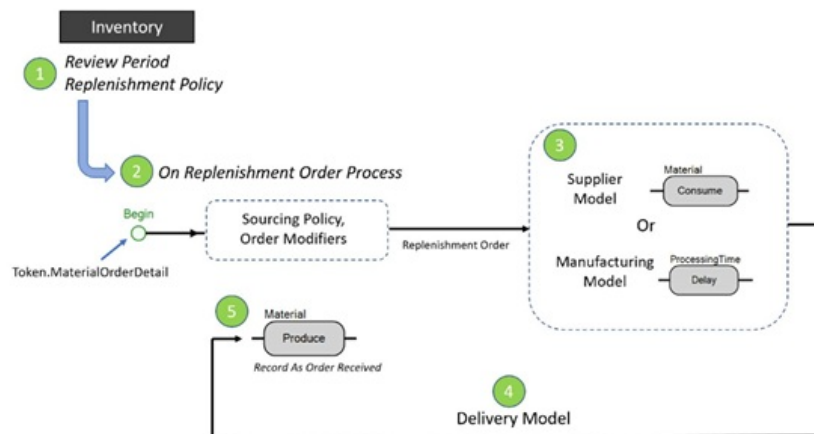
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Inventory Replenishment

Inventory Replenishment Framework and Overview

The graphic below which illustrates the framework that is available in Simio to model inventory replenishment. See the [MultiEchelonSupplyChain](#) example under Support ribbon / Examples as well.



Inventory is material stored, waiting for processing, or experiencing processing. In a typical supply-and-demand system, customers (or demand transactions) request quantities of material. The material consumption requests are filled from inventory on hand. If insufficient inventory is available, then unsatisfied requests are backordered. In some cases, a material shortage may be only a small inconvenience, while for other cases it may cause a severe problem such as cancelled sales orders or interruptions at a production line.

The primary function of inventory is to prevent or limit material shortages caused by uncertainties inherent in demand and/or supply. In a perfect world, a company would always have infinite inventory on hand to reduce the risk of a shortage to zero. However, the reality is excessive inventories can incur high storage costs, tie up significant business capital, and may be subject to spoilage or obsolescence. Because of this, a major focus for many businesses is implementing inventory replenishment methods that strike the right balance between meeting desired service levels and minimizing inventory-related costs.

Simply defined, inventory replenishment is decision making that is focused on answering two fundamental questions:

- When should orders be placed to replenish inventory?
- How much should be ordered?

The remainder of this section introduces some key terms and common inventory replenishment policies.

Inventory Position

Returns the inventory's quantity in stock plus quantity on order minus quantity backordered minus quantity reserved (excluding reserved units already backordered).

Inventory Position = Quantity In Stock + Quantity On Order - Quantity Backordered - Quantity Reserved (excluding reserved units already backordered)

Continuous or Periodic Inventory Review

These terms refer to the frequency of inventory review to determine when orders must be placed for replenishment.

A *continuous inventory review* keeps a constant track of the inventory position; as soon as it falls below a pre-determined level (*the reorder point*), a replenishment order is placed. Tracking inventory levels in real-time is typically more expensive to administer but allows for a lower level of safety stock, as the only uncertainty is the magnitude of demand during the delivery period.

Alternatively, a *periodic inventory review* evaluates the inventory position at discrete points in time to determine if a

replenishment order needs to be placed. Replenishment decisions can be made only at those points. The time between two review points is called the *review period*. A periodic review system is cheaper to administer compared to continuous review since inventory counts take place only at fixed times, but a higher level of safety stock is typically required to buffer against a longer period of uncertainty in demand.

Replenishment Policies

At the time of an inventory review, a decision strategy often referred to as the replenishment policy is used to determine whether replenishment is required and, if so, then by how much. Some commonly used replenishment policies included within Simio:

Min/Max Replenishment Policy

This policy is sometimes referred to as the (s, S) policy where s is the reorder point, the 'Min', and S is the order-up-to level, the 'Max'. When the inventory position falls to or below the reorder point, s , then replenishment is required so as to bring the inventory position to the order-up-to level, S . In other words, if the inventory position is y , and $y \leq s$, then a replenishment order of size $S - y$ is required.

Figures 1 and 2 illustrate the Min/Max replenishment policy using either continuous or periodic inventory review.

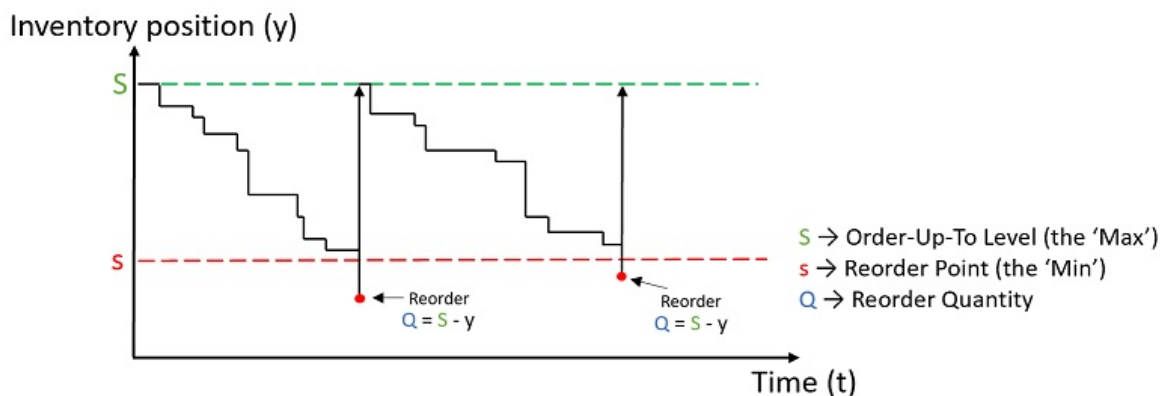


Figure 1 – Continuous Review Min/Max Replenishment Policy

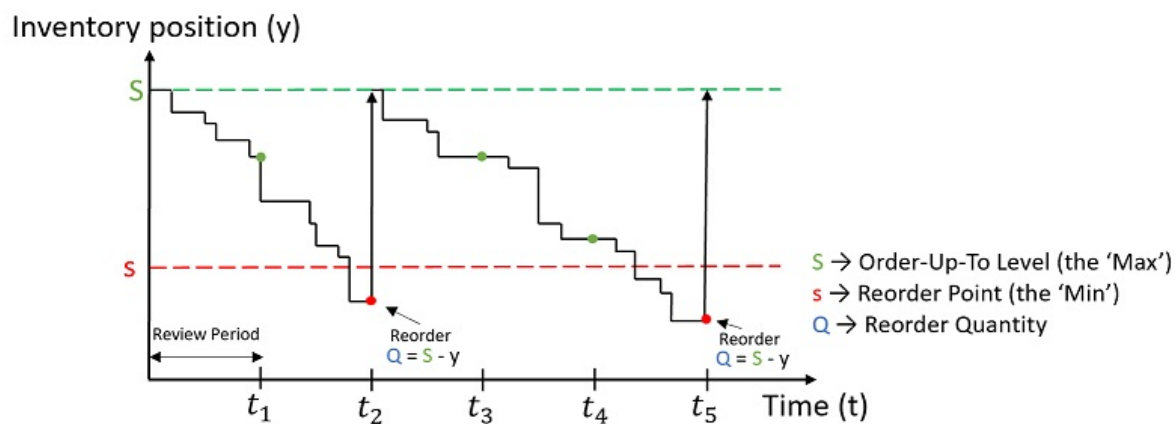


Figure 2 – Periodic Review Min/Max Replenishment Policy

Order-Up-To Replenishment Policy

This policy is sometimes referred to as the *base-stock policy* or "one-for-one" policy. When the inventory position decreases below the order-up-to level, S , then replenishment is required so as to bring the inventory position back to the order-up-to level. In other words, if the inventory position is y , and $y < S$, then a replenishment order of size $S - y$ is required.

Figures 3 and 4 illustrate the Order-Up-To replenishment policy using either continuous or periodic inventory review.

Inventory position (y)

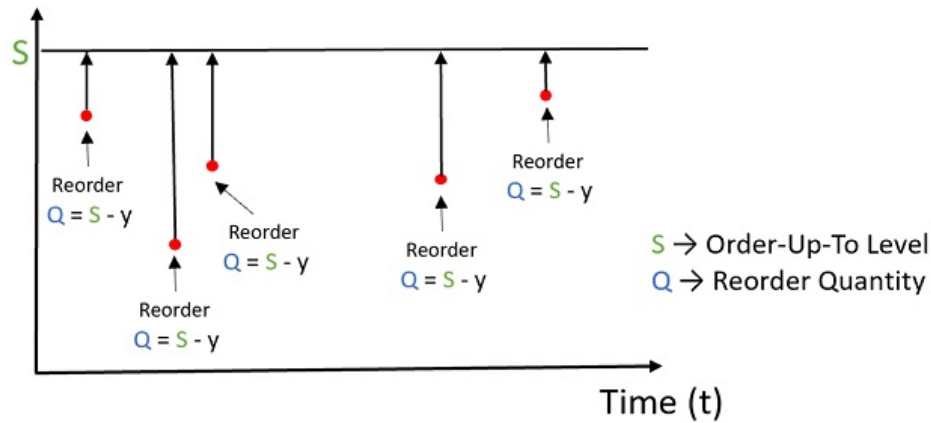


Figure 3 - Continuous Review Order-Up-To Replenishment Policy

Inventory position (y)

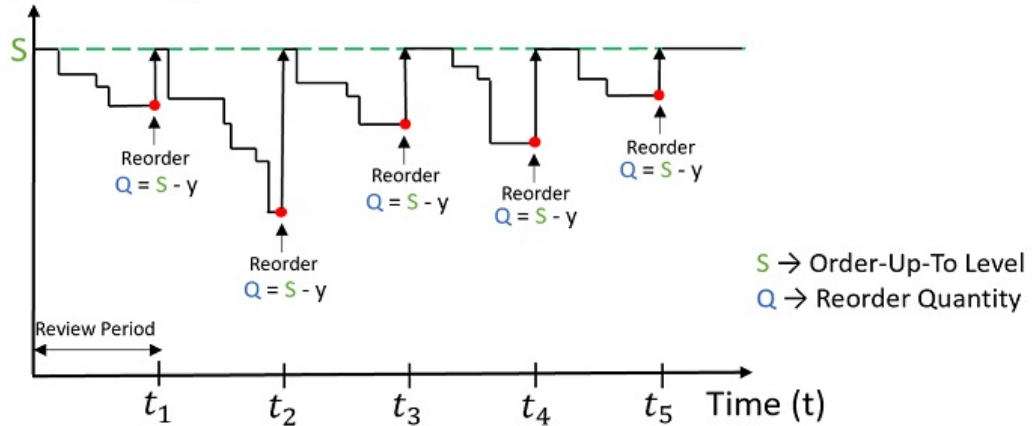


Figure 4 - Periodic Review Order-Up-To Replenishment Policy

Reorder Point/Reorder Qty Replenishment Policy

This policy is sometimes referred to as the (s, nQ) policy where s is the reorder point and Q is a fixed reorder quantity (a fixed lot size). When the inventory position falls to or below the reorder point, s , then replenishment is required so as to bring the inventory position just above s . The size of the replenishment order is a multiple of the reorder quantity, Q . In other words, if the inventory position is y , and $y \leq s$, then a replenishment order of size nQ is required, where n is the smallest integer such that $y + nQ > s$.

Figures 5 and 6 illustrate the Reorder Point/Reorder Qty replenishment policy using either continuous or periodic inventory review.

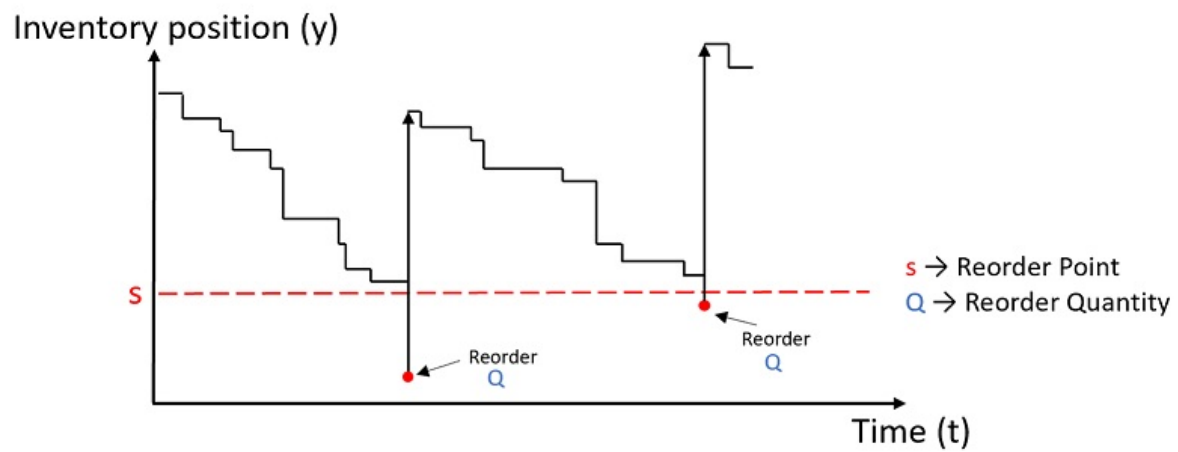


Figure 5 - Continuous Review Reorder Point/Reorder Qty Replenishment Policy

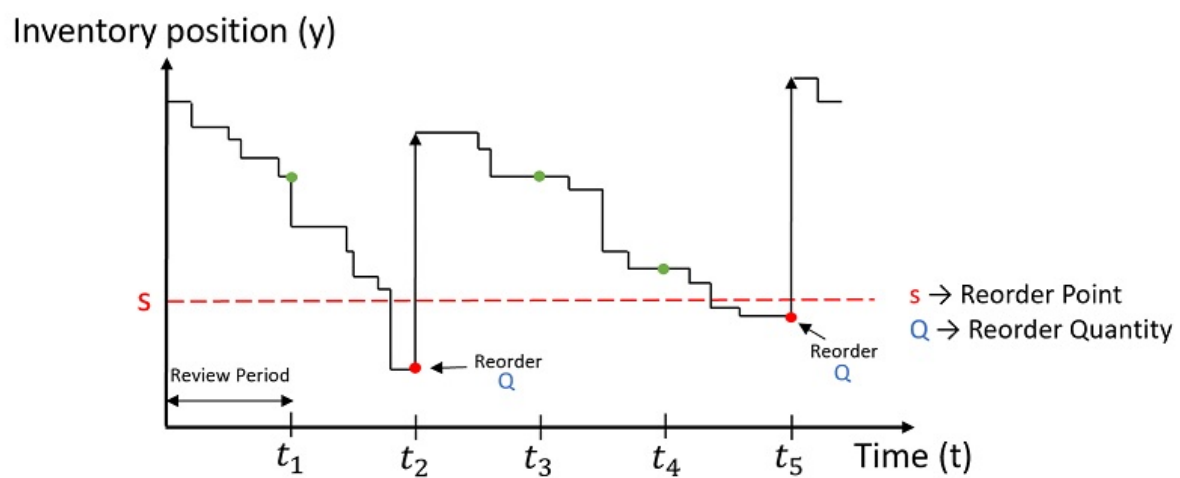


Figure 6 - Periodic Review Reorder Point/Reorder Qty Replenishment Policy

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Inventory and Material Backorders

Allow Backorder Policy - Balking or Reneging

When an entity attempts to consume material at a [Consume](#) step, the material consumption request is first inserted into the [Material](#) or [Inventory](#) element's allocation queue and then a late priority event scheduled on the simulation's current events calendar to try allocating material to waiting requests in queue rank order. If a new material consumption request arrival is still waiting in the allocation queue once that triggered allocation pass has been completed, then the *Allow Backorder Policy* property is checked to determine whether the request can be backordered. If it can, then the inventory's quantity backordered is incremented and the request remains in the allocation queue. Otherwise, the material consumption request is removed from the allocation queue and treated as a balked request.

A process token created to execute an *On Balked At Backorder Process* will have the following characteristics:

- Its associated object reference will be set to the object that was attempting to consume the material.
- It will be initialized with the same table row references as the token that executed the [Consume](#) step.
- Its material order detail reference will provide the detail of the balked material consumption request.

The *Backorder Max Wait Time* property will be evaluated when a material consumption request is backordered, and a late priority event scheduled on the simulation's future events calendar to renege the backorder at time 'Run.TimeNow + Backorder Max Wait Time' if the request has not been completed beforehand.

A backordered material consumption request can also be reneged by using a [Remove](#) step to manually remove an entity from a [Material](#) or [Inventory](#) element's allocation queue.

A process token created to execute an *On Reneged Backorder Process* will have the following characteristics:

- Its associated object reference will be set to the object that was attempting to consume the material.
- It will be initialized with the same table row references as the token that executed the [Consume](#) step.
- Its material order detail reference will provide the detail of the reneged material consumption request.

Note that if a material consumption request balks or reneges, then the original token executing the [Consume](#) step will be automatically destroyed (end processing) if all material consumption request requirements to continue execution of that token have been cancelled.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Measuring Service Level

Measuring Service Level

Measuring Service Level - Fill Rate

Fill rate is the fraction of demand that is immediately satisfied from inventory on hand (no balking or backordering). It is one of the most frequently used service measures in practice to evaluate an inventory's ability to meet demand. For example, a target service level might be a 95% fill rate.

In Simio, two types of fill rate statistics for each inventory will be automatically collected and reported:

Quantity Fill Rate: The percentage of the total quantity demanded that was immediately satisfied from inventory on hand. Sometimes referred to as *unit fill rate* or *case fill rate* in industry.

Order Fill Rate: The percentage of material consumption requests (orders) whose full quantity demanded was immediately satisfied from inventory on hand.

Table 1 shows an example of calculating quantity fill rate versus order fill rate.

Order	Quantity Demanded	Quantity Immediately Consumed	Order Fill Rate Count
1	10	10	1
2	50	40	0
3	30	0	0
4	60	60	1
Total	150	110	2
Quantity Fill Rate = $110/150 = 73.33\%$			
Order Fill Rate = $2/4 = 50\%$			

Table 1: Calculating Quantity Fill Rate vs. Order Fill Rate

Measuring Service Level – Stockout Frequency

A stockout is when there is no inventory on hand to fulfill demand. Besides fill rate, another frequently used service measure is a desired probability of no stockout occurrences during a defined risk period, a criterion sometimes referred to as the *ready rate*. For example, a target service level might be a 95% ready rate.

In Simio, time-dependent statistics on the stockout frequency for each inventory will be automatically collected and reported.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Material

Material

A Material element may be used to define a material that is consumed or produced in the system. For a material representing an end product, the material list may be entered that specifies the component materials required to make one unit of the parent material.

Separate storage locations for holding stock of a specific material may be defined using [Inventory](#) elements. Refer to the [Consume](#) and [Produce](#) steps to consume or produce material quantities.

Material may also be reserved and unreserved by using the [Reserve](#) and [UnReserve](#) steps.

Only the first waiting token may consume material. SmallestValueFirst and MaxValue rankings are based on the priority specified on the Consume step. Material is hierarchical and may hold a bill of material that is required to produce the top level material. Material may be consumed/produced at the top level, or the expanded bill of material. Materials may be consumed or produced through the Workstation (Deprecated) object within the Standard Library also, as well as through the Task Sequences within Server, Combiner and Separator objects.

See the [Inventory](#) element description and [Inventory Replenishment Policies](#) page for more information regarding the Review Period and Replenishment Policy properties.

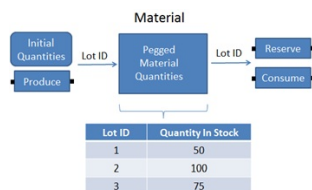
The Material element's *Assume Infinite Availability If* expression property will be evaluated in the context of the entity attempting to consume the material.

The *Allow Partial Allocation If* property may be specified as a conditional expression that is evaluated in the context of the entity requesting the material. If the specified expression contains a data table reference, then that table value may be a row reference that has been set for either the Material element, the process token that is executing the Consume step, or the entity (in that search priority order).

The Material functions, QuantityConsumed and QuantityProduced, are statistics functions that return the total material quantity consumed or produced during the run. These values are cleared at the end of a warm-up period. If the *Location Based Inventory* property of a Material element is set to 'True', then its QuantityInStock and QuantityOnOrder state values, QuantityBackordered function value, etc. are the aggregate quantities across all inventory sites.

Material Pegging

Pegging links material demand to material supply. Simio will allow material quantities in stock to be assigned to lot identifiers, with those lot identifiers then referenceable by material reservation and consumption requests (see below figure).



Pegging relationships between material demand and material supply such as shown above can provide more detailed logging of raw material usage as well as prevent material inventory from being allocated to unlinked demand.

As an example, suppose a specific production order requires 100 units of raw material A. That material quantity can be linked (pegged) to the order using a unique lot number. Entities associated with other production orders will then be unable to reserve or consume from that specific lot of material.

The *Lot ID* property may be specified as a string literal enclosed in double quotes (e.g., "Lot1"), the name of a string state variable or string column in a table, or any other expression that returns a string value.

Refer to the [Inventory Replenishment Policies](#), [Inventory and Material Backorders](#) and [Measuring Service Level](#) sections in help for additional details.

For examples of using the Material element, please refer to the SimBits [ScheduledMaterialArrivals](#) and [InventoryAndMaterials](#).

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **Material**:

Property	Description
Location Based Inventory	Indicates whether the material is stored at multiple physical locations in the system. If this property is set to 'True', then Inventory elements must be added to the model to define the separate inventory locations. Otherwise, if set to 'False', then the material is treated as a single global inventory (no physical location).
Initial Quantity	The quantity of the material present in the system at the beginning of the simulation run.
Initial Quantities (More)	Additional initial quantities of the material.
Initial Quantities (More).Initial Quantity	The initial quantity of the material present in the system.
Initial Quantities (More).Lot ID	Optional string value indicating the lot identifier of the initial material.
Review Period	The frequency of inventory review to determine whether a replenishment order is required. If the review period is 'Continuous', then an inventory review is triggered once at the start of the simulation and then whenever the inventory's position decreases. If the review period is 'Timer', then an inventory review is triggered whenever the specified timer fires its event.
Review Timer Name	The name of the timer used to trigger the inventory reviews.
Replenishment Policy	The replenishment policy checked at the time of an inventory review to determine whether a replenishment order is required and, if so, the recommended order size.
Reorder Condition	The condition that, if true, signals the need to place a replenishment order.
Reorder Quantity	The fixed reorder quantity when placing a replenishment order.
Red Zone Size	The expression used to calculate and update the buffer red zone size. The red zone is the safety embedded in the buffer.
Yellow Zone Size	The expression used to calculate and update the buffer yellow zone size. The top of the

	yellow zone is the minimum threshold for the net flow position that signals the need to place a replenishment order.
Green Zone Size	The expression used to calculate and update the buffer green zone size. The top of the green zone is the target level to return the net flow position to if at or below the top of the yellow zone.
Qualified Spike Demand	The expression used to calculate and update the qualified spike demand value for calculating the net flow position.
Reorder Point	The minimum threshold for the inventory position that signals the need to place a replenishment order.
Order-Up-To-Level	The target level to return the inventory position to if at or below the reorder point.
Log Inventory Reviews	Indicates whether inventory review events are to be automatically logged. Go to Results -> Logs -> Inventory Review Log to view the logged data.
On Replenishment Order Process	The name of the process that will be executed to handle an inventory replenishment order. Note that whenever this process is triggered by the replenishment policy, the inventory's quantity on order will have been automatically increased by the recommended order size. NOTE: The created token's material order detail reference will provide the detail of the replenishment request. For example, the function Token.MaterialOrderDetail.Quantity may be used to get the recommended order size.
Volume Per Unit	The volume per unit of the material.
Weight Per Unit	The weight per unit of the material.
Capacity Usage Per Unit	The capacity usage per unit of the material.
Display Color	The color used for the material in animation.
Bill Of Materials	The bill of materials for the material.
Bill Of Materials.Component Material Name	The name of the component material.
Bill Of Materials.Component Quantity	The quantity of the component material that is required to make one unit of the parent material.
Cost Per Unit	The cost per unit of material that is charged if a quantity of the material is consumed.
Allocation Ranking Rule	The ranking rule used to allocate the material to competing material consumption requests.
Allocation Ranking Expression	The expression used to rank the queue holding the member tokens waiting for material.
Assume Infinite Availability If	Optional condition that is evaluated whenever an entity is attempting to consume the material. If the condition is true, then the entity will be allowed to immediately consume its full required quantity of the material regardless of the actual quantity in stock. Note that this may result in negative inventory levels.
Allow Partial Allocation If	Optional condition that is evaluated whenever an entity is attempting to consume the material. If the condition is 'True', then the entity will be allowed to immediately consume only a portion of its requested quantity if the full quantity is not available. Leaving this property blank (no condition) is equivalent to entering 'False'.
Allow Backorder Policy	Indicates whether an entity attempting to consume the material can be backordered if its full requested quantity is not immediately satisfied from inventory on hand. If the backordering of an entity is rejected, then its outstanding requested quantity will be automatically cancelled and the balked demand observation recorded in the statistics.
Balk Condition or Probability	The balk at backorder condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
On Balked At Backorder Process	The name of the process that will be executed to handle an entity that has balked at being backordered. The created token's material order detail reference will provide the detail of the balked material consumption request.
Backorder Max Wait Time	An optional limit on the total time that a backordered entity will wait for its full requested quantity of the material. If the max wait time for a backordered entity is reached, then its outstanding requested quantity will be automatically cancelled and the renege demand observation recorded in the statistics.
On Reneged Backorder Process	The name of the process that will be executed to handle an entity whose backorder has been cancelled. The created token's material order detail reference will provide the detail of the reneged material consumption request.
Log Material Usage	Indicates whether usage related events for this material are to be automatically logged. Go to Results > Logs to view logged data.
Report Statistics	Specifies if statistics are to be automatically reported for this element.
Listed below are the states of Material :	
State	Description
QuantityInStock	Gets/sets the quantity of the material that is currently present in the system.
QuantityOnOrder	The current quantity of material that has been ordered to replenish the inventory but has not yet been received. This state variable may be assigned using an Assign step.
AllocationQueue	The queue of requests waiting to be allocated quantities of this material.
Listed below are the functions of Material :	
Function	Description

Inventory (site)	Returns a reference to the inventory holding this material at the specified site (a fixed object). If there is no such inventory defined, then the Nothing keyword is returned.
QuantityConsumed	Returns the total quantity of the material consumed during the simulation run.
QuantityProduced	Returns the total quantity of the material produced during the simulation run.
QuantityBackordered	Returns the current quantity of material required to satisfy all material consumption requests waiting in allocation queues.
QuantityReserved	Returns the current quantity of the material that is reserved.
QuantityReservedTo(owner)	Returns the current quantity of the material that is reserved for use by a specified owner object.
AverageQuantityInStock	Returns the average quantity of the material present in the system during the run.
MinimumQuantityInStock	Returns the minimum quantity of the material present in the system during the run.
MaximumQuantityInStock	Returns the maximum quantity of the material present in the system during the run.
AverageQuantityBackordered	Returns the material's average quantity backordered during the simulation run.
MinimumQuantityBackordered	Returns the material's minimum quantity backordered during the simulation run.
MaximumQuantityBackordered	Returns the material's maximum quantity backordered during the simulation run.
Pegged.QuantityInStock(lotID*)	Returns the current quantity of the material of a specified lot identifier that is present in the system.
Pegged.QuantityReserved(lotID*)	Returns the current quantity of the material of a specified lot identifier that is reserved.
Pegged.QuantityReservedTo(owner, lotID*)	Returns the current quantity of the material of the material of a specified lot identifier that is reserved for use by a specified owner object.
Demand	Provides functions for accessing additional statistics on the material's demand.
Demand.NumberOrders	Returns the total number of material consumption requests (i.e., orders) received during the simulation run.
Demand.QuantityDemanded	Returns the total quantity that was demanded to satisfy the material consumption requests received during the simulation run.
Demand.NumberBalked	Returns the total number of material consumption requests that balked at being backordered in an allocation queue.
Demand.QuantityBalked	Returns the total demand quantity cancelled due to balked material consumption requests.
Demand.NumberReneged	Returns the total number of material consumption requests that abandoned waiting as a backorder in an allocation queue.
Demand.QuantityReneged	Returns the total demand quantity cancelled due to reneged material consumption requests.
Demand.OrderFillRate	Returns the percentage of material consumption requests (i.e., orders) whose full quantity demanded was immediately satisfied from inventory on hand (no balking or backordering).
Demand.QuantityFillRate	Returns the percentage of the total quantity demanded that was immediately satisfied from inventory on hand (no balking or backordering).
Stockout	Provides functions for accessing time-dependent statistics on the material's stockout frequency.
Stockout.TotalTime	Returns the total time that no material was present in the system.
Stockout.PercentTime	Returns the percent time that no material was present in the system.
Stockout.NumberOccurrences	Returns the number of occurrences where no material was present in the system.
Stockout.AverageTime	Returns the average time of the occurrences where no material was present in the system.
UsageCostCharged	Returns the total cost that has been charged to consumers of the material.

* Specifying the lotID argument as an empty string will return the current quantity of unassigned (not pegged) material that is present in the system or is reserved.

Example 1 – Material/Inventory Allocation Queue – Ranking By Demand Priority

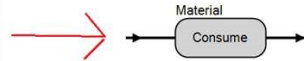
A supply chain model is using a token-only based approach to model demand for inventory (no entities). The demand queue ranking rule is not FIFO but instead needs to be a priority-based queue.

You can now use the modeling technique illustrated below, whereby the Material or Inventory element's *Ranking Expression* property is a token-based table row reference.

Example Workflow #1

Demand				
	Material Name	Site Name	Quantity	Priority
1	Material1	Site1	10	2
2	Material1	Site1	12	1
3	Material1	Site1	8	1
4	Material1	Site1	14	2
5	Material1	Site1	14	1
6	Material1	Site1	16	2

Each token executing the Consume step has a reference to a specific row in the Demand table (No entities are being used to model the demand for inventory)



Properties: Inventory1 (Inventory Element)

Basic Logic	
Material Name	Material1
Site Object Name	Site1
Initial Quantity	0.0
Review Period	None
Advanced Options	
Allocation Ranking Rule	SmallestValueFirst
Allocation Ranking Expression	Demand.Priority
Assume Infinite Availability If	
Allow Partial Allocation If	
Allow Backorder Policy	Always

The static ranking rule for allocating Material1 at Site1 to waiting demand is smallest Demand.Priority first.

A Material element may be used to define a material that is consumed, produced, or stored in the system. For a material representing an end product, the bill of material list may be specified to define the component materials required to make one unit of the parent material. Material storage can be defined using either Inventory elements or StorageBin elements.

Storing Material with Inventory Elements

When Material is stored using Inventory elements, refer to the Consume and Produce steps to consume or produce material quantities. Material stored by Inventory elements may also be reserved and unreserved by using the Reserve and UnReserve steps.

Only the first waiting token may consume material. SmallestValueFirst and MaxValue rankings are based on the priority specified on the Consume step. Material is hierarchical and may hold a bill of material that is required to produce the top level material. Material may be consumed/produced at the top level, or the expanded bill of material. Materials may be consumed or produced through the Workstation (Deprecated) object within the Standard Library also, as well as through the Task Sequences within Server, Combiner and Separator objects.

See the Inventory element description and Inventory Replenishment Policies page for more information regarding the Review Period and Replenishment Policy properties.

The Material element's *Assume Infinite Availability If* expression property will be evaluated in the context of the entity attempting to consume the material.

The *Allow Partial Allocation If* property may be specified as a conditional expression that is evaluated in the context of the entity requesting the material. If the specified expression contains a data table reference, then that table value may be a row reference that has been set for either the Material element, the process token that is executing the Consume step, or the entity (in that search priority order).

The Material functions, QuantityConsumed and QuantityProduced, are statistics functions that return the total material quantity consumed or produced during the run. These values are cleared at the end of a warm-up period. If the Location Based Inventory property of a Material element is set to 'True', then its QuantityInStock and QuantityOnOrder state values, QuantityBackordered function value, etc. are the aggregate quantities across all inventory sites.

Storing Material with StorageBin Elements

A storage bin represents a capacity-constrained storage location for holding stock of one or more materials, such as a storage location in a warehouse. The Material's *Volume Per Unit*, *Weight Per Unit*, and *Capacity Usage Per Unit* constrain the amount of material than can be stored in a single storage bin. Refer to the AddStock and RemoveStock steps to add or remove material. Space in storage bins can be reserved with the ReserveBins step, and material stored by storage bins can be reserved with the ReserveStock step.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Material Storage - Discussion and Examples

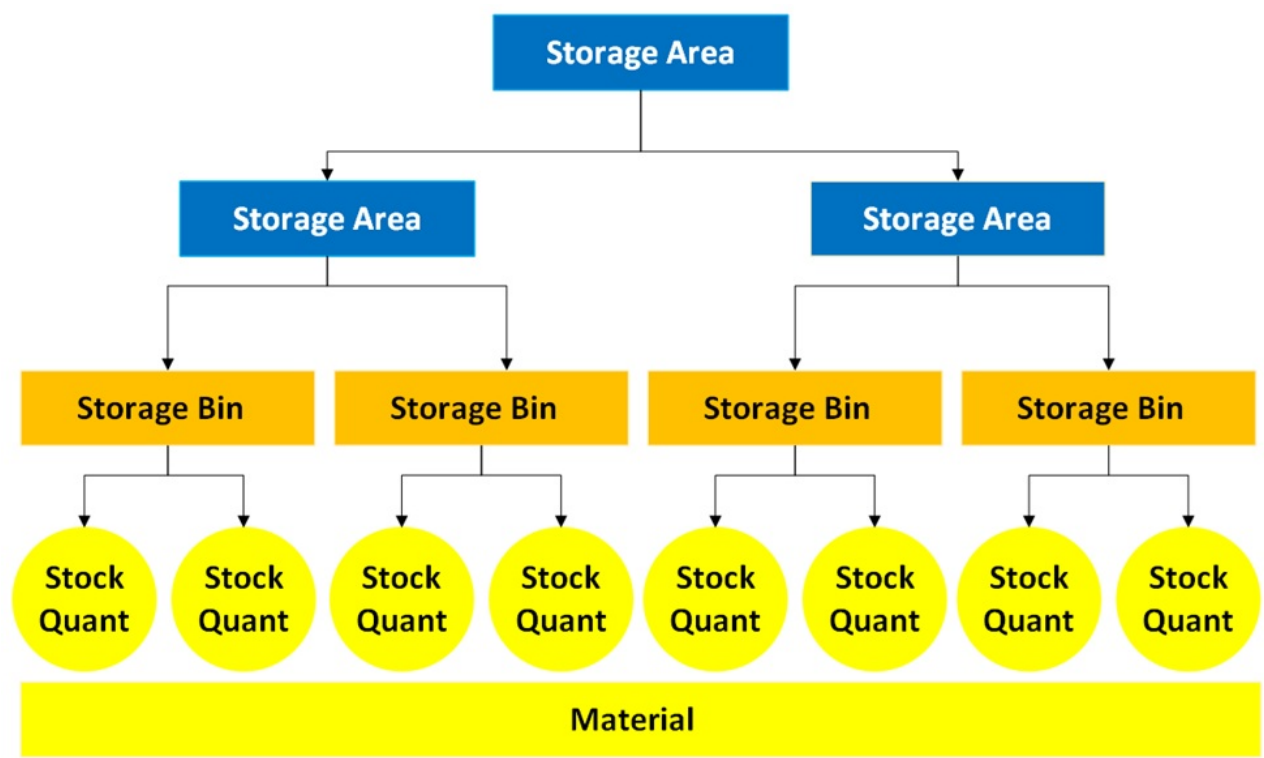
Material Storage – Discussion and Examples

Material storage features may be used to model the material storage structures and stock putaway or removal processes typically found in warehouse facilities. It may be used separately or in conjunction with other existing Simio features related to inventory control and replenishment. The core material storage components are summarized below.

Feature	Description
Material Element	Defines a material type that can be stored in a storage bin.
Storage Area Element	Defines a logical grouping of other storage areas or storage bins for modeling material stock putaway or removal processes.
Storage Bin Element	Defines a capacity-constrained storage location for holding stock of one or more materials.
Stock Quant	Represents stock of a specific material with the same characteristics in a storage bin.
Stock Reservation	Represents a material stock putaway or removal reservation.
ReserveBins Step	May be used to reserve storage bin capacity for an entity's material stock putaway requirements.
AddStock Step	May be used to add material stock to a storage bin.
ReserveStock Step	May be used to reserve material stock in a storage bin for an entity's material stock removal requirements.
RemoveStock Step	May be used to add material stock to a storage bin.

Defining Storage Structures

Material storage structures may be implemented using storage areas, storage bins, and stock quants as shown below.



Each material type that may be physically stored is defined using a Material element. Each Material element instance is therefore analogous to a SKU or other unique part identifier.

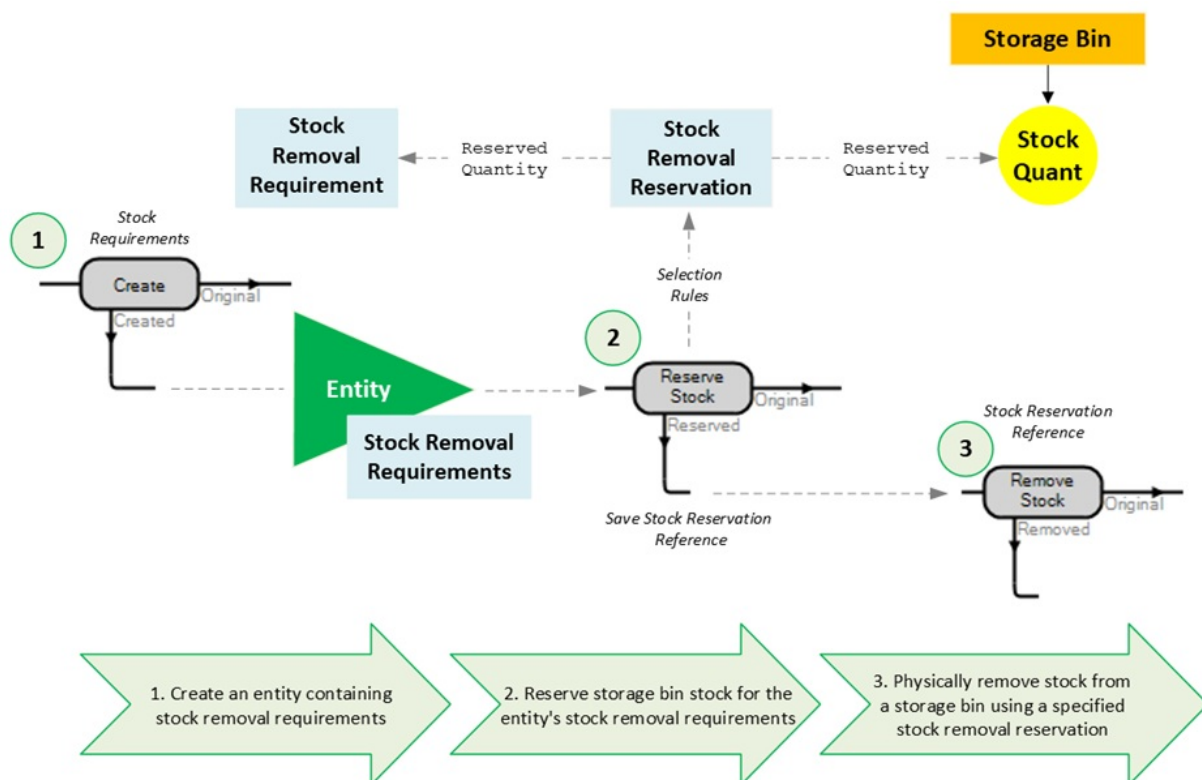
Capacity-constrained storage locations for holding material stock are defined using Storage Bin elements. Depending on the use case, storage bins might correspond to a physical bin, a shelf, a rack, or any other location where material is stored.

Logical groupings of storage locations are implemented using Storage Area elements. The definition of storage areas is a key aspect of modeling material storage structures. Storage areas can simultaneously contain storage bins and storage areas, and a storage structure can have an arbitrary number of layers. A storage bin can also be a member of an arbitrary number of storage areas. Therefore, a storage area could contain every storage location in a warehouse, all storage locations on a single rack or shelf, or each storage location where a specific material type can be stored.

A single storage location can be contained within multiple storage areas. For instance, a storage bin representing a rack could simultaneously be a member of a "Zone1" storage area and a "Aisle5" storage area. However, a single storage location cannot be added to the same storage area multiple times. Additionally, circular references are not allowed. For instance, if "StorageArea1" is defined as a member of "StorageArea2" and "StorageArea2" is simultaneously defined as a member of "StorageArea1".

Stock Removal

The general approach used to model a material stock removal process is illustrated in the figure below.



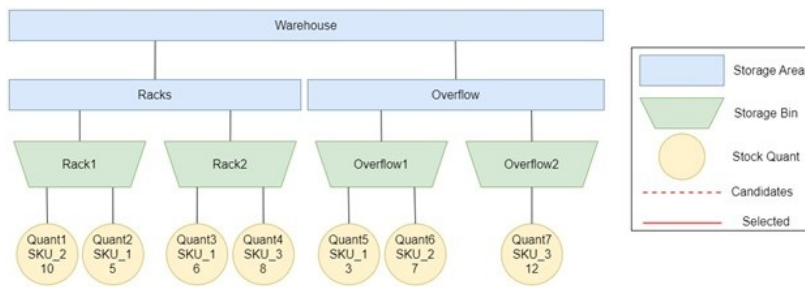
First, an entity is created containing one or more stock requirements. The stock requirements can only be applied to the entity when it is created, such as through the Source's or Create step's Stock Requirements repeat group.

A ReserveStock step is then executed to reserve storage bin stock for the entity's stock requirements. The ReserveStock step's Selection Rules repeat group can be used to determine which stock is reserved. Once stock is reserved, it cannot be reserved for any other requirement. Each time stock is reserved, a new stock removal reservation is created, and a reference to the reservation will be saved to the state variable specified by the ReserveStock step's Save Stock Reservation Reference property.

Finally, a RemoveStock step is used to remove the reserved stock from a storage bin using a specified stock removal reservation. Once the reserved stock is removed, the corresponding stock reservation is destroyed. This last step is typically executed once simulation logic has completed any prerequisite material handling tasks.

Stock Removal – Selection Rule Example

The following figure depicts an example storage structure with four storage bins: Rack1, Rack2, Overflow1, and Overflow2. Three Material elements are also defined: SKU_1, SKU_2, and SKU_3. The initial QuantityInStock of each stock quant is shown. The QuantityReserved for each quant is zero.



The system's material retrieval process first specifies that material should be retrieved from a rack before an overflow bin, then specifies that material should be retrieved from a storage bin that can fulfill the entirety of a requirement if possible. These rules could be implemented in ReserveStock step's repeat group as shown below.

Properties:	
Selection Rule	
Starting Storage Area Name	Warehouse
Rule Type	Area
Sub Rule Type	Sort
Sort Value Expression	Candidate.StorageArea == Racks
Sort Order	Descending
Advanced Options	
Skip Selection Rule If	
Stock Requirement Condition	
Parent Storage Area Condition	

Rule 1 – Prefer Racks area over the Overflow Area. This rule sorts areas in descending order by the expression "Candidate.StorageArea == Racks", which returns '1' if the candidate storage area is the Racks area, and '0' otherwise.

Properties:	
Selection Rule	
Starting Storage Area Name	Warehouse
Rule Type	Bin
Sub Rule Type	Sort
Sort Value Expression	Candidate.StorageBin.CurrentStock.QuantityAvailable(Stock.Requirement.Material) >= Stock.Requirement.UnreservedQuantity
Sort Order	Descending
Advanced Options	
Skip Selection Rule If	
Stock Requirement Condition	
Parent Storage Area Condition	

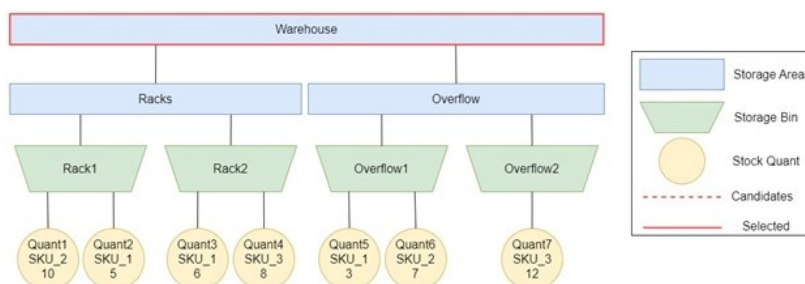
Rule 2 – Prefer storage bins where the available quantity of the stock requirement's material is at least equal to the unreserved quantity of the stock requirement. This rule sorts storage bins in descending order by the expression "Candidate.StorageBin.CurrentStock.QuantityAvailable(Stock.Requirement.Material) >= Stock.Requirement.UnreservedQuantity", which returns '1' if the quantity available of the material which the step is currently attempting to reserve is greater than or equal to the quantity of material which the step is currently attempting to reserve.

Properties:	
Selection Rule	
Starting Storage Area Name	Warehouse
Rule Type	Quant
Sub Rule Type	Condition
Selection Condition	
Advanced Options	
Skip Selection Rule If	
Stock Requirement Condition	
Parent Storage Area Condition	

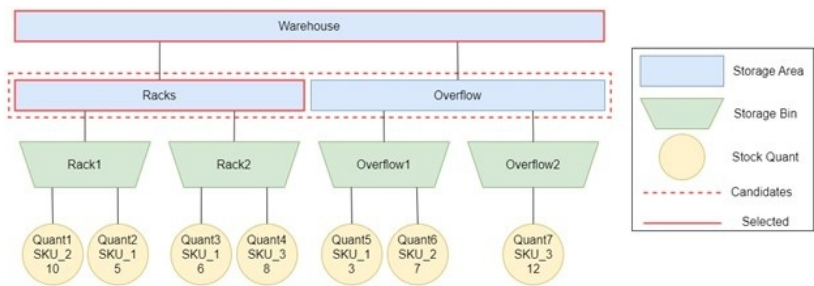
Rule 3 – Empty rule with 'Quant' Rule Type. A quant rule is always required when reserving stock, since the ReserveStock step must always search through quant to find eligible stock. This blank rule will cause the step to select the first eligible quant, i.e. the first quant of the same material. Adding a valid Condition or Sort would cause the step to filter or sort the quant candidates, affecting the final selected quant.

Given an entity arrives to the system with a stock requirement for 6 of material SKU_1, the following logic will be applied to select what material is reserved for removal.

The ReserveStock step will find the first group of selection rules using the Starting Storage Area Name of each repeat group row. In this case, there is only one group since each rule has the same starting storage area, Warehouse. The step will then begin to evaluate the selection rules, starting from the Warehouse storage area, as shown below.

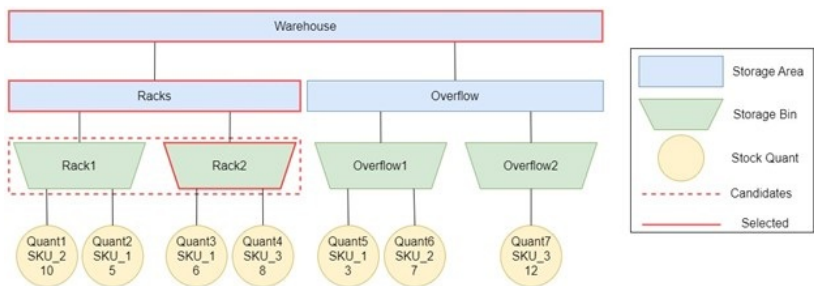


The first rule the ReserveStock step will evaluate is the area rule. The step will look at the candidate storage areas, Racks and Overflow, and sort them based on the specified Sort Value Expression. Since the rule expression returns '1' for the Racks storage area and '0' for the Overflow area and the Sort Order is 'Descending', the Racks area will be selected.

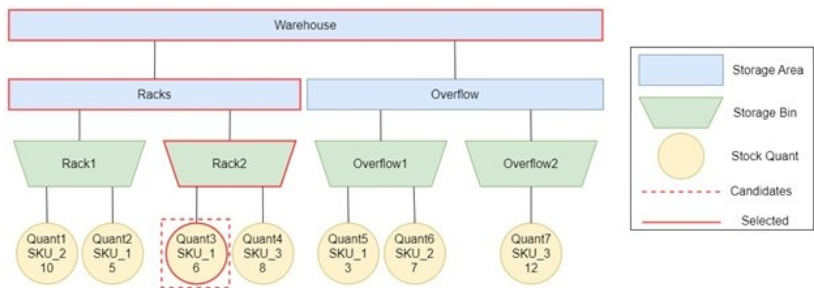


The ReserveStock step then re-resolves the rule with the Racks area as the parent area. At this stage, the Parent Storage Area condition could cause a rule to no longer be applicable to this case. However, because all rules are applicable, the step will continue evaluating rules, again starting with the first rule. Since there are no more areas within the Racks area, no other area can be selected. The step then moves to the next rule.

The second rule is the bin rule. Because the Racks area has already been selected, the step will look at the Rack area's child bins, Rack1 and Rack2, and sort them based on the specified Sort Value Expression. Since the expression will return '1' if the storage bin has at least 6 available units of SKU_1 and '0' otherwise, Rack2 will get sorted before Rack1, since Rack1 only has 5 of SKU_1. Therefore, Rack2 will be selected.

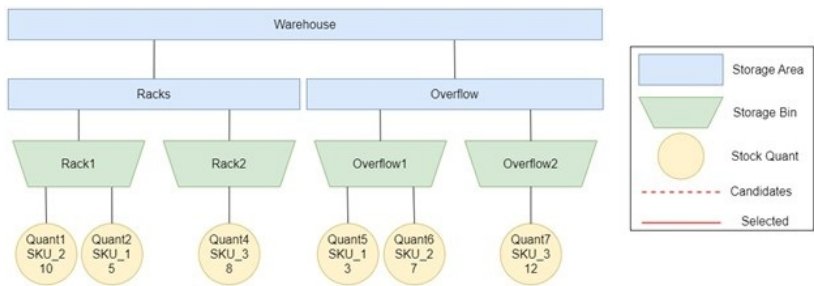


Once Rack2 is selected, the ReserveStock step will look for eligible quants, since there is a quant rule present in the group of rules. Because there is no specified Selection Condition or Sort Value Expression, the step will simply find the first eligible quant. In this case, the only eligible quant is Quant3 since it is the only quant in Rack2 containing stock of SKU_1.



Once a quant with the required material is selected, the ReserveStock step will increase the QuantityReserved of the selected quant by the minimum of the quant's QuantityAvailable and the stock requirement's unreserved quantity to ensure that no stock is reserved for multiple stock requirements. In this case, both the quant's QuantityReserved and QuantityAvailable are equal to six, so a new stock reservation will be created for six units of SKU_1 from Quant3.

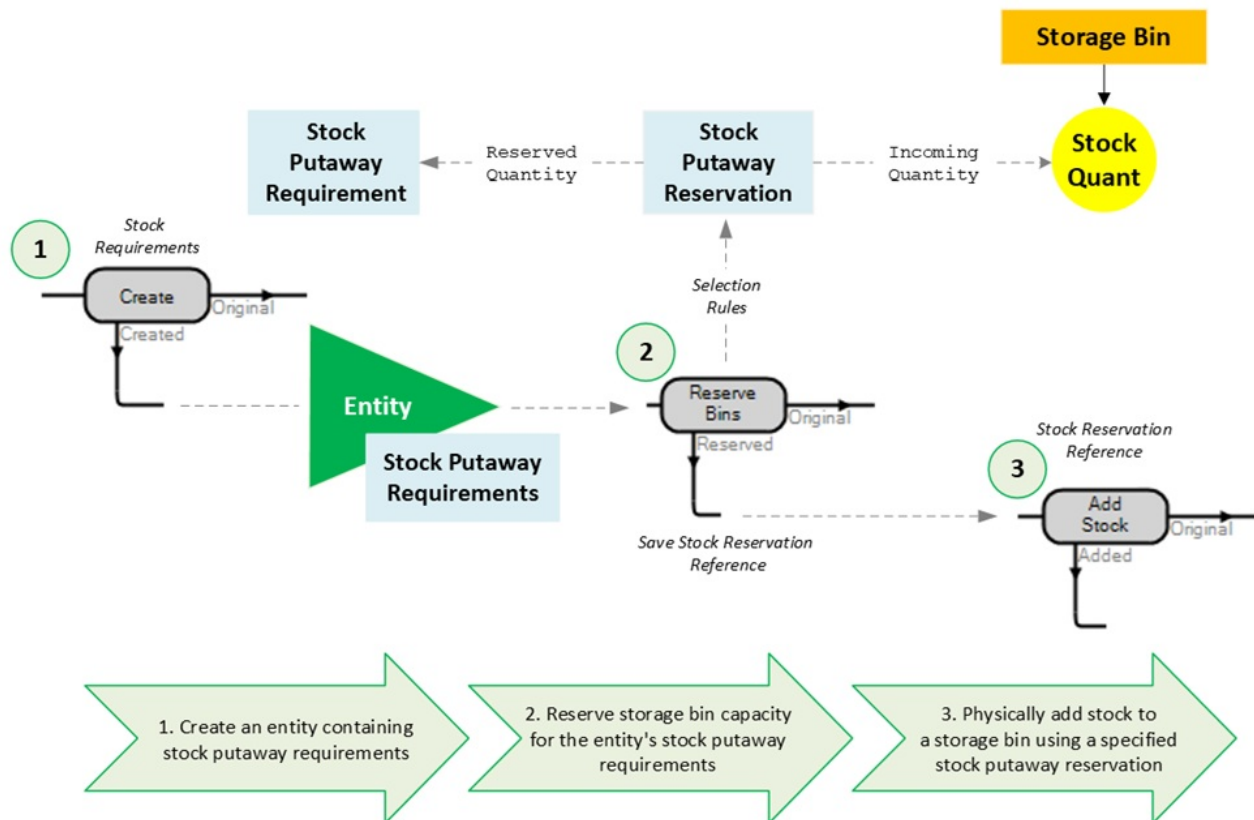
Once the stock is removed from Quant3 using a RemoveStock step and the stock reservation, the final storage structure will be updated as shown below.



All units of SKU_1 from Rack2 have been removed, and because the QuantityInStock and IncomingQuantity of Quant3 were both zero, the quant was automatically removed from Rack2.

Stock Putaway

The general approach used to model a material stock putaway process is illustrated in the figure below.



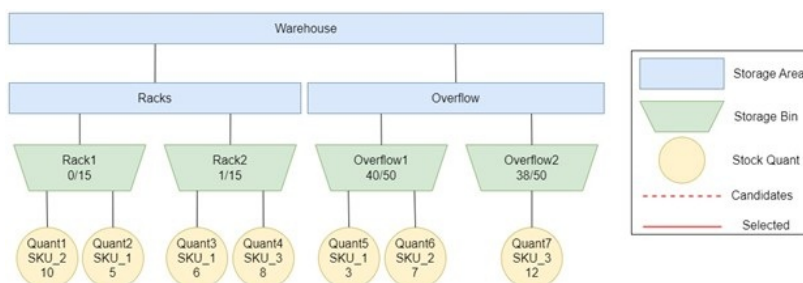
First, an entity is created containing one or more stock requirements. The stock requirements can only be applied to the entity when it is created, such as through the Source's or Create step's Stock Requirements repeat group.

A ReserveBins step is then executed to reserve storage bin capacity for the entity's stock requirements. The ReserveBins step's Selection Rules repeat group can be used to determine which storage bin's capacity is reserved. Once storage bin capacity is reserved, no other material can be placed in the storage bin unless the storage bin has remaining unreserved capacity. Each time storage bin capacity is reserved, a new stock putaway reservation is created, and a reference to the reservation will be saved to the state variable specified by the ReserveBins step's Save Stock Reservation Reference property.

Finally, an AddStock step is used to physically add stock to a storage bin using a specified stock putaway reservation. This last step is typically executed once simulation logic has completed any prerequisite material handling tasks.

Stock Putaway – Selection Rule Example

The following figure depicts an example storage structure with four storage bins: Rack1, Rack2, Overflow1, and Overflow2. Three Material elements are also defined: SKU_1, SKU_2, and SKU_3. The Capacity Usage Per Unit of each material type is '1'. Each storage bin's TotalCapacityAvailable out of its CurrentTotalCapacity is shown below the storage bin name. The initial QuantityInStock of each stock quant is given, and the QuantityReserved for each quant is zero.



The system's material putaway process specifies that material should first be put away in the Racks area before any remaining material is placed in the Overflow area. When placing material in the Racks area, material should be first placed in the bin with the most capacity available. When placing material in the Overflow area, material should be grouped with material of the same type if possible. These rules could be implemented in the ReserveStock step's repeat group as shown below.

Properties:	
Selection Rule	
Starting Storage Area Name	Racks
Rule Type	Bin
Sub Rule Type	Sort
Sort Value Expression	Candidate.StorageBin.CurrentTotalCapacityAvailable
Sort Order	Descending
Advanced Options	
Skip Selection Rule If	
Stock Requirement Condition	
Parent Storage Area Condi...	

Rule 1 – In Racks area, prefer storage bins with the greatest total capacity available.

Properties:	
Selection Rule	
Starting Storage Area Name	Racks
Rule Type	Quant
Sub Rule Type	Condition
Selection Condition	
Advanced Options	
Skip Selection Rule If	
Stock Requirement Condition	
Parent Storage Area Condi...	

Rule 2 – In Racks area, place material in first eligible quant. If no quant rule is specified, the ReserveBins step will automatically create a new quant for each reservation provided that a storage bin is selected.

Properties:	
Selection Rule	
Starting Storage Area Name	Overflow
Rule Type	Bin
Sub Rule Type	Sort
Sort Value Expression	Candidate.StorageBin.CurrentStock.QuantityInStock(Stock.Requirement.Material)
Sort Order	Descending
Advanced Options	
Skip Selection Rule If	
Stock Requirement Condition	
Parent Storage Area Condi...	

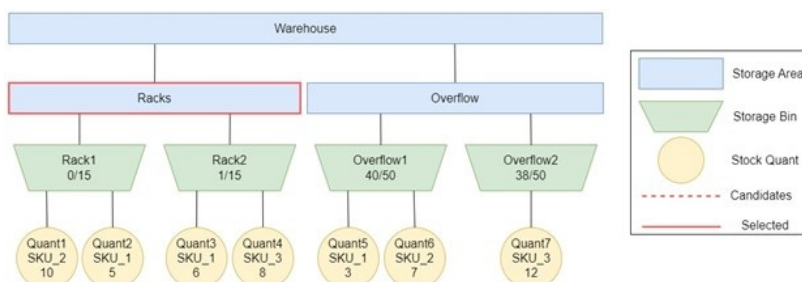
Rule 3 – In Overflow area, prefer storage bins with the greatest quantity of matching material currently in the bin.

Properties:	
Selection Rule	
Starting Storage Area Name	Overflow
Rule Type	Quant
Sub Rule Type	Condition
Selection Condition	True
Advanced Options	
Skip Selection Rule If	
Stock Requirement Condition	
Parent Storage Area Condi...	

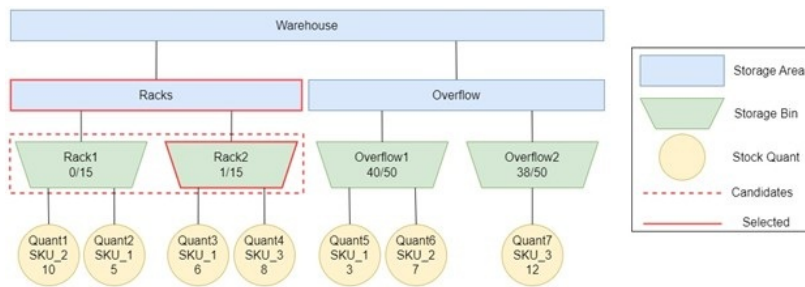
Rule 4 – In Overflow area, place material in the first eligible quant. If no quant rule is specified, the ReserveBins step will automatically create a new quant for each reservation provided that a storage bin is selected.

Given an entity arrives to the system with a stock requirement to put away 10 of SKU_3, the following logic will be applied to select what storage bin capacity is reserved.

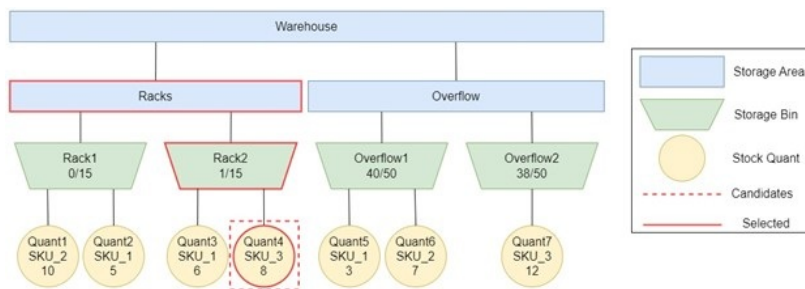
The ReserveBins step will find the first group of selection rules using the Starting Storage Area Name of each repeat group row. In this case, there are two groups of rules. The first group includes Rules 1 and 2 since Rules 1 and 2 specify Racks as the starting storage area and the second group includes Rules 3 and 4 as the starting storage area and Rules 3 and 4 specify Overflow instead. The step will then begin to evaluate the first group of selection rules, which start from the Racks area.



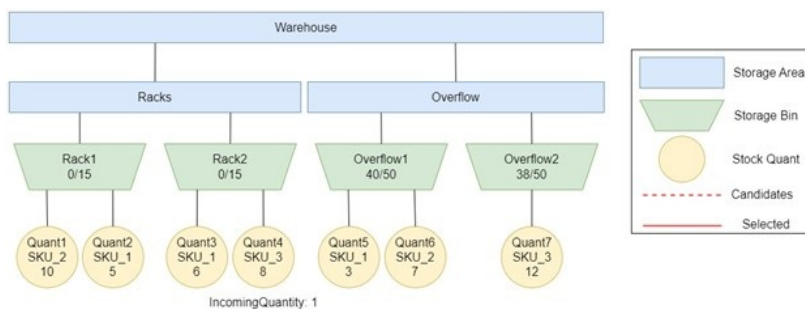
The first rule the ReserveBins step will evaluate is Rule 1, the bin rule. The step will evaluate the candidate storage bins, Rack1 and Rack2, and sort them based on the specified Sort Value Expression. Since the rule expression returns '0' for Rack1 and '1' for Rack2 and the Sort Order is 'Descending', Rack2 will be selected.



If no quant rules were specified, the ReserveBins step would simply create a new quant in Rack2 and immediately create a stock reservation. However, the inclusion of Rule 2 causes the step to find the first eligible quant to reserve for the material instead. Quant4 is the only eligible quant since it is the only quant within Rack2 of the same material as the stock reservation, SKU_3.

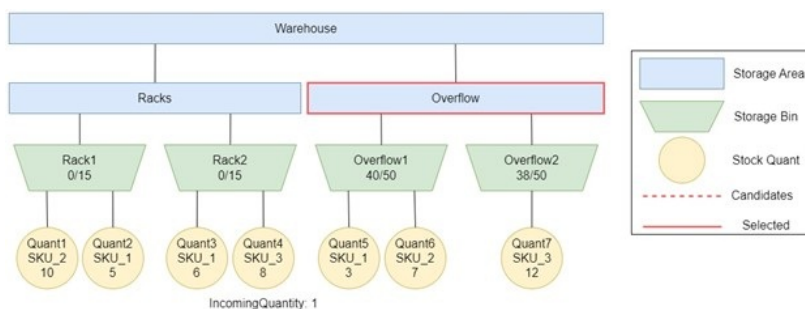


Once Quant4 is selected, the ReserveBins step will increase the IncomingQuantity of the quant by the minimum of the stock requirement's unreserved quantity and the maximum quantity of material that can fit in the storage bin. In this case, the unreserved quantity is '10' and the maximum quantity of material that can fit in the storage bin is '1', so the step will create a reservation to place one unit of SKU_3 in Quant4. Once the reservation is created, the storage structure is updated as shown below. Since one unit of SKU_3 is being added to Rack2, and because SKU_3's Capacity Usage Per Unit is '1', Rack2's TotalCapacityAvailable will decrease from '1' to '0'.

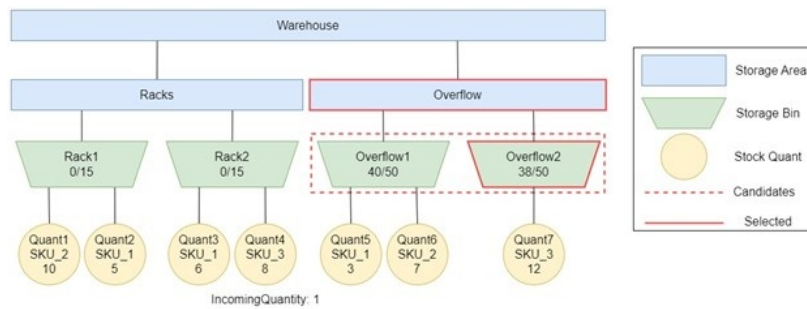


Because the unreserved quantity of the stock requirement is greater than zero, the ReserveBins step will restart its search to find storage bin capacity to reserve for the remaining quantity of SKU_3.

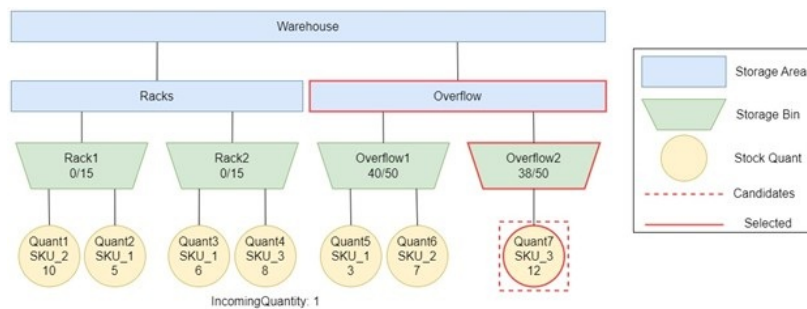
The ReserveBins step will restart the search starting from the first group of selection rules. However, because no storage bins in the Racks area have available capacity, the step will fail to reserve storage bin capacity using this rules group. The step will proceed to the second group of rules, whose starting storage area is Overflow.



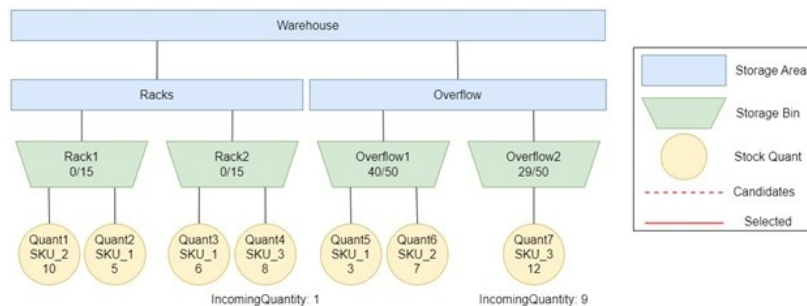
The ReserveBins step will proceed to evaluate Rule 3. The step will evaluate the candidate storage bins, Overflow1 and Overflow2, and sort them based on the specified Sort Value Expression. Since the rule expression returns '0' for Overflow1 and '12' for Overflow 2 and the Sort Order is 'Descending', Overflow2 will be selected.



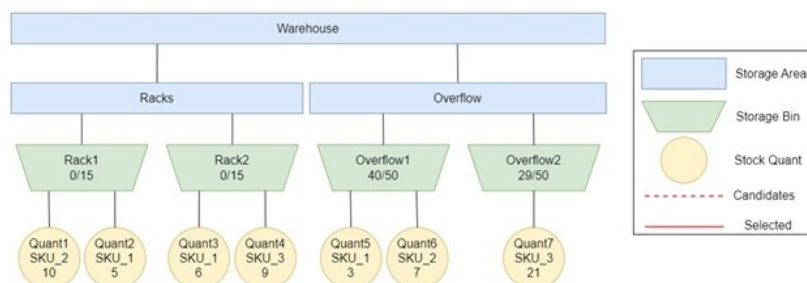
If no quant rules were specified, the ReserveBins step would simply create a new quant in Rack2 and immediately create a stock reservation. However, the inclusion of Rule 4 causes the step to find the first eligible quant to reserve for the material instead. Quant7 is the only eligible quant since it is the only quant within Overflow2 and matches the stock reservation's material, SKU_3.



Once the ReserveBins step selected Quant, it will increase the IncomingQuantity of Quant7 by the minimum of the stock requirement's unreserved quantity and the maximum quantity of the material that can fit in the storage bin given the bin's weight, volume, and total capacity constraints. In this case, the unreserved quantity of the stock requirement is '9' and the maximum quantity of the material that can fit in the storage bin is '38', so the step will create a reservation to place nine units of SKU_3 in Quant7. Once the reservation is created, the storage structure will be updated as shown below. Since nine units of SKU_3 are being added to Overflow2, and because SKU_3's Capacity Usage Per Unit is '1', Overflow2's TotalCapacityAvailable will decrease from '38' to '29'.



Once stock is added to Quant4 and Quant7 using an AddStock step and the created stock reservations, the final storage structure will be updated as shown below.



Modeling Custom Stock Attributes

Simio's material storage framework provides features that can facilitate the modeling of custom stock attributes that are pertinent to stock putaway or removal selection rules. For example, a custom stock attribute that is a position in the storage bin, a batch number, a best-before date, or an expiration date.

An example table-driven approach for modeling custom stock attributes is illustrated in the figure below. Note that the ability to define a primary key in an output table requires a Simio RPS license. More information about modeling custom stock attributes can be found in the SimBit.

Stock Requirements

Custom stock attributes can be included in a data table defining stock putaway or removal requirements.

Each stock requirement has a specified unique ID number that is the primary key for table row lookups.

Data Table

Stock Requirements			
ID Number	Material Name	Quantity	Batch Number
1	Material1	5	100
2	Material2	10	200

Primary Key Custom Stock Attribute

Selection rule expression syntax to reference the custom batch number attribute of the current stock requirement being evaluated:

`StockRequirements[StockRequirements.IDNumber.RowForKey(Stock.Requirement.IDNumber)].BatchNumber`

Current Stock Quants

Custom stock attributes can be included in an output table whose data is synchronized with the simulation's current stock quants.

Each stock quant has an automatically generated unique ID number that is the primary key for table row lookups.

Use tokens created from **ReserveBins**, **ReserveStock**, **AddStock**, and **RemoveStock** steps to execute additional process logic that keeps the stock quants table synchronized.

Output Table

Current Stock Quants							
ID Number	Storage Bin Name	Material Name	Quantity In Stock	Quantity Reserved	Quantity Available	Incoming Quantity	Batch Number
1	StorageBin1	Material1	25.0000	0.0000	0.0000	0.0000	100
2	StorageBin2	Material2	50.0000	0.0000	0.0000	0.0000	200

Primary Key

Custom Stock Attribute

Selection rule expression syntax to reference the custom batch number attribute of a candidate stock quant:

`CurrentStockQuants[CurrentStockQuants.IDNumber.RowForKey(Candidate.StockQuant.IDNumber)].BatchNumber`

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Monitor

Monitor

The Monitor element may be used to detect a discrete value change or threshold crossing of a specified state variable or group of state variables. It is added to a model from the Elements panel found within the Definitions window.

The Monitor has an Enabled state that allows the Monitor to be enabled/disabled.

A single Monitor element can monitor more than one state variable. This enables users to model a single 'Status Changed' event that pertains to some logical grouping of state variables in the system, such as a set of variables that relate to the status of an area, a processing location, etc. For example, suppose a single Monitor element named PackingAreaStatusChanged is defined that is monitoring discrete changes in the state variables Packer1.AllocationQueue, Packer1.ResourceState, Packer2.AllocationQueue, or Packer2.ResourceState. Somewhere in the modeled process logic, perhaps a Wait step is then used to hold a process token until event name PackingAreaStatusChanged occurs and event condition 'Packer1.ResourceState==0 && Packer2.ResourceState==0' is true.

When a Monitor event is fired, if there are any processes specified to be triggered by that event, then the created tokens are associated with the object that owns the Monitor element. For additional information, visit the [Monitor - Discussions and Examples](#) page.

The Monitor's *Initial Threshold Value* may be changed during the simulation run by assigning the 'MonitorName.CurrentThresholdValue' state to a new value.

For examples of using the Monitor element, please refer to the SimBits [DynamicallyCreatingVehicles](#), [BatchingProcessUsingScanOrWaitStepToControlBatchSize](#), [UsingaMonitor](#) and [WritingToAFile](#).

The State Variable Name (or multiple State Variable Names) to be monitored may be a member of an object or element reference state variable (i.e., EntityReferenceState.StateName) which allows the state variable(s) monitored by an individual Monitor element to dynamically change. The Monitor element will 're-wire' to the state variable to be monitored each time the Monitor is enabled (i.e., its Enabled state changes from False to True).

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **Monitor**:

Property	Description
State Variable Name	The state variable to be monitored.
Monitor Type	The type of state change to detect.
Crossing Direction	The direction that the monitored state value must approach the specified threshold for a crossing state change to be detected.
Initial Threshold Value	The initial threshold value whose crossing is to be detected. Note that the monitor's 'CurrentThresholdValue' state may be used to dynamically get or set the current threshold value during a simulation run.
Monitored State Variables (More)	Additional state variables to be monitored.
Monitored State Variables (More).State Variable Name	The state variable to be monitored.
Monitored State Variables (More).Monitor Type	The type of state change to detect.
Monitored State	The crossing direction.

Variables (More).Crossing
Direction

Monitored State Variables (More).Initial Threshold Value	The initial threshold value whose crossing is to be detected.
Monitored State Variables (More).Threshold State Variable Name	Optional user-assignable state variable that will hold the threshold value during a simulation run. NOTE: This state variable must be a real and of the same unit type as the monitored state variable.
Trigger Condition	Optional condition that must be true when a discrete or threshold crossing state change is detected, in order for the moitor to fire its event and trigger any process logic.
Triggered Process Name	The process to be immediately executed when a discrete or crossing state change is detected, before the execution of any other simulation logic in the system.
Initially Enabled	Specifies whether the monitor is enabled when the system is initialized.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Listed below are the States of **Monitor**:

State	Description
CurrentThresholdValue	State that may be assigned a value (in the base units per the monitored state variable's unit type) to dynamically change the monitor's threshold value.

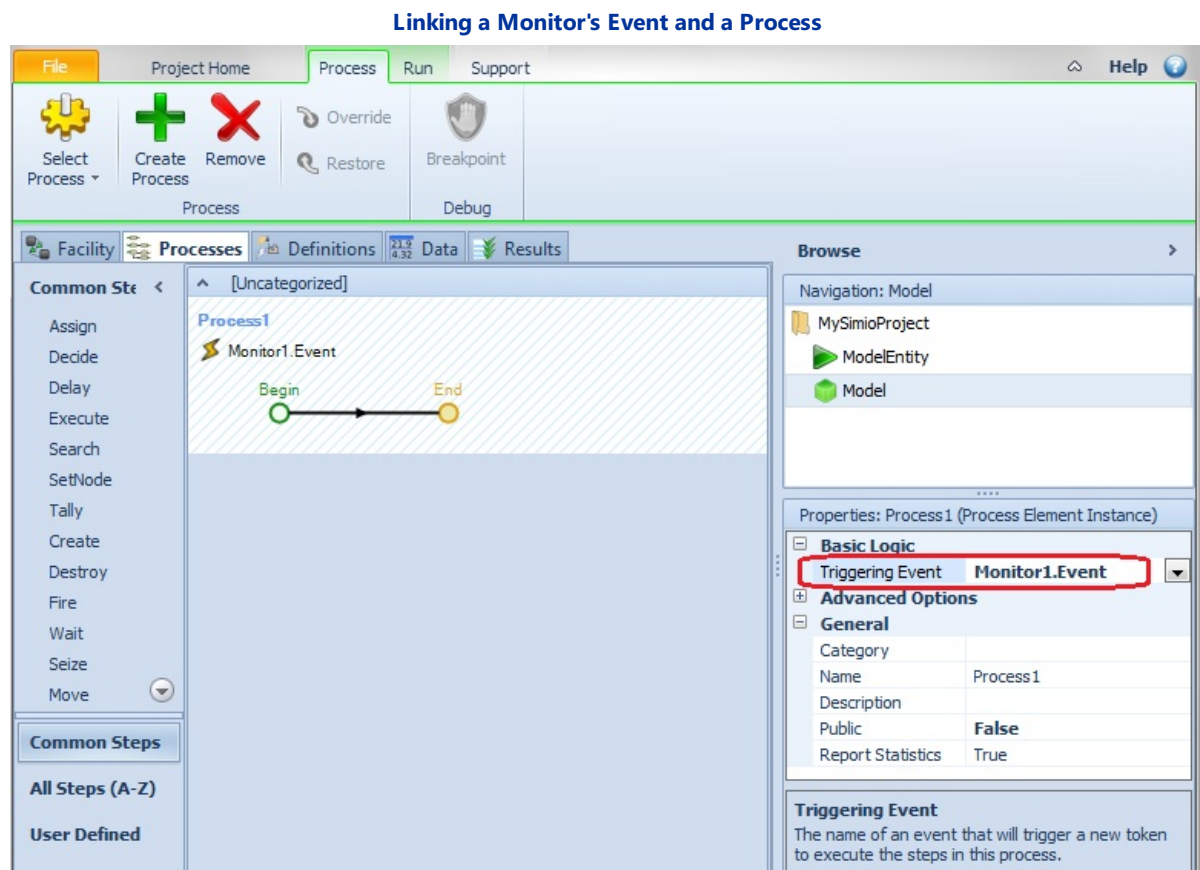
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Monitor - Discussion and Examples

Discussion

In order to have a process execute when a Monitor fires its event, the *Triggering Event* property on the Process must be set to the name of the Monitor's event. A new process can be created from the Steps tab of the Processes Window by clicking the Create Process button in the ribbon. In the Properties window of this new process, select the name of the Monitor (.Event) from the drop down of the *Triggering Event* property. This will cause this process to be executed each time the specified Monitor fires an event.



Monitor Example 1

The following is an example of a Monitor that watches a predefined Simio queue state, called `Server1.InputBuffer.Contents`, which holds the value for the number of entities currently waiting in the InputBuffer of the object called `Server1`.

- Since its *MonitorType* property is set to 'CrossingStateChange', this Monitor will fire an event when the value of `Server1.InputBuffer.Contents` crosses over the specified *Threshold Value*, which is 3.
- Because the *CrossingDirection* property is set to 'Positive', the Monitor will fire when the value crosses in the positive direction from 3 to 4, not in the negative direction from 3 to 2.
- The *On Change Detected Process* property is set to the process named 'Monitor1_OnChangeDetectedProcess' so as soon as the Monitor's event is fired, it will execute 'Monitor1_OnChangeDetectedProcess' before any other logic is executed.

Properties Window of a Monitor Element

Properties: Monitor1_QueueLength (Monitor Element Instance)

Basic Logic	
Monitor Type	CrossingStateChange
State Variable Name	Server1.InputBuffer.Contents
Initial Threshold Value	3
Crossing Direction	Positive
On Change Detected Process	
Advanced Options	
General	
Name	Monitor1_QueueLength
Description	
Public	True
Report Statistics	True

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Network

Network

A Network is a list of links to which an entity may be assigned to follow via the shortest path to its destination.

A link may be in multiple networks. The Network keeps a master list of network nodes for each node that is connected in the network. A node is considered connected if it has at least one incoming or outgoing link that is a member of the network. Each intersection has a list of network nodes - one for each network that the node is a member (by being the starting or ending node of a member link). Hence each network node is both on the master list of network nodes owned by the network, and the local list of network nodes owned by the node. Each network node keeps a list of outbound member links, and a count of inbound member links. When both counts are zero, the node is removed from both the master list and the node list. Each network node has a vector of shortest path links from this node to all nodes in the master network node list. Each vector is indexed in the same order as the master network node list owned by the network. The shortest path link is found by locating the network node for the destination transfer station in this master list, and then using the index of this node to index into the shortest path vector.

The NextLink and NextNode functions listed below will allow process logic to 'walk along' the shortest path from one node to another node, link by link and/or node by node.

When using a Shortest Paths Tie Breaker Rule of 'Turns', inbound links not on the entity's current network will not be used to consider the number of turns along the entity's path.

For more information regarding the Turnaround Method, see the [Path - Discussion and Examples](#) page.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the Properties of **Network**:

Property	Description
Links Type	The method for specifying the links that comprise the network.
Member Links	The links that comprise the network. NOTE: If this repeat group property is bound to a data table or parent repeat group, then all other user interface tools for adding, removing, or visualizing the network's member links are disabled.
Shortest Paths Graph Type	Indicates how the directions of member links are determined when calculating the network's shortest paths. LinkTypes - The direction of each member link is determined by its Type property. LinkDesiredDirections - The direction of each member link is determined by its DesiredDirection state. LinkCurrentDirections - The direction of each member link is determined by its current or planned traffic direction.
Shortest Paths Exclusion Condition	Optional condition evaluated for each member link whenever calculating the network's shortest paths. If true will exclude the link from the calculations. In the expression, use the syntax [LinkClass].[Attribute] to reference an attribute of each member link (e.g., Path.Contents).
Shortest Paths Weight Multiplier	Expression evaluated for each member link whenever calculating the network's shortest paths. Is a multiplier that is applied to the link's length to determine the link's weighted length (a.k.a. cost). This expression must return a non-negative number. In the expression, use the syntax [LinkClass].[Attribute] to reference an attribute of each member link (e.g., Path.SelectionWeight).
Shortest Paths Tie Breaker Rule	The rule used to break ties between shortest paths with equal weighted lengths (a.k.a. costs). None - The first path found will be given priority. FewestTurns - The path with the fewest turns at nodes will be given priority. A turn is counted at a node if travel from the inbound link to outbound link is not approximately a straight line.
Update Shortest	Optional event-driven triggers that signal an update of the network's shortest path calculations.

Paths

Update Shortest Paths Triggers	Optional event-driven triggers that signal an update of the network's shortest path calculations.
Triggering Event Name	The name of the event whose occurrence will signal an update of the network's shortest path calculations.
Condition	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to signal an update of the network's shortest path calculations.
Update Type	Indicates the type of update to the network's shortest path calculations whenever the triggering event occurs. Immediate - All shortest paths calculations are immediately updated. Deferred - Shortest paths calculations are delayed until actually required.
Turnaround Method	The method used by travelers on this network to turn around at nodes in order to move in the opposite direction on bidirectional links. If the method is 'Default', then the Network Turnaround Method specified for the entity type will be assumed.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Listed below are the States of **Network**:

State	Description
VisitRequestQueue	A queue of visit requests associated with this network.

Listed below are the Functions of **Network**:

Function	Description
Distance(fromNode,toNode)	Returns the shortest path distance between two nodes using this network. If the fromNode == toNode, the function returns 0.0.
NextLink(fromNode,toNode)	Returns a reference to the next link on the shortest path between two nodes using this network. If the fromNode == toNode, the function returns Nothing.
NextNode(fromNode,toNode)	Returns a reference to the next node on the shortest path between two nodes using this network. If the destination node is returned, then the two nodes are directly connected by a link. If the fromNode == toNode, the function returns the node.
PathExists(fromNode,toNode)	Returns 'True' if a followable travel path exists between two nodes using this network. If the fromNode == toNode, the function returns 'True'.
Links.NumberItems	Returns the number of links that are members of this network.
Links.FirstItem	Returns a reference to the first link in the collection of links that are members of this network.
Links.LastItem	Returns a reference to the last link in the collection of links that are members of this network.
Links.ItemAtIndex(index)	Returns a reference to the link at a specified index position in the collection of links that are members of this network.

Links.IndexOfItem(link)	Returns the one-based index of a specified link in the collection of links that are members of this network. If the link is not a member of this network then the value 0 is returned.
Links.Contains(link)	Returns True(1) if the network's collection of link members contains the specified link. Otherwise the value False (0) is returned.

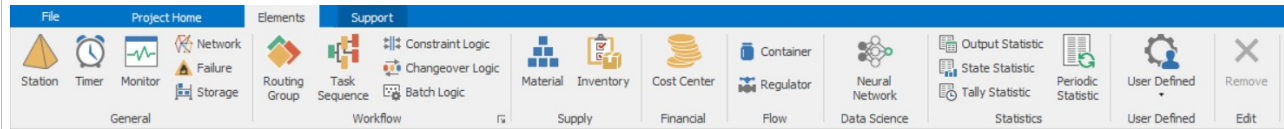
[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Neural Network Element

A Neural Network Element may be used to integrate a neural network regression model into simulation logic.

The purpose of the Neural Network Element is to provide expressions that specify the inputs to the [Neural Network Model](#), define an expression for the actual value to be recorded for comparison to the predicted value, and to define triggers to initiate the recording the input values and actual value. An optional unique training key is also provided to map the input values to the corresponding actual value.



Listed below are the properties of **Neural Network Element**:

Property	Description
Neural Network Model Name	The name of the associated neural network model. For more information on creating and training a neural network regression model, refer to the Data, Neural Network Models view.
Input Value Expressions	The expressions used to get a set of input values for the associated neural network model. Note that the order of these expressions should always match the order of the input values provided during the training of the model.
Untrained Predicted Value Expression	The expression used as a substitute to return a predicted output value if the associated neural network model is not trained.
Save Inputs Triggers	Optional event-driven triggers that will save a set of input values for the associated neural network model. The Training Key Expression is used to uniquely identify the saved input data.
Save Actual Triggers	Optional event-driven triggers that will save an actual observed value for the associated neural network model. The Training Key Expression is used to pair the actual observed value with a previously saved set of input values so that a new record may be added to the neural network model's training data repository.
Actual Value Expression	The expression used to record an actual observed value for the associated neural network model when a Save Actual trigger occurs.
Training Key Expression	The expression used to return a key for pairing a saved set of input values with a saved actual observed value so that a new record may be added to the associated neural network model's training data repository. Can be any expression that evaluates to a string value. If left unspecified, defaults to the string identifier name of the object associated with the Save Inputs or Save Actual triggering event occurrence.
Name	The name of this instance.
Description	Description text for this instance.
Public	Indicates if this instance is publicly accessible outside of its containing object.
Report Statistic	Indicates whether statistics are to be automatically reported for this object or element.

Listed below are the functions of **Neural Network Element**:

Function	Description
IsTrained	Returns True (1) if the associated neural network model has a defined computational graph that returns a predicted output value given a set of input values. Otherwise returns False (0).
PredictedValue	Evaluates the neural network's mapped input value expressions in the context of the specified object. Then, for the resulting set of input values, runs the associated neural network model and returns a predicted output value. Note that the neural network's Untrained Predicted Value Expression property is used as a substitute to return a predicted output value if the associated neural network model is not trained. By default, that property will return a random number between 0 and 1, but any custom expression can alternatively be used.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

OutputStatistic

OutputStatistic

An Output Statistic element may be used to record and report statistics on a specified expression.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **OutputStatistic**:

Property	Description
Report Statistics	Specifies if statistics are to be automatically reported for this element.
Expression	The expression to be recorded.
Data Source	Optional string used as the Data Source field for classifying results reported by this statistic element. If defaulted, then the element name is assumed.
Category	Optional string used as the Category field for classifying results reported by this statistic element. If defaulted, then the string 'UserSpecified' is assumed.
Data Item	Optional string used as the Data Item field for classifying results reported by this statistic element. If defaulted, then the string 'Output' is assumed.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

OutputStatistic - Discussion and Examples

OutputStatistic Example 1

The following is an example of a [OutputStatistic](#) element that is evaluating an expression which uses three different state variables to determine a price for bad customer service.

- The *Expression* property is evaluated and recorded at the end of each replication of the simulation. It multiplies the value of the state *WaitForTable* and multiplies it by a factor of 3.5. It does the same for the states *WaitForGreeting* and *WaitForFood*, using different pricing factors and adds these all together.
- The *Category* property is set to 'Costs', indicating that this OutputStatistic will appear in the user defined Category called 'Costs' in the pivot table within the Results Window.

Properties Window of the OutputStatistic Element

Properties: OutputStatistic1_CustomerService (OutputStatistic Element Instance)	
Basic Logic	
Unit Type	Time
Expression	$(\text{WaitForTable} * 3.5) + (\text{WaitForGreeting} * 2) + (\text{WaitForFood} * 4)$
Units	Hours
Results Classification	
Data Source	CustomerServiceCost
Category	Costs
Data Item	
General	
Name	OutputStatistic1_CustomerService
Description	
Public	True
Report Statistics	True

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Path Planner

Path Planner Element

A Path Planner element may be used in conjunction with a Transfer step to find and reserve a deadlock-free shortest network path for an entity from its current node to a destination node.



Listed below are the properties of the Path Planner Element:

Property	Description
Plan Path Condition	Condition expression that must evaluate to true for an entity to use the path planner.
Request Ranking Rule	The static rule used to rank competing entities waiting in the path planner's request queue.
Request Ranking Expression	The expression used for the smallest or largest value first ranking rule.

Path Planner elements can be incorporated into a model by referencing a Path Planner element on the *Path Planner Name* property of a BasicNode, TransferNode, or Transfer step. A Path Planner element referenced by a Transfer step's *Path Planner Name* property will be ignored unless an entity is attempting to transfer from a Node to an outbound link. Note that the Path Planner element is incompatible with flow entities, and if a Flow Regulator element is specified at the same Transfer step as a Path Planner element, the Path Planner element will be ignored.

When a Path Planner element is specified and the entity attempting the Transfer is assigned a valid destination node, the entity will evaluate the Path Planner element's *Plan Path If* condition expression to determine if it should plan a path from its current node to its destination. If this condition evaluates to 'false', it will Transfer out of its current node normally. If the condition evaluates to 'True', the entity will submit a request to plan a path to its destination node.

Entities requesting planned paths submit the request to the Path Planner element. Information about the entities waiting for planned paths can be accessed via the Path Planner element's RequestQueue, where the waiting requests are stored. Requests in the queue are prioritized according to the *Request Ranking Rule* and *Request Ranking Expression*, therefore entities at the front of the queue will get priority when competing with other entities for the same path. If an entity's request for a planned path cannot be immediately completed the entity will remain at its current node, and the token will wait at the Transfer step until a path to the entity's destination node is reserved.

When a path is found the links along the path are reserved, request for a planned path is completed, and the planned path is assigned to the entity. Under some conditions multiple entities can reserve the same path. The entity will then transfer to the first outbound link along the planned path to its destination node, and the token executing the Transfer step will exit the step through the "OK" exit. The entity will continue traversing links along its planned path until it reaches its destination node, its *Plan Path If* Condition expression resolves to true when attempting to Transfer to an outbound link, or its Planned Path is cancelled. A cancellation can occur if the entity's network travel information has changed, such as its destination node or its current network. If a planned path cancellation occurs, replanning may occur at the next node if the *Plan Path If* Condition expression resolves to 'True'.

Information about an entity's planned path can be accessed using the syntax "Entity.PlannedPath", and the expression "Link.PlannedPathLoad" will return the number of entities who have currently reserved a path including this link but have not yet entered the link.

See the [Path Planner - Discussion and Examples](#) page for more information about network path planning.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

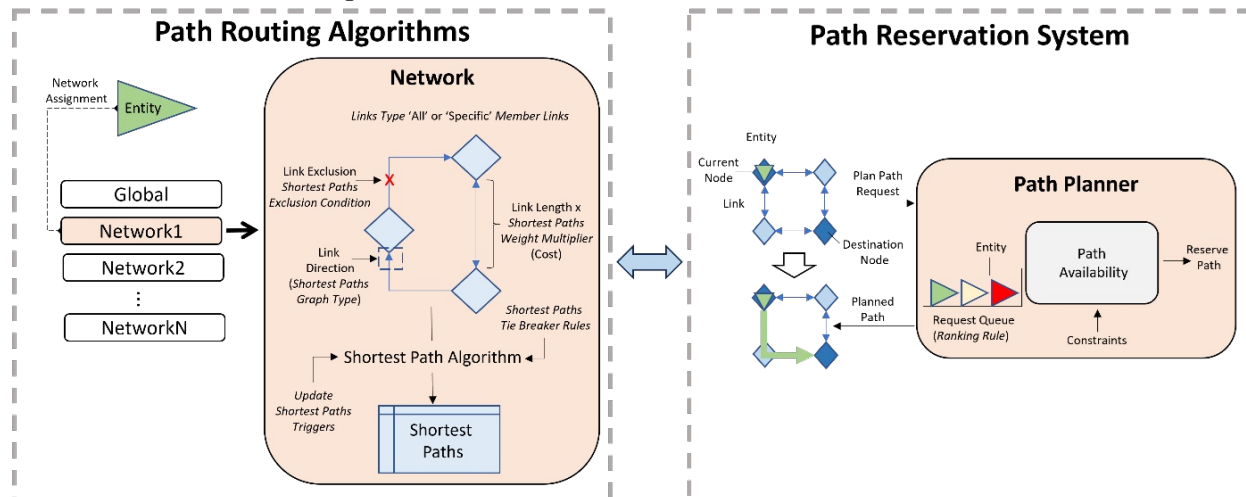
Path Planner - Discussion and Examples

Network Path Planning - Discussion

The objective of network path planning is to find and reserve a deadlock free shortest path for an entity on a bidirectional network from its current node to a destination node. In bidirectional networks such as those commonly found in warehouse systems, modeling logic that avoids head-on deadlocks on bidirectional paths can be very challenging. Network Path Planning features may help simplify the deadlock avoidance problem.

The figure below describes how path routing and path reservations interact within the context of network path planning. Path routing and path reservations are decoupled, such that entities independently find the shortest path to their destinations using a Network Element, then reserve that path using a Path Planner element. Therefore, the logic to modify which paths are selected by entities is contained within the Network element, and the logic to modify how entities reserve paths is contained within the Path Planner element. See the Network Element page for more information about path routing and the Path Planner Element page for more information about path reservations.

Network Path Planning in Simio



To use a Path Planner element, create a Path Planner element instance and set its *Plan Path If* and *Plan Path Request Ranking Rule* properties. The *Request Ranking Rule* will determine how entities competing to reserve paths will be prioritized. The *Plan Path If* condition expression will determine when an entity plans or replans a path to its destination. A value of 'True' will cause entities to replan their path to their destination node every time they transfer from their current node, if able. Although this replanning behavior might be desirable, it could also cause entities' planned paths to be reserved by higher priority entities. An alternative approach to constant replanning is to use an expression like `'Entity.PlannedPath.Links.NumberItems == 0'`, which will cause entities to only plan a path if the entity does not already have a planned path. Once a Path Planner element instance has been created, it can be referenced via the *Path Planner Name* property of a BasicNode, TransferNode, Transfer step.

Consideration of the value of a Network element's *Shortest Paths Graph Type* is paramount when implementing network path planning. A common configuration is setting *Shortest Paths Graph Type* to 'LinkCurrentDirections', which will create behavior where entities will select the shortest available paths to their destinations. Alternatively, using a *Shortest Paths Graph Type* value of 'Link Types' will cause entities to always plan the shortest possible paths to their destinations. Using *Shortest Paths Graph Type* value of 'LinkDesiredDirections' will allow logic to set the desired directions of links to affect the paths planned by entities.

In models with many paths and many entities moving simultaneously, dynamic shortest path calculations during a run could slow down the model. Use of a Path Planner element with its *Plan Path If* condition expression set to an expression such as `'Entity.PlannedPath.Links.NumberItems == 0'` could speed up the model since each entity's path is planned immediately at its origin node, rather than planned dynamically as the entity travels. Another approach can be used in models with distinct collections of links when entities are isolated to only one set of links. Under these circumstances, adding a unique Network and Path Planner element for each set of links will increase the performance of shortest path calculations. For example, if modeling two separate facilities in the same model, you could define a unique Network and Path Planner element for each facility.

Although proper use of the Path Planner and Network elements can help avoid deadlocks, logical deadlocks may still occur. For instance, in highly congested systems there may be situations where no entity can find a deadlock-free path to its destination. Deadlocks might also occur when one entity attempts to turn around on a path reserved by another entity. Specifying the Traveler Capacity of such a link to '1' can help avoid deadlocks caused by entity turnarounds.

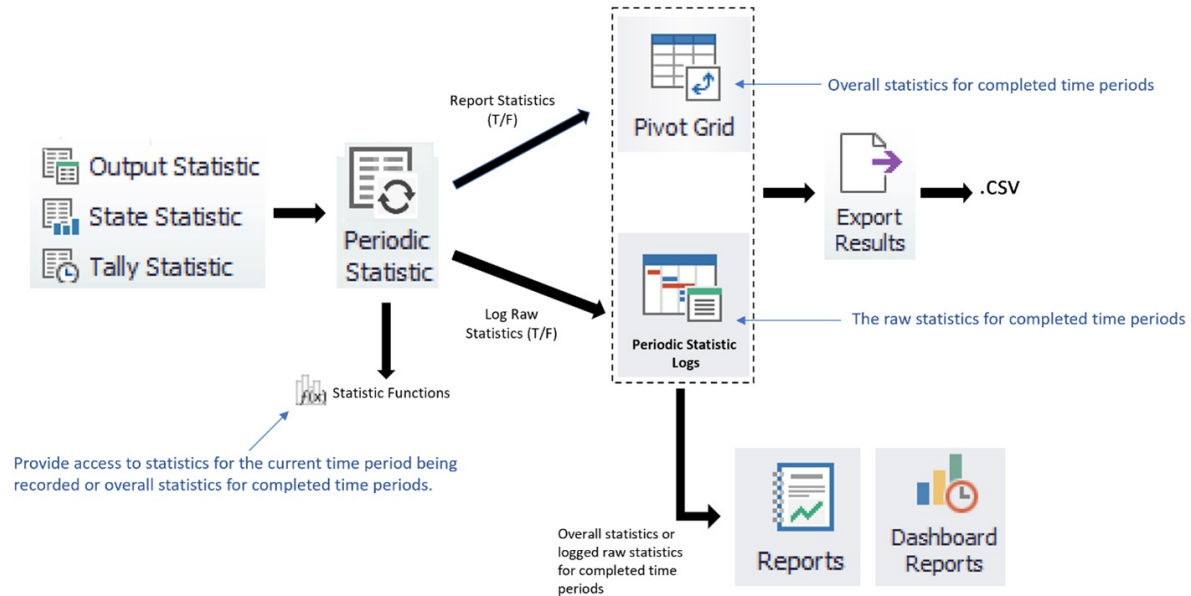
PeriodicStatistic

Periodic Statistic

A **Periodic Statistic** element may be used in conjunction with an **Output Statistic**, **State Statistic**, or **Tally Statistic** element to periodically record and report statistics for distinct time periods (e.g., daily or weekly statistics recording).

[Worker - Discussion and Examples](#)

Periodic Statistic Element Overview



Adding a Periodic Statistic Element

Listed below are the Properties of **Periodic Statistic** element:

Property	Description
Statistic Element Type	The statistic element type for which to record statistics.
Output Statistic Name	The name of the output statistic for which to record periodic statistics.
State Statistic Name	The name of the state statistic for which to record periodic statistics.
Tally Statistic Name	The name of the tally statistic for which to record periodic statistics.
Anchor Date Type	Indicates the start date of the time period pattern to record statistics. RunStart - The anchor date is the simulation run's start date and time. Specific - The anchor date is a specific date and time. If the Repeating Period Pattern property is true, then the time period pattern repeats infinitely from the anchor date in both directions (i.e., both forward and backward in time).
Anchor Date Time	The specific anchor date and time.
Period Lengths	The lengths of the time periods to record statistics.
Number of Periods	The number of distinct time periods.
Period Length	The length of each of the time periods.
Time Offset Before	The time offset before the start of the first time period.
Time Offset Between	The time offset between the time periods.
Time Offset After	The time offset after the end of the last time period.
Repeating Period Pattern	Indicates whether the time period pattern repeats infinitely from the anchor date in both directions (i.e., both forward and backward in time).
Period Name Prefix	The text added to the front of a time period's number to form the period name, which is used to distinguish individual time periods in reports and logs.
Log Raw	Indicates whether the raw statistics for completed time periods are to be automatically logged. Go to

Statistics Results or Planning -> Logs -> Periodic Output Statistic Log, Periodic State Statistic Log, or Periodic Tally Statistic Log to view the logged data.

Periodic Statistic Element Properties

Properties: PeriodicStatistic1 (Periodic Statistic Element)	
Statistic Element	
Element Type	OutputStatistic
Output Statistic Name	
Time Period Pattern	
Anchor Date Type	RunStart
Period Lengths	0 Rows
Repeating Period Pattern	True
Period Name Prefix	Period
Reporting & Logging	
Log Raw Statistics	False

Period Lengths - Repeating Property Editor

Items: 1, 1.0, 0.0, 0.0, 0.0

Properties:

Time Period Pattern	
Number Of Periods	1
Period Length	1.0
Time Offsets	
Time Offset Before	0.0
Time Offset Between	0.0
Time Offset After	0.0

Time Period Pattern

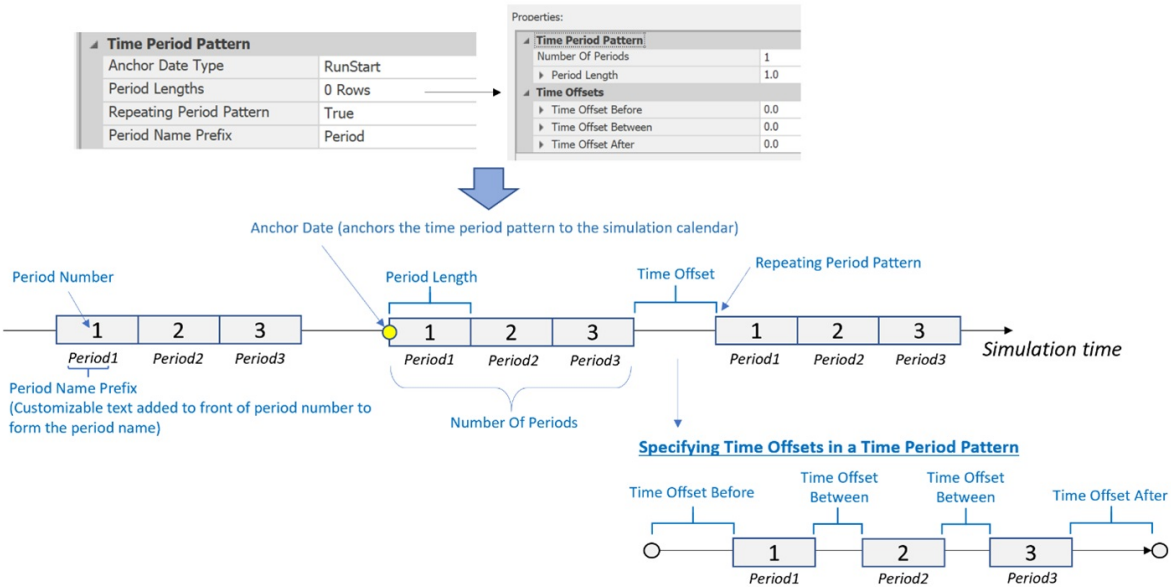
Time Period Pattern

Close

Defining the Time Period Pattern

A **Periodic Statistic** element provides several properties for defining its time period pattern.

Periodic Statistic Element - Defining the Period Pattern



Anchor Date

The anchor date is the date and time when the time period pattern starts. This date may be specified as the simulation run's start date and time or as a specific date and time.

If the *Repeating Period Pattern* property is 'True', then the pattern repeats infinitely from the anchor date in both directions (i.e., both forward and backward in time).

Period Lengths

A repeat group data structure is provided to define the lengths of the time periods, as well as any time offsets or "gaps" before, between, or after time periods.

During a simulation run, the distinct time periods in the pattern will be automatically numbered sequentially, starting at 1 for the first time period and then incrementing by 1 for each successive time period until the end of the pattern is reached.

Repeating Period Pattern

Indicates whether the time period pattern repeats infinitely from the anchor date in both directions (i.e., both forward and backward in time).

Period Name Prefix

Specifies the text added to the front of a time period's number to form the period name, which is used to distinguish individual time periods in reports and logs.

For example, if the specified prefix text is 'Day' then the period names will be 'Day1', 'Day2', etc. If the specified prefix text is 'Q', then the period names will be 'Q1', 'Q2', etc.

See Time Period Pattern Examples below.

Periodic Statistic Element Functions

The following functions will be provided by a **Periodic Statistic** element:

Function Name	Description
PeriodName	Returns the name of the time period currently being recorded.
PeriodNumber	Returns the number of the time period currently being recorded.
PeriodStartTime	Returns the simulation time (in hours) that was the start of the time period currently being recorded.
PeriodEndTime	Returns the simulation time (in hours) that is the end of the time period currently being recorded.
Average	Returns the average value recorded for the statistic element during

	the current time period. Is only applicable to a periodic state or tally statistic.
Minimum	Returns the minimum value recorded for the statistic element during the current time period. Is only applicable to a periodic state or tally statistic.
Maximum	Returns the maximum value recorded for the statistic element during the current time period. Is only applicable to a periodic state or tally statistic.
NumberObservations	Returns the number of observations recorded for the statistic element during the current time period. Is only applicable to a periodic tally statistic.
ValueChange	Returns the change in value of the statistic element's output expression from the start of the current time period. Is only applicable to a periodic output statistic.
OverallAverage(periodNumber)	Returns the overall average value recorded for the statistic element during completed time periods of the specified period number. Is only applicable to a periodic state or tally statistic.
OverallMinimum(periodNumber)	Returns the overall minimum value recorded for the statistic element during completed time periods of the specified period number. Is only applicable to a periodic state or tally statistic.
OverallMaximum(periodNumber)	Returns the overall maximum value recorded for the statistic element during completed time periods of the specified period number. Is only applicable to a periodic state or tally statistic.
OverallNumberObservations(periodNumber)	Returns the overall number of observations recorded for the statistic element during completed time periods of the specified period number. Is only applicable to a periodic tally statistic.
OverallTotalValueChange(periodNumber)	Returns the total change in value of the statistic element's output expression during completed time periods of the specified period number. Is only applicable to a periodic output statistic.
OverallAverageValueChange(periodNumber)	Returns the average change in value of the statistic element's output expression during completed time periods of the specified period number. Is only applicable to a periodic output statistic.

Time Period Pattern Examples

This section contains some examples of defining the time period pattern of a **Periodic Statistic** element.

Example 1: Three 8-Hour Shifts, 7 Days Per Week

Time Period Pattern

Anchor Date Type	Specific
Anchor Date Time	1/1/2020 12:00:00 AM
Period Lengths	1 Row
Repeating Period Pattern	True
Period Name Prefix	Shift

Period Lengths - Repeating Property Editor

Items: 3, 8.0, 0.0, 0.0, 0.0

Properties:

Time Period Pattern

Number Of Periods

3

Period Length

8.0

Units

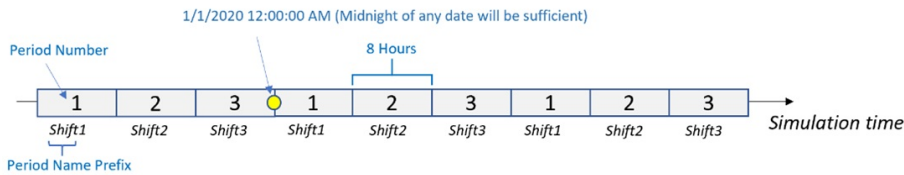
Hours

Time Offsets

Time Period Pattern

Time Period Pattern

Time Period Pattern



Example 2: Weekly, Monday Through Friday (Excludes Weekends)

Time Period Pattern

Anchor Date Type	Specific
Anchor Date Time	1/6/2020 12:00:00 AM
Period Lengths	1 Row
Repeating Period Pattern	True
Period Name Prefix	Week

Period Lengths - Repeating Property Editor

Items: 1, 5.0, 0.0, 0.0, 2.0

Properties:

Time Period Pattern

Number Of Periods

1

Period Length

5.0

Units

Days

Time Offsets

Time Offset Before

0.0

Time Offset Between

0.0

Time Offset After

2.0

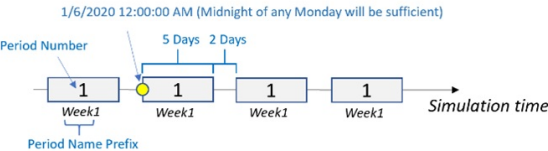
Units

Days

Time Period Pattern

Time Period Pattern

Time Period Pattern



Example 3: Monthly, 4-4-5 Calendar

Time Period Pattern

Anchor Date Type

Anchor Date Time

Period Lengths

Repeating Period Pattern

Period Name Prefix

Specific

1/1/2020 12:00:00 AM

8 Rows

True

Month

Period Lengths - Repeating Property Editor

Items:

2, 4, 0, 0, 0, 0, 0, 0

1, 5, 0, 0, 0, 0, 0, 0

2, 4, 0, 0, 0, 0, 0, 0

1, 5, 0, 0, 0, 0, 0, 0

2, 4, 0, 0, 0, 0, 0, 0

1, 5, 0, 0, 0, 0, 0, 0

2, 4, 0, 0, 0, 0, 0, 0

1, 5, 0, 0, 0, 0, 0, 0

Properties:

Time Period Pattern

Number Of Periods

Period Length

Units

Time Offsets

2

4.0

Weeks

Time Period Pattern

Time Period Pattern

Period Lengths:

2 Periods, 4 Weeks

1 Period, 5 Weeks

2 Periods, 4 Weeks

1 Period, 5 Weeks

2 Periods, 4 Weeks

1 Period, 5 Weeks

2 Periods, 4 Weeks

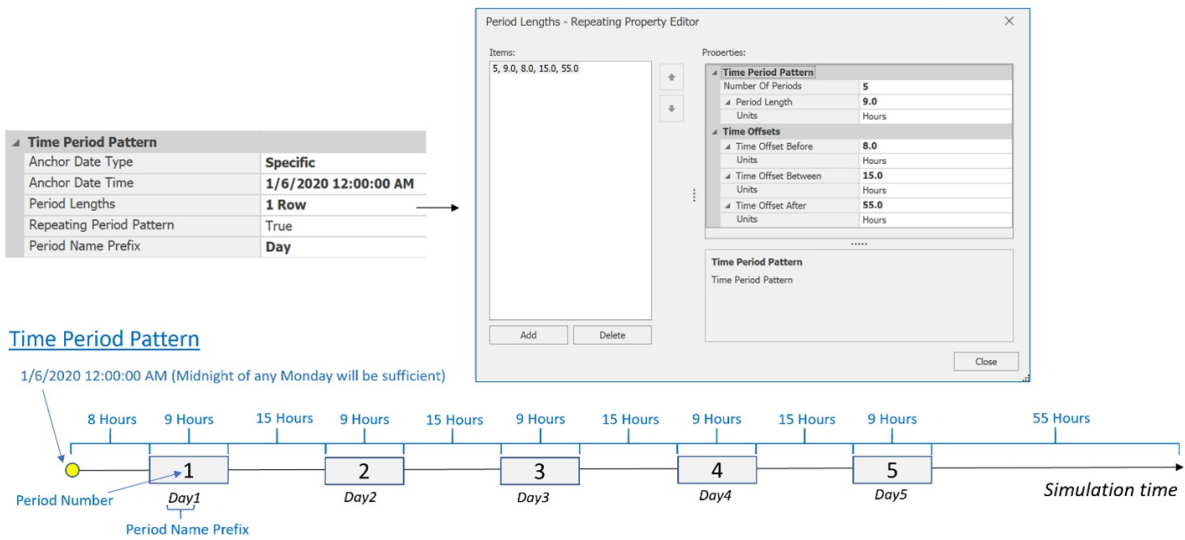
1 Period, 5 Weeks

12 Periods each representing a 'Month'

Time Period Pattern

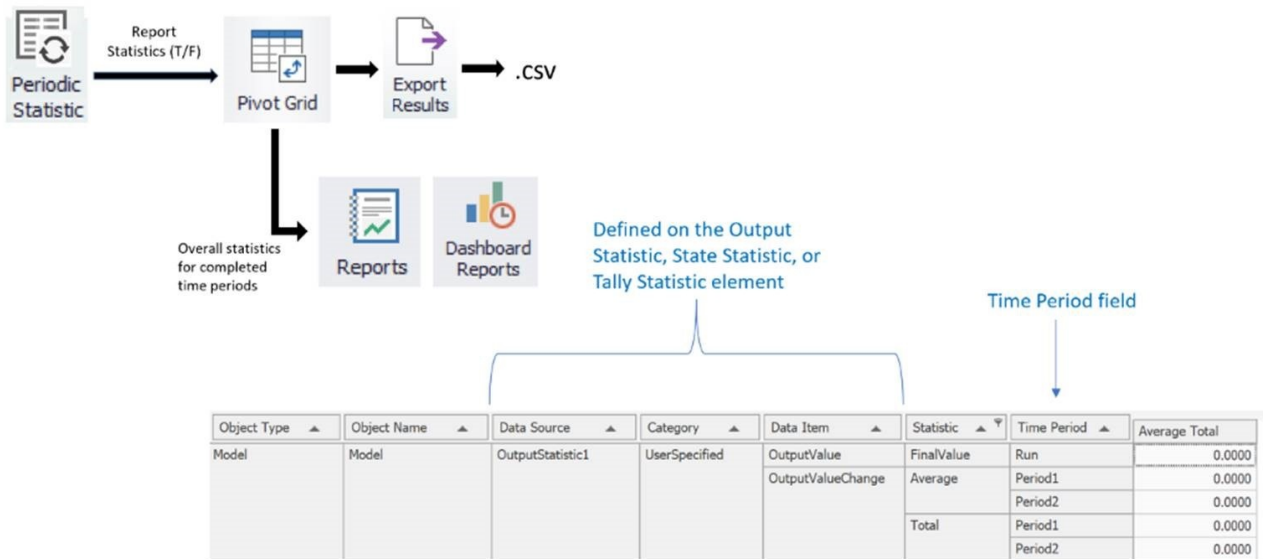


Example 4: Daily 8AM to 5PM, Monday Through Friday



Reporting Overall Statistics for Completed Time Periods

If the *Report Statistics* property of a **Periodic Statistic** element is set to 'True', then overall statistics for time periods completed during the run will be automatically reported by period name. That statistical data may be explored in the Pivot Grid, exported to a comma separated value file (.csv), or used as a data source to create reports.

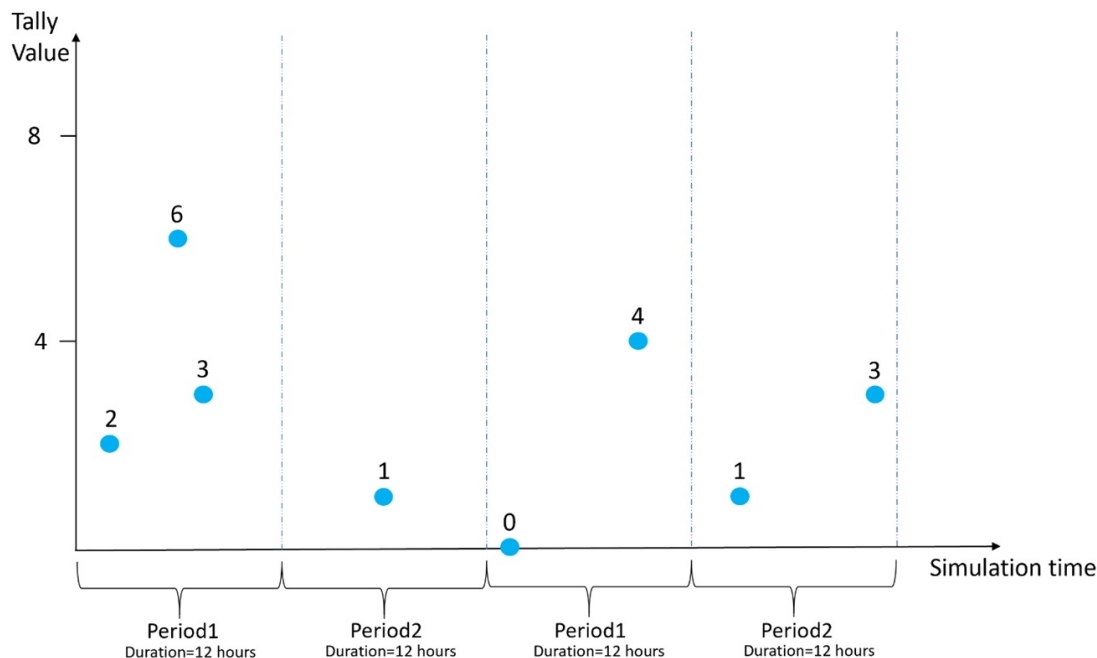


The following overall statistics will be reported by a **Periodic Statistic** element:

Reported Statistic	Periodic Output Statistic	Periodic State Statistic	Periodic Tally Statistic
Average	No	Yes	Yes
Minimum	No	Yes	Yes
Maximum	No	Yes	Yes
TotalValueChange	Yes	No	No
AverageValueChange	Yes	No	No
Observations	No	No	Yes

Periodic Tally Statistic Example

The image below illustrates an example of a periodic tally statistic. The periodic statistics are collected using a repeating period pattern that consists of two distinct time periods named 'Period1' and 'Period2'.



The logged raw statistics for the completed time periods are as follows:

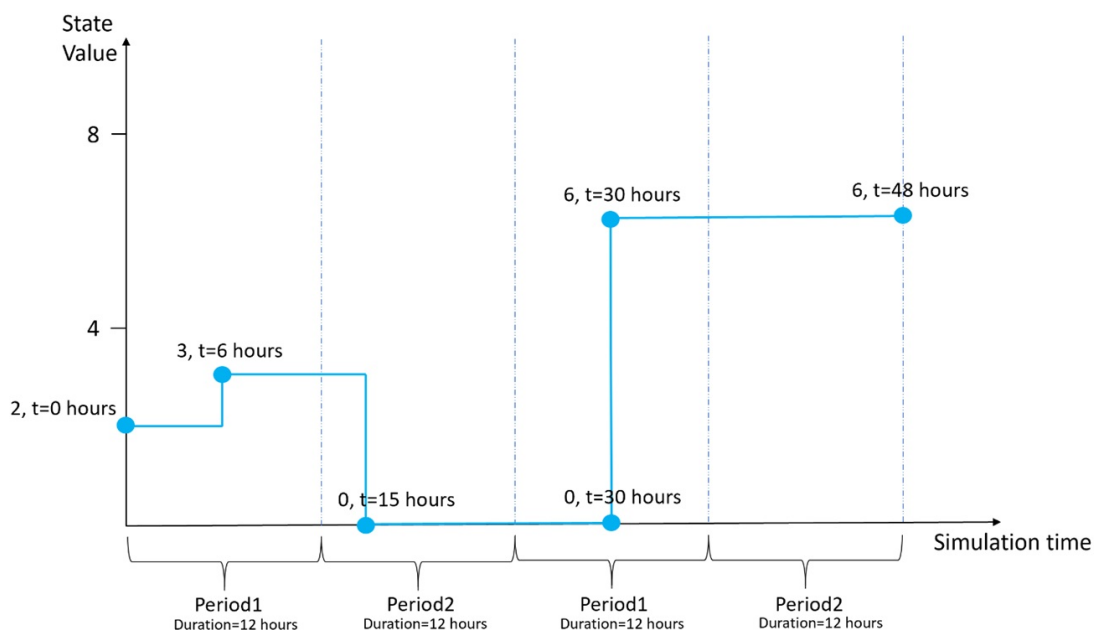
Period Name	Average	Minimum	Maximum	Observations
Period1	3.667	2	6	3
Period2	1	1	1	1
Period1	2	0	4	2
Period2	2	1	3	2

The reported overall statistics for the completed time periods are as follows:

Period Name	Average	Minimum	Maximum	Observations
Period1	3	0	6	5
Period2	1.67	1	3	3

Periodic State Statistic Example

The image below illustrates an example of a periodic state statistic. The periodic statistics are collected using a repeating period pattern that consists of two distinct time periods named 'Period1' and 'Period2'.



The logged raw statistics for the completed time periods are as follows:

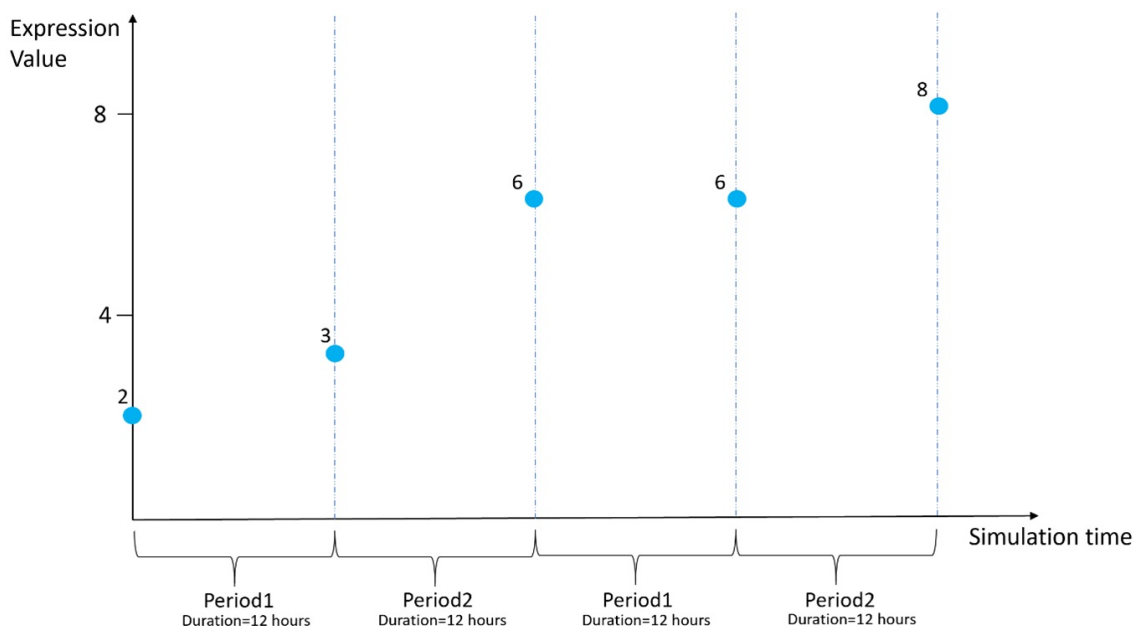
Period Name	Average	Minimum	Maximum	Value Change
Period1	2.5	2	3	1
Period2	0.75	0	3	-3
Period1	3	0	6	6
Period2	6	6	6	0

The reported overall statistics for the completed time periods are as follows:

Period Name	Average	Minimum	Maximum
Period1	2.75	0	6
Period2	3.375	0	6

Periodic Output Statistic Example

The image below illustrates an example of a periodic output statistic. The periodic statistics are collected using a repeating period pattern that consists of two distinct time periods named 'Period1' and 'Period2'.



The logged raw statistics for the completed time periods are as follows:

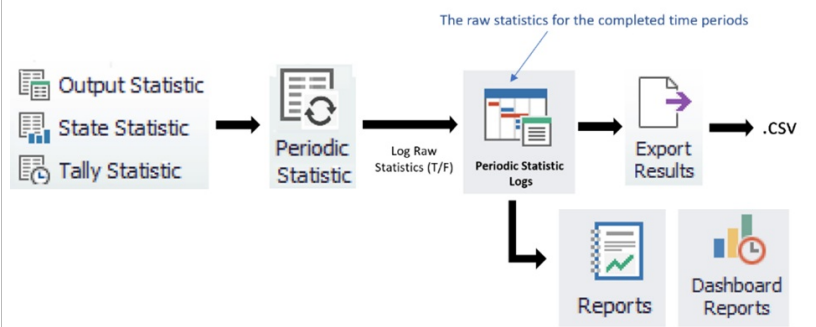
Period Name	Value Change
Period1	1
Period2	3
Period1	0
Period2	2

The overall statistics for the completed time periods are as follows:

Period Name	Average Value Change	Total Value Change
Period1	0.5	1
Period2	2.5	5

Logging the Raw Statistics for Completed Time Periods

If the *Log Raw Statistics* property of a **Periodic Statistic** element is set to 'True', then the raw statistics for each time period completed during the run will be logged. That logged data may be explored in the associated periodic statistic log, exported to a comma separated value file (.csv), or used as a data source to create reports.



New Logs

The following new logs will be provided:

Periodic Output Statistic Log

Field	Description
Time Period	The name of the time period.

Start Time	The simulation time (in hours) that the time period started.
End Time	The simulation time (in hours) that the time period ended.
Duration	The duration of the time period.
Object Type	Object type classification field.
Object Name	Object name classification field.
Data Source	Data source classification field.
Category	Category classification field.
Data Item	Data item classification field.
Start Value	The value of the output expression at the start of the time period.
End Value	The value of the output expression at the end of the time period.
Value Change	The change in value of the output expression during the time period.
Units	The units of the output expression values.

Periodic State Statistic Log

Field	Description
Time Period	The name of the time period.
Start Time	The simulation time (in hours) that the time period started.
End Time	The simulation time (in hours) that the time period ended.
Duration	The duration of the time period.
Object Type	Object type classification field.
Object Name	Object name classification field.
Data Source	Data source classification field.
Category	Category classification field.
Data Item	Data item classification field.
Start Value	The value of the state variable at the start of the time period.
End Value	The value of the state variable at the end of the time period.
Value Change	The change in value of the state variable during the time period.
Average	The time-weighted average of the values recorded for the state variable during the time period.
Minimum	The minimum value recorded for the state variable during the time period.
Maximum	The maximum value recorded for the state variable during the time period.
Units	The units of the state variable values.

Periodic Tally Statistic Log

Field	Description
Time Period	The name of the time period.
Start Time	The simulation time (in hours) that the time period started.
End Time	The simulation time (in hours) that the time period ended.
Duration	The duration of the time period.
Object Type	Object type classification field.
Object Name	Object name classification field.
Data Source	Data source classification field.
Category	Category classification field.
Data Item	Data item classification field.
Observations	The number of observation values recorded for the tally statistic during the time period.
Average	The average of the observation values recorded for the tally statistic during the time period.
Minimum	The minimum observation value recorded for the tally statistic during the time period.
Maximum	The maximum observation value recorded for the tally statistic during the time period.
Units	The units of the state observation values.

How to Record Periodic Statistics that are Time-Weighted Averages

A *time-weighted* average takes into consideration not only the values of a particular variable but also assigns a weight to each observed value according to the amount of time that it occurred.

Several statistics automatically recorded and reported by Simio are time-weighted averages, such as average queue lengths, average inventories, or scheduled utilizations of resources.

For example, suppose for a 24-hour period, the number waiting in a particular queue was 5 entities for 4 hours, 8 entities for 8 hours, and 2 entities for 12 hours. The average number waiting in that queue during the time period would be calculated and reported as:

$$(5 \text{ entities} \times 4 \text{ hours}) + (8 \text{ entities} \times 8 \text{ hours}) + (2 \text{ entities} \times 12 \text{ hours}) = 108 \text{ entity hours}$$

$$\text{average number waiting (during the 24 hour period)} = \frac{108 \text{ entity hours}}{24 \text{ hours}} = 4.5 \text{ entities}$$

There may be cases where a simulation's output needs to include periodic statistics that are time-weighted averages (e.g.,

daily or weekly resource utilizations). To satisfy such output requirements, the following modeling approach in Simio may be used:

1.) Add an Output Statistic element whose expression is in the form:

$$(Average * Run.TimeNow) / (PeriodicStatisticName.PeriodEndTime - PeriodicStatisticName.PeriodStartTime)$$

where *Average* is the expression that returns the time-weighted average for the entire run and *PeriodicStatisticName* is the name of the Periodic Statistic element that will be used to record the value change of the Output Statistic's expression for each distinct time period.

2.) Specify the Report Statistics property of the Output Statistic element as 'False' so that the Output Statistic does not try to report the final value of its expression as an actual result, which potentially would be a Divide-By-Zero error.

3.) Add the Periodic Statistic element with the desired time period pattern that is referencing the Output Statistic element.

The figure below illustrates the recording of a Server's daily scheduled utilization using this modeling approach.

Periodic Statistic Example - Recording a Server's Daily Scheduled Utilization

Properties: OutputStatistic1 (Output Statistic Element)

Basic Logic

Unit Type	Unspecified
Expression	(Server1.Capacity.ScheduledUtilization * Run.TimeNow) / (PeriodicStatistic1.PeriodEndTime - PeriodicStatistic1.PeriodStartTime)

Properties: PeriodicStatistic1 (Periodic Statistic Element)

Statistic Element

Element Type	OutputStatistic
Output Statistic Name	OutputStatistic1

Time Period Pattern

Anchor Date Type	RunStart
Period Lengths	1 Row
Repeating Period Pattern	True
Period Name Prefix	Day

Period Lengths - Repeating Property Editor

Items:

7, 1, 0.0, 0.0, 0.0

AddDelete

Properties:

Time Period Pattern

Number Of Periods	7
Period Length	1
Units	Days

Time Offsets

Time Offsets

Close

If the Capacity of the Resource is not fixed, it is necessary to only account for the time that the Resource is On-Shift. One approach would be to use "ResourceName.ResourceState.TotalTime(1)" to represent the total time the Resource spent in the Processing Resource State. It is important to note that if there are multiple units of Capacity in a Server, the Resource State will change to 1 even if only one unit of Capacity is being used. Alternatively, replacing Run.TimeNow with a custom State that tracks only the time the Resource is On-Shift is another approach to not consider the Off-Shift time.

Proof of the Modeling Approach

Prove:

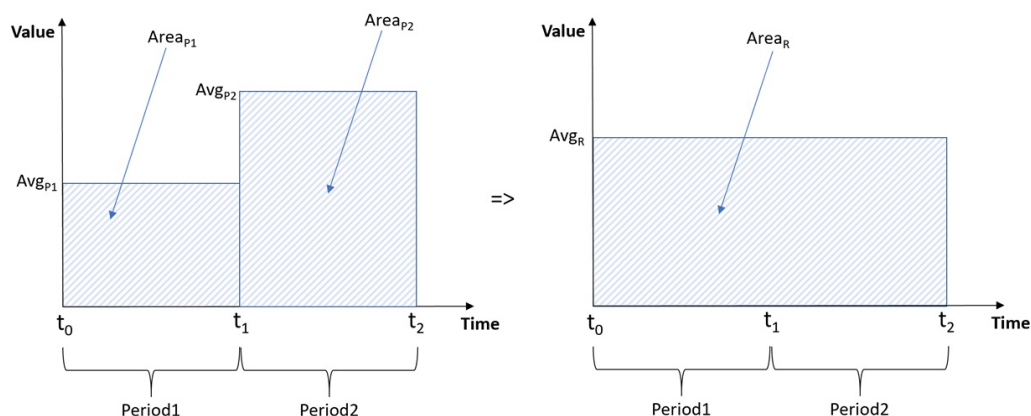
A time-weighted average during any time period in a simulation run can be calculated as the difference in the following expression's value between the start and end of the time period:

$$\frac{AvgR * t_{NOW}}{(t_E - t_S)}$$

Where *AvgR* is the time-weighted average for the run, *tNOW* is the current time, *tS* is the start of the time period, and *tE* is the end of the time period.

Proof:

Suppose a simulation run consists of two distinct time periods as shown in the figure below.



Where $Av_{g_{P1}}$ is the time-weighted average during the first time period from t_0 to t_1 , $Av_{g_{P2}}$ is the time-weighted average during the second time period from t_1 to t_2 , and Av_{g_R} is the time-weighted average during the entire run (t_0 to t_2).

The time-weighted average during the run up to time t_1 is simply equal to the average during the first time period, as shown here:

$$Av_{g_{R_{t_1}}} = \frac{Area_{P1}}{t_1} = \frac{Av_{g_{P1}} t_1}{t_1} = Av_{g_{P1}}$$

The time-weighted average during the run up to time t_2 is equal to:

$$Av_{g_{R_{t_2}}} = \frac{Area_{P1} + Area_{P2}}{t_2} = \frac{Av_{g_{P1}} t_1 + Av_{g_{P2}} (t_2 - t_1)}{t_2} = \frac{Av_{g_{R_{t_1}}} t_1 + Av_{g_{P2}} (t_2 - t_1)}{t_2} \text{ since } Av_{g_{R_{t_1}}} = Av_{g_{P1}}$$

Therefore, using the above equation to solve for the average during the second time period, the result becomes:

$$Av_{g_{P2}} = \frac{Av_{g_{R_{t_2}}} t_2 - Av_{g_{R_{t_1}}} t_1}{(t_2 - t_1)} = \frac{Av_{g_{R_{t_2}}} t_2}{(t_2 - t_1)} - \frac{Av_{g_{R_{t_1}}} t_1}{(t_2 - t_1)}$$

A similar calculation would apply to solve for the average during any subsequent time period.

Thus, in a general form, a time-weighted average during any time period in a simulation run can be calculated as the difference in the following expression's value between the start and end of the time period:

$$\frac{Av_{g_R} t_{NOW}}{(t_E - t_S)}$$

Where Av_{g_R} is the time-weighted average for the run, t_{NOW} is the current time, t_S is the start of the time period, and t_E is the end of the time period.

Regulator

Regulator

The Regulator element may be used to regulate flow transfers of entities into or out of a location.

The Regulator is modeled after a "pump", where the `CurrentMaximumFlowRate` state is the throttle/speed setting, and the `Enabled` state is the on/off switch.

If the *Input Flow Control Mode* of a Regulator is set to 'MergeFlow', entities with different Destination Nodes will not be merged. Simio dynamically stops/starts merging if an entity waiting or in a merge at a regulator changes its Destination Node.

Merge Allocation Rules when merging inflow include 'ProportionalBasedOnInflowRates', 'PreferredOrderByQueueRank' and 'ProportionalBasedOnExpression'.

The states listed below for the flow out of Regulators are read-only. These may be monitored to detect when a target volume or weight quantity has been transferred (e.g., new monitor threshold = `RegulatorXYZ.CurrentVolumeFlowOut + SomeVolumeQuantity`). The rate values of the state variables might be monitored to detect flow rate changes (e.g., you could monitor `Regulator.CurrentVolumeFlowOut.Rate` for changes).

The *Initial Output Yield Factor* property specifies the yield factor for the start of the simulation run. The `Regulator.CurrentOutputYieldFactor` can be assigned to change the yield over the course of the simulation run.

When a regulator is disabled and then enabled again, the outbound link selections are reevaluated.

Statistics are generated automatically for the flow out of a regulator, based on the flow states listed below, if the value is greater than 0. Statistics will also be reported on the flow in for the regulator, if the value is not equal to the flow out (that is, if a yield factor is applied).

For examples of using the Regulator element, please refer to the Flow Example models on the Simio User Forum (`xFlowNode`). The Regulator element is used inside the `FlowNode` object.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **Regulator**:

Property	Description
Flow Rate Unit Type	The unit type to be used for specifying the regulator's maximum flow rate.
Initial Maximum Flow Rate	The initial maximum flow rate for the regulator.
Initial Output Yield Factor	The initial output yield factor for the regulator. This factor is entered as a ratio of outflow to inflow, and may be used to scale the flow into the regulator such that there is a physical loss or gain represented in the regulator's output flow.
Initial Output Entity Type	Optional property specifying the initial desired entity type of the outflow produced by the regulator. If unspecified, then the outflow flow entity type will be determined by the inflow entity types.
Flow Request Ranking Rule	The static rule used to rank competing flow requests using the regulator.
Flow	The expression used with a Smallest Value First or Largest Value First ranking rule.

Request
Ranking
Expression

Input Flow Control Mode The mode used by the regulator to control the flow requests in its flow request queue.

Merge Matching Rule Specifies the match conditions required to merge flow requests in the regulator's flow request queue into a single output flow. Available when the *Input Flow Control Mode* is *MergeFlow*. If the matching rule is specified as 'AnyEntityType', then all inflows may be merged together regardless of entity type. If the matching rule is specified as 'SameEntityType', then only inflows with the same entity type may be merged.

Merge Allocation Rule The allocation rule used to indicate how flow requests in the regulator's flow request queue are merged into a single output flow. Available when the *Input Flow Control Mode* is 'MergeFlow'.

Merge Proportion Expression The expression used to return the desired merge proportion for a flow request in the regulator's flow request queue. Available when the *Input Flow Control Mode* is 'MergeFlow' and *Merge Allocation Rule* is 'ProportionalBasedOnExpression'.

Initially Enabled Specifies whether the regulator is enabled when the system is initialized.

Report Statistics Specifies if statistics are to be automatically reported for this element.

Listed below are the States of **Regulator**:

State	Description
CurrentDesiredOutputEntityType	State used to assign or to change the type of entity coming out of the Regulator. If set to 'Nothing', the Regulator will pick the type of entity with the most incoming flow into the regulator.
CurrentMaximumFlowRate	State used to get or set the current maximum flow rate of this regulator.
CurrentOutputYieldFactor	State used to get or set the current output yield factor of this regulator.
CurrentVolumeFlowIn	State to get the current total volume flowed into this regulator. The 'rate' from this state is also available.
CurrentWeightFlowIn	State to get the current total weight flowed into this regulator. The 'rate' from this state is also available.
CurrentVolumeFlowOut	State to get the current total volume flowed out of this regulator. The 'rate' from this state is also available.
CurrentWeightFlowOut	State to get the current total weight flowed out of this regulator. The 'rate' from this state is also available.
Enabled	Indicates whether the regulator is currently enabled.

Listed below are the Functions of **Regulator**:

Function	Description
OutputFlowReceivers.NumberItems	Returns the current number of entities receiving outflow from the

	regulator.
OutputFlowReceivers.FirstItem	Returns a reference to the first entity in the list of entities currently receiving outflow from the regulator.
OutputFlowReceivers.LastItem	Returns a reference to the entity at a specified index position in the list of entities currently receiving outflow from the regulator.
OutputFlowReceivers.IndexOfItem(entity)	Returns a reference to the entity at a specified index position in the list of entities currently receiving outflow from the regulator.
OutputFlowReceivers.ItemAtIndex(index)	Returns the one-based index of a specified entity in the list of entities currently receiving outflow from the regulator. If the entity is not a current output flow receiver of the regulator then the value 0 is returned.
OutputFlowReceivers.Contains(entity)	Returns True (1) if the list of entities currently receiving outflow from the regulator includes the specified entity. Otherwise, the value False (0) is returned.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RoutingGroup

RoutingGroup

The RoutingGroup element is used with a Route step to route an entity object to a destination selected from a list of candidate nodes (a Node [List](#)). If using only a static Route Request Ranking Rule on a RoutingGroup, when the destination in the RoutingGroup becomes available, the algorithm is to search through the RouteRequestQueue starting at Rank 1 and find the first eligible entity that meets eligibility requirements.

The Standard Library TransferNode object uses a RoutingGroup element in its internal logic to determine where the entities will travel when they leave the node. A user can create their own RoutingGroup element for use in a the TransferNode or a custom object.

The TransferNode includes an option to 'Use Custom Routing Group' within the *Entity Destination Type* property. This advanced option allows users to reference a custom RoutingGroup element directly within the TransferNode. There are several use cases for using this feature which are noted within the [TransferNode - Discussion and Examples](#) page. One use case is if multiple routing entities are blocked and waiting for an available destination (i.e., select the next entity to route from multiple blocked waiting locations), the RoutingGroup element provides a Dynamic Selection rule to dynamically select one of the waiting entities. This is a similar feature to a resource object being able to dynamically select an entity waiting to seize.

By default, for each candidate destination node in a Routing Group element's node list that is an 'Input' node into an associated object station location, the Routing Group automatically listens for station state changes (such as entities exiting the station or the station's capacity increases) and then does a selection search through its RouteRequestQueue. The default *Destination Blocked Condition* that is specified for the RoutingGroup element includes a default definition of blocked that is based on a candidate node being an 'Input' node with an associated station and it is considered 'blocked' if the associated station does not have any remaining capacity. There may, however, be modeling situations where just listening for station state changes at 'Input' nodes is not enough triggering events for the RoutingGroup to cover all possible selection search scenarios. Sometimes, an additional selection search may need to be triggered based on a conditional upstream system event (i.e., the number of entities waiting on a path larger than 3). The *Select Route Requests Triggers* repeat group of Triggering Events/Conditions allows users to define additional triggers that will cause re-evaluation of the RoutingGroup logic. If a [ConstraintLogic Element](#) is used, the RoutingGroup will be reevaluated when an item in the ConstraintLogic becomes available.

There are four processes that may be executed from within this RoutingGroup element, one prior, one during and two after the decision making process for the destination. These include *On Starting Route Request Queue Search*, *On Evaluating Route Request*, *On Confirming Destination Assignment* and *On Destination Assigned* process properties. The logical flow charts for how the RoutingGroup logic and add-on processes work can be found in [RoutingGroup - Discussions and Examples](#). When using a dynamic selection rule, the *On Evaluating Route Request* process is executed for every possible entity and destination node combination, to filter the list of possible destination node assignments. The best destination node assignment from the candidates that made it through the filtering is then selected per the specified rule. The *On Confirming Destination Assignment* process on the other hand is only executed whenever a destination node assignment has been selected, to confirm it. If this decision process returns False, then the routing group goes back to the drawing board and using the dynamic selection rule again looking for the next best entity/node combination (which would then have to be confirmed and so forth).

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **RoutingGroup**:

Property	Description
Destination Node List Name	Identifies the list of possible destination nodes in this routing group. This may be a node list or table reference including node objects.
Route Request Ranking Rule	The static rule used to rank competing entities waiting in the routing groups' route request queue.
Route Request	The expression used with a Smallest Value First or Largest Value First ranking rule.

Ranking Expression

Route Request Dynamic Selection Rule	The rule used to dynamically select an entity from competing requests waiting in the routing group's route request queue. Can be one of several Dynamic Selection Rules .	
Repeat Group	True, False	Indicates whether a repeat group data structure is used to define the primary dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax <code>Candidate.[EntityClass].[Attribute]</code> to reference an attribute of the candidate entities (e.g., <code>Candidate.Entity.Priority</code>).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.
Evaluate Route Requests Triggers	Optional event-driven triggers that will schedule an immediate evaluation of the routing group's route request queue, in order to assign any available destinations to eligible entity route requests.	
Triggering Event Name (Select Route Requests Triggers)	The name of the event whose occurrence will schedule an immediate evaluation of the routing group's route request queue, in order to assign any available destinations to eligible entity route requests.	

Condition (Select Route Requests Triggers)	Optional condition to be evaluated whenever the triggering event occurs, and which also must be true to trigger an evaluation of the routing group's route request queue.
--	---

Destination Blocked Condition	Logical condition used when evaluating an entity route request to determine whether a candidate location in the destination node list is considered to be blocked (i.e., has no space available). In the expression, use the syntax Candidate . [NodeClass] . [Attribute] to reference an attribute of either the candidate destination nodes themselves or the objects associated with those candidate nodes (e.g., Candidate.Node.AssociatedStation.Capacity or Candidate.Server.Capacity).
-------------------------------	---

On Starting Route Request Queue Search	Occurs when the routing group is about to search its route request queue to try to select a waiting entity and assign its destination node.
--	---

On Evaluating Route Request	Occurs when the routing group is executing whether an entity waiting in its route request queue is eligible to be assigned a particular destination node. In the executed decision process, assigning a non-positive value to the executing token's <code>ReturnValue</code> state indicates that the route request is not eligible.
-----------------------------	--

On Confirming Destination Assignment	Occurs when the routing group is about to assign a destination node to an eligible entity selected from its route request queue and is confirming the assignment. In the executed decision process, assigning a non-positive value to the executing token's <code>ReturnValue</code> state indicates to cancel the destination node assignment and re-evaluate the queue for a different selection (either different entity or different node).
--------------------------------------	---

On Assigned Destination	Occurs when an entity has been assigned a destination node by the routing group.
-------------------------	--

Report Statistics	Specifies if statistics are to be automatically reported for this element.
-------------------	--

Listed below are the states of **RoutingGroup**:

State	Description
RouteRequestQueue	The queue of entities waiting to route to a destination in this routing group's node list.

Listed below are the Events of **RoutingGroup**:

Event	Description
Assigning Destination	Routing Group Event triggered immediately before an entity is assigned a destination.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RoutingGroup - Discussion and Examples

Route Selection Algorithm

When a Route step is executed, the arriving entity is placed into the route request queue. The entity arrival event will trigger a routing group to schedule a late priority current event to try to select entities from its route request queue.

When checking its route request queue, the route selection algorithm used by the routing group is as follows:

If no dynamic selection rule is being used (static queue ranking rule only)

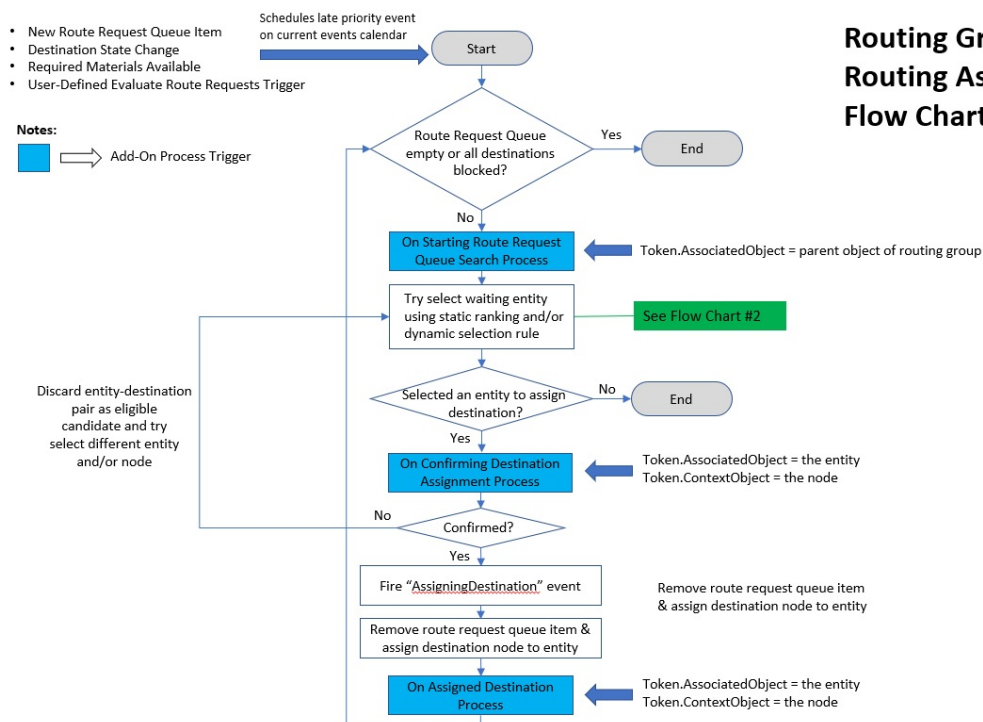
- Use the route request queue's static ranking rule to select a queue item.
- If there are multiple possible destination nodes available for the selected queue item, then use the entity's destination selection goal to select one of those nodes.

If dynamic selection rule is being used

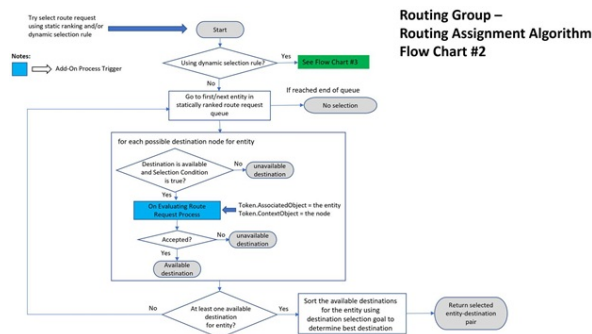
- Find the best route request queue item for each possible destination node using the specified dynamic selection rule.
- Use the route request queue's static ranking rule to select a queue item from the best queue items found.
- If there are multiple destination nodes whose best queue item is the selected queue item, then use the entity's destination selection goal to select one of those nodes.

RoutingGroup Element – Routing Assignment Algorithm Flow Charts

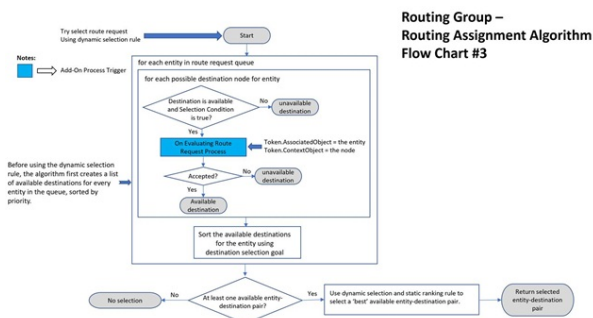
The below flow charts describe the logic of the RoutingGroup element decision making, as well as the timing of each of the Add-On Processes Triggers.



Routing Group – Routing Assignment Algorithm Flow Chart #1



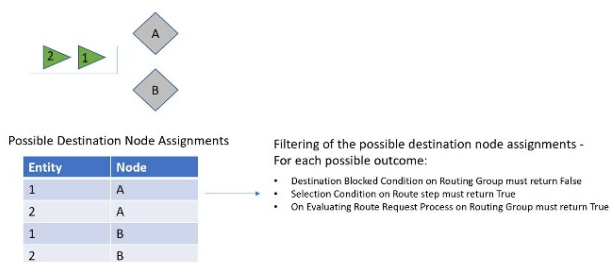
Routing Group – Routing Assignment Algorithm Flow Chart #2



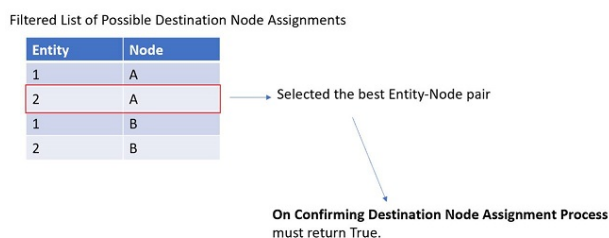
Example 1

In this example, we have a relatively simple case of 2 entities waiting in a RoutingGroup's RouteRequestQueue and there are two possible destinations in the RoutingGroup's destination node list. And some triggering event (built-in or user-defined on the RoutingGroup element) causes an evaluation of the RouteRequestQueue.

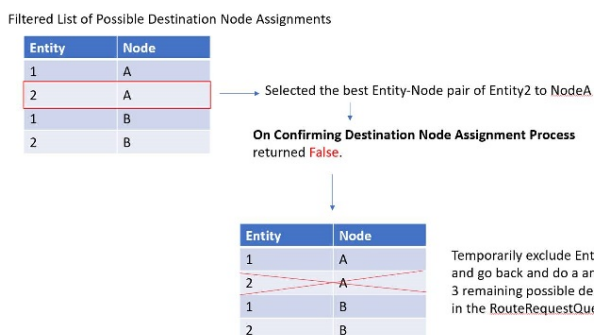
In the table below, there are four (4) possible destination node assignments from which the RoutingGroup can choose. The RoutingGroup does a selection pass using the queue static ranking rule, dynamic selection rule (if specified), and Selection Goal on the Route step to find the best choice of those 4 alternatives.



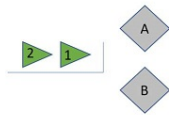
Note that as part of the selection pass, those 4 choices are going to be filtered based on whether a destination node is considered blocked, the *Selection Condition* on the Route step is 'True', and whether the *On Evaluating Route Request* process on the RoutingGroup element returns 'True'. Let's say that after that selection pass, the RoutingGroup has picked Entity2 to NodeA as the best choice. Now what happens is the *On Confirming Destination Assignment* process is executed to give the modeler a chance to 'confirm' that destination assignment. That process must return 'True' for the assignment to happen.



Let's say in this example, the *On Confirming Destination Assignment* process returned 'False'.



The RoutingGroup temporarily excludes that particular Entity-Destination Node pair as a possible choice and goes back to the queue to do an entirely new selection pass (with now just 3 possible choices instead of 4 to select from).



Possible Destination Node Assignments

Entity	Node
1	A
2	A
1	B
2	B

Filtering of the possible destination node assignments -
For each possible outcome:

- Destination Blocked Condition on Routing Group must return False
- Selection Condition on Route step must return True
- On Evaluating Route Request Process on Routing Group must return True

In the second selection pass, the RoutingGroup now selects Entity2 to NodeB as the best choice of those three remaining alternatives and the *On Confirming Destination Assignment* process returns 'True'. Entity2 is removed from the RouteRequestQueue, the destination node assignment is made, and the RoutingGroup's *On Destination Assigned* process is called.

Filtered List of Possible Destination Node Assignments

Entity	Node
1	A
2	A
1	B
2	B

Selected the best Entity-Node pair of Entity2 to NodeB

On Confirming Destination Node Assignment Process
returned **True**.

Entity2.DestinationNode = NodeB
On Destination Node Assigned Process

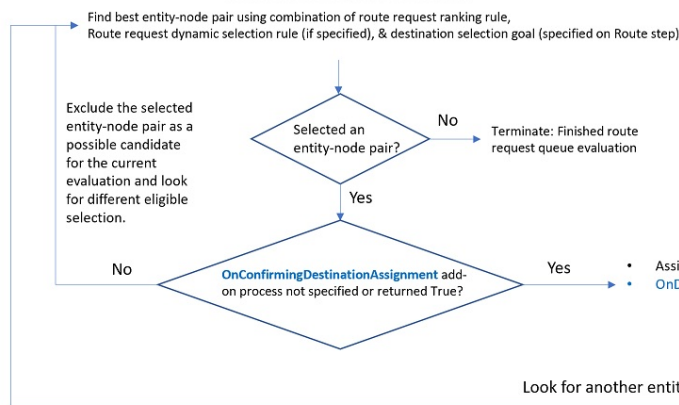
The RoutingGroup keeps checking its RouteRequestQueue using the same selection behavior until it is unable to make any destination node assignment. It then goes back to sleep until woken up again by another triggering event.

Example 2

- Node State Changed (e.g., associated server's resource state changed, server's input buffer or processing contents changed, etc.)
- Required Materials for one or more route requests now available.
- New arrival to route request queue.
- User-Defined Evaluate Route Requests Triggering Event

Schedules late priority current event to evaluate route requests queue

Route Request Queue Evaluation



Eligibility Checks (To be considered for selection, an entity-node pair must satisfy the following conditions)

Conditions checked in this order (returns ineligible result on first failed condition. Trace will tell you that first condition which failed):

- The entity's required materials must be available (if required materials specified on Route step)
- The Destination Blocked Condition specified on Routing Group must return False for the node
- Selection Condition on Route step (if specified) must return True
- Must satisfy any stipulated node list routing/requirement dependencies (Stipulate step)
- *OnEvaluatingRouteRequest* decision process must return True (this check is done last because is typically the most expensive execution so we only get this far if all of the other checks have passed)

Suppose a routing group has two entities waiting in its route request queue with two possible available server destinations that have sequence dependent setups and both the server and job selection rules are intended to be 'Least Setup Time' as shown in Example 2. The destination node list is ordered Server1 then Server2.

The route selection algorithm will select the entities from the route request queue as follows:

-Use the dynamic selection rule to determine that the best entity for Server1 is the blue entity with setup time of 1 hour.

-Use the dynamic selection rule to determine that the best entity for Server2 is the blue entity with setup time of 0 hours.

-Since the best entity (the blue entity) has multiple possible destination nodes available that consider blue entity the best (either Server1 or Server2), then use the destination selection goal to select the server with the least setup time which is Server2.

-The blue entity is routed to Server2 incurring no setup time.

-The red entity is thus routed to Server1 incurring a setup time of 2 hours.

StateStatistic

StateStatistic

A State Statistic element may be used to record and report statistics on a specified state variable. It is added to a model from the Elements panel within the Definitions window.

NOTE: If the state variable is a continuous-change variable, then a trapezoidal integration method is used to calculate the state variable's time-weighted average. Otherwise, a rectangular integration method is used.

NOTE: Runtime confidence interval half-widths are never calculated when running an experiment scenario for multiple replications or when running the model interactively. The Confidence Level property of a StateStatistic element thus only applies if running an experiment scenario for a single replication. For more information, see the Batch Means section of [Experiments](#) page.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For examples of a StateStatistic, see the [StateStatistic - Discussion and Examples](#) page.

Listed below are the properties of **StateStatistic**:

Property	Description
State Variable Name	The name of the state variable for which to record and report statistics.
Data Source	Optional custom value written to the Data Source field in reports and logs, specified as a text string. If defaulted, then the state statistic name is assumed.
Category	Optional custom value written to the Category field in reports and logs, specified as a text string. If defaulted, then the string 'UserSpecified' is assumed.
Data Item	Optional custom value written to the Data Item field in reports and logs, specified as a text string. If defaulted, then the string 'StateValue' is assumed.
Confidence Level	The confidence level used to calculate the confidence interval half-width statistic for the average of the recorded state values. If specified as 'Default', then the confidence level used will be either 'None' if running the model interactively, or will be the Experiment's specified confidence level if running an experiment scenario for a single replication.
Stopping Condition	Logical expression that is evaluated each time a discrete change of the state variable's value or parameters has occurred. The simulation run is ended if the expression evaluates to true.
Log Observations	Indicates whether discrete changes of the state variable's value or parameters are to be automatically logged. Go to Results or Planning -> Logs -> State Observation Log to view the logged data.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

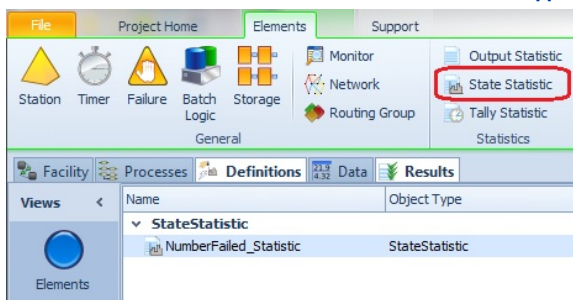
StateStatistic - Discussion and Examples

StateStatistic Example 1

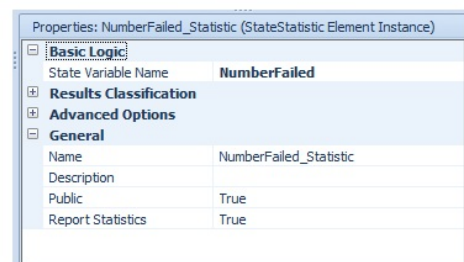
The following is an example of a [StateStatistic](#) element.

- Since its *State Variable Name* property is set to 'NumberFailed', it will collect statistics on the state variable called NumberFailed.
- The *Category* property determines what will appear in the Pivot table of the Results Window in the Category field. In this example, it will show the value of 'User'. The user can put any word into this field in order to add custom organization of the Pivot Table.
- The *Data Item* property is set to 'General' which indicates that in the Pivot Table of the Results Window, the word 'General' will appear in the Data Item category. The user can put any word into this field in order to add custom organization of the Pivot Table.

StateStatistic Element and How it Appears in the Results Window



Add a State Statistic to the Definitions Window, Elements Panel



Properties Window of the State Statistic

The screenshot shows the 'Results' window with the 'Pivot Grid' view. The grid displays statistics for the 'NumberFailed_Statistic' element. The 'Object Type' is 'Model', 'Object Name' is 'Model', 'Data Source' is 'NumberFailed_Statistic', 'Category' is 'UserSpecified', and 'Data Item' is 'StateValue'. The 'Statistic' column shows 'Average', 'FinalValue', and 'Maximum'. The 'Average Total' column shows values 470.3595, 930.0000, and 930.0000 respectively.

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Model	Model	NumberFailed_Statistic	UserSpecified	StateValue	Average	470.3595
					FinalValue	930.0000
					Maximum	930.0000

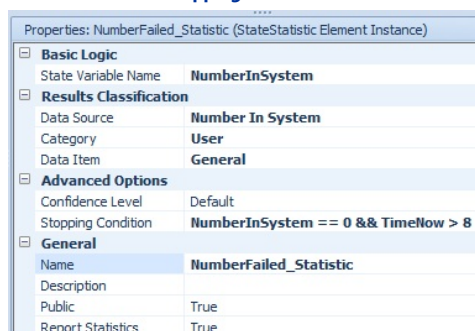
Results Window, Pivot Grid

StateStatistic Example 2

The following is an example of a [StateStatistic](#) element that is also being used to stop the simulation run.

- Since its *State Variable Name* property is set to 'NumberInSystem', it will collect statistics on the state variable called NumberInSystem.
- The *Stopping Condition* property is evaluated each time a value is recorded for the statistic NumberInSystem. When it is evaluated, if the expression evaluates to 'True' then the simulation run is ended. In this case, the simulation run ends whenever the NumberInSystem equals 0 and the simulation has run for at least 8 hours.

StateStatistic Element with Stopping Condition to End the Simulation Run



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Station

Station

The Station element may be used to define a capacity constrained location for holding entities representing discrete items. The [Transfer step](#) is used to transfer entities into and out of a station. The [EndTransfer](#) step should be used to end the entity transfer into a Station. A Station element is added to a model from the Elements panel within the Definitions window.

If the *Balk Entering Decision Type* is 'Blocked', then an entity's transfer attempt into the station will be allowed to proceed as normal. This includes being redirected to another station or to a parent external node if the station's capacity is zero and a redirect location has been specified. However, a late priority event will also be scheduled on the current events calendar for the entity. When that event is processed, the following conditions for the entity will be checked:

- a) Was the entity's Transferring event fired?
- b) Is there a queue item associated with the entity in a station's entry queue, link's entry queue, node's entry queue, or a routing group's route request queue?

If the answer to both of the above questions is true, then the entity is determined to have immediately attempted a discrete transfer that was blocked and it will balk. The queue item associated with the entity's blocked transfer attempt will be automatically cancelled (reneged) and the station's *On Balked Entering Process* then executed.

Station elements are built into some of the Simio Standard Library objects. Some examples are: `Server.InputBuffer`, `Server.Processing`, `Server.OutputBuffer`, `Combiner.ParentInputBuffer`, `Separator.MemberOutputBuffer`, `Vehicle.RideStation`, etc.

The contents of a station queue can be accessed by utilizing the 'Contents' state of the station. For example, `'Server1.InputBuffer.Contents'` displays the contents of the queue at the `Server1 InputBuffer` station. See the [Functions in Simio - Automatic](#) page for various functions that can be accessed for queues.

The Station element's Contents queue also has several read-only parameters (not for all queues, only Station content queues). This includes `Volume`, `Volume.Rate`, `Weight` and `Weight.Rate`. These parameters represent the cumulative `Weight` and `Volume` of the contents of the queue. Since they are parameters, they can also be used in a Monitor, so you can see when the volume of objects in a given station exceeds a specified value. As an example, `'Server1.OutputBuffer.Contents.Weight'` would provide the cumulative weight of all the entities waiting in the output buffer of `Server1`.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For additional information on reneging, see the [Station - Discussions and Examples](#) page.

Listed below are the Properties of **Station**:

Property	Description
Initial Capacity	The initial maximum number of entities that can be simultaneously located in the station.
Entry Ranking Rule	The static rule used to rank entry into this station among competing entities.
Entry Ranking Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Entry Dynamic Selection Rule	Indicates whether this station, when capacity to enter the station becomes available, dynamically selects the next station entry from its statically ranked entry queue using a dynamic selection rule. See the page regarding Dynamic Selection Rules for more information.
Repeat Group	True, False

Indicates whether a repeat group data structure is used to define the primary

		dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax <code>Candidate.[EntityClass].[Attribute]</code> to reference an attribute of the candidate entities (e.g., <code>Candidate.Entity.Priority</code>).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.
Parent Cost Center	The cost center that costs allocated to entities located in this station are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this station.	
Cost Per Use	The cost for an entity to occupy this station location irrespective of the time spent in the station.	
Holding Cost Rate	The cost per unit time to hold an entity in this station location.	
Balk Entering Decision Type	The method used by an arriving entity to decide whether to balk at entering the station. If the decision type is 'Blocked', then an entity attempting to enter the station will balk if the station is full rather than be blocked. Or, if a zero capacity station, the entity will balk if it immediately attempts a transfer to or from a redirected location and that transfer attempt is blocked.	
Balk Condition Or Probability	The balk condition or probability specified as an expression. If a probability then enter the chance	

	of balking as a value between 0.0 (0%) and 1.0 (100%).
On Balked Entering Process	Optional process that is executed if an entity has balked at entering the station.
Redirect Location Type	Indicates the type of location that an arriving entity will be automatically redirected to if the capacity of this station is zero.
Redirect Station Name	The name of the station that an arriving entity will be automatically redirected to if the capacity of this station is zero.
Redirect External Node Name	The name of the parent external node that an arriving entity will be automatically redirected to if the capacity of this station is zero.
Contents Ranking Rule	The static ranking rule used to rank the station's 'Contents' queue of entity objects currently located in the station. This Ranking Rule for the InputBuffer station of the standard Server, Combiner, Separator and Workstation (Deprecated) match the resource allocation and batching ranking rules (in the Combiner case) that the user specifies in the properties of these objects.
Renege Triggers	Optional time or event-driven triggers that can cause entities to abandon waiting in the station.
Trigger Type.Renege Triggers	The type of trigger. A time based trigger can cause an entity waiting in the station to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the station to renege if a specific event occurs.
Wait Duration.Renege Triggers	The tolerable wait duration until a renege decision by an entity waiting in the station.
Triggering Event Name.Renege Triggers	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the station.
Renege Decision Type.Renege Triggers	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the station.
Renege Condition Or Probability.Renege Triggers	The renege condition or probability specified as an expression. If a probability then enter the chance of reneging as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).

On Reneged Process.Renege Triggers	Optional process that is executed if an entity reneges.
Trigger Start Boundary.Renege Triggers	Indicates when the trigger becomes active for an entity occupying the station. Can be either immediately when the entity has entered the station or, alternatively, not until its transfer into the station has been ended (e.g., after a transfer-in time or any other entry related logic).
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Listed below are the States of **Station**:

State	Description
TransferInState	State indicating whether an entity is currently transferring into the station (Idle = 0, Busy = 1).
Contents	The queue of entities currently located in the station.
EntryQueue	The queue of entities waiting for capacity to enter the station.
CurrentCapacity	The current capacity of the station.

Listed below are the Events of **Station**:

Event	Description
Entered	The Entered event provides notification that an entity has entered the station.
Exited	The Exited event provides notification that an entity has exited the station.
CapacityChanged	The CapacityChanged event provides notification that the capacity of the station has changed.

Listed below are the Functions of **Station**:

Function	Description
NumberArrivals	Returns the number of entities that have arrived at this station.
NumberDepartures	Returns the number of entities that have departed from this station.
NumberTransferringOut	Returns the current number of entities attempting to transfer out of the station.
NumberBalked	Returns the total number of entities that balked at entering the station.
NumberReneged	Returns the total number of entities that have abandoned waiting in the station.
Capacity	Returns the current capacity of the station.
Capacity.Initial	Returns the initial capacity of the station.
Capacity.Previous	Returns the previous capacity of the station.
Capacity.Remaining	Returns the current capacity that is remaining at this station.

Capacity.Minimum	Returns the minimum capacity of this station during the run.
------------------	--

Capacity.Maximum	Returns the maximum capacity of this station during the run.
------------------	--

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Station - Discussion and Examples

Reneging

For performance optimization reasons, the Station element will initialize the Renege Triggers repeat group once on run initialization, subscribing to any triggering events. The same set of renege triggers will then apply to every entity that enters the station.

If a time-based renege trigger has been specified, the trigger's *Wait Duration* expression will be evaluated for an entity and a late priority event then scheduled on the event calendar to make a renege decision after the specified *Wait Duration* has elapsed. The calendar item will be created and scheduled either when the entity enters the station or when its transfer into the station has been ended, depending on the *Trigger Start Boundary* setting.

If an event-based renege trigger has been specified, then any occurrence of the specified *Triggering Event Name* will cause the station to evaluate the entities in its contents queue in queue rank order for possible renege decisions. The trigger will be considered active for an entity either when it enters the station or when its transfer into the station has been ended, depending on the *Trigger Start Boundary* setting. Once the evaluation of all entities in the station's contents queue has been completed, an early priority event to handle any entities that decided to renege will be scheduled on the current events calendar.

An entity that is occupying the station but which has already begun to transfer out to a new location, or which has successfully reserved a transporter to eventually transfer out, will not be renegable and thus ignored by any renege trigger.

If a conditional or probabilistic renege decision is being evaluated, the *Renege Condition Or Probability* expression is evaluated in the context of the entity.

For each reneging entity, the station will first automatically remove and dispose of any queue items associated with the entity that can be removed using the Remove step approach, excluding any Storage element queue items. This action will automatically cancel the entity's queuing for any constraints such as resources, materials, a blocked destination, etc. Additionally, if the parent object containing the station is a resource that is owned by the entity then that resource will be automatically released.

Finally, the *On Reneged Process* (if specified) will be executed by a token that is associated with the reneged entity, to allow user-defined process logic to complete its handling, such as destroying it or transferring it to another location.

Station Example 1

The following is an example of a [Station](#) element that uses a Dynamic Selection Rule to select which entity can enter in to this Station next.

- Since its *Dynamic Selection Rule* property is set to 'Largest Value First', if this Stations capacity becomes available and there are entities in the Entry Queue, this Station will select entities from the Entry Queue based on the rule of Largest Value First.
- The *Value Expression* property is set to 'Candidate.Entity.Priority', so the Station will select the entity with the largest value for the state ModelEntity.Priority.
- The *Filter Expression* property is set to 'Candidate.ModelEntity.PartType == 2' so it will only look to select entities that meet this criteria.

Properties Window of a Station Element

Properties: Station1 (Station Element Instance)

Basic Logic	
Initial Capacity	Infinity
Financials	
Advanced Options	
Entry Ranking Rule	FirstInFirstOut
Dynamic Selection...	Largest Value First
Value Expression	Candidate.Entity.Priority
Filter Expression	Candidate.ModelEntity.PartType == 2
Contents Ranking Rule	FirstInFirstOut
Redirect Station	
General	
Name	Station1
Description	
Public	True
Report Statistics	True

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Storage

Storage

The Storage element may be used to define a queue for temporarily holding one or more objects. A Storage element is added to a model from the Elements panel in the Definitions window. The Insert and Remove steps are used to insert and remove objects from the Storage element's queue. Entities in a storage are an **image** of the entity that is actually located elsewhere in the model. There are [functions](#) that can be used to get information about a Queue State.

A queue can be [animated](#) from the Drawing Tab by selecting Detached Queue.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **Storage**:

Property	Description
Report Statistics	Specifies if statistics are to be automatically reported for this element.
Ranking Rule	The rule used to rank the queue holding the objects in the storage.
Ranking Expression	The expression used with a Ranking Rule of Smallest Value First or Largest Value First, to rank the queue holding the objects in the storage.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

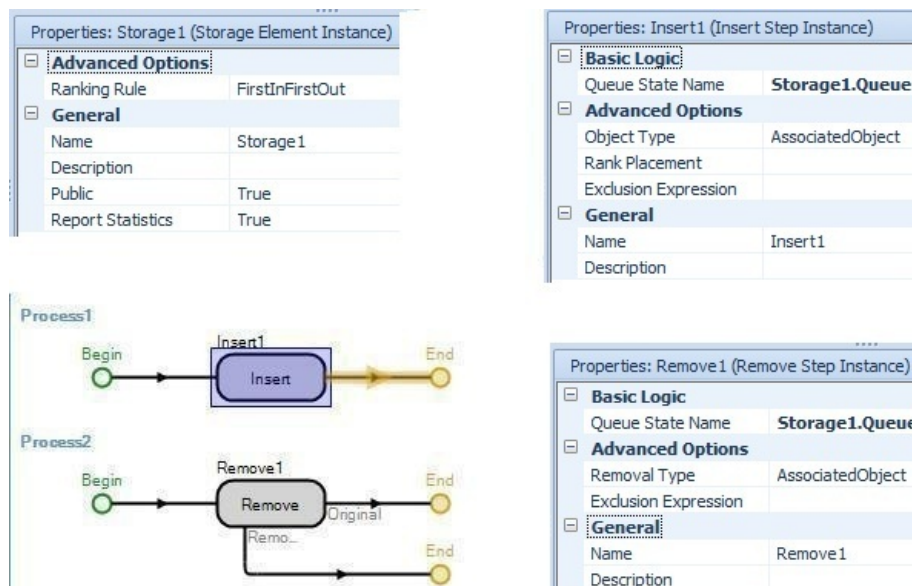
Storage - Discussion and Examples

Storage Example 1

The following is an example of a basic [Storage](#) element.

- Since its *Ranking Rule* property is set to 'FirstInFirstOut', the first entity that enters this queue will be the first to leave, when a Remove step is used to take entities out of the queue.
- An [Insert Step](#) is used to add entities to the Storage queue. In this example, an entity is the associated object of the token that is executing the process, so the *Object Type* property of the Insert Step is set to 'AssociatedObject'.
- Since *Rank Placement* property is blank, the entities simply follow the static ranking rule of FirstInFirstOut to determine their placement in the queue.
- A [Remove Step](#) takes the entity out of the queue. An entity also leaves the queue if it has been destroyed elsewhere in the system.

Storage Element and Related Steps



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

StorageArea

Storage Area

A Storage Area element defines a logical grouping of material storage areas or storage bins. The element can be used in conjunction with the ReserveBins and ReserveStock steps for modeling stock putaway and removal processes. The contents of a storage area can be animated using the Stack animation. For more information, see [Status Labels, Plots, Gauges and Buttons](#).

For more information about using storage areas for material storage processes, see [Material Storage – Discussion and Examples](#).

Listed below are the properties of **Storage Area Element**:

Property	Description
Child Storage Locations	The child storage locations of the storage area.
Child Storage Locations.Child Storage Location Type	The storage location type.
Child Storage Locations.Child Storage Area Name	The name of the storage area.
Child Storage Locations.Child Storage Bin Name	The name of the storage bin.

Listed below are the properties of **Function Area Element**:

Function Name	Description
CurrentVolumeCapacity	The current maximum volume of materials that can be present in the storage area. NOTE: Refer to the Storage Bin element to define a capacity constrained storage location in a storage area.
CurrentWeightCapacity	The current maximum weight of materials that can be present in the storage area. NOTE: Refer to the Storage Bin element to define a capacity constrained storage location in a storage area.
CurrentTotalCapacity	The current total capacity of the storage area. NOTE: Refer to the Storage Bin element to define a capacity constrained storage location in a storage area.
CurrentVolumeCapacityAvailable	The current available volume capacity of the storage area. $\text{CurrentVolumeCapacityAvailable} = \text{CurrentVolumeCapacity} - \text{CurrentStock.Volume} - \text{IncomingStock.Volume}$
CurrentWeightCapacityAvailable	The current available weight capacity of the storage area. $\text{CurrentWeightCapacityAvailable} = \text{CurrentWeightCapacity} - \text{CurrentStock.Weight} - \text{IncomingStock.Weight}$
CurrentTotalCapacityAvailable	The current available total capacity of the storage area. $\text{CurrentTotalCapacityAvailable} = \text{CurrentTotalCapacity} - \text{CurrentStock.CapacityUsage} - \text{IncomingStock.CapacityUsage}$
CurrentStock.Volume	The total volume of the materials currently present in the storage area.
CurrentStock.Weight	The total weight of the materials currently present in the storage area.
CurrentStock.CapacityUsage	The total capacity usage of the materials currently present in the

	storage area.
CurrentStock.QuantityInStock(material)	The current quantity of the specified material that is present in the storage area.
CurrentStock.QuantityReserved(material)	The current reserved quantity of the specified material that is present in the storage area.
CurrentStock.QuantityAvailable(material)	The current unreserved quantity of the specified material that is present in the storage area. $\text{QuantityAvailable} = \text{QuantityInStock} - \text{QuantityReserved}$
CurrentStock.Quants.NumberItems	The number of stock quants currently present in the storage area.
CurrentStock.Quants.FirstItem	The first quant in the first-in-first-out list of stock quants currently present in the storage area.
CurrentStock.Quants.LastItem	The last quant in the first-in-first-out list of stock quants currently present in the storage area.
CurrentStock.Quants.IndexOfItem(quant)	Returns the one-based index of the specified quant in the first-in-first-out list of stock quants currently present in the storage area, if found. Otherwise, returns 0.
CurrentStock.Quants.ItemAtIndex(index)	Returns the quant at the specified one-based index position in the first-in-first-out list of stock quants currently present in the storage area.
CurrentStock.Quants.Contains(quant)	Returns True (1) if the first-in-first-out list of stock quants currently present in the storage area contains the specified quant. Otherwise, returns False (0).
IncomingStock.Volume	The total volume of current incoming materials to the storage area.
IncomingStock.Weight	The total weight of current incoming materials to the storage area.
IncomingStock.CapacityUsage	The total capacity usage of current incoming materials to the storage area.
IncomingStock.Quantity(material)	The current incoming quantity of the specified material to the storage area.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

StorageBin

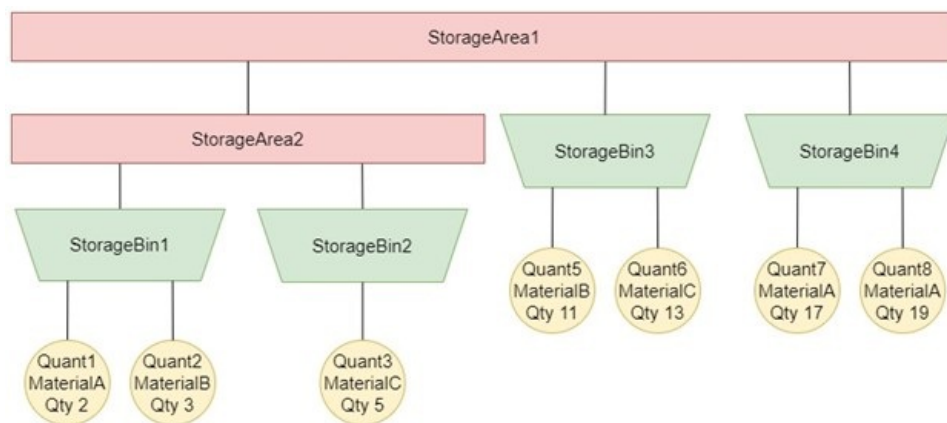
Storage Bin

A Storage Bin element defines a capacity-constrained storage location for holding stock of one or more material types. Material can be added to a storage bin using the AddStock step and removed from a storage bin using the RemoveStock step. The amount of material stock in a storage bin is concurrently constrained by the storage bin's CurrentVolumeCapacity, CurrentWeightCapacity, and CurrentTotalCapacity, corresponding to the Material element's Volume Per Unit, Weight Per Unit, and Capacity Use Per Unit properties. Volume and weight capacity correspond to the maximum volume and weight of stock that the storage bin can hold at once. Total capacity represents a generic storage bin capacity, which could represent a physical quantity or any other attribute. The contents of a storage bin can be animated using the Stack animation. For more information, see [Status Labels, Plots, Gauges and Buttons](#).

For more information about modeling material storage, see [Material Storage – Discussion and Examples](#).

Stock Quants

Material stock in a storage bin is tracked using dynamically created stock quants. Each stock quant represents stock of a specific material type with the same characteristics stored in the same storage bin. For example, in the storage structure pictured below, StorageBin1 contains two quants, Quant1 and Quant2. Quant1 contains 2 units of MaterialA, and Quant2 contains 3 units of MaterialB. A single storage bin might contain two or more quants of the same material, but a single quant cannot contain stock of multiple material types. For instance, StorageBin4 contains Quant7 and Quant8, both of which contain units of MaterialA, while no quant contains multiple material types.



Information about the quants within a storage bin can be accessed using the storage bin's 'CurrentStock.Quants' function. For more information about quants, see [Stock Quant](#).

Listed below are the properties of **Storage Bin Element**:

Property Name	Description
Initial Volume Capacity	The initial maximum volume of materials that can be present in the storage bin. A material's volume per unit may be specified on the Material element. NOTE: Refer to the user-assignable 'CurrentVolumeCapacity' state variable of a storage bin to dynamically change its volume capacity during a simulation run.
Initial Weight Capacity	The initial maximum weight of materials that can be present in the storage bin. A material's weight per unit may be specified on the Material element. NOTE: Refer to the user-assignable 'CurrentWeightCapacity' state variable of a storage bin to dynamically change its weight capacity during a simulation run.
Initial Total Capacity	The initial total capacity of the storage bin. A material's capacity usage per unit may be specified on the Material element. NOTE: Refer to the user-assignable 'CurrentTotalCapacity' state variable of a storage bin to dynamically change its total capacity during a simulation run.

Listed below are the state variables of **Storage Bin Element**:

State Variable Name	Description
CurrentVolumeCapacity	The current maximum volume of materials that can be present in the storage bin.
CurrentWeightCapacity	The current maximum weight of materials that can be present in the storage bin.
CurrentTotalCapacity	The current total capacity of the storage bin.

Listed below are the functions of **Storage Bin Element**:

Function Name	Description
CurrentVolumeCapacityAvailable	The current available volume capacity of the storage bin. $\text{CurrentVolumeCapacityAvailable} = \text{CurrentVolumeCapacity} - \text{CurrentStock.Volume} - \text{IncomingStock.Volume}$
CurrentWeightCapacityAvailable	The current available weight capacity of the storage bin. $\text{CurrentWeightCapacityAvailable} = \text{CurrentWeightCapacity} - \text{CurrentStock.Weight} - \text{IncomingStock.Weight}$
CurrentTotalCapacityAvailable	The current available total capacity of the storage bin. $\text{CurrentTotalCapacityAvailable} = \text{CurrentTotalCapacity} - \text{CurrentStock.CapacityUsage} - \text{IncomingStock.CapacityUsage}$
CurrentStock.Volume	The total volume of the materials currently present in the storage bin.
CurrentStock.Weight	The total weight of the materials currently present in the storage bin.
CurrentStock.CapacityUsage	The total capacity usage of the materials currently present in the storage bin.
CurrentStock.QuantityInStock(material)	The current quantity of the specified material that is present in the storage bin.
CurrentStock.QuantityReserved(material)	The current reserved quantity of the specified material that is present in the storage bin.
CurrentStock.QuantityAvailable(material)	The current unreserved quantity of the specified material that is present in the storage bin. $\text{QuantityAvailable} = \text{QuantityInStock} - \text{QuantityReserved}$
CurrentStock.Quants.NumberItems	The number of stock quants currently present in the storage bin.
CurrentStock.Quants.FirstItem	The first quant in the first-in-first-out list of stock quants currently present in the storage bin.
CurrentStock.Quants.LastItem	The last quant in the first-in-first-out list of stock quants currently present in the storage bin.
CurrentStock.Quants.IndexOfItem(quant)	Returns the one-based index of the specified quant in the first-in-first-out list of stock quants currently present in the storage bin, if found. Otherwise, returns 0.
CurrentStock.Quants.Contains(quant)	Returns True (1) if the first-in-first-out list of stock quants currently present in the storage bin contains the specified quant. Otherwise, returns False (0).

CurrentStock.Quants.ItemAtIndex(index)	Returns the quant at the specified one-based index position in the first-in-first-out list of stock quants currently present in the storage bin.
IncomingStock.Volume	The total volume of current incoming materials to the storage bin.
IncomingStock.Weight	The total weight of current incoming materials to the storage bin.
IncomingStock.CapacityUsage	The total capacity usage of current incoming materials to the storage bin.
IncomingStock.Quantity(material)	The current incoming quantity of the specified material to the storage bin.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

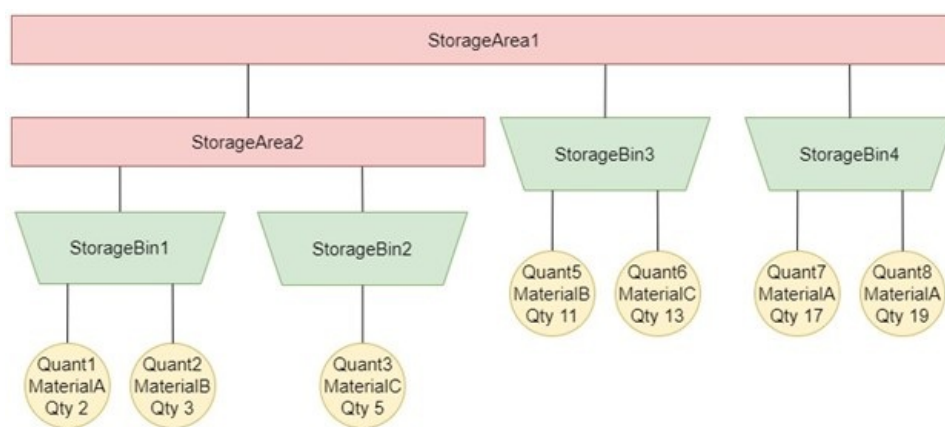
Send comments on this topic to [Support](#)

StockQuant

Stock Quant

A stock quant, often referred to as a quant, represents a group of stock of a specific material type with the same characteristics stored in the same storage bin. Quants are automatically created and destroyed, and thus cannot be manually instantiated by the user.

Quants are used to track the material stock within storage bins, allowing multiple materials to be stored in the same storage bin concurrently. For example, in the storage structure pictured below, StorageBin1 contains two quants, Quant1 and Quant2. Quant1 contains 2 units of MaterialA, and Quant2 contains 3 units of MaterialB. A single quant cannot contain two different materials, but a single storage bin could also contain two quants of the same material. For instance, StorageBin4 contains Quant7 and Quant8, both of which contain units of MaterialA.



A quant tracks multiple quantities for the stored material: IncomingQuantity, QuantityInStock, QuantityReserved, and QuantityAvailable. These quantities can be accessed using the quant function of the same name. When stock is added to a storage bin using the AddStock step, it will either be added to an existing quant, or a new quant will be created and added to the storage bin. When stock is removed from a storage bin using the RemoveStock step, it will be removed from an existing quant. When stock contained by a quant is reserved using the ReserveStock step, the quant's QuantityAvailable is decreased by the reservation quantity, and the quant's QuantityReserved is increased by the reservation quantity. When capacity of a storage bin is reserved using the ReserveBins step, the quant's IncomingQuantity is increased by the reservation quantity.

Quants are not static and are created and destroyed dynamically as material is added to and removed from a storage bin. Quants are created as necessary when material is added to a storage bin that does not have an existing matching quant and destroyed when the quant's QuantityInStock and IncomingQuantity are '0'.

A quant is automatically removed from a storage bin if both its QuantityInStock and IncomingStock are zero. The removed quant is then automatically destroyed when it no longer has state variables (if any) referencing it. For example, if a removed quant has one or more state variables referencing it then that quant is automatically destroyed once those state variables have all been cleared or assigned different values.

Listed below are the functions of the **Stock Quant**:

Function Name	Description
IDNumber	Unique ID number for the stock quant.
StorageBin	The quant's storage bin.
Material	The quant's material type.
QuantityInStock	The current quantity of the material that is present in the quant.
QuantityReserved	The current reserved quantity of the material that is present in the quant.

QuantityAvailable	The current unreserved quantity of the material that is present in the quant.
-------------------	---

IncomingQuantity	The current incoming quantity of the material to the quant.
------------------	---

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

TallyStatistic

TallyStatistic

A Tally Statistic element may be used in conjunction with the Tally step to record and report statistics on a set of observed values. It is added to a model from the Elements panel within the Definitions window.

The TallyStatistic element is referenced by the Tally step which specifies a value to be tallied. A single TallyStatistic element may be referenced by multiple Tally Steps.

NOTE: Runtime confidence interval half-widths are never calculated when running an experiment scenario for multiple replications or when running the model interactively. The Confidence Level property of a TallyStatistic element thus only applies if running an experiment scenario for a single replication. For more information, see the Batch Means section of [Experiments](#) page.

For additional information, including functions for tally statistics, examples and logging observations, see the [TallyStatistic - Discussion and Examples](#) page.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

Listed below are the properties of **TallyStatistic**:

Property	Description
Unit Type	The unit type of the observation values to record.
Data Source	Optional custom value written to the Data Source field in reports and logs, specified as a text string. If defaulted, then the tally statistic name is assumed.
Category	Optional custom value written to the Category field in reports and logs, specified as a text string. If defaulted, then the string 'UserSpecified' is assumed.
Data Item	Optional custom value written to the Data Item field in reports and logs, specified as a text string. If defaulted, then the string 'TallyValue' is assumed.
Confidence Level	The confidence level used to calculate a confidence interval half-width statistic of the recorded observations. If specified as 'Default', then the confidence level used will be either 'None' if running the model interactively, or will be the Experiment's specified confidence level if running an experiment scenario for a single replication.
Stopping Condition	Logical expression that is evaluated each time an observation value is recorded for the tally statistic. The simulation run is ended if the expression evaluates to true.
Log Observations	Indicates whether the observation values recorded for the tally statistic are to be automatically logged. Go to Results or Planning -> Logs -> Tally Observation Log to view the logged data.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TallyStatistic - Discussion and Examples

Discussion

The following functions are available for a TallyStatistic and can be used in an Expression within your model logic or displayed in a Status Label or Floor Label in the Facility window (see [Functions](#) for information on what value they return):

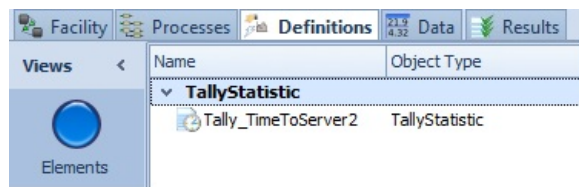
- TallyStatistic.Average
- TallyStatistic.HalfWidth
- TallyStatistic.LastRecordedValue
- TallyStatistic.Maximum
- TallyStatistic.Minimum
- TallyStatistic.NumberObservations

TallyStatistic Example 1

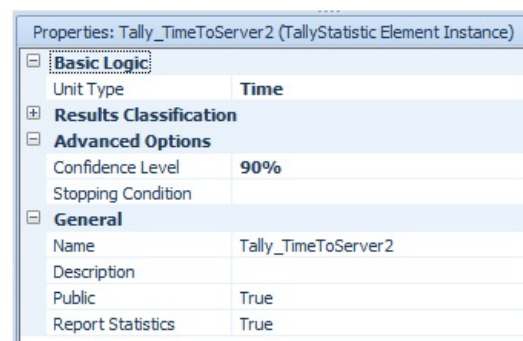
The following is an example of a [TallyStatistic](#) element that tallies the time it takes for an entity to arrive at Server2.

- The TallyStatistic is created in the Elements panel within the Definitions window. It is simply given a name, an optional label, category and data item in its properties window. In this example, the *Confidence Level* property is set to 90%, which will determine how the Half Width is calculated in the results.
- In order to use a TallyStatistic, a Tally Step must be inserted into the logic of the model. In this example, the Tally Step is added to a process that is triggered by the Entered Add-On process trigger of the Input Node of Server2.
- The *Value* property is set to the expression 'Run.TimeNow - ModelEntity.TimeCreated'. Run.TimeNow is the current simulation time and ModelEntity.TimeCreated is a Simio function that returns the simulation time when this entity was first created. The result of this expression is recorded into the TallyStatistic each time this Tally Step is executed.

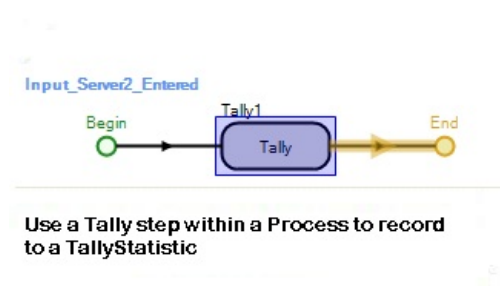
Using a TallyStatistic Element



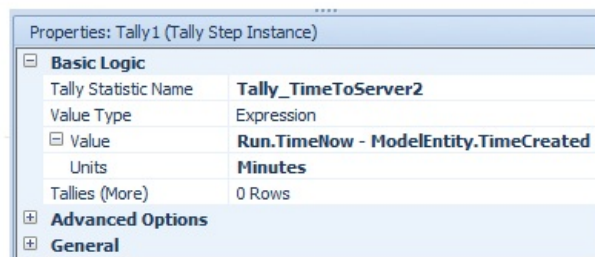
Define the TallyStatistic from the Elements panel in the Definitions window



Properties Window of the TallyStatistic



Use a Tally step within a Process to record to a TallyStatistic



TallyStatistic, in the Properties window of the Tally step

Send comments on this topic to [Support](#)

TaskSequence

TaskSequence

A Task Sequence element is used to define and execute structured sequences of processing tasks. A [StartTasks](#) step references a Task Sequence element and is used to start a task sequence that is associated with an object in the system.

The Task Sequence Number Scheme

The task sequence numbering scheme supported by the Task Sequence element has the following format:

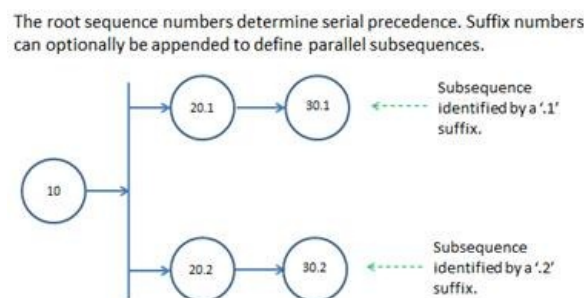
XX.YY.YY.YY...

XX is an integer referred to as the root sequence number. This number is the primary key that determines the serial precedence requirements of a task.

YY are optional integer suffixes separated by dots (periods) that can be appended to the root sequence number in order to define parallel subsequences in the task workflow. As many suffixes as needed may be appended to a root number to represent the desired task sequence structure accurately.

The precedence rules used to determine task order are as follows:

1. If the task sequence number consists of only a root number (only XX), then all tasks with lower root numbers (ignoring any suffixes) must precede that task. Note: It is possible to have one or more tasks with the same root sequence number, with or without suffixes.
2. If the task sequence number consists of a root number and one or more suffixes, then all tasks with lower root numbers must precede that task, as long as those tasks either have no suffixes or all shared suffixes match.
3. If the task sequence number consists of a root number and one or more suffixes, then all tasks whose sequence number is the same root number without any suffixes must precede that task.



The Task ID Number Scheme for Predecessors or Successors

An alternative to the task sequence numbering scheme described above is to use the Task Precedence Method of either 'ImmediatePredecessorsMethod' or 'ImmediateSuccessorsMethod'. With these methods, either the immediate predecessors for each task, or the immediate successors for each task may be specified. This is done through the use of the ID Number property instead of the Sequence Number property, as well as the Immediate Predecessors or Immediate Successors properties. A list of immediate predecessor ID numbers which must be finished (or cancelled) or a list of immediate successor ID numbers which cannot start until the task is finished (or cancelled) may be specified.

When using the Task ID method, a runtime error will occur if any of the following conditions are detected when initializing the start of the task sequence:

1. The integer ID numbers specified for the tasks are not unique.
2. A task ID number is specified more than once as an immediate predecessor or successor for another task.
3. Task predecessor or successor dependencies are specified such that there is a loop back to a previous task, which is not allowed.

Conditional And Probabilistic Branching

The Task Sequence element allows conditional or probabilistic branching to be included in the modeled workflow structure. Such branching may be required for example if you have scenarios where there are two or more possible tasks that may succeed another task, and a decision will have to be made at runtime regarding which branch in the workflow to take.

To indicate that a task in the task list is to be treated as the start of a conditional or probabilistic branch in the workflow, you specify the task's *Branch Type* property as either 'Conditional', 'Probabilistic' or 'Independent Probabilistic' and then enter the appropriate logical condition or probability value.

When executing a task sequence during a simulation run, the decision to conditionally or probabilistically select a task (branch) is made when all precedence dependencies for the task have been satisfied and it is time to either start or cancel the task. If the *Branch Type* is 'Conditional', then the task is performed if the specified logical condition evaluates to True. Otherwise, the task is cancelled. The 'Probabilistic' *Branch Type* may be used to identify the task as one of possibly several mutually exclusive alternatives at a probabilistic decision point. The sum of the branch probabilities at a probabilistic decision point cannot be greater than one. In the case of a 'Probabilistic' branch, a single task will be probabilistically selected from the set of candidate tasks (branches) that share the same immediate predecessor dependencies. If the *Branch Type* is 'Independent Probabilistic', then the task's specified probability is independent of and thus not affected by other probabilistic branching. For example, if a decision point has two task branches each with an independent probability of 0.5 (50% chance), then that indicates either, both or neither branches might be required to occur. And the probability of both branches being required is $(0.5) \times (0.5) = 0.25$ (25% chance).

Whenever a task is cancelled due to a conditional or probabilistic decision, if that action has cancelled all immediate predecessors of another unstarted task then that task may be likewise automatically cancelled by specifying its *Auto Cancel Trigger* property as 'AllImmediatePredecessorsCancelled'. For example, cancelling the first task in a sequence of serially performed tasks can automatically cancel all of the successor tasks in that branch of the workflow.

Loopback Branches

Multiple loopback branches can be specified for a given task. A loopback branch simply allows the user to specify a conditional or probabilistic method of looping an entity to another area of a task sequence. A loopback branch evaluation is triggered when all tasks with the specified *From Task Sequence Number* or *From Task ID Number* have finished or been cancelled. In the case where the tasks were all cancelled, then the loopback branch evaluation is cancelled.

Because looping back is a 'roll back' of the task sequence execution, loopback branch evaluations are always triggered first before successor branch evaluations. A probabilistic decision point in the task sequence selects from a set of mutually exclusive loopback and/or successor branches. If a loopback branch is selected, then all tasks with the specified *To Task Sequence Number* or *To Task ID Number* must be finished or cancelled. Otherwise, a runtime error occurs. A runtime error will also occur if any successor of those tasks in the precedence graph is currently executing, to avoid creating a task dependency violation.

A loopback changes the state of all tasks with the specified *To Task Sequence Number* or *To Task ID Number*, as well as the state of all transitive successors in the loop, to NotReady. The tasks with the specified *To Task Sequence Number* or *To Task ID Number* are then started, resuming normal execution of the task sequence.

General Information

There are a number of token functions available when a token's process execution is part of a task sequence. These functions for accessing information about the task assigned to the token can be found within the [Tokens](#) page.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For examples of the Task Sequence element, see the [Task Sequence - Discussion and Examples](#) page. A SimBit is available that includes multiple model examples, please refer to the [ServerUsingTaskSequence](#) SimBit.

Listed below are the properties of **Task Sequence**:

Property	Description
Task Precedence Method	The method used to define the task precedence dependencies. Can be either by specifying task sequence numbers, specifying the immediate predecessors for each task, or specifying the immediate successors for each task.
Tasks	The set of processing tasks to be performed.
Tasks.Sequence Number	Sequence Number used to define the task precedence constraints. To model a task sequence that is simply a serially ordered set of tasks, define the tasks with increasing sequence numbers (e.g., the first task is numbered '10', the second task is numbered '20', and so forth). Shown if the <i>Task Precedence Method</i> is 'SequenceNumberMethod'.

Tasks.ID Number	Integer identifier number for the task. Shown if the Task Precedence Method is 'ImmediatePredecessorsMethod' or 'ImmediateSuccessorsMethod'.
Tasks.Name	The name of this task.
Tasks.Branch Type	Indicates whether the task is always performed, or whether it is instead treated as the first task of a conditional or probabilistic branch in the process workflow.
Tasks.Condition or Probability	The branch condition or probability specified as an expression. If a condition, then enter the logic condition. If a probability, then enter the chance of selecting the task as a value between 0.0 (0%) and 1.0 (100%).
Tasks.Process Name	The name of the process that will be executed in order to perform the action(s) required for the task. If this property is left blank, the task will be ignored and will not be executed.
Tasks.Immediate Predecessors	Lists the ID numbers of the tasks which must be finished or cancelled before this task can start. To specify more than one predecessor, enter multiple task ID numbers separated by commas. Shown when the <i>Task Precedence Method</i> is 'ImmediatePredecessorsMethod'.
Tasks.Immediate Successors	Lists the ID numbers of the tasks which can't start until this task is finished or cancelled. To specify more than one successor, enter multiple task ID numbers separated by commas. Shown when the <i>Task Precedence Method</i> is 'ImmediateSuccessorsMethod'.
Tasks.Auto Cancel Trigger	Event type that will trigger an automatic cancellation of the task. The default setting is "AllImmediatePredecessorsCancelled". If left unchanged, then the task will be automatically cancelled if all its immediate predecessors in the task sequence are cancelled. This default setting is recommended if you want conditional or probabilistic branching to automatically cancel all tasks in unselected branches. For a more in-depth discussion of this option including examples, please refer to the Task Sequence element documentation.
Loopback Branches	Conditional or probabilistic loopback branches in the task workflow.
Loopback Branches.From Task Sequence Number	The task sequence number that is the beginning of the loopback. Shown if the <i>Task Precedence Method</i> is 'SequenceNumberMethod'.
Loopback Branches.To Task Sequence Number	The task sequence number that is the end of the loopback. Shown if the <i>Task Precedence Method</i> is 'SequenceNumberMethod'.
Loopback Branches.From Task ID Number	The task ID number that is the beginning of the loopback. Shown if the Task Precedence Method is 'ImmediatePredecessorsMethod' or 'ImmediateSuccessorsMethod'.
Loopback Branches.To Task ID Number	The task ID number that is the end of the loopback. Shown if the Task Precedence Method is 'ImmediatePredecessorsMethod' or 'ImmediateSuccessorsMethod'.
Loopback Branches.Branch Type	Indicates whether the loopback is a conditional or probabilistic branch on the task workflow. If the <i>Branch Type</i> is 'conditional', then the loopback occurs if the specified logical condition evaluates to True. The 'Probabilistic' branch type may be used to identify the loopback as one of possibly several mutually exclusive alternatives at a probabilistic decision point. The sum of the branch probabilities at a probabilistic decision point cannot be greater than one.
Loopback Branches.Condition	The branch condition or probability specified as an expression. If a condition, then enter the logic condition. If a probability, then enter the chance of the loopback as a value

or Probability	between 0.0 (0%) and 1.0 (100%).
Log Tasks	Indicates whether executed tasks are to be automatically logged. Go to Results > Logs > Task Log to view the logged data.
Invalid Task Dependency Notification Type	Indicates the level of alert at runtime if the task sequence has missing, duplicate, or otherwise invalid data defining the task dependency relationships. Error - Throw runtime error. Warning - Throw runtime warning and skip the invalid data. Trace - Write to trace and skip the invalid data.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TaskSequence - Discussion and Examples

Task Sequences Within Standard Library Objects

The Task Sequences element is utilized within the Server, Combiner and Separator objects in the Standard Library. The *Process Type* property on each of those objects may be specified as 'Specific Time' or 'Task Sequence'. When 'Task Sequence' is used, a repeating group of *Processing Tasks* may be specified. See the specific objects for more detail; the *Processing Tasks* are explained in greater detail within the [Task Sequence - Processing Tasks](#) section.

Task Precedence Method Alternatives

Within the Task Sequences element, as well as within the Server, Combiner and Separator objects using task sequences, there is a *Task Precedence Method* property that must be specified.

The default method of 'SequenceNumberMethod' allows for very easily defining sequential tasks by simply numbering the tasks using the *Sequence Number* property.

There are two alternative methods for defining the task precedences, that is, defining the *Task Precedence Method* property as either 'ImmediatePredecessorMethod' or 'ImmediateSuccessorMethod'. With these two options, an *ID Number* is defined (instead of *Sequence Number*) for a given task. One or more *ID Numbers* may then be referenced as either the *Immediate Predecessors* or *Immediate Successors* of other tasks. Users may find it slightly easier to define parallel tasks using these methods. The examples below are shown using the various methods.

Task Sequence Example 1

The following is an example of a *Task Sequence* element which defines the following tasks, all of which are in series (have precedence constraints). There are no tasks that are done in parallel in this example. The diagram below shows the information needed for setting up this sequence of tasks using either the 'SequenceNumberMethod' or the 'ImmediatePredecessorMethod'/'ImmediateSuccessorMethod'.

Example Workflow #1

SequenceNumberMethod

Sequence Number	Name
10	Task1
20	Task2
30	Task3

ImmediatePredecessorsMethod or ImmediateSuccessorsMethod

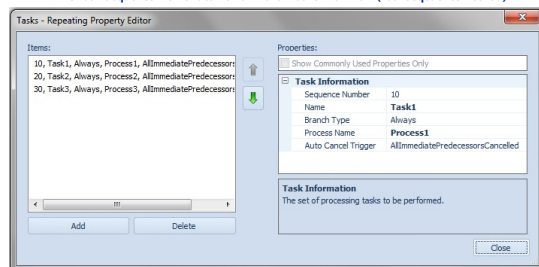
ID Number	Name	Immediate Predecessor	Immediate Successor
10	Task1		20
20	Task2	10	30
30	Task3	20	



The following description and screen shot are only for the 'SequenceNumberMethod'.

- There are three rows in the *Tasks* repeat group property; one row for each task in this workflow.
- The *Sequence Number* property determines the order in which the tasks are performed and if any tasks should proceed others. In this example, Task1 must be performed first, followed by Task2, then finishing with Task3.
- The *StartTasks* step is used to start the tasks within a specified Task Sequence element.

The Task Sequence Element to Perform the Above Workflow (TaskSequenceMethod)



Task Sequence Example 2

The following is an example of a *Task Sequence* element which defines the following tasks, some of which are in series (have precedence constraints) and others which can be done in parallel.

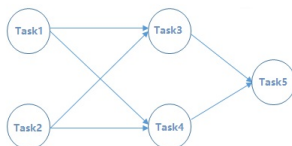
Example Workflow #2

SequenceNumberMethod

Sequence Number	Name
10	Task1
10	Task2
20	Task3
20	Task4
30	Task5

ImmediatePredecessorsMethod or ImmediateSuccessorsMethod

ID Number	Name	Immediate Predecessor	Immediate Successor
1	Task1		3, 4
2	Task2		3, 4
3	Task3	1, 2	5
4	Task4	1, 2	5
5	Task5	3, 4	



The following description and screen shot are only for the 'SequenceNumberMethod'.

- There are five rows in the *Tasks* repeat group property; one row for each task in this workflow.
- The *Sequence Number* property determines the order in which the tasks are performed and if any tasks should proceed others. In this example, Task1 and Task2 are performed in parallel and are started first. Once both of the Sequence Number 10 tasks have been completed, then Task3 and Task4 can be started in parallel (both indicated as Sequence 20's). Once the 20's have been completed, then Task5 may begin (Sequence Number 30).
- Note that this example follows Rule #1 in the Task Sequence element, that is, if the task sequence number consists of only a root number (only XX), then all tasks with lower root numbers (ignoring any suffixes) must precede that task. Note that it is possible to have one or more tasks with the same root sequence number, with or without suffixes.

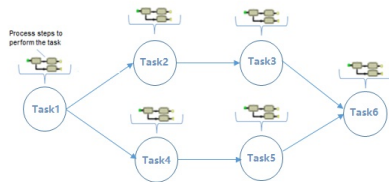
Task Sequence Example 3

The following is an example of a *Task Sequence* element which defines the following tasks, some of which are in series (have precedence constraints) and others which can be done in parallel.

Example Workflow #3

SequenceNumberMethod	
Sequence Number	Name
10	Task1
20.1	Task2
30.1	Task3
20.2	Task4
30.2	Task5
40	Task6

ImmediatePredecessorsMethod or ImmediateSuccessorsMethod			
ID Number	Name	Immediate Predecessor	Immediate Successor
1	Task1		2, 4
2	Task2	1	3
3	Task3	2	6
4	Task4	1	5
5	Task5	4	6
6	Task6	3, 5	



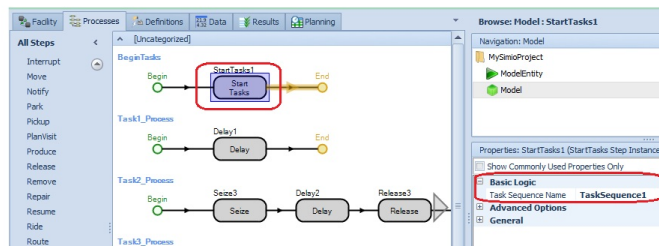
The following description and screen shot are only for the 'SequenceNumberMethod'.

- There are six rows in the *Tasks* repeat group property; one row for each task in this workflow.
- The *Sequence Number* property determines the order in which the tasks are performed and if any tasks should proceed others. In this example, Task1 must be performed first, followed by Task2 and Task4, since sequence number 20 is greater than 10, but less than 30. These two tasks can be performed in parallel, as determined by the number after the 20. Task3 must not begin until Task2 is complete and Task5 must wait until Task4 is complete. Task6 can be performed after both Task3 and Task5 are complete, since 40 is greater than 30.

The Task Sequence Element to Perform the Above Workflow

Tasks - Repeating Property Editor

Items:	Properties:
10, Task1, Always, Task1_Process, AllImmediatePredecessorsCancelled	Task Information
20.1, Task2, Always, Task2_Process, AllImmediatePredecessorsCancelled	Sequence Number: 30.1
30.1, Task3, Always, Task3_Process, AllImmediatePredecessorsCancelled	Name: Task3
20.2, Task4, Always, Task4_Process, AllImmediatePredecessorsCancelled	Branch Type: Always
30.2, Task5, Always, Task5_Process, AllImmediatePredecessorsCancelled	Process Name: Task3_Process
40, Task6, Always, Task6_Process, AllImmediatePredecessorsCancelled	Auto Cancel Trigger: AllImmediatePredecessorsCancelled



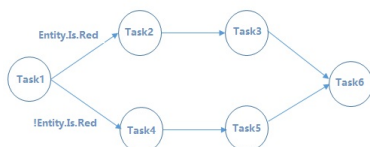
Task Sequence Example 4

The following is another example of a *Task Sequence* element which illustrates conditional branching.

Example Workflow #4

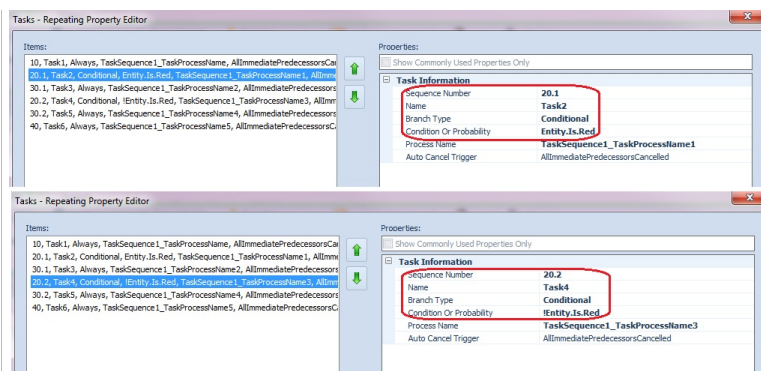
SequenceNumberMethod

Sequence Number	Name	Branch Type	Condition or Probability
10	Task1	Always	
20.1	Task2	Conditional	Entity.Is.Red
30.1	Task3	Always	
20.2	Task4	Conditional	!Entity.Is.Red
30.2	Task5	Always	
40	Task6	Always	



- There are six rows in the *Tasks* repeat group property; one row for each task in this workflow.
- If the entity object associated with the active task sequence is a 'Red' entity, then the tasks performed (in sequential order) will be Task1, Task2, Task3 and finally Task6. Otherwise, the tasks performed will be Task1, Task4, Task5 and finally Task6.

The Task Sequence Element to Perform the Above Workflow



Task Sequence Example 5

The following is another example of a **Task Sequence** which illustrates Secondary Resource Allocation Queue – ranking requests to Seize a resource by Processing Task priority.

A model is using Task Sequences to model processing at Servers. Sometimes, when a secondary resource (Worker) comes on-shift or is released, there are multiple processing tasks at Servers waiting to seize it, and those individual processing tasks have a priority that is used to determine which task the Worker should select first. So, priority is a characteristic of the processing tasks rather than the entities.

You can now use the modeling technique illustrated below, whereby the resource object's **Ranking Expression** property is a processing task-based table row reference.

Example Workflow #5

Processing Tasks		
Task Name	Sequence Number	Priority
1 Task1	10	2
2 Task2	10	4
3 Task3	10	1
4 Task4	10	3

Entities are executing task sequence at Server,
each individual processing task in task sequence has a Priority value.



Properties: Worker1 (Worker)	
Resource Logic	
Capacity Type	Fixed
Ranking Rule	Smallest Value First
Ranking Expression	ProcessingTasks.Priority
Dynamic Selection Rule	None
Park While Busy	False

The static ranking rule for allocating Worker1 to waiting processing tasks is smallest ProcessingTasks.Priority first.

Properties: Server1 (Server)	
Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Task Sequence
Processing Tasks	ProcessingTasks
Off Shift Rule	Suspend Processing

Timer

Timer

The Timer element fires a stream of events according to a specified *Interval Type*. It is added to a model from the Elements Window from within the Definitions Tab.

If the *Interval Type* is 'Time' or 'TimeInState', then the first event occurs at *Time Offset*. If the *Interval Type* is 'RateTable', 'EventCount' or 'ArrivalTable', then the first event time is determined by the rate pattern, event count or the numeric table property.

The maximum number of timer events is limited to *Maximum Events*. The *Maximum Events* expression will be re-evaluated each time it is required to be looked at. The *Reset Event Name* event will reset the timer event count.

The Timer has an Enabled state that allows the Timer to be enabled/disabled. To disable a Timer, set its Enabled property (Timer.Enabled) to 0. If the timer is disabled and it is an EventCount timer, then the triggering event fire notification is stopped. If the timer is disabled and it is a Time, TimeInState, or RateTable timer, then the calendar event stream is stopped.

For 'ArrivalTable' type Timer elements, if the *Repeating Arrival Pattern* property is 'True', the timer re-evaluates the *Arrival Events Per Time Slot* property at each discrete time in the table on every recurrence. Thus, you can now reasonably specify a random distribution for the number of arrival events per time slot. Previous to sprint 170, that property was evaluated once at initialization, and the number of arrival events for the time slot was then fixed to that initial value on each recurrence. It is important to note that the *Arrival Events Per Time Slot* expression is evaluated for all time slots of first cycle at run initialization (time 0.0). Then, for each recurrence of the time slot if the pattern is repeated, the *Arrival Events Per Time Slot* is evaluated one cycle in advance. So, if time slot is 8:00 am and the length of the pattern is 24 hours, then the number of arrival events is evaluated and scheduled one day in advance. Because of this, users should only use a constant or a random distribution for that expression.

When a Timer event is fired, it can be used as the triggering event that executes a Process. If there are any processes specified to be triggered by that event, then the created tokens are associated with the object that owns the Timer element. For additional information on using Timers, see the [Timer - Discussions and Examples](#) page.

The standard Source object uses a Timer element to generate arrivals. For examples of how it uses the timer, see the [Source - Discussion and Examples](#) page.

See the [Table-Based Elements \(AutoCreate\)](#) page for information on automatically creating elements from table data.

For an example of using the Timer element, please refer to the SimBit [ScheduledMaterialArrivals](#).

Listed below are the properties of **Timer**:

Property	Description
Interval Type	The method used to determine the time interval between two successive timer events.
Time Offset	The time offset until the first timer event.
Time Interval	The time interval between two successive timer events.
State Variable Name	The name of the state variable that will determine which time periods are included in calculating the timer's elapsed time.
State Value	The state variable value that will determine which time periods are included in calculating the timer's elapsed time. Only time periods that the state variable has this value will be included.
Rate Table	The rate table that defines how the rate of timer events changes over time, the interarrival times following a non-stationary poisson process.

Rate Scale Factor	The factor used to scale the rate values specified in the rate table.
Triggering Event Name	The name of the event which determines when the next timer event is fired.
Triggering Event Count	The number of Triggering Event occurrences required to trigger the next timer event.
Arrival Time Property	<p>The name of the numeric table property (table column) that contains the discrete times when timer events are expected to be fired.</p> <p>The number of timer events to be fired at each scheduled time is determined by the Arrival Events Per Time Slot property. When the timer's event is fired, a reference to the table row holding the arrival time will be automatically assigned to any process tokens created by the event.</p>
Arrival Events Per Time Slot	<p>The number of arrival events expected to occur at each of the discrete times defined in the Arrival Time property.</p> <p>To vary the number of arrival events for each time slot, add a numeric column that contains those values to the referenced arrival table. Then specify the name of that table column here.</p>
Repeating Arrival Pattern	Indicates whether the arrival pattern in the table should be repeated when the end of the pattern is reached.
Maximum Events	The maximum number of times after initialization or reset that the time can fire its event
Maximum Time	The maximum time allowed after initialization or reset that the timer can fire events.
Arrival Time Deviation	<p>Expression used to model differences (typically random) between the scheduled times in the arrival table and the actual times during the simulation run that the arrival events occurs.</p> <p>The specified expression may return a negative or positive value. A negative value indicates that the actual arrival time will occur earlier than the scheduled time by that duration. A positive value indicates that the actual arrival time will occur later than the scheduled time by that duration.</p> <p>To vary the arrival time deviation for each time slot, add an expression column that contains those expressions to the referenced arrival table. Then specify the name of that table column here.</p> <p>NOTE: This feature is automatically disabled if random sampling in the simulation run is disabled.</p>
Arrival No-Show Probability	<p>The probability that a scheduled arrival in the arrival pattern will be a no-show and thus the timer's event not actually fired. Enter the chance of a no-show as a value between 0.0 (0%) and 1.0 (100%).</p> <p>To vary the no-show probability for each time slot, add a numeric column that contains those probabilities to the referenced arrival table. Then specify the name of that table column here.</p> <p>NOTE: This feature is automatically disabled if random sampling in the simulation run is disabled.</p>
Random Number Stream	The random number stream used to generate interarrival times (if rate table) or make probabilistic no-show decisions (if arrival table).
Reset Event Name	The name of the event whose occurrence will trigger a reset of the timer.
Initially Enabled	<p>Indicates whether the timer is enabled when the system is initialized.</p> <p>NOTE: Refer to the user-assignable 'Enabled' state variable of a timer to dynamically enable or disable it during a simulation run.</p>
Report	Specifies if statistics are to be automatically reported for this element.

Statistics

Listed below are the states of **Timer**:

State	Description
Enabled	Indicates whether the timer is currently enabled. Assignable values are True(1) or False(0).
ElapsedTime	The time (or time in state if applicable) that has elapsed since the last timer event or reset.

Listed below are the events of **Timer**:

Event	Description
Event	The timer event.
Reset	Resets the timer.

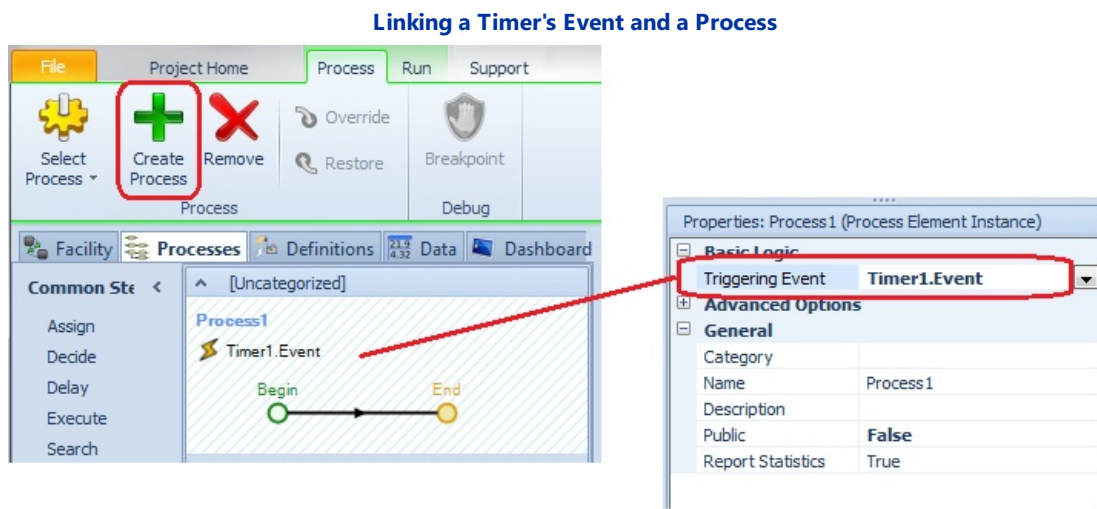
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Timer - Discussion and Examples

Discussion

In order to have a process execute when a Timer fires its event, the *Triggering Event* property on the Process must be set to the name of the Timer's event. A new process can be created from the Processes Window by clicking the Create Process button in the ribbon. In the Properties window of this new process, select the name of the Timer (.Event) from the drop down of the *Triggering Event* property. This will cause this process to be executed each time the specified Timer fires an event.



If the Timer element's *Interval Type* is 'TimeInState', then *State Value* expression is re-evaluated on every state change. This allows the user to model an expression whereby the timer's *Time Offset* and *Interarrival Time* values are based on multiple possible state values instead of a single state value. For example, if *Interval Type* = 'TimeInState', *State Variable Name* = 'myState', *State Value* = 'Math.If((myState==3)||((myState==4),myState,Math.NaN))'. This would cause the *TimeOffset* and *Interarrival Time* values of the timer to be based on the time that the state variable 'myState' had values of 3 or 4.

Timer Example 1

The following is an example of a Timer that fires 5 events, an hour apart, starting at 2 hours after the beginning of the run.

- The *Time Offset* property is set to '2 hours' so this Timer does not fire its first event until 2 hours into the simulation run.
- The *Time Interval* property is set to '1 hour' which indicates that the events are fired with one hour in between events.
- The *Maximum Events* property tells the Timer to stop firing after 5 events have been fired.

Properties: HourlyTimer (Timer Element Instance)	
Basic Logic	
Interval Type	Time
Time Offset	2
Units	Hours
Time Interval	1
Units	Hours
Stopping Conditions	
Maximum Events	5
Maximum Time	Infinity
Advanced Options	
General	
Name	HourlyTimer
Description	
Public	True
Report Statistics	True

Timer Example 2

The following is an example of a Timer that fires an event every time 5 entities have entered the Input Node of Server1. Therefore, the Timer fires an event after 5 entities have entered the input node, then again when 10 entities have entered, again when 15 entities have entered, etc.

- The *Interval Type* property is set to 'Event Count' so this Timer counts the number of specified events to determine when to fire its own event next.
- The *Triggering Event* property is set to 'Input@Server1.Entered' which determines that this is the event that is counted by the Timer.
- The *Triggering Event Count* property tells the Timer to fire its event after 5 occurrences of Input@Server1.Entered.

Properties: EventCount (Timer Element Instance)	
Basic Logic	
Interval Type	EventCount
Triggering Event	Input@Server1.Entered
Triggering Event Count	5
Stopping Conditions	
Maximum Events	Infinity
Maximum Time	Infinity
Advanced Options	
General	
Name	EventCount
Description	
Public	True
Report Statistics	True

Timer Example 3

The following is an example of a Timer that fires its event based on information in a Data Table. The Timer reads information from a Date Time, an Integer or Real property in a Data Table to determine when to fire its next event. If a Date Time property is used, the timing of the event will be calculated by the time span from the starting date of the simulation to the value in the table. So if the starting date of the simulation was 1/1/2008 12:00:00 AM, and the Date Time in the table was 1/1/2008 2:30:00AM, the event would be fired at time 2.5 hours. If a Real or Integer property is used, the event will fire at the simulation start time plus the numeric value of the property.

Note: If the DateTime is before the Simulation Start, or the value is negative, the arrival event does not occur.

- The *Interval Type* property is set to 'Arrival Table' so this Timer looks to a property in a Data Table to determine when to fire its next event.
- The *Arrival Time Property* property is set to 'Table1.Notification' which tells this Timer to read the property called Notification from the table called Table1, to determine when to fire its next event.

Properties: ArrivalTimer (Timer Element Instance)	
Basic Logic	
Interval Type	ArrivalTable
Arrival Time Property	Table1.NotificationTime
Arrival Events Per Time...	1
Arrival Time Deviation	0.0
Arrival No-Show Proba...	0
Repeat Arrival Pattern	False
Stopping Conditions	
Advanced Options	
General	
Name	ArrivalTimer
Description	
Public	True
Report Statistics	True

Timer Example 4

The following is an example of a Timer that fires its event based on the amount of time spent in a state. In this example, the timer will fire when the total time that Server1's ResourceState = 1 reaches 3 hours. In other words, when Server1 has been busy for a total of 3 hours, the timer will fire.

- The *Interval Type* property is set to 'TimeInState' so this Timer looks to the state variable Server1.Resource.TotalTime to determine when to fire its next event.

Properties: StateTimer (Timer Element Instance)	
Basic Logic	
Interval Type	TimeInState
Time Offset	0.0
Units	Hours
Time Interval	3
Units	Hours
State Variable Name	Server1.ResourceState
State Value	1
Stopping Conditions	
Advanced Options	
General	
Name	StateTimer
Description	
Public	True
Report Statistics	True

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Activity - DEPRECATED

Activity - DEPRECATED

An activity is performed by an operation that is owned by an object.

An operation is a sequence of activities that are performed by an object over time. Time slots for operations/activities may be reserved in the future. Activities may be started and ended. An activity time scales by a work schedule (if any). Activity times can be specific, change dependent, or sequence dependent. Dependent times are based on an entity list value. This list value (numeric) is typically provided by a list property (color, size, type, etc.) on the entity.

NOTE: It is recommended that the [TaskSequence](#) element and [StartTasks](#) step be used instead of the deprecated Operation / Activity elements and associated steps.

Listed below are the properties of **Activity**:

Property	Description
Duration Type	The method used to determine how the time duration for this activity is specified.
Duration	The time duration of this activity.
Comparison Expression	The expression evaluated for each entity that is used to determine the change-dependent or sequence-dependent activity time. Note that, for a sequence-dependent time, the comparison expression is evaluated to a zero-based index into the changeover matrix string list for which 'from-to' pairs are defined. For example, the comparison expression value of '0' means the entity's comparison value is the first value in the changeover matrix string list. Referencing a list property on the processing entity as the comparison expression is particularly useful for modeling sequence-dependent activity times.
Duration If Same	This activity's time duration if there has not been a change in the value of the Entity Property from the previous entity to the current entity.
Duration If Different	This activity's time duration if there has been a change in the value of the Entity Property from the previous entity to the current entity.
Changeover Matrix	The name of the changeover matrix that defines the activity times dependent on the value of the Entity Property from the previous entity to the current entity.
Usage Cost Rate	The cost per capacity unit per unit time to use the operation's primary resource for the activity. This cost rate will only apply during activity time that the primary resource is in a utilized state, and will be charged to the cost of the entity performing the activity.
Batch Size	The batch size for this activity. If not specified, then defaults to the operation quantity.
Material Consumption	The material consumption of this activity.
Consumed Material Name	The material which is to be either specifically consumed or whose bill of materials is to be consumed by this activity.
Consumed Quantity	The quantity to be consumed.
Material Production	The material production of this activity.

Produced Material Name	The material which is to be either specifically produced or whose bill of materials is to be produced by this activity.
Produced Quantity	The quantity to be produced.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Listed below are the Functions of **Activity**:

Function	Description
ExpectedTotalDurationFor(entity)	Returns an estimate of the expected total time duration (in hours) for the specified entity to perform the activity, calculated under the assumption that the entity would be next to start the activity.
NumberBatchesRequiredFor(entity)	Returns the total number of sequential batches required for the specified entity to perform the activity.
CurrentBatchDurationFor(entity)	Returns the time duration (in hours) to perform the activity for the specified entity's current batch. If the entity is not currently performing the activity then NaN is returned.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Operation - DEPRECATED

Operation - DEPRECATED

An operation is a sequence of activities that are performed by an object over time. An operation may have one or more activities.

NOTE: It is recommended that the [TaskSequence](#) element and [StartTasks](#) step be used instead of the deprecated Operation / Activity elements and associated steps.

Listed below are the properties of **Operation**:

Property	Description
Quantity	The item quantity assumed for each entity executing this operation.
Activities	This operation's activities in the sequence that they must be performed.
Activities.Activity Name	The name of the activity.
Activity Resources	This operation's activities in the sequence that they must be performed.
Activity Resources.Object Type	The method for specifying the object(s) required as a secondary resource.
Activity Resources.Object Name	The name of the object type that is required.
Activity Resources.Object List Name	The name of the object list from which to select objects.
Activity Resources.Activity Range	The range of activities that the secondary resource is required for.
Activity Resources.Activity Index	The specific activity specified as an index in the operation's activity list.
Activity Resources.Request Move	Indicates whether a move to a specified location will be requested from the seized resource(s). The executing token will be held in the Seize step until the resources have arrived to the requested location.
Activity Resources.Destination Node	The name of the specific node location that the seized resource(s) will be requested to move to. If not specified, and the owner object performing the seize is an entity object at a node, then this property defaults to that owner entity's current node.
Activity Resources.Move Priority	The priority of the move request if a PlanVisit step or SelectVisit step is used by the seized resource(s) to choose a visit destination from the multiple candidates. If not specified, and the owner object performing the seize is an entity object, then this property defaults to that entity's Priority state value.
Activity	The number of objects that are required.

Resources.Number Of
Objects

Activity
Resources.Units Per
Object

The number of capacity units that are required per object.

Activity
Resources.Selection
Goal

The goal used to rank object preference when multiple candidates are available.

Activity
Resources.Selection
Expression

The expression criteria, evaluated for each candidate object, used with a SmallestValue or LargestValue selection goal. Use the keyword [Candidate](#) at the beginning of the expression.

Activity
Resources.Selection
Condition

An optional condition evaluated for each candidate object that must be true for the object to be eligible for seizing. In the condition, use the keyword [Candidate](#) to reference an object in the collection of candidates (e.g., Candidate.Object.Capacity.Remaining).

Activity Resources.On
Seized Process

Optional process that is executed by a token associated with a seized object after the seize occurs.

Activity Resources.On
Released Process

Optional process that is executed by a token associated with a released object after the release occurs.

Activity
Resources.Accrue
Usage Costs

Indicates whether costs for the use of the seized resource(s) will be charged, or accrued, to the cost of the owner object.

Activity
Resources.Per-Use
Cost Accrual

This property indicates when the one-time 'cost per use' for a seized resource will be charged, or accrued, to the cost of the owner object. Note that, if the owner is an entity object, then the 'AtNextStation' accrual method will charge the cost to the entity the next time it enters a station location.

Maximum Makespan

Defines the maximum amount of time which this operation can be performed, from the start of the first activity until the end of the last activity.

Report Statistics

Specifies if statistics are to be automatically reported for this element.

Listed below are the Functions of **Operation**:

Function	Description
EstimatedMakespanFor(entity)	Returns the estimated amount of time for the specified entity to perform this operation, from the start of the first activity to the end of the last activity, assuming the operation's activities will be spanned over (i.e., extended by) any off-shift periods for the operation's primary resource. Note that if the operation has not yet been started by the entity, then the estimate will be based on if the operation was started immediately.

*Note: The above function is currently used by the Workstation in its logic to not allow an entity to seize the Workstation unless the following condition is true: (Operation.EstimatedMakespanFor(Entity) = (MaximumMakespan - MakespanBufferTime)

User Defined Elements

Simio provides an open architecture that allows users to add new steps and elements to the system. Steps and elements may be coded in any .NET language (Visual Basic, C#, J#, etc.). You would simply add your *.dll files to the UserExtensions folder of Simio. For additional information on creating your own custom Elements, see [Custom Simio Extensions](#) and **C:\Program Files\Simio LLC\Simio\Simio API Reference Guide.chm**. See [Simio Visual Studio Templates](#) for information on how to use the templates provided by Simio to get started with User Defined Elements.

Simio provides several sample User Defined elements including [BinaryGate](#) and [File](#).

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

BinaryGate

BinaryGate

The BinaryGate user defined element defines the gate name and its characteristics. A BinaryGate is initially closed and must be opened before any object can flow through it. The BinaryGate element is used in conjunction with the OpenGate, CloseGate and PassThruGate user defined steps.

Listed below are the properties of **BinaryGate**:

Property	Description
Public	Specifies if this element is publicly accessible outside of its containing object.
Report Statistics	Specifies if statistics are to be automatically reported for this element.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

DbConnect

DbConnect

The DbConnect user defined element defines the database provider and connection to the database. It is used in conjunction with several of the user defined steps, including [DbExecute](#), [DbQuery](#), [DbRead](#), and [DbWrite](#).

Listed below are the properties of **DbConnect**:

Property	Description
Connection String	Specifies the connection string used to establish the connection to the database. If you need assistance, use the database bind wizard in the Data tab, Tables panel. The connection string is available if you select the 'Custom connection string' option in the Import Your Database dialog.
Provider Name	Specifies the provider type to specify what database provider to use to connect to the database. Typically MySQL data provider is not available unless the user installs the MySQL.net client (mysql-connector-net-6.6.5.msi). Other providers come with the .NET framework. If you enter an incorrect provider, when you run the model, the list of available providers is shown.
Report Statistics	Specifies if statistics are to be automatically reported for this item.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ExcelConnect

ExcelConnect

The ExcelConnect user defined element may be used in conjunction with the user defined [ExcelRead](#) and [ExcelWrite](#) steps to read and write to an Excel spreadsheet.

Listed below are the properties of **ExcelConnect**:

Property	Description
Excel Workbook	Specifies the location of the spreadsheet. The ExcelConnect handles the following (workbook in same Folder, workbook in relative folder and workbook in absolute folder). ExcelReadWrite.xlsx, .\Test\ExcelReadWrite.xlsx, C:\temp\ExcelReadWrite.xlsx
Report Statistics	Specifies if statistics are to be automatically reported for this item.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

File

File

The File user defined element defines the file name and its characteristics. The user should ensure that they have sufficient privileges to write to the location that they specify for the file path.

The File element is used in conjunction with the [Read](#) and [Write](#) user defined steps. See the [WritingToAFile](#) SimBit for an example on how to use the File Element.

Listed below are the properties of **File**:

Property	Description
Public	Specifies if this element is publicly accessible outside of its containing object.
Report Statistics	Specifies if statistics are to be automatically reported for this element.
File Path	The full path of the file to read from or write to.
Auto Merge Write Step Files For Experiment	Specifies whether to automatically merge all the experiment output files per File Element into one new file. If enabled, the new file will be located in the same folder.

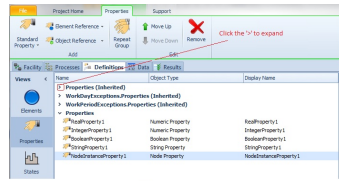
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Properties

Properties

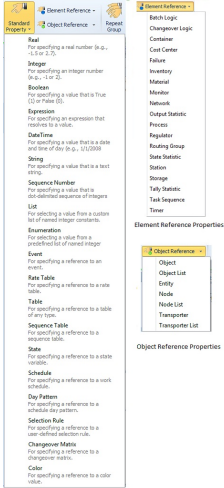
Properties of a model object are the input parameters associated with that object. They are added to a model from the Properties panel of the Definitions window.



Clicking to the right of a column allows for sorting and filtering

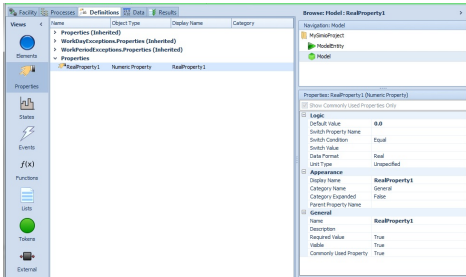
A Property holds a specific data type. It is a static input parameter (e.g. processing time, batch size) for the object and does not change during the running of the simulation. There are four different types of Properties that can be added to the model by selecting the appropriate icon in the ribbon menu: Standard Properties, Element Reference Properties, Object Reference Properties and Repeat Group Properties. User Defined Elements will also have properties appear in the Element Reference drop-down in a section at the bottom. Learn more about the properties of the Simio standard objects by viewing the details of each [Standard Library Object](#).

Properties may also be put into repeating groups. An example of a repeating property is the [Assignments\(More\)](#) property of the [Assign](#) step in the [Processes](#) window. These repeat group type properties allow the user to specify multiple, similar properties within a different instance. See [Repeat Groups](#) for more information.



The main model properties may be referenced by properties of objects within your model – i.e. they get their values from these properties. The easiest way to simultaneously add the main level properties and create the reference to it is to right-click on a property of interest for an object within your model and select "Add Reference Property". This adds a new property to the parent model and sets the reference to this new property. You can see this new property by clicking on the Definitions tab and selecting the Properties panel for your main model. Changing the default value of the property affects the model if that property is referenced by objects in the model.

Different types of properties (as shown above, for example, Real, Integer, Expression, Entity Object Reference, etc.) may have different properties that can be specified to determine their behavior. The properties for a Real type Property are shown below and include Logic, Appearance and General type properties.



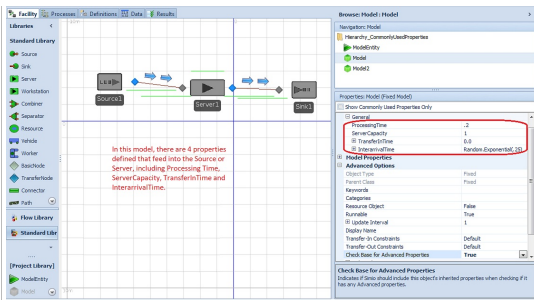
Within the Logic portion of properties, the **Default Value** can be specified. This is the initial value that the property has and can typically be changed. Each property also has three properties called **Switch Property Name**, **Switch Condition** and **Switch Value**. These are used if this particular property is only used based on the value of another property. For example, the property of a **Fixed** object type called **InitialCapacity** is only displayed if the property **CapacityType** is equal to the value **Fixed**. It does not make sense to display **Capacity** if the **CapacityType** is **WorkScheduler**. Therefore, on the property **Capacity**, the **Switch Property Name** is set to **CapacityType**, the **Switch Condition** is set to **Equal** and **Switch Value** is set to **Fixed**. See the main Model's Definitions window, Properties panel and select the property **InitialCapacity** to see this example. The **Switch Value** property can have multiple values associated with it. To specify multiple values within the **Switch Value** field, enter the values, separated by commas. For example, if a property, **PropertyC**, is displayed only if another property, **PropertyA**, has a value of either 1, 2 or 4, then the **Switch Property Name** for **PropertyC** would be **PropertyA**, the **Switch Condition** would be **Equal** and the **Switch Value** would be **"1, 2, 4"**.

Within the Appearance category of information, the **Display Name** is used to show what is displayed to the user within the object. A property may be categorized, such that it appears within a given section based on the **Category Name** of properties for that model object. Properties will be shown in the model, as well as the experiment, based on their **Category Name**. It may also have a **Parent Property Name**, such that it is listed 'underneath' another property and accessible with the **^**.

Within the General section of properties for a Property are two other important pieces of information for the Property. The **Required Value** determines whether the user must include a value for this property to run the simulation. The **Visible** determines whether or not the user actually sees the value with the list of properties for the object.

Listed below are the commonly used properties of **Properties**:

Property	Description
Default Value	The default value for the property.
Switch Property Name	The name of the enum property that is used as an on-off switch to make this property visible.
Switch Condition	The conditional operator used to compare Switch Property Name to Switch Value .
Switch Value	The value of the Switch Property Name that is checked against the Switch Condition to indicate whether this property is visible. To use multiple values of Switch Property Name in the comparison, enter the values separated by commas.



A Property can be used as an input parameter into an Experiment. Once a Referenced Property is added to a model, it then appears in the table of the Experiment Design Window. The user can enter a value into the table for that Property and might choose to run multiple scenarios, each with a different value entered for the Property(s).

HasValue function - For element and object type properties, this function would be referenced as `TemplateName.PropertyName.HasValue` or `ObjectName.PropertyName.HasValue`. This function is useful for checking in model logic whether an optional property has been specified a value or not. The function returns True (1) if a value has been specified for the property, otherwise the value false (0) is returned. If the property is a repeat group, HasValue will return True (1) if one or more entries have been entered into the repeat group, otherwise the value false (0) is returned.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Properties. Please refer to the [User Interface](#) for more details.

Property Default Values and Macros

There are macros that can be put in the default value of a property that will expand to the correct value when an object with that property is instantiated. These macros include:

- `$(InstanceName)`
- `$(DefinitionName)`
- `$(AssociatedObjectNameName)`
- `$(AssociatedObjectDefinitionName)`

These macros make it easier to design a library object that can be reused without errors occurring.

Note

An **Expression Property** may contain random variables or state variables that return a different value each time. Hence although the expression definition does not change during the run, the value of the expression does.

Both a **State Property** and an **Expression Property** have dual behavior because they are derived from a Numeric Property. A State Property can be used to reference a state variable (e.g. statistic element) or it can be used to return the value of the state variable. An Expression Property can be used to reference an expression or it can be used to return the value of the expression.

The user has the ability to hide values for an **Enum Property** or to change the caption that is displayed in the Enum property. This is done by expanding the Captions category in the Enum properties window and selecting or unselecting values or changing the display name of the value.

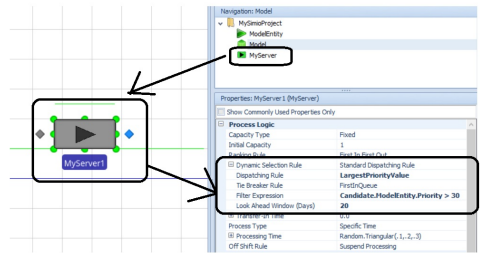
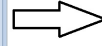
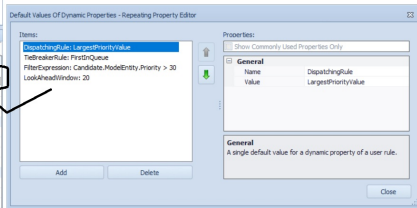
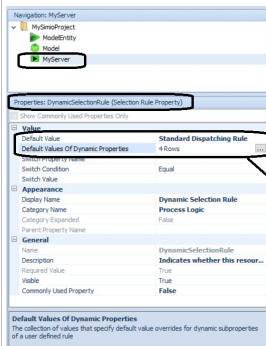
In order to add rows to a **Repeating Group Property**, first add a Repeat Group property in the Properties window. Then ensure that it is selected (highlighted) and add the properties that you wish to be part of the repeating group. The sub-properties will appear in the Properties window under a category such as **RepeatingProperty** Properties.

The **DateTime** is a Numeric Property that returns the time span from the starting date of the simulation to itself. For example, if the starting date of the simulation was 1/1/2008 12:00:00 AM, and the DateTime is 1/1/2008 2:30:00 AM, the value 2.5 is returned if DateTime is referenced in an expression or an Actual Table.

Property Default Values for Dynamic Sub-Properties

There are a few properties for objects that include dynamic sub-properties, for example, the Dynamic Selection Rule and the Travel step's Steering Behavior. Within an object's properties, if one of these property types is selected and the Default value property is specified as one which would include additional information, the Default Values of Dynamic Properties can be used.

For example, when an object property, such as Dynamic Selection Rule, has a Default Value of 'Standard Dispatching Rule', a number of additional properties, such as Dispatching Rule and Tie Breaker Rule become available for the user. In a typical Server object, the default values for these are 'FirstInQueue'. To change those default values for an object, such as a sub-classed MyServer (shown below), the user would add the property Name and new default Value in the repeating property editor.



Repeat Groups

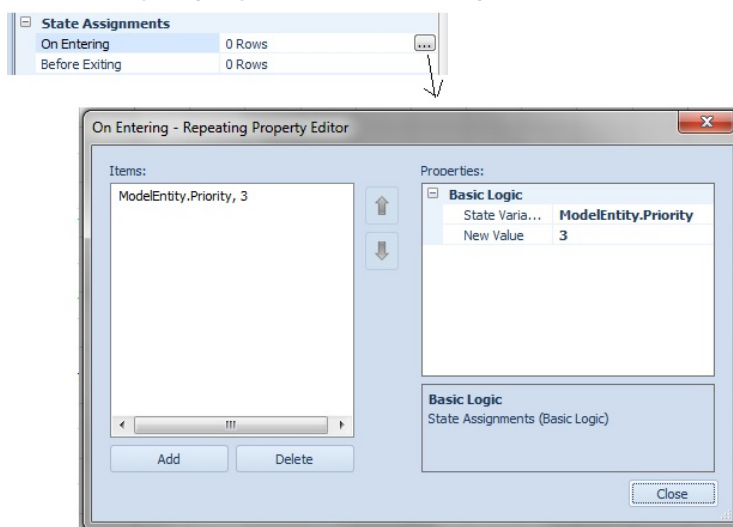
Repeat Group Properties

Properties may be put into repeating groups. Examples of a repeating property are the *Assignments(More)* property of the Assign step, as well as the *Seizes* property of the Seize step, in the Processes window. These repeat group type properties allow the user to specify multiple, similar properties within a different interface.

Repeat Groups are also used if a model uses a Data Table, but then is used as a submodel inside another model. Since Data Tables should exist at the top level model and not the submodel, a Repeat Group is created on the submodel instead of keeping the reference to the Data Table. To learn how to Convert a Table to a Repeat Group, see [Using A Custom Object in a Model](#) page.

Within the Standard Library objects, many of the objects have repeating properties in the State Assignments section. For example, within the Server, state assignments can be made On Entering or Before Exiting the Server. A repeating property is shown in the user interface with a default value of '0 Rows' with a small button on the right side containing '...'. By pressing the button, the repeating property dialog opens to allow any number of repeating property assignments.

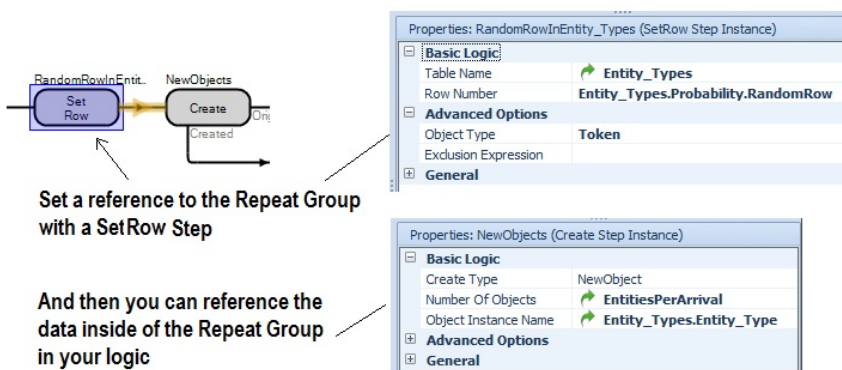
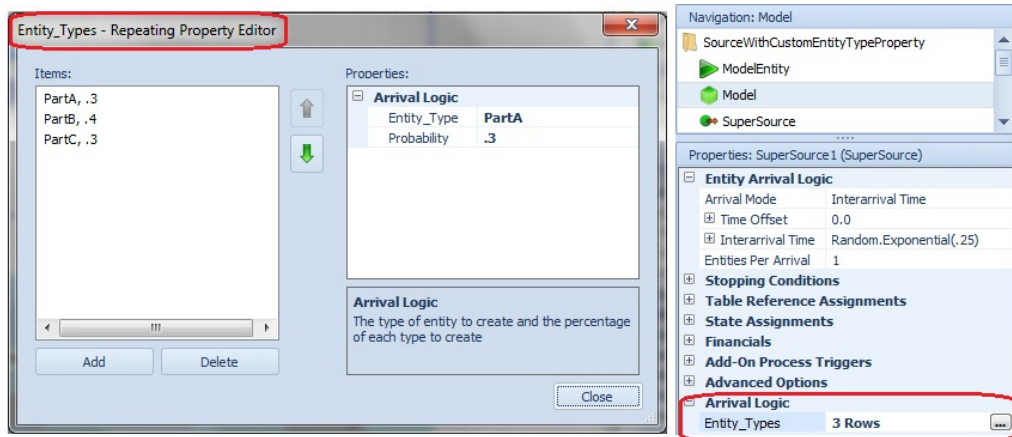
Repeating Properties within the State Assignments of a Server



Setting a Reference to the contents of a Repeat Group Property

You can set a reference to the contents of a Repeat Group Property, similarly to how you set a reference to a Data Table. This will allow you to reference data inside of a Repeat Group property, from elsewhere in your model. The following screen shot is from the [SourceWithCustomEntityTypeProperty](#) SimBit. It demonstrates how to set a reference to a Repeat Group property (Entity.Types, in this example) with a SetRow Step. This example then references the Entity_Type property of that Repeat Group in the *Object Instance Name* property of a Create Step to determine which type of entity to create. This is similar to setting a reference to a row in a table in order to utilize the data in a Simio table. In this case, you are setting a reference to a particular index in a Repeating Group property so you can utilize the data in that Repeat Group.

Setting a Reference to a Repeat Group Property



Referencing Tables within Repeating Properties

The procedure for referencing a table entry from within a repeating property depends upon how the table is referenced. Typically, when referencing a table column, the entity previously was assigned a pointer to a specific row in the table. This is done using the SetRow step in the Processes window or through the Table Row Referencing within the Source object. Below are the steps for referencing a simple table and a relational table from within a repeating property.

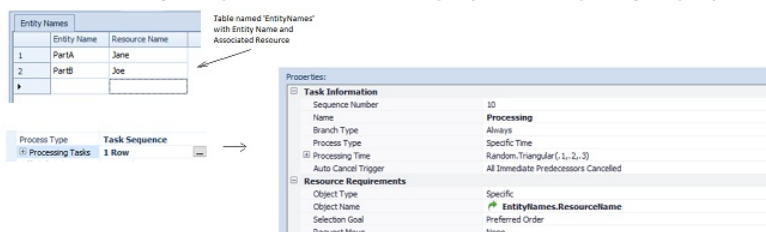
For more information on referencing tables within repeating properties, please see the SimBits [TableReferenceInRepeatingProperty](#), [RelationalTableInRepeatingProperty](#) and [SourceWithCustomEntityTypeProperty](#).

Simple Table References

Tables may be referenced within a repeating property and repeat group field. When referencing a table column within a particular repeating property, you would specify `MyTableName.MyPropertyName`, where `MyPropertyName` is the name of the property column within the table. This referencing assumes that the user has already put an explicit SetRow to this **same** table earlier in the logic. In the below example, there are two entity types (PartA and PartB) which require a different resource for processing at a Server (PartA requires Jane, PartB requires Joe). The information about the entity types and associated resource needed is stored in a table, EntityNames. The processing entity, prior to referencing the table, used a SetRow of EntityNames and the particular row number.

The *Object Name* references the table by using 'EntityNames.ResourceName'. Once the table is referenced within the repeating property, '1 Row' will appear in the *Processing Tasks* (or similarly *Secondary Resources areas*) property of the Server. When Simio encounters the table reference in the repeat group tuple, the resource can be determined because it already has a row set for that table for the current entity.

Referencing a Simple Table and Column Property within a Repeating Property



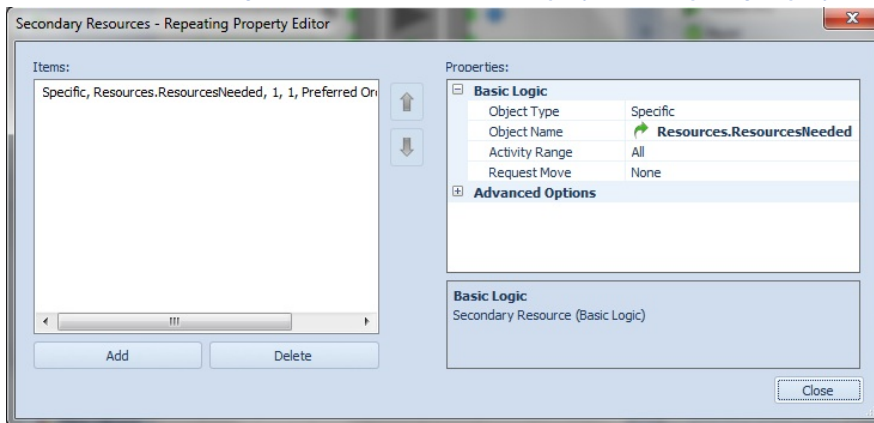
Relational Table References

Relational tables may also be referenced within a repeating property and repeat group field by using `MyRelationalTableName.MyPropertyName`. This referencing assumes that the user has done a SetRow to a **different, but related** table earlier in the logic.

In the diagram shown below, there are two tables in the model, EntityTable and Resources. The EntityTable simply lists the Entity Names, PartA and PartB and is set as a key column. When entities in the model are created, their SetRow should be set to either PartA or PartB from the EntityTable. See [Tables](#) for more information. There is also a table named Resources that contains the names of the resources that each of the two part types requires for processing at a given Workcenter. In this Resources table, the Entity Reference is a Foreign Key, and Resource Needed specifies one or more resources for each

entity type.

Referencing a Relational Table and Column Property within a Repeating Property



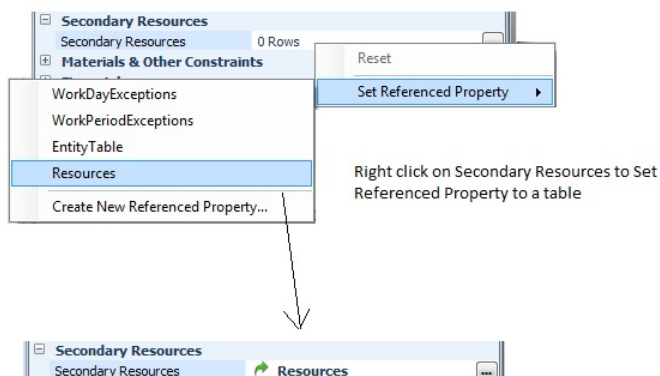
This is a repeating property within the Workstation object where Secondary Resources for processing are specified.

Within the Data tab, the Resources table specifies which resources are needed for each Entity Name

Entity Table	
Resources	
Entity Names	
PartA	
Resources	
Entity Name	Resources
PartA	Jane
*	
PartB	
Resources	
Entity Name	Resources
PartB	Joe
PartB	Jane
*	

In this case, because the entity has no known row within the Resources table, one additional step is needed. In addition to referencing the table and property column within the actual repeating property, the table name must be referenced in the main property. For example, as shown in the diagram below, the table, Resources, is referenced in the Secondary Resources property. To reference a table within a property, right click on the property and select the Set Referenced Property and select the table name to reference, as shown below. By doing this, the software will know that there are not a fixed number of entries based on what has been entered into the repeat group, but it is based on what is shown in the table.

Referencing the Table Name within a Repeat Group Property



Right click on Secondary Resources to Set Referenced Property to a table

Repeating Groups and Object Building

When building an object, such as a Source or Server, you may need to define a repeat group and repeating properties, for such steps as Assign, Seize or Release. One feature of repeat groups and their properties is that they may be referenced similar to referencing tables. For example, let's say you have a repeat group named MyRepeatGroup with two properties within the repeat group, MyTime and MyEntityReference. You may utilize the SetRow step with the Table Name set as 'MyRepeatGroup'. Other table functions, as shown below, may also be referenced within an expression.

- MyRepeatGroup.MyTime
- MyRepeatGroup[x].MyTime
- MyRepeatGroup.MyTime.RandomRow
- MyRepeatGroup.AvailableRowCount
- MyRepeatGroup.MyEntityReference, as a reference property in places that take an entity reference

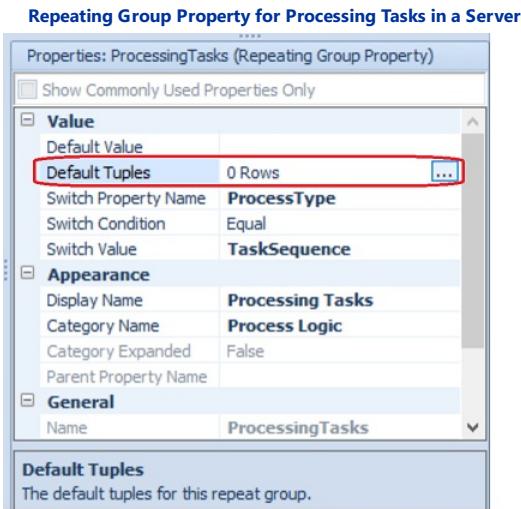
For more information on using repeating properties, please see the SimBits [TableReferenceInRepeatingProperty](#), [RelationalTableInRepeatingProperty](#) and [SourceWithCustomEntityTypeProperty](#).

Repeating Groups and Default Tuples

As mentioned above, repeating groups and their properties are used when building many different objects, such as Source,

Server or an object using an Assign or Seize step. This is also the case when sub-classing or copying an existing object from within one of the Standard or Flow libraries of objects. Similar to defining a *Default Value* for a property, the Repeating Group property has a *Default Tuples* repeating property to allow users to define pre-existing default values for one or more tuples in the repeat group.

This feature allows users to pre-define one or more 'tuples' within a repeat group. For example, this could be useful with multiple assignments that are made in an Assign step, multiple resources/workers/vehicles that are seized within a Seize step, or multiple tasks are performed within a task sequence in a Server.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

States

States

Simio allows the user to add either a Discrete State or a Continuous State to the model. States are added to a model from the States panel within the Definitions window. For additional information on how to assign a State to an object, see the help topic [Assigning States](#).

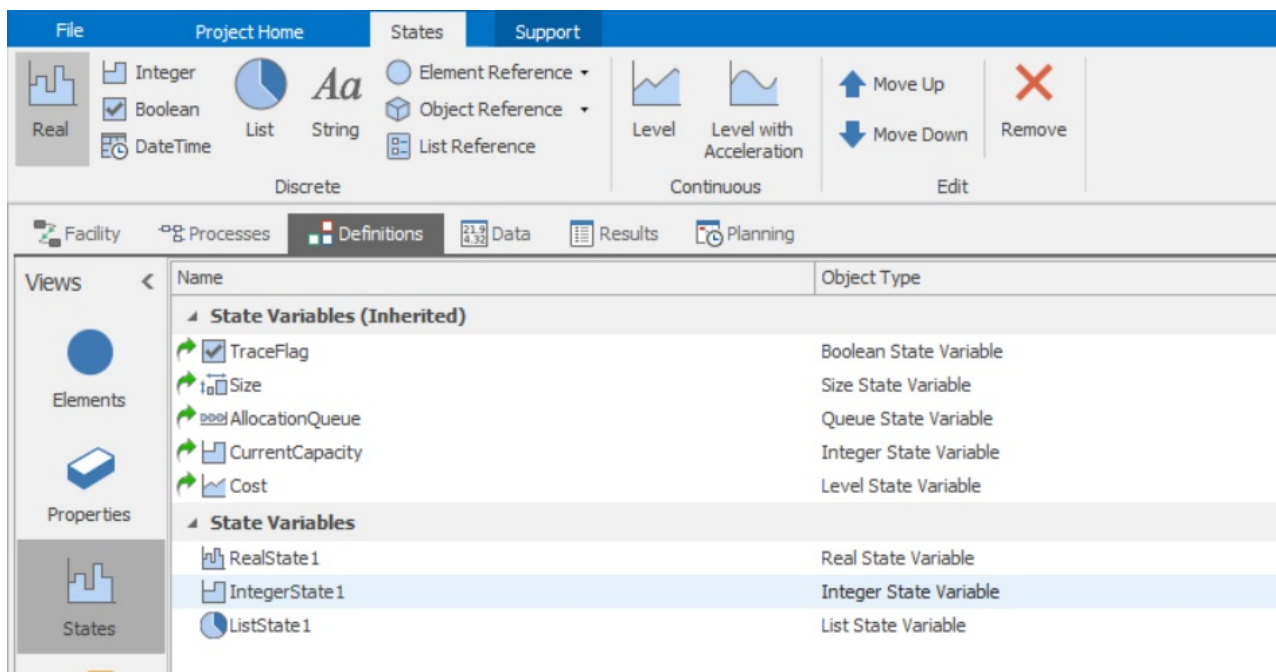
States are categorized as either Inherited or not. Inherited states automatically are generated with a given object. Other states are added by the user. The user interface of the States window allows the user to sort and/or filter the columns (Name, Object Type and Display Name). To the right of each column name is an arrow icon for sorting and a filter icon for filtering. This allows users to more easily organize an object's states when there are many defined.

Discrete States

- A **Real** state variable represents a real numeric value that may change by assignment logic at discrete times during a model run.
- An **Integer** state variable represents an integer numeric value that may change by assignment logic at discrete times during a model run.
- A **Boolean** state variable represents a boolean (true or false) value that may change by assignment logic at discrete times during a model run.
- A **DateTime** state variable represents a datetime value that may change by assignment logic at discrete times during a model run.
- A **List** state variable represents an integer variable with a discrete set of possible values from 0 to N. A zero-based indexed string list is used to define the possible integer values for the state. A list state's value may be changed by assignment logic at discrete times during a model run, and the state will automatically collect and report time-persistent statistics for each of its state values. For more information, see [List States](#).
- A **String** state variable represents a string value that may change by assignment logic at discrete times during a model run. For more information, see [String States](#).
- An **Element Reference** state variable defines an element reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to an element reference state using an Assign step in a process. The expression used as the value for the assignment must return an element reference.
- An **Object Reference** state variable defines an object reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to an object reference state using an Assign step in a process. The expression used as the value for the assignment must return an object reference.
- A **List Reference** state variable defines a list reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to a list reference state using an Assign step in a process. The expression used as the value for the assignment must return a list reference, for example from an Object List Property or another List Reference State.
- A **Stock Quant Reference** state variable defines a material stock quant reference that may be changed by assignment logic at discrete times during a model run. A stock quant represents a stock of specific material with the same characteristics in a storage bin. You can assign a value to stock quant reference state using an Assign step in a process. The expression used as the value for the assignment must return a Stock Quant reference.
- A **Stock Reservation Reference** state variable defines a material stock putaway or removal reservation reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to stock reservation reference state using an Assign step in a process. The expression used as the value for the assignment must return a Stock Reservation reference.

Continuous States

- A **Level** continuous-change state variable defines a numeric variable that may change continuously over time based on the current value of its rate. An automatic rate variable, based on the LevelStateName.Rate can be used to modify the rate of change throughout the simulation run. The state's rate parameter is a discrete-change value that may be changed by assignment logic at discrete times during a model run.
- A **Level with Acceleration** continuous-change state variable defines a numeric variable that may change continuously over time based on the current value of its rate and acceleration. The state's rate, acceleration and acceleration duration parameters are discrete-change values that may be changed by assignment logic at discrete times during a model run.



A State might represent a count of completed parts, the status of a machine selected from an enumeration state list, the temperature of an ingot heating in a furnace, the level of oil in a ship being filled, or the accumulation level on a conveyor belt.

A Real or Integer state has a property called *Unit Type*. This classifies the units of the values stored in this state variable. The choices are Unspecified, Time, TravelRate, Length, Currency and CurrencyPerTimeUnit. Time units available in Simio are Hours, Minutes, Seconds, Days and Weeks. Travel Rate units available in Simio are Meters Per Hour, Meters Per Minute, Meters Per Second, Kilometers Per Hour, Feet Per Second, Feet Per Minute and Miles Per Hour. Length units available in Simio are Meters, Kilometers, Centimeters, Inches, Feet, Yards and Miles. Currency units include over 100 world-wide currency units.

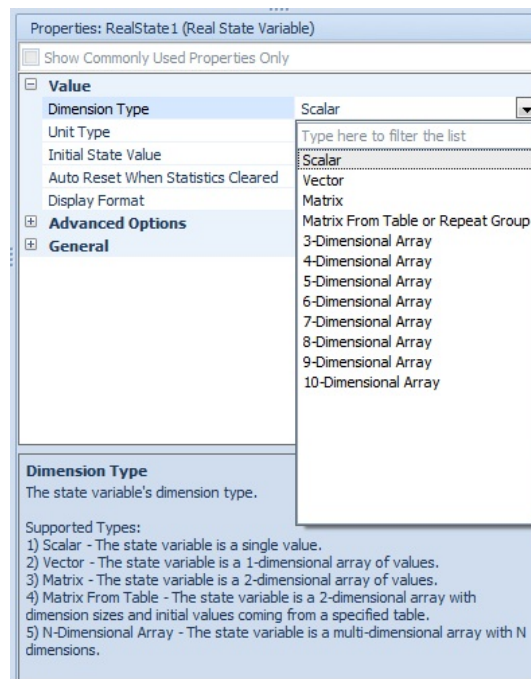
A State variable is assigned a value using an [Assign Step](#) in a process. Statistics can be recorded on state variables by creating a [StateStatistic](#) element in the [Elements Window](#).

Every object in Simio has States that are inherited from its base class. To learn more about inherited States on each type of object, see the following Functions, States and Events help pages for [Entity](#) object, [Node](#) objects, [Link](#) objects, [Transporter](#) objects and [Resource](#) objects.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for all States. Please refer to the [User Interface](#) for more details.

State Arrays

All reference states, with the exception of a List state, can become arrays or a matrices by changing the *Dimension Type* property. If the *Dimension Type* property is set to 'Scalar', the State does not have a dimension and is therefore a scalar Discrete State. If the *Dimension Type* property is set to 'Vector', the State is 1 dimensional and the *Rows* property will determine the number of rows in the array. If the *Dimension Type* property is set to 'Matrix', the state is a 2-dimensional matrix and the *Rows* and *Columns* properties will determine the number of rows and columns in the matrix. The *Dimension Type* property may also be set to 'Matrix From Table or Repeat Group' meaning that there is an associated table or repeat group from which the 2-dimensional array is sized and initialized. Finally, up to 10-dimensional arrays may be specified by selecting the appropriate *Dimension Type* property, as shown below.



In order to reference a State Array, use the syntax `StateArray[2]`, where `StateArray` is the name of the State variable and 2 is the value of the row that is referenced. A multi dimensional State Array can be referenced similarly, such as, `StateArray[2,3]` where 2,3 is the index to the matrix in this 2 dimensional array. State Array indexing is 1 based.

In order to initialize a State Array, the user should bind the array to a Data Table or Repeat Group. When bound, the array size will be initialized from the number of rows and numeric columns in the table or repeat group, and the array data will be initialized from the data in the table or repeat group. To bind an array to a table or repeat group, set the *Dimension Type* property of the state to 'Matrix From Table or Repeat Group' and select the appropriate table or repeat group from the drop down of the *Table or Repeat Group Name* property.

Listed below are the properties of **States**:

Property	Description
Dimension Type	The state variable's dimension type, see the above section titled State Arrays for more information.
Rows	The size of dimension 1.
Columns	The size of dimension 2.
Table or Repeat Group Name	The table or repeat group from which the array is sized and initialized (used for 'Matrix from Table or Repeat Group' <i>Dimension Type</i>). The array is given the same number of rows as the table and a column for each numeric column (number, expression, state reference) in the table. Numeric values are then copied from the table to the array. Expressions are evaluated to get their numeric value and the state references get the value from the state.
Dimension 3-10 Size	The size of dimension 3-10.
Unit Type	Classifies the units of the values stored in this state variable.
Initial State Value	The initial value for this state.
String List Name	The name of the string list that defines the list state's discrete set of possible integer values and their associated labels. Note that the integer equivalent of each value label in the string list will be the zero-based index into the list.
Automatic Assignment Type	Indicates whether automatic assignments to this list state will be performed by the simulation engine, with each possible list state value being associated with an autostate.
Resource	Indicates the method used to calculate capacity utilization statistics for the resource object

Utilization Calculation	containing this list state.
Auto Reset When Statistics Cleared	Indicates whether or not this state variable will be automatically reset to its initial state value(S) whenever statistics for the state's parent object or element are cleared. This could be from warm-up Period in an experiment or from using the ClearStatistics step.
Display Format	The optional format specifier used when displaying this state's value. Specifiers includ N or F followed by an optional number of decimal places. If left blank, the format will be N4 (Number, 4 digits after the decimal point).
Unit Type Property	The name of the property whose value will be used to determine the units classification of this state.

Queue States

A Queue State holds a list of objects or tokens. Queue states are used in such things as [Station](#) and [Storage](#) elements. Queue states in Storage elements can be modified by using the [Insert](#) and [Remove](#) step to add and remove objects from them. There are a number of [functions](#) that can be used with Queue States.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

List States

List States

There are two types of List States including Automatic List States and User Defined List States. There are also functions available to use with List States.

Automatic List States for Standard Library - Server, Combiner, Separator, Resource, Vehicle, Worker and Workstation (Deprecated)

With many of the Standard Library objects, there is an automatic List State that is updated during the simulation, namely the 'ObjectName.ResourceState'. This list state references a string list named 'ResourceStateName' and the states are automatically assigned as 'ResourceAutoStates'; therefore, no user assignments to the list state are required.

These automatic resource states have statistics calculated for the number of occurrences and percentage of time spent in each given state. The [Status Pie](#) may be used to graphically display the various states of an object. A Status Pie may be added by highlighting the desired object, i.e., Server1, and clicking on the Status Pie on the Symbols tab. Specify the *Data Type* property as 'ListState' and the *List State* property is 'ResourceState' as selected from the pulldown.

Below are the contents of the string list 'ResourceStateName' for the various objects and the object states that may be assigned.

Server, Combiner and Separator

'ResourceStateName' string list for the Server, Combiner and Separator object 'ResourceState' list state and the numerical equivalents include:

- Starved (0)
- Processing (1)
- Blocked (2)
- Failed (3)
- Offshift (4)
- FailedProcessing (5)
- OffshiftProcessing (6)
- Setup (7)
- OffshiftSetup (8)

Resource

'ResourceStateName' string list for the Resource object 'ResourceState' list state and the numerical equivalents include:

- Idle (0)
- Busy (1)
- Failed (3)
- Offshift (4)
- FailedBusy (5)
- OffShiftBusy (6)

Vehicle

'ResourceStateName' string list for the Vehicle object 'ResourceState' list state and the numerical equivalents include:

- Idle (0)
- Busy (1)
- Failed (3)
- Offshift (4)
- FailedBusy (5)
- OffShiftBusy (6)
- Transporting (7)
- FailedTransporting (8)
- OffShiftTransporting (9)

Worker

'ResourceStateName' string list for the Worker object 'ResourceState' list state and the numerical equivalents include:

- Idle (0)
- Busy (1)
- Blocked (2)
- Failed (3)
- Offshift (4)
- FailedBusy (5)
- OffShiftBusy (6)
- Transporting (7)

- FailedTransporting (8)
- OffShiftTransporting (9)

Workstation (Deprecated)

'ResourceStateName' string list for the Workstation object 'ResourceState' list state and the numerical equivalents include:

- Starved (0)
- Processing (1)
- Blocked (2)
- Failed (3)
- Offshift (4)
- Setup (5)
- Teardown (6)
- WaitingForSecondaryResource (7)
- WaitingForMaterial (8)

Automatic List States for Flow Library - Tank, ItemToFlowConverter, FlowToItemConverter, ContainerEntity, Filler, Emptier

As with the Standard Library objects, many of the Flow Library objects also include automatically updated states. For the flow objects with a Container element, there is an automatic List State that is updated during the simulation, namely the 'ObjectName.FillStatus'. Each of these objects has an internal FlowContainer element and the states are automatically assigned as 'ContainerAutoStates' associated with that FlowContainer element; therefore, no user assignments to the list state are required. The Filler and Emptier objects have an automatic List State that is updated during the simulation, namely the 'ObjectName.ResourceState'. This list state references a string list named 'ResourceStateName' and the states are automatically assigned as 'ResourceAutoStates'; therefore, no user assignments to the list state are required.

These automatic container states have statistics calculated for the number of occurrences and percentage of time spent in each given state. The [Status Pie](#) may be used to graphically display the various states of an object. A Status Pie may be added by highlighting the desired object, i.e., Tank1, and clicking on the Status Pie on the Symbols tab. Specify the *Data Type* property as 'ListState' and the *List State* property is 'FillStatus' as selected from the pulldown.

Below are the ContainerAutoStates for the various objects and the object states that may be assigned.

Tank, ItemToFlowConverter, FlowToItemConverter, ContainerEntity

The 'FlowContainer' within each of the objects has the following list states and the numerical equivalents:

- Empty (0)
- PartiallyFull (1)
- Full (2)
- Cleaning (3) - not included in ContainerEntity"

Below are the contents of the string list 'ResourceStateName' for the various objects and the object states that may be assigned.

Filler, Emptier

'ResourceStateName' string list for the Filler and Emptier object 'ResourceState' list state and the numerical equivalents include:

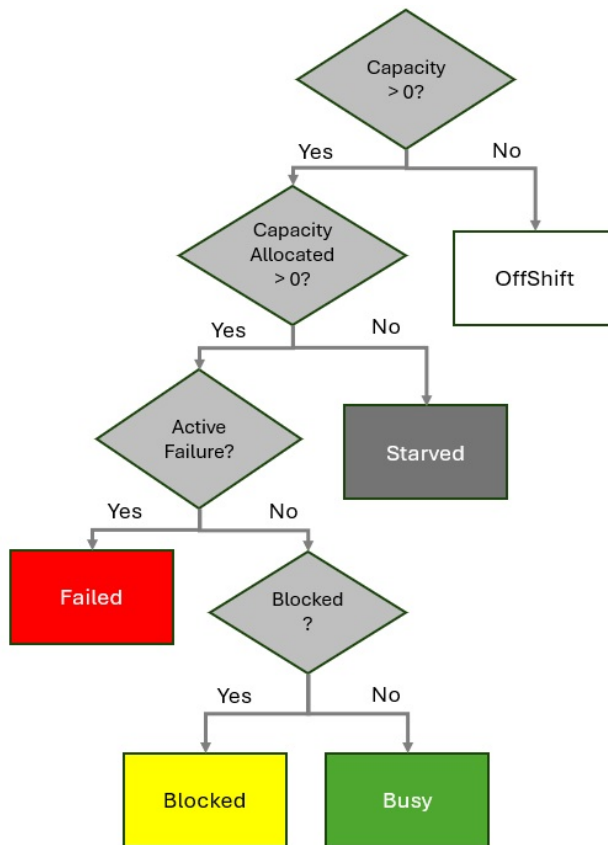
- Starved (0)
- Processing (1)
- Blocked (2)
- Failed (3)
- Offshift (4)
- FailedProcessing (5)
- OffshiftProcessing (6)

Resource Auto States Overview

A resource's current auto state is updated whenever the result of one of the auto state conditions has changed.

Auto State Condition Descriptions

Condition	Description
Capacity > 0?	Is true if the resource's capacity is greater than zero.
Capacity Allocated > 0?	Is true if the resource's capacity allocated is greater than zero.
Active Failure?	Is true if the resource has a child Failure element that is currently active. A Failure element is active when a Fail or Repair step is being used.
Blocked?	Is true if the resource has one or more child Station elements and the resource's current capacity is fully allocated to entities attempting to transfer out of those stations.



For example, if the resource's capacity and capacity allocated are both greater than zero and there is no active failure and the resource is not blocked, then the resource's current auto state is 'Busy'.

List States

A list state may be added to a resource-enabled object with *Automatic Assignment Type* set to 'ResourceAutoStates'.

The list state's *String Name* may be used to change an auto state's default name. For example, from 'Busy' to 'Processing'.

A list state's *String List Name* may be used to define alternative states for a specific auto state. For example, 'Setup' and 'Processing' states that are alternatives of the 'Busy' auto state.

The list state's current value may be manually changed using an Assign step.

The list state's current value will be automatically assigned to match the resource's current auto state. If there are alternative states for the new auto state, then the list state's current value will be automatically assigned to the same alternative when the resource was last in that same auto state.

For example, the resource has 'Setup' and 'Processing' states that are alternatives of the 'Busy' auto state. Suppose the resource's list state has first automatically changed from 'Setup' to 'OffShift' because the resource's auto state changed to 'OffShift'. When the off-shift period ends and the resource's auto state has changed back to 'Busy', the list state's current value is automatically changed back to 'Setup' as that was the alternative state assigned when the resource was last 'Busy'.

A list state will automatically collect and report time- persistent occurrence statistics for each assigned state value.

User Defined List States

The user may define one or more List States for a model under the [States](#) panel in the Definitions window. The List State references a String List that is defined in the [List](#) panel in the Definitions window. This is done by specifying the *List Name* property as 'MyStringListName', and *Automatic Assignment Type* as 'None'. The *Initial State Value* must be a value from the String List and should represent the state of the model when it initially starts running.

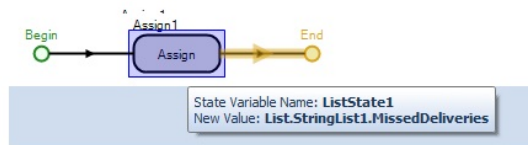
The String List should include any possible states that the user may want to assign to the List State using the [Assign](#) step. There are no automatic assignments made during the simulation, as with the above automatic ResourceStates. Thus, the Assign step in the Processes Window should be used to change the List State value from one string value to another. Statistics can be categorized automatically (default) or by using the Results Classification section of properties.

Assignments to the List State in the Assign step are made by specifying the *State Variable Name* property as 'ListStateName' and the *New Value* property as 'List.ListName.Value', similar to referencing a list elsewhere in the model.

User Defined List State using Lists, States and Assign step

Name	Object Type	Display Name
▼ Strings		
StringList1	Strings	StringList1
.....		
String		
OnTimeDeliveries		
MissedDeliveries		
▶		

String List in Lists panel



Assign step in Processes window to Assign the ListState1 to one of the values within the string list (MissedDeliveries)

Views	Name	Object Type	Display Name
States (Inherited)			
▼ States			
ListState1	List State Variable	ListState1	

Properties: ListState1 (List State Variable)	
Show Commonly Used Properties Only	
Value	
String List Name	StringList1
Automatic Assignment Type	None
Initial State Value	OnTimeDeliveries
Results Classification	
Data Source	
Category	
Data Item	
General	
Name	ListState1
Description	
Public	True

List State within States panel references StringList1

User defined List States will automatically generate statistics for the model (not specific object) on the number of occurrences and percentage of time the List State has a value for each of the given string values. The List State can also be graphically displayed by using the Status Pie found on the Animation tab of the ribbon.

List States Functions

The following functions are available for List States:

AverageTime(stateValue)
 NumberOccurrences(stateValue)
 PercentTime(stateValue)
 TotalTime(stateValue)

Therefore, to reference the percentage of time that an object was in a certain state, use the syntax [ObjectName].PercentTime(stateValue), where stateValue is the number referencing the state in the String List. The first value in the String List is value 0.

Utilization Statistics

The ResourceState list states provide time-persistent statistics for many of the objects in the simulation system. In addition to tracking these utilization statistics for the Results page, the ResourceStates can also be used during the simulation run.

Because Simio automatically tracks the list states for the various objects, the information is available to track resource utilization statistics during a simulation run to be displayed with a status label. While a status pie graphic for an object's list state can give you a pie chart of the various states that the object is in, using the object's list state in an expression can show the percentage of time in each state.

For example, the expression 'Server1.ResourceState.PercentTime(1)' utilizes the resource state for Server1 to determine the percentage of time that the server has the state value of 1 (which is Processing, as shown in the above section). Likewise, 'Vehicle1[1].ResourceState.PercentTime(0)' would show the percentage of time that the first unit of Vehicle1 has the state value of 0, which is Idle.

It may be desirable to determine the percentage of time in multiple states for an object. For example, if a worker is used for transporting tasks as well as stationary tasks, use the expression 'Worker1[2].ResourceState.PercentTime(1)+Worker1[2].ResourceState.PercentTime(7)' to display the percentage of time that the second unit of Worker1 is either busy at a stationary task or busy transporting or moving between nodes.

Refer to the SimBits [ExamplesOfFunctions_StaticObjects](#) and [ExamplesOfFunctions_DynamicObjects](#) for examples of using and displaying various functions with static objects(Source, Sink, Path, Server, and Resource) and dynamic objects (Entity, Vehicle, Worker).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

String States

String Expression Functions Available

There are a number of functions that are available when utilizing strings in an expression.

Listed below are the **String Expression** Functions available:

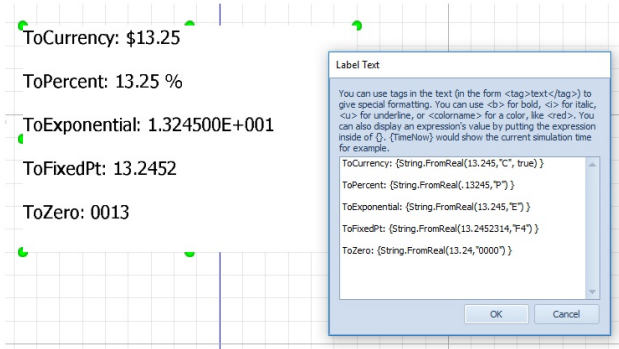
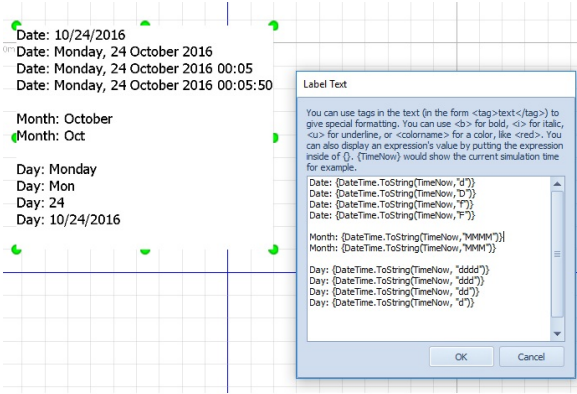
Function	Return Value
String.Length(string)	returns the length of a string
String.Contains(string, substring)	return true if substring is in string, otherwise false
String.Substring(string, startindex, length)	returns a substring of string starting at startindex and of the specified length
String.FromReal(value)	returns a string representation of a real value (String.FromReal(6) turns into "6")
String.ToReal(string)	returns a real value representation of a string (String.ToReal("6.2") turns into 6.2)
String.FromDateTime(value)	returns a string representation of a datetime value based on the start date and time of the simulation (String.FromDateTime(6) turns into "October 1, 2010 06:00:00" if the start date of the simulation is "October 1, 2010 00:00:00")
String.ToDateTime(string)	returns a datetime representation, in hours, of a string based on the start date and time of the simulation model (String.ToDateTime("October 11, 2010") turns into 0 if that is the start date of the simulation, 24 if the start date is October 10, -24 if the start date is October 12, etc.)
String.Format(string, arg1, arg2,...)	returns a new string formatted with the specified format string and arguments. We follow the .NET string formatting syntax here: String.Format("Hello {0}, you are {1} years old", "Friend", 30) turns into "Hello Friend, you are 30 years old"
String.Compare(string1, string2)	returns 0 if the strings are equal, -1 if string1 < string2, 1 if string1 > string2
String.CompareIgnoreCase(string1, string2)	same as above, but ignores casing in comparison, so String.Compare("fred", "FRED") does not return 0, but String.CompareIgnoreCase("fred", "FRED") does return 0.
String.IndexOf(string, substring)	Returns the one-based index of the first occurrence of a specified substring within the string. If the substring is not found, then the value 0 is returned.

Simio supports "+" for strings, so "Hello" + " World" turns into "Hello World". Also supported are the logical comparison operators such as ==,>, etc. For example, the expression MyStringstate=="Red" and MyStringState > "A" are valid for use in a Decide step.

See the [StringStates](#) SimBit to see an example of using String States in Simio.

The following links have additional information on .NET standard and custom formatting of datetime and real values, including [.NET Standard Date and Time Format Strings](#), [.NET Custom Date and Time Format Strings](#), [.NET Standard Numeric Format Strings](#), and [.NET Custom Numeric Format Strings](#).

See also below examples of formatting within a Simio floor label.



Reference States

Element Reference States

An Element Reference state variable defines an element reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to an element reference state using an Assign step in a process. The expression used as the value for the assignment must return an element reference.

Element Reference states can be arrays. An element reference state that is defined as a scalar is referenced as 'ElementReferenceStateName'. An element reference state that is defined as a vector is referenced as 'ElementReferenceStateName[rowindex]'. An element reference state that is defined as a matrix is referenced as 'ElementReferenceStateName[rowindex, columnindex]'. Refer to the [States](#) page section on arrays for more information.

If an element reference state has no current value (null), it returns the keyword 'Nothing'. Users may wish to use this keyword in a comparison statement to check whether an element reference exists (e.g., Model.ElementReferenceValue == Nothing).

Element Reference States

Batch Logic
Changeover Logic
Constraint Logic
Container
Cost Center
Failure
Inventory
Material
Monitor
Network
Neural Network
Output Statistic
Periodic Statistic
Process
Regulator
Routing Group
State Statistic
Station
Storage
Tally Statistic
Task Sequence
Timer

The types of element reference states and how they may be used are described below:

- **Element** - This generic element reference can be used to refer to any of the below element types.
- **Activity** - This refers to the name of an activity and may be used within the Operation element.
- **BatchLogic** - This refers to the batchlogic element and may be used within the Changeover step's *Changeover Logic Name* property. This may be useful when various entity types require different changeover logic information that is

then stored within the ChangeoverLogic element reference state.

- **ChangeoverLogic** - This refers to the changeoverlogic element and may be used within the Batch step's *BatchLogic Name* property. This may be useful when various entity types require different batching information that is then stored within the BatchLogic element reference state.
- **Container** - This element reference state is used to store a container name. Containers are utilized within the Flow Library's Tank object.
- **CostCenter** - This element reference state is used to store the name of a Cost Center.
- **Failure** - This element reference state is used to store failure names. An object may wish to utilize this state within the Fail and/or Repair steps within the Processes window.
- **Material** - This refers to a material element name and maybe be used in conjunction with the AddStock, RemoveStock, ReserveBins, Produce, Consume, or Create steps, or within the Source's Stock Requirements repeat group.
- **Monitor** - This refers to the name of a monitor element. An entity may have a monitor element reference state that tracks a particular state of the entity.
- **Network** - This refers to a network name. Entities may utilize a network element reference state to store individual network names that may later be used in a SetNetwork step in the Processes window.
- **Neural Network** - This refers to an elemnt used to integrate a neural network regression model into simulation logic.
- **Operation** - This refers to an operation name and may be used within the StartOperation and EndOperation steps within the Processes window.
- **OutputStatistic** - This refers to an output statistic name and may be used within a table that store statistics on various states.
- **Process** - This refers to a process name within the Processes window. Entities may have a state variable that stores a different process name for each entity type, where within a Server for processing, the element reference state is used within an add-on process property to point to different processes to undergo.
- **Regulator** - This refers to a regulator element name. Regulators are used within the Flow Library's FlowNode (and therefore Tank, FlowSource and FlowSink).
- **Routing Group** - This refers to a routing group name that may include various nodes in which the object will go through. Routing groups refer to node lists. The routing group element reference would then be used within the Route step as the *Route Group Name* property.
- **StateStatistic** - This refers to the name of a state statistic element. One application of a state statistic element reference would be within a table to store the various state statistics.
- **Station** - This refers to the name of a station element. An entity may have a station element reference and utilize that variable within the Transfer step in the Processes window to transfer to a given *Station Name* based on the element reference state.
- **Storage** - This refers to a storage element name and may be used within the Insert or Remove steps as the *Queue State Name* property.
- **Task Sequence** - This element reference state refers to a task sequence element. An entity may have a reference to the task sequence element and utilize it within the StartTasks step as the *Task Sequence Name* property.
- **Tally Statistic** - This element reference state refers to a tally statistic element. An entity may have a reference to the tally statistic element and utilize it within the Tally step as the *Tally Statistic Name* property.
- **Timer** - This refers to a timer element name. This type of reference may be useful within a table, for example.
- **Path Planner** - This refers to the name of a path planner element and may be used in conjunction with the Transfer step or the BasicNode and TransferNode objects.
- **Storage Area** - This refers to the name of a storage area element and may be used in conjunction with the Storage Area element, ReserveBins step, or ReserveStock step.
- **Storage Bin** - This refers to the name of a storage bin element and may be used in conjunction with the Storage Area element, AddStock step, or ReserveStock step.

With the above element reference states, the state can be referenced by simply using the object name and element reference name. For example, if you have a tally statistic element reference state on ModelEntity, then it is referenced 'ModelEntity.TallyStatisticStateName' within the Tally step in the Processes window. Likewise, if you have a Process reference state named 'P1' on an entity, it would be referred to as 'ModelEntity.P1' within a step or property requiring a process name.

Object Reference States

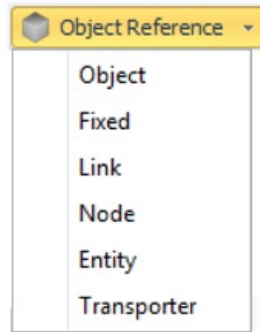
An Object Reference state variable defines an object reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to an element reference state using an Assign step in a process. The expression used as the value for the assignment must return an object reference.

Object Reference states can be also be arrays. An object reference state that is defined as a scalar is referenced by

'ObjectReferenceStateName'. An object reference state that is defined as a vector is referenced as 'ObjectReferenceStateName[rowindex]'. An object reference state that is defined as a matrix is referenced as 'ObjectReferenceStateName[rowindex, columnindex]'. Refer to the [States](#) page section on arrays for more information.

If an object reference state has no current value (null), it returns the keyword 'Nothing'. Users may wish to use this keyword in a comparison statement to check whether an object reference exists (e.g., ModelEntity.ObjectReferenceValue == Nothing).

Object Reference States



Object reference states may also be used with functions for various objects. The types of object reference states and how they may be used are described below:

- **Object** - This general purpose object reference can be used for any of the below object reference states. An object reference simply means that the state is some type of object.
- **Fixed** - This general purpose object reference can be used for fixed type object reference states. Standard Library objects such as Server, Combiner, and Separator are fixed type objects.
- **Link** - This refers to a link reference and may be used to access information related to a particular link, such as a conveyor or path.
- **Node** - This object reference state is used to reference a node name. An entity may wish to reference a node within the Transfer step or any object's property that requires a node name reference. The reference can also be used to get function information on the node.
- **Entity** - This refers to an entity reference and may be used to access function information related to entities.
- **Transporter** - This refers to a transporter name. Entities may utilize a transporter reference state to store a worker or vehicle name. This type reference may also be used with transporter type functions.

With the above object reference states, if the object is clearly known (i.e., entity, transporter), then the state can be referenced by simply using the object name and object reference name. For example, if you have a transporter object reference state on ModelEntity, then it is referenced 'ModelEntity.TransporterStateName' within a *Transporter Name* property.

As with the element reference states, it is important to note that if you have an Object type reference state, which is more 'generic', you may need to explicitly cast it to a reference of some type. For example, if you have an Object reference state called Object1, and you wish to utilize this state variable to store a Transporter object reference, you would reference 'ModelEntity.Object1.Transporter'. Whether or not you need to include the extra 'type' of reference depends upon the type of property field where it is being utilized. If a property asks for Object Name, for example, then an object reference state can be utilized without the extra 'type'. With object references, this 'typing' is required because an object is a more 'generic' type of reference state and the various properties in which object reference states may be utilized may require the reference to be of a specific type.

Please refer to the two SimBits on Object Referencing, [ObjectReferenceOnEntity](#), and [SeizingSameResourceFromList](#) for examples.

List Reference States

A List Reference state variable defines a list reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to a list reference state using an Assign step in a process. The expression used as the value for the assignment must return a list reference, for example from an Object List property or another List Reference State.

If a list reference state has no current value (null), it returns the keyword 'Nothing'. Users may wish to use this keyword in a comparison statement to check whether a list reference exists (e.g., ModelEntity.MyWorkerList == Nothing).

Stock Quant Reference States

A Stock Quant Reference state variable defines a material stock quant reference that may be changed by assignment logic at discrete times during a model run. A stock quant represents a stock of specific material with the same characteristics in a storage bin. You can assign a value to stock quant reference state using an Assign step in a process. The expression used as the value for the assignment must return a Stock Quant reference.

If a stock quant reference state has no current value (null), it returns the keyword 'Nothing'. Users may wish to use this keyword in a comparison statement to check whether a list reference exists (e.g., `Model.StockQuantReferenceValue == Nothing`).

Stock Reservation Reference State

A Stock Reservation Reference state variable defines a material stock putaway or removal reservation reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to stock reservation reference state using an Assign step in a process. The expression used as the value for the assignment must return a Stock Reservation reference.

If a stock reservation reference state has no current value (null), it returns the keyword 'Nothing'. Users may wish to use this keyword in a comparison statement to check whether a list reference exists (e.g., `Model.StockReservationReferenceValue == Nothing`).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Events

Events

An Event is a notification that can be given by one object and responded to by several. An Event is added to a model from the Events panel within the Definitions window. An Event is fired with a [Fire](#) step in a process. A Process waits for a fired event by having its *Triggering Event* property set to the name of the event that it is waiting for. A [Wait](#) step can also be used to wait for an event to be fired. The [Subscribe](#) step can be used to dynamically add a triggering Event to a process.

For an example of adding new Events, please refer to the SimBit [UsingButtonsToAffectSystem](#).

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Events. Please refer to the [User Interface](#) for more details.

There are a number of events that are automatically fired by Simio. The following events are summarized by object type, where ObjectName is replaced by the name of the actual object:

Standard Server and Workstation (Deprecated) Objects

- ObjectName.CapacityChanged
- Input@ObjectName.CapacityChanged
- Input@ObjectName.Entered
- Input@ObjectName.Exited
- Output@ObjectName.CapacityChanged
- Output@ObjectName.Entered
- Output@ObjectName.Exited
- Output@ObjectName.RiderWaiting
- ObjectName.Failed
- ObjectName.Repaired

Note: Source and Sink objects are similar except that Source does not have an Input Node and Sink does not have Output Node. Combiner and Separator are also similar except that Combiner has both ParentInput and MemberInput and Separator has both ParentOutput and ParentInput.

Standard Resource Object

- ObjectName.CapacityChanged
- ObjectName.Allocated
- ObjectName.Released
- ObjectName.Failed
- ObjectName.Repaired

Standard Entity Objects

- ObjectName.CapacityChanged
- ObjectName.Destroyed
- ObjectName.Engaged
- ObjectName.Transferred
- ObjectName.Transferring

Standard Vehicle and Worker Objects

- ObjectName.CapacityChanged
- ObjectName.Destroyed
- ObjectName.Engaged
- ObjectName.Transferred
- ObjectName.Transferring
- ObjectName.RiderLoaded
- ObjectName.RiderUnloaded
- ObjectName.Allocated
- ObjectName.Released
- ObjectName.RemainInPlaceEnded

-
- ObjectName.MinimumDwellTimeExpirationReset
 - ObjectName.Failed ** Vehicle only
 - ObjectName.Repaired ** Vehicle only

Standard Node Objects

- ObjectName.Entered
- ObjectName.Exited
- ObjectName.RiderWaiting ** TransferNode only

Standard Link Objects

- ObjectName.CapacityChanged
- ObjectName.Failed ** Conveyor only
- ObjectName.Repaired ** Conveyor only

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions

Functions

A function is a query of a calculated value that can be called by another object. To query for the value of a function, use it in an expression.

Any functions defined within this window are then listed under the 'User Functions' section of the Watch window during a simulation run.

Functions are most useful for sharing an expression across several objects. For example if you have a complex expression for calculating a processing time, and you use that expression in several different places, you can instead make that expression a function. This can make your model easier to maintain, since you can now update the expression in just one place and the new value will be returned everywhere that uses the function.

You can also use functions simply to make your model easier to read. Rather than having the complex expression directly referenced in your model, you can define the expression as a function, and give that function a meaningful name.

By combining the two above features, you can simplify routine use of your model. Say, for example, you had two different common ways of calculating processing time. To avoid typing and retyping the appropriate expression each time you change, you could define functions named "PTime_MethodA" and "PTime_MethodB" with the appropriate expressions and just change between those names as desired.

Listed below are the properties of **Functions**:

Property	Description
Expression	The expression to evaluate for the value of this function.
Return Type	The type of the result returned from the expression. Return Type can be Number, String, ElementReference or Any.
Unit Type	The type of units for the value of this function. Available for Number and Any.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Lists

Lists

Lists are used to define a collection of strings, objects, nodes, or transporters. List are also used to define the possible changeover states for a [changeover](#) matrix (e.g. color, size, etc.), or to provide a list from which a selection is to be made (e.g. a resource to seize, transporter to select, etc.). A List is added to a model from the Lists panel within the Definitions window. It can also be added by right-clicking on an object(s) within the Facility window, as described below.

The members of list have a numeric index value that begins with 0. A List value may be referenced in an expression using the format **List.ListName.Value**. For example, if we have a list named Color with members Red, Green, Blue, Yellow, then List.Color.Yellow returns a value of 3. If we have a ListProperty (i.e. a property whose possible values are list members) named BikeColor we can test conditions like `BikeColor == List.Color.Yellow`.

One can also reference a List within another object. For example, if there is a list on a ModelEntity and you want to assign a value from this List to a ListState within the Model, you can use the syntax `ModelEntity.ListProperty1 == ModelEntity.List.MyStringList.Value1`, where Value1 is a value from MyStringList.

The `String.FromList(stringList, intConstant)` function can be used to return the string representation of a specified integer constant in the specified String List. The 'intConstant' input can be the value from a List State or Property.

For example, if a String List contains entries "Green" = 0, "Red" = 1, and "Blue" = 2, then `'String.FromList(StringListName, 2)'` will return 'Blue'.

Note: The StringListName must be just the Name and not List.Name. The keyword 'List' is only needed if you are trying to access the member of the list.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for all Lists. Please refer to the [User Interface](#) for more details.

String List

Defines a string list (e.g. Small, Medium, Large) that is used to define a list property that can have any value from the list, or the possible from-to states in a changeover table. Examples of using string lists is the SimBit

WorkstationWithSequenceDependentSetup. String lists must be defined in Definitions window for both the ModelEntity, as well as the Model.

String List Example

String list within the ModelEntity object as well as within the Model

Property of ModelEntity is named Color, which references the 'Colors' list

Selection Weight on the Path checks to see if the entity's color property is 'Red'

Object List

Defines an object list (e.g. Fred, Sue, Tom) referenced by steps (e.g. Search step) and objects (e.g. Server object) that allow for a selection of an object from a list of objects. Examples of using object lists include the SimBits [SelectingResourcesAsAGroup](#) and [SelectingResourceFromList](#).

Object List Example

Facility

Processes

Definitions

Views

Elements

Properties

States

Events

$f(x)$

Functions

Lists

Name

Objects

Mechanics

Object

Resource1

Resource2

*

Secondary Resources

Resource for Processing

Object Type

FromList

Object List Name

Mechanics

Selection Goal

Preferred Order

Request Move

None

Referring to the 'Mechanics' Object List from within the Server

Node List

Defines a node list (e.g. Input@Drill, Input@Lathe, MergePoint) that is used by the TransferNode step and the RoutingGroup element to dynamically route an entity to a node based on the current state. There are many examples of using the node list including the SimBits [SelectServerWithShortestLine](#), [OneQueueForMultipleServers](#) and [UsingAStorageQueue](#).

Node List Example

Facility

Processes

Definitions

Views

Elements

Properties

States

Events

$f(x)$

Functions

Lists

Name

Nodes

Servers

Node

Input@Server1

Input@Server2

Input@Server3

Routing Logic

Outbound Link Preference

Any

Outbound Link Rule

Shortest Path

Entity Destination Type

Select From List

Node List Name

Servers

Selection Goal

Smallest Value

Selection Expression

Candidate.Node.AssociatedStationOverload

Selection Condition

Blocked Destination Rule

Select Available Only

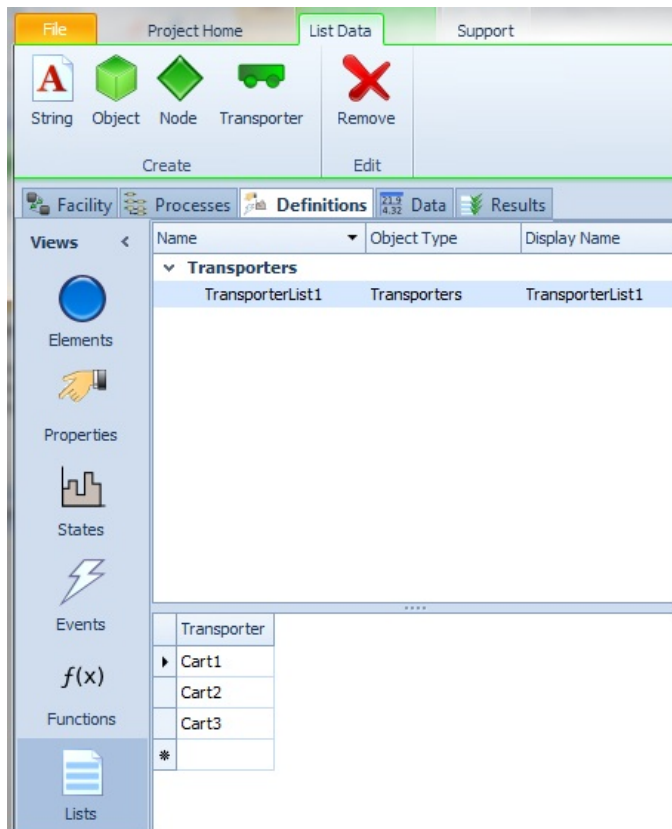
Routing Logic within a Transfer Node that selects from Node List Name 'Servers'

Transporter List

Defines a transporter list (e.g. Cart1, Cart2, Cart3) that is used by steps (e.g. Ride step) and objects (e.g. TransferNode) that

allow for the selection of a transporter from a list of transporters.

Transporter List Example



Transporter List in Model

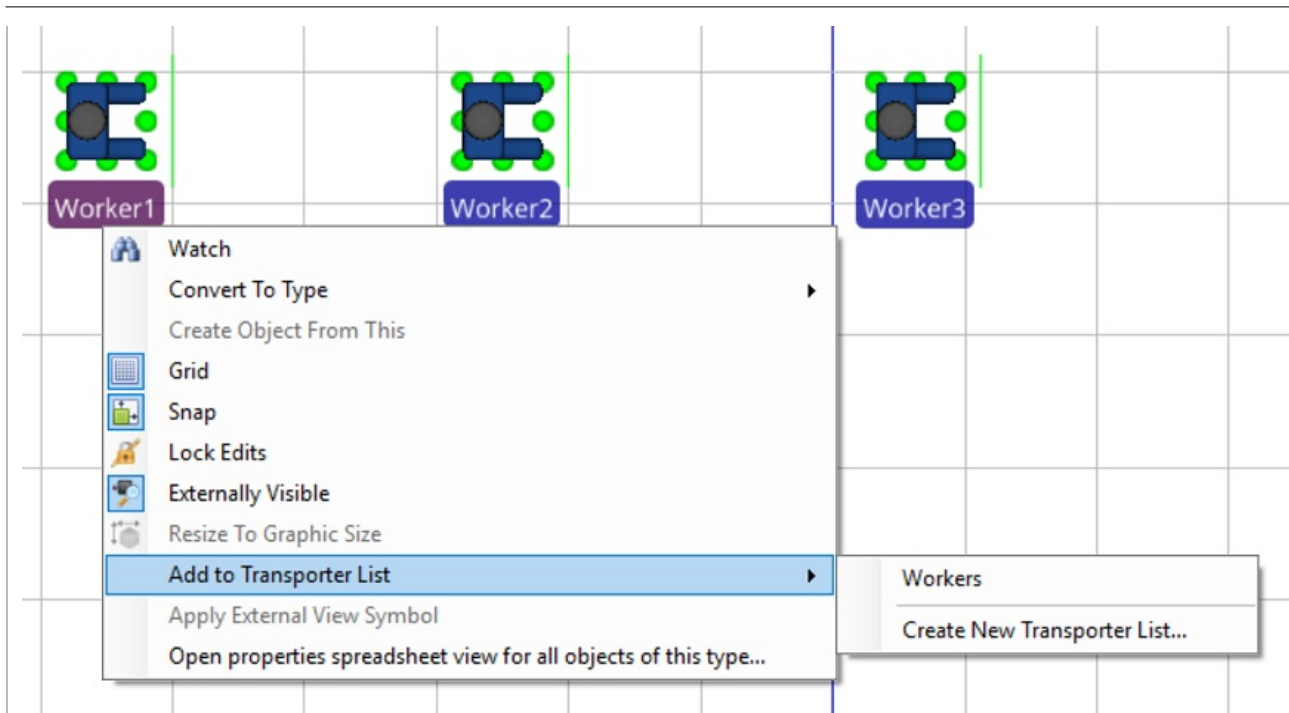
Transport Logic	
Ride On Transporter	True
Transporter Type	From List
Transporter List Name	TransporterList1
Reservation Method	Reserve Closest
Selection Goal	Preferred Order
Selection Condition	

Within the TransferNode, the Transporter that is selected will be based on the list 'TransporterList1' as defined to the left

Adding Objects to Lists from the Facility Window

In addition to adding members to a List from the Definitions window, Lists panel, user's also have the option to select a single object or group select multiple objects (using Ctrl-click to select multiple objects or Ctrl-click and drag to select an area). Once an object or group of objects is selected, the right-click menu includes an 'Add to List' option as shown below. If only nodes are selected, the option to 'Add to Node List' will be shown. Likewise, if only transporters are selected, the option to 'Add to Transporter List' will be shown. If multiple object types are selected, the more generic 'Add to Object List' is displayed. Objects can be members of multiple lists. Once on a list, the right-click menu provides a 'Remove from List' option as well. The right-click menu provides existing list names as well as a 'Create New List' option, where the user is then prompted for a new list name.

Right-Click Menu to Add Objects to a List from the Facility Window



Note: Resource, Transporter, and Node lists can also be defined in a table column. Read more in [Data Tables](#)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Tokens

Tokens

A Token executes the steps in a process flow. A Token may have one or more user defined states that carry information from Step to Step. A Token may also reference an associated object such as an [Entity](#) that is visiting the parent object. Simio provides you with a default token named Token that you can use for most of your processes. The function `ProcessName.TokensInProgress.NumberItems` returns the number of tokens currently executing a given process.

Every token executing a process in Simio has an associated object and a contextual object (ContextObject). For a token that has been created by a Create or Search step for example, the token's associated object reference is the created entity or found object, while its contextobject reference is the object that was associated with the original token that executed the Create or Search step.

If the token's process execution is part of a task sequence, the Task namespace will provide a set of functions (see table below) for accessing information about the task assigned to the token.

You only need to add a Token through the Token panel of the Definitions tab if you require a Token with one or more custom states or if you wish to use a Token as a counter in a process loop.

If you use a custom token with custom states, the states of the receiving token in the called process get set to match, much like passing function arguments. If you set the ReturnValue of the token in the executed process, that value will be set (returned) to the token that started the execute, much like a function return value. This allows the user to pass information between the processes, much like you would with a function. The [Execute](#) step is used to move from one process to another.

An example of why a user might want to create a custom Token is if they are using a Create step, with the *Create Type* set to 'NewObject', to create a new Entity and they would like the new entity to have one of the same States that the original Entity has. Before the Create step, the Entity State can be saved into a custom Token State. And then from the Created segment of the Create step, the Token State can be applied to a State on the new Entity.

If a token has been created to execute an *On Replenishment Order Process* specified for an Inventory element, then the destination in its material order detail will be the inventory site requesting the replenishment. Since built-in features for modelling sourcing policies will not yet be available, the source in the token's material order detail (`MaterialOrderDetail.SourceInventory` or `MaterialOrderDetail.SourceSiteObject`) will always be set to nothing.

If a token has been created to execute an *On Consumed Process* specified for an Consume step, then the source in its material order detail will be the inventory site from which that the material was consumed. The destination in its material order detail (`MaterialOrderDetail.DestinationInventory` or `MaterialOrderDetail.DestinationSiteObject`) will be copied from the material order detail of the original token, if applicable.

Listed below are the States of **Token**:

State	Description
ReturnValue	<p>A special-purpose, user-assignable numeric state value.</p> <p>If the token is executing a decision process, then this state variable is used to return a 'True' or 'False' value for the decision. It may also be used in the Search step to return the results of an expression evaluated across the searched collection.</p> <p>Additionally, if the token is waiting at an Execute step until a specified process is completed, then this state variable may be used to pass back a 'return value' from the other token that executed the separate process.</p>

Listed below are the Functions of **Token**:

Function	Description
Name	Returns the string name of this item.
Is	The "Is" keyword followed by a class or type name returns the value 'True' if the item is of that class or type. Otherwise the value 'False' is returned.

ContextObject	Returns the reference to the token's contextual object.
TimeCreated	Returns the time that this token was created in hours.
TimeInProcess	Returns the time duration (in hours) that this token has been in process.
AssociatedObject	Returns the object that this token is associated with.
MaterialOrderDetail	Returns a reference to the material order detail associated with the token, if applicable.
MaterialOrderDetail.Material	Reference to the material.
MaterialOrderDetail.SourceInventory	Reference to the inventory that is the source of the material.
MaterialOrderDetail.SourceSiteObject	Reference to the inventory site object that is the source of the material.
MaterialOrderDetail.DestinationInventory	Reference to the inventory that is the destination of the material.
MaterialOrderDetail.DestinationSiteObject	Reference to the inventory site object that is the destination of the material.
MaterialOrderDetail.Quantity	The quantity of the material.
MaterialOrderDetail.LotID	The lot identifier of the material.
Task	Returns a reference to the active task associated with the token, if the token's execution is part of a task sequence. Refer to the Task Sequence element for more information.
Task.AssociatedObject	Returns the associated object reference for the task.
Task.SequenceNumber	Returns a string representing the sequence number used to determine the task's precedence constraints.
Task.IDNumber	Returns the integer identifier number used to identify the task in the Immediate Predecessors or Immediate Successors field of another task.
Task.Name	Returns the name for the task.
Task.PrimaryToken	Returns a reference to the token that is executing the task's primary process.
Task.ExecutionID	Returns the unique integer identifier number automatically assigned to the task when the StartTasks step was executed.
Task.TimeStarted	Returns the simulation time (in hours) that the task was started.
Task.TimeInProcess	Returns the elapsed time duration (in hours) since the task was started.
Task.SeizedResources	Provides functions for accessing the resources currently seized by the task's associated object, filtered to only include the resource seizes that occurred specifically due to the task's execution.

Task.SeizedResources.NumberItems	Returns the number of resources currently seized by the task's associated object, filtered to only include the resource seizures that occurred specifically due to the task's execution.
Task.SeizedResources.FirstItem	Returns a reference to the first resource in the list of resources currently seized by the task's associated object, filtered to only include the resource seizures that occurred specifically due to the task's execution.
Task.SeizedResources.LastItem	Returns a reference to the last resource in the list of resources currently seized by the task's associated object, filtered to only include the resource seizures that occurred specifically due to the task's execution.
Task.SeizedResources.IndexOfItem(resource)	Returns the one-based index of the first occurrence of a specified resource in the list of resources currently seized by the task's associated object, filtered to only include the resource seizures that occurred specifically due to the task's execution. If the resource is not found then the value 0 is returned.
Task.SeizedResources.ItemAtIndex(index)	Returns the one-based index of the first occurrence of a specified resource in the list of resources currently seized by the task's associated object, filtered to only include the resource seizures that occurred specifically due to the task's execution. If the resource is not found then the value 0 is returned.
Task.SeizedResources.Contains(resource)	Returns True (1) if the specified resource is in the list of resources currently seized by the task's associated object, filtered to only include the resource seizures that occurred specifically due to the task's execution. Otherwise, the value False (0) is returned.
Task.SeizedResources.CapacityOwnedOf(resource)	Returns the total number of capacity units of a specified resource that are currently seized by the task's associated object, filtered to only include the resource seizures that occurred specifically due to the task's execution.
Task.SeizedResources.AggregateEfficiency(type)	<p>Calculates and returns an aggregate efficiency value for the list of resources currently seized by the token's associated object, filtered to only include the resource seizures that occurred specifically due to the token's execution.</p> <p>The aggregate type is an integer argument. Possible values: 0 = None, 1 = Average, 2 = Count, 3 = Maximum, 4 = Minimum, 5 = Sum.</p> <p>Note that if the aggregate type is None or if there are no seized resources with defined efficiency values, then the value NaN is returned.</p>
<p>NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Tokens. Please refer to the User Interface for more details.</p>	

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)





Tokens and Processes - Discussion and Examples

Token's Object References

- A token executing a process will have an associated object (AssociatedObject) and a contextual object (ContextObject). The token's associated object is the primary object reference and is often the entity/object that triggered the execution of the process. The context object is the secondary object associated with the token and is often the related object that owns the executed process. For example, if an entity triggers a Server's *Entered* Add-On Process Trigger, the entity is the associated object, and the Server is the context object. If a Process is triggered by a model's Timer, the Model will be the associated and context object.
- The token's object references can change as a process enacts logic. Create or Search steps might change the references on the Token that exits the second branch of this step.
- The token's object references can be changed and assigned as the process enacts logic. A Create step can be used with a *Create Type* of 'NewToken'. The *Token Associated Object* or *Token Context Object* can then be specified in this step.
- Some steps allow the *Owner Type* or *Object Type* to be specified on the *Advanced Options* property group. By default, this is typically the 'AssociatedObject'. But this could be change to the Token's Context Object or any other reference.
- A useful debugging tool to understand a Process Token's associated and contextual objects is a Notify step. In the Notify step, you can use the expression "Token.AssociatedObject.Name" or "Token.ContextualObject.Name" to read out this information during the run.

In this example, the Model is referencing a Data Table with a Foreign Key Column. The user uses the SetRow step to change the row for the Data Table with a Key.

Consider the following Key Table.

Key Table	Table2
	 Name
1	 One
2	 Two
3	 Three
▶	

In Table2, there is a Foreign Key Column.

Key Table	Table2	
	Name	Name2
1	One	OneOne
2	One	OneTwo
3	One	OneThree
4	Two	TwoOne
5	Two	TwoTwo
6	Two	TwoThree
7	Three	ThreeOne
8	Three	ThreeTwo
9	Three	ThreeThree
▶		

There is a Status Label in the Facility Window with the expression "Table2[2].Name2" which resolves to "OneTwo". Since the Model does not currently have any row references, the absolute row 2 in Table2 is referenced.

In the Model, there is a Button that, when clicked, executes a Process with a SetRow step that sets the row to 3 in the KeyTable. When the button is clicked, the Status Label changes to "ThreeTwo". This is because the AssociatedObject running the Process is the Model. The Model now has an explicit reference to the third row in the KeyTable. When the Model resolves the "Table2[2].Name2" expression on the Status Label, the Model will only see the section of Table2 that relates to the 'Three' Key and will pick the second related row.

Depending on the objective of the Process and Model, this could be avoided by changing the SetRow step's *ObjectType* property to 'Token'. Then the SetRow step would not assign the Model an explicit KeyTable row reference.

Row Reference Hierarchy for Processes

As row references are needed to resolve an expression, there is a hierarchy that the Token will use. The order is:

- Owner of the Property / Property Parent
- Token
- Parent Token
- Token's Parent Process Element
- Token's Associated Object
- Token's Context Object
- Entity Instance

Note, if it has no references, it is skipped, and the next item is checked in the hierarchy list.

Hierarchy Example for 'Owner of the Property'

A Data Table is fed into a submodel through a Repeating Group Property; the submodel owns the Repeating Group Property. A process is run that assigns a Repeating Group row reference to the submodel. After that row reference is created, when the Repeat Group Property is referenced in the submodel, the submodel's row references will supersede any others. Even if the Token has an explicit row reference, this will be ignored for the submodel's references.

This is important to consider for certain process steps, such as Search steps. When Searching through a table, the Token's row references are incremented to look at the different rows in the table. This is so that any related Search criteria using table references, such as the *Match Condition* or *Search Expression*, uses the candidate row being searched. When the Property Parent has the reference to the table, this row reference will be used in the Search step rather than the token's candidate row references.

External

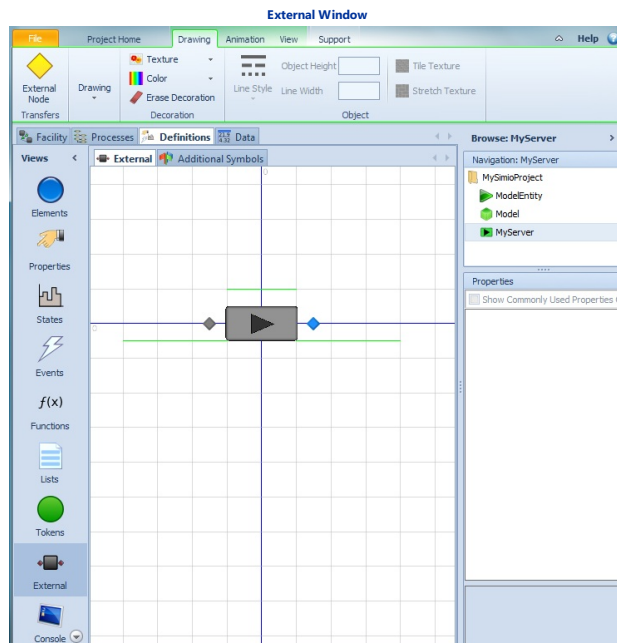
External Window

The External panel of the Definitions window consists of two windows that allow the user to define the graphical representation of a model that is instantiated as an object into another model. The External window is comprised of a 3D drawing that depicts the object, along with external node symbols that define associated node objects that serve as entry and exit points for the object. These associated node objects are automatically created whenever this object is instantiated into a model. The External window may also contain animated queue states for the model. This window is available for Fixed, Agent, Entity, and Transporter type models.

When creating the graphical representation for a model, first place a graphical symbol representing the model by using the Place Symbol button or drawing features on the Drawing tab. Depending on the type of model you are building, you often will also place one or more external node symbols. An external node defines an input and/or output point that may be used to enter or exit the model.

When utilizing hierarchy in building a model (for example, building an object from two servers connected with a path), objects that are placed within the model's Facility window are, by default, shown in the External window. Within the Facility window, right-clicking on a particular object or group of objects allows the 'Externally Visible' option to be turned on and off. Users may wish to selectively show some objects, and not others, from the Facility window into the External window.

Additionally, Simio provides menu features to easily auto-create external node symbols in an object's External window that are bound to nodes placed in that object's Facility window. If you right-click on a node in the Facility window and select the option 'Bind to External Input Node', then a new external node with *Node Class Name* of 'BasicNode' and *Input Location Type* of 'Node' will be automatically created in the object's External Window. Similarly, by right-clicking on a node in the Facility window and selecting the option 'Bind to External Output Node', a new external node with *Node Class Name* of 'TransferNode' will be automatically created. Refer to the [Creating New Objects](#) page for more information on building objects with hierarchy from the Facility window.



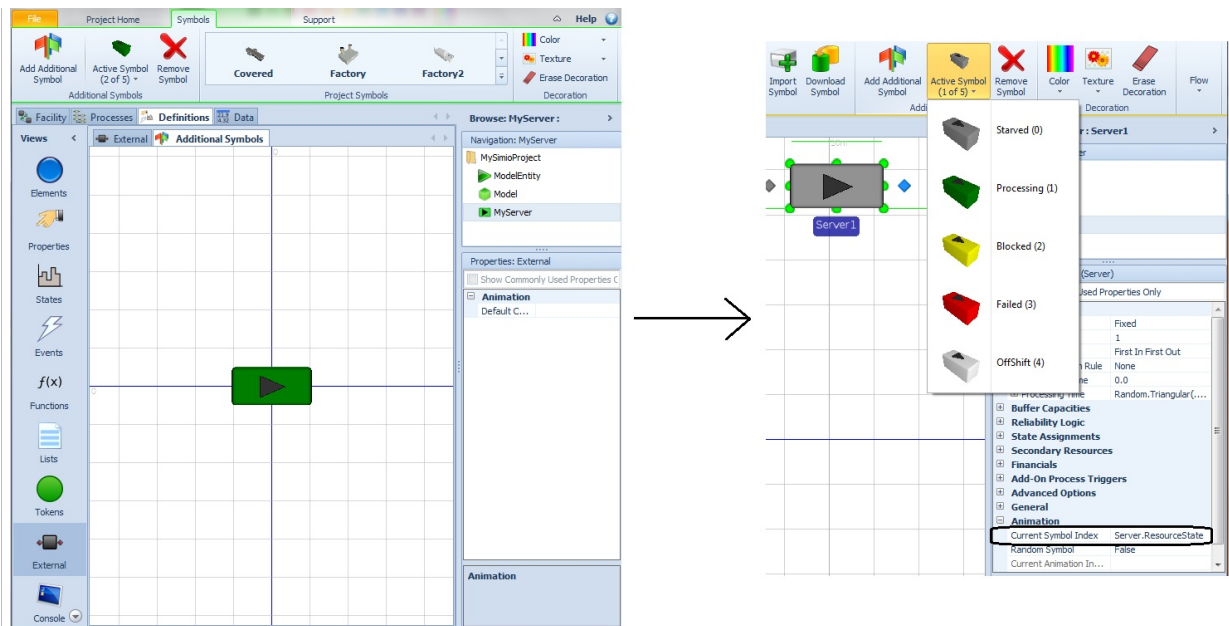
Additional Symbols Window

The Additional Symbols window provides a method for defining the additional graphical symbols for the *Current Symbol Index* expression of the object. For example, if the *Current Symbol Index* for an object is based on the 'Object.ResourceState', then default symbols may be defined for the various resource state values for the object. The *Current Symbol Index* may alternatively have an expression that evaluates to a value, for example, 'Object.Capacity.Allocated > 0'. For such an object, if the capacity of the associated object resource has been allocated, the expression returns a 'True' or '1', whereas if the capacity has not been allocated, the expression returns a 'False' or '0'. In that case, symbols for the values 0 and 1 can be defined.

When creating Additional Symbols, the symbol animation is stored and referenced separately from the size and rotation of that symbol. The animation appearance can be applied to an object by setting the *Current Symbol Index* during run time. The symbol will be fit to the current object's size and orientation. However, the *Current Size and Orientation Index* property can also be set to reference the Additional Symbols and will pull the size and rotation of that symbol as defined in the Additional Symbols window. The *Current Size and Orientation Index* property can only reference a State Variable. It is recommended to set the *Current Symbol Index* and *Current Size and Orientation Index* property to the same value.

Within the Standard Library of objects, the Server, Combiner, Separator, Resource, Worker, Vehicle and Workstation (Deprecated) have additional symbols defined based on the 'Object.ResourceState' expression specified within the *Current Symbol Index* property of the object (under the Animation group of properties). You can find the states associated with the various Standard Library objects in the [List States](#) section of the help. The Filler and Emptier objects within the Flow Library also contain additional symbols based on the state of the associated resource.

Additional Symbols Window

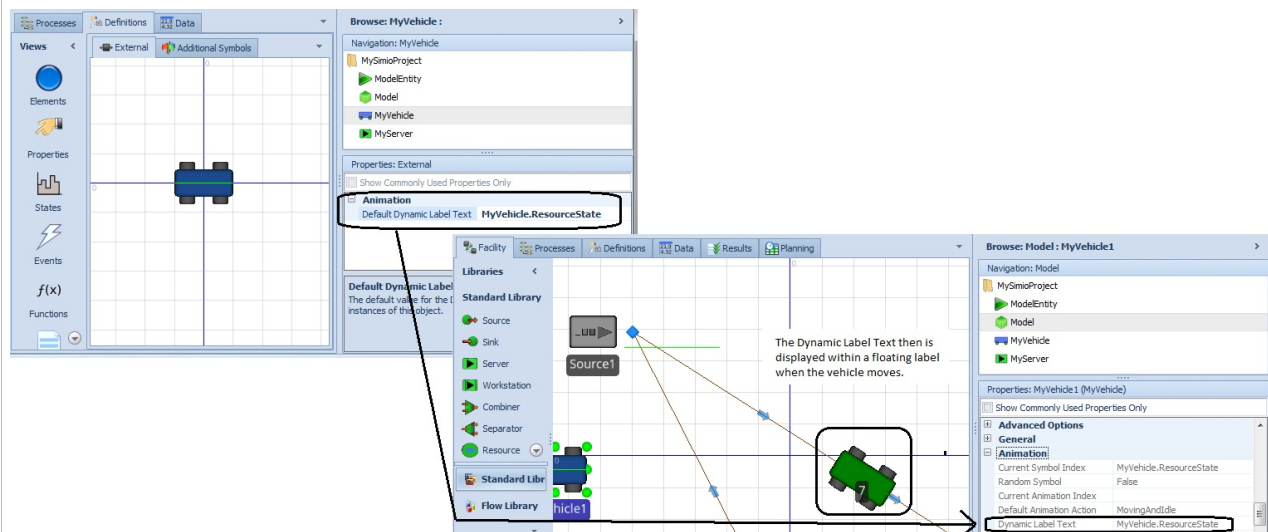


All Objects - Properties

Within the External window of any object, select the empty space to view the *Vertical Offset* property that is the offset (in meters) that the object should be instantiated above the floor. This may be useful if you are building a custom object, and want that object to come in by default several meters above the floor. Under the *Advanced Options*, there is also a property named *Pad Generated Symbol Bounding Box* that, if set to 'True' will add "padding" to the symbol generated from the external view (and any objects from the Facility view set to view in external). This was implicitly 'True' in older versions of Simio (prior to Sprint 164/165) and for new objects should be left as 'False'. This property was added mainly for compatibility reasons in Sprint 164/165.

Entity / Transporter Objects - Animation Defaults

Within the External window of an Entity or Transporter, select the empty space to view the animation property default values for that object type. There is a *Default Dynamic Label Text* that provides a default value for the *Dynamic Label Text* of instances of the object. These labels can be turned on/off for a simulation run by using the *Dynamic Labels* button within the *Visibility* ribbon.



Additionally, the *Default Link Segment Transition Type* property provides the default value for the *Link Segment Transition Type* for instances of this object.

NOTE: Multi-Select of objects, as well as Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste), are supported for all objects in this window. Please refer to the [User Interface](#) for more details.

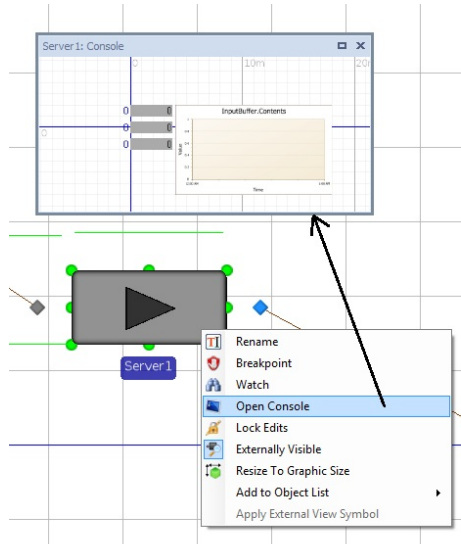
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

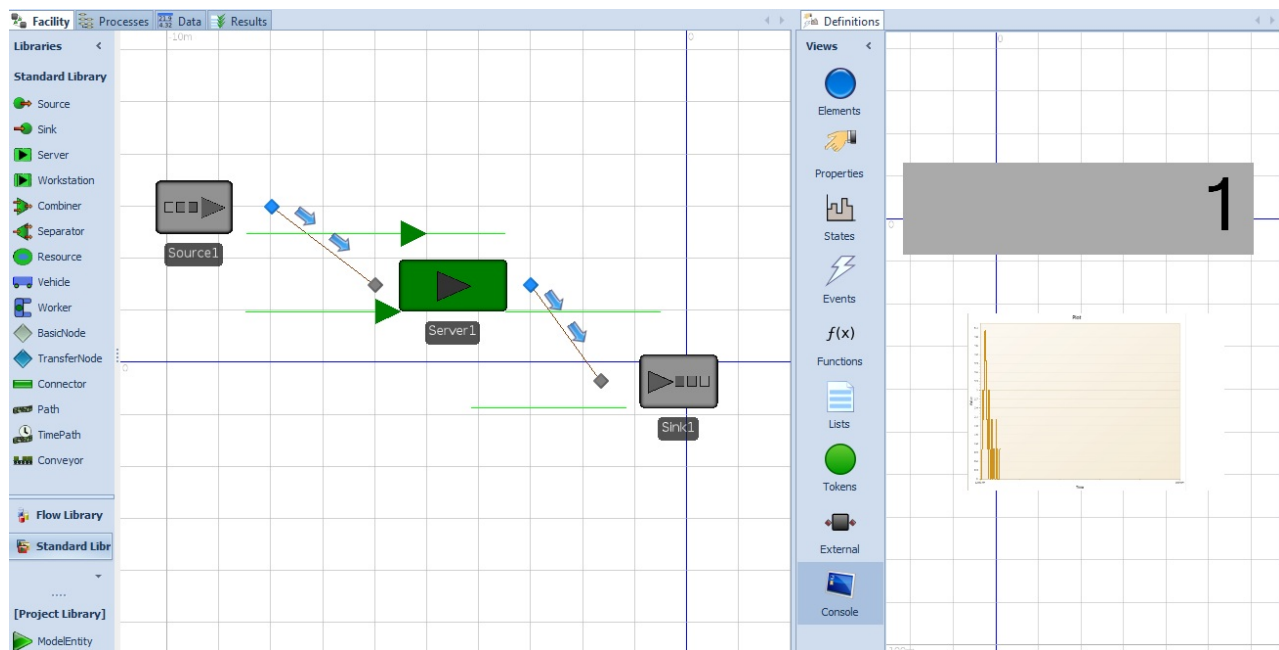
Console

The Console allows you to define run time status and control mechanisms for your model. It can be used to track the status of your running model, as well as define a status display when your model is used inside another one.

Several of the objects within the Standard Library include a console. If an object is placed inside another object, such as a model, the console can be opened by right-clicking on the object and selecting Open Console.



A convenient way to look at the Console window of a model during a run is to tile the windows. To do this, click on the Definitions tab and move the window to the right side of the screen for docking and select the Console panel. Click on the Facility tab on the left to activate the Facility window and Definitions window simultaneously, as shown below. The two windows will now be displayed vertically next to each other. This will allow you to watch both views during the interactive run. To return the windows to their original position, right click on the Definitions tab and select Move to Previous Tab Group.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

The Data Window

The Data Window

The Data window allows you to access a number of panels to create and edit data to be used in your model, including the following:

[Tables](#) hold model data that can be referenced by entities/tokens.

[Data Connectors](#) displays two types of table bindings for importing data.

[Lookup Tables](#) define lookup tables that return a function value based on a lookup value.

[Rate Tables](#) define time-varying arrival rates for firing events or creating entities.

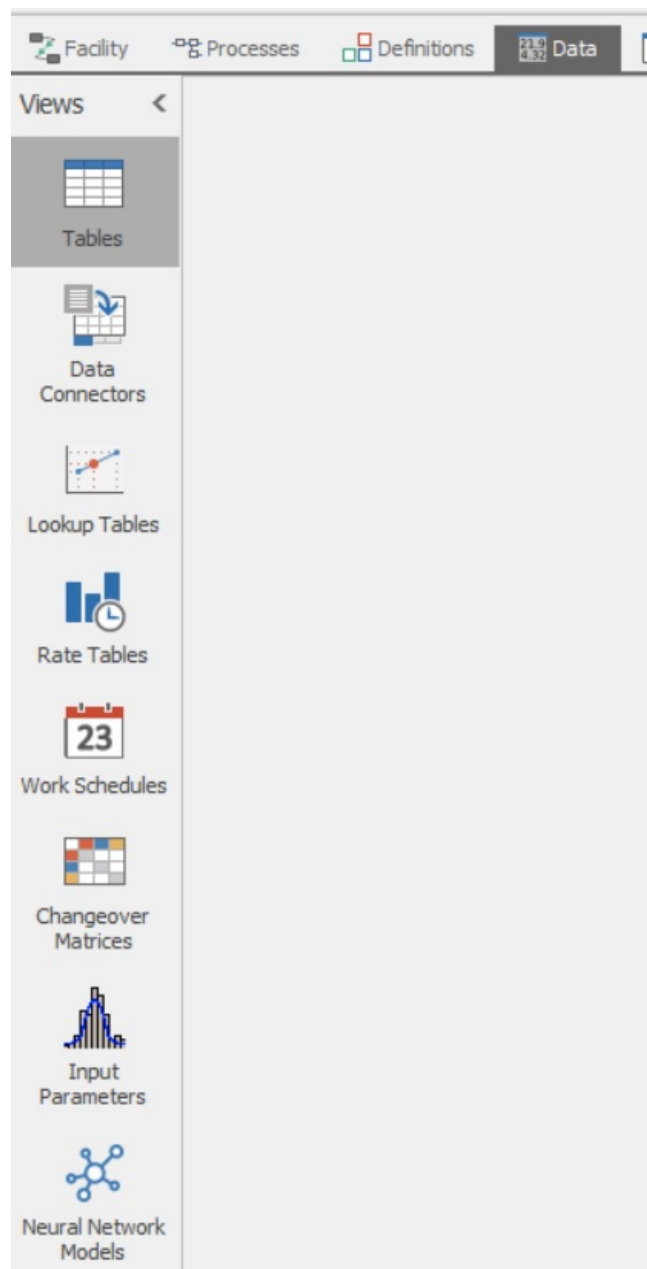
[Work Schedules](#) allow modeling resource capacities that change over time. Both pattern based and table based work schedules can be defined.

[Changeover Matrices](#) are used to model time durations that are dependent upon changes from one type of operation to another (e.g., sequence-dependent setup times).

[Input Parameters](#) provide data to sample from during the model run.

[Neural Network Models](#) allows you to use Simio's machine learning features.

The Data Window



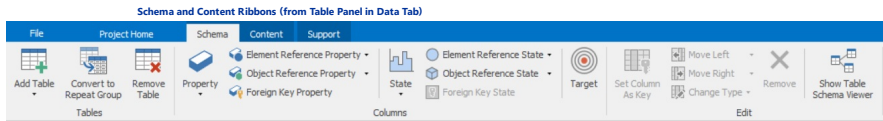
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Tables

Tables

Within the Data tab, selecting the Table panel allows a user to add either a [Data Table](#), a [Data Table From Schema](#), [Sequence Table](#) or [Output Table \(Professional/RPS Only\)](#) by clicking on the appropriate icon in the Add Table drop-down.



To reference a particular row in a table, the following syntax is used: **DataTable[1].NumericProperty** where the table name is DataTable, the column (property) in the table is NumericProperty and the row number is 1. An expression that returns an integer can be used in the brackets to specify a row. See [Data Tables](#) for more information about table functions and table referencing.

Add Property Columns

The data columns within a data or sequence table are added by selecting a Property, Element Reference Property, Object Reference Property or Foreign Key Property from the Columns area of the Schema ribbon.

The **Property** list includes data such as integers, reals, expressions, strings, model events and states, and references to other data such as lists, rate tables and schedules. These properties are typically used to store process times, arrival times or percentages for incoming part types, quantities, or other numeric data.

An **Element Reference Property** includes data such as stations, timers, material, failures and other elements that are defined in the Elements panel of the Definitions window.

An **Object Reference Property** includes data such as Entity types, Objects such as Resource Names, Nodes and Transporters, as well as references to Node and Transporter Lists. These properties are typically used to store entity types for arriving entities, resources to use for processing, or vehicles to request for movement. See also the section within [Data Tables](#) regarding using table columns for List contents. The object references can also auto-create objects into the Facility window and may use the Object Type of object reference. See the below section on Auto-Creating objects, as well as the [Table-Based Objects](#) page for more detailed information.

A **Foreign Key Property** reference is used to relate data from multiple data and/or sequence tables. When a foreign key property is added, the Table Key property is used to make a row in this table link directly to another row in the table specified by the Table Key. When the row in this table is referenced using a SetRow step in the Processes Window, the data in the row and the row it points to (via the TableName.Column listed in the Table Key property) will be immediately available. The TableName.Column that is referenced must be specified as 'Set Column As Key'. It is important to note that the type of property this column becomes is based on the Table Key's TableName.Column property. Therefore, if you are referencing an Entity object column, this column will also be Entity objects.

A table can be searched by using a [Search Step](#) to browse the table looking for certain criteria. Tables may also be used with the [Candidate](#) keyword in an expression while evaluating a condition. For example, it may be used when selecting a Worker to seize from a list of workers, where each worker is bound to a particular table row, the Selection Expression may read "Candidate.TableName.ColumnName > 100". Here you need the candidate keyword to tell Simio you want to know about the row each candidate worker is bound to, NOT whatever row the executing entity is bound to.

You can use the [Import DateTime format](#) property of a DateTime Property column to specify the regional format to align with your locale. This property should contain a format specifier character (see [NET Standard Date and Time Format Strings](#)).

Add State Columns

Within the Simio Professional and RPS Editions, users can also add State type columns to data tables. See the [State Columns](#) page for more information.

Editing Columns

Under the Edit section of information on the Schema ribbon are multiple options for editing column information. The **Set Column As Key** button allows the user to indicate that the highlighted column may be referenced by another data and/or sequence table. This button makes the current column a Key for this table (you may have multiple columns that are Key in a single table). This column's data can then be referenced by Foreign Key properties in other tables, which will link the rows in those tables to a row in this table. The Set Column as Key and Foreign Key are used together for relational type data tables. See [Relational Tables](#), as well as the [Using Relational Tables](#) SimBit and the [EntityFollowsSequenceWithRelationalTables](#) SimBit for examples on using the relational table functionality.

Once a column has been added to the table, it may be moved to the left or to the right to reorganize the table. This can be done by selecting the table column you wish to move by highlighting the column header. The Move Left and Move Right buttons on the Schema ribbon can be used to move the column within the table. Columns can also be moved all the way to the left or all the way to the right, in their respective group, i.e., Properties, States, Targets, with the Move Leftmost or Move Rightmost buttons.

The Change Type drop down button on the ribbon is available when a column is selected. It allows the user to change an existing column to a new type (real -> expression, string -> real, etc...). When converting, it attempts to keep all values in the rows of that column; however, the user may have to edit the values based on the column type that is being used.

Any column may be deleted by simply highlighting the column header and pressing the Remove Column button.

Editing Row Data

Highlighting one or more rows in a table will select that row data into the property grid to allow editing of one or more columns. If a single row is selected, the data value within each column will be displayed. If multiple rows are selected (by using Shift-click or Ctrl-click), any common data from within each column will be displayed.

The selected row (3) data is shown in the property grid with all column data displayed.

Resources	Routing Destinations	Materials	Material Lots	Manufacturing Orders	Routings	Bill Of Materials	Work In Proce
	Routing Key	Sequence	Material Name	Route Number	SetupTime (Hours)	ProcessTime (Hours)	
1	FinishedGoodA_10	Cut	FinishedGoodA	10	3	0	0.4
2	FinishedGoodA_20	Weld	FinishedGoodA	20	0	0	0.7
3	FinishedGoodA_30	Shape	FinishedGoodA	30	0	0	0.44
4	FinishedGoodA_40	Finish	FinishedGoodA	40	0	0	0.4
5	FinishedGoodA_50	Ship	FinishedGoodA	50	0	0	0
6	FinishedGoodB_10	Cut	FinishedGoodB	10	3	0	0.35
7	FinishedGoodB_20	Weld	FinishedGoodB	20	0	0	0.3
8	FinishedGoodB_30	Finish	FinishedGoodB	30	0	0	0.46
9	FinishedGoodB_40	Ship	FinishedGoodB	40	0	0	0
10	FinishedGoodC_10	Cut	FinishedGoodC	10	3	0	0.56
11	FinishedGoodC_20	Shape	FinishedGoodC	20	0	0	0.44
12	FinishedGoodC_30	Finish	FinishedGoodC	30	0	0	0.4
13	FinishedGoodC_40	Ship	FinishedGoodC	40	0	0	0

The selected rows (1, 6, 10) data is shown in property grid with common column data displayed.

Resources	Routing Destinations	Materials	Material Lots	Manufacturing Orders	Routings	Bill Of Materials	Work In Proce
	Routing Key	Sequence	Material Name	Route Number	SetupTime (Hours)	ProcessTime (Hours)	
1	FinishedGoodA_10	Cut	FinishedGoodA	10	3	0	0.4
2	FinishedGoodA_20	Weld	FinishedGoodA	20	0	0	0.7
3	FinishedGoodA_30	Shape	FinishedGoodA	30	0	0	0.44
4	FinishedGoodA_40	Finish	FinishedGoodA	40	0	0	0.4
5	FinishedGoodA_50	Ship	FinishedGoodA	50	0	0	0
6	FinishedGoodB_10	Cut	FinishedGoodB	10	3	0	0.35
7	FinishedGoodB_20	Weld	FinishedGoodB	20	0	0	0.3
8	FinishedGoodB_30	Finish	FinishedGoodB	30	0	0	0.46
9	FinishedGoodB_40	Ship	FinishedGoodB	40	0	0	0
10	FinishedGoodC_10	Cut	FinishedGoodC	10	3	0	0.56
11	FinishedGoodC_20	Shape	FinishedGoodC	20	0	0	0.44
12	FinishedGoodC_30	Finish	FinishedGoodC	30	0	0	0.4
13	FinishedGoodC_40	Ship	FinishedGoodC	40	0	0	0

When editing multiple row data, any value changes will affect all selected rows. In the above diagram, for example, changing the *SetupTime* from '3' to '6' will change it for all selected rows (1, 6, 10). Note that in the above example, the *ProcessTime* value is not shown in the property grid, as it is different for each selected row (row 1 - 0.4, row 6 - 0.35, row 10 - 0.56). Using the property grid to change this *ProcessTime* will change the value for all selected rows. This is similar functionality to the property grid with multiple selected objects within the Facility window.

Content Ribbon - Importing, Exporting and Binding To Tables

Within the Content ribbon, the user can **import, export and bind** to data tables to utilize data external to Simio. See the [Data Connectors](#) page for more information.

Auto-Creating Elements

The Element reference columns within either a Data Table or Sequence Table can be used to automatically create new Elements (within the Definitions window / Elements panel), as well as their associated properties. See the [Table-Based Elements](#) section for more detailed information.

Auto-Creating Objects

The Object reference columns within a Data Table can be used to automatically create new objects within the Facility window, as well as their associated location. See the [Table-Based Objects](#) page for more detailed information.

General Table Right-Click Features

The below sections on Filtering, Exporting and Splitting can be done with table type formats including Data Tables, Output Tables, Sequence Tables, as well as the Experiment Design tables (see Experiments).

Table Filtering

The columns within the tables include a small filter icon that can be used to filter the data within the particular column. Click the filter glyph in a column header and select an option from it. You may filter based on a particular entry in the column, or specify a custom filter. There is a bar at the bottom to turn the filter on/off, see your most recent filters, and edit the filter. Additionally, right-clicking on any column will allow users to select the auto filter row to allow custom filtering by typing, as shown below.

Filtering a Table Column

Filtering on 'Oil' will result in this view

Right-click menu for sizing and filtering columns

Show Auto Filter Row will result in this view where the user may begin typing to filter a column

Table Exporting

The table contents may be exported to several different applications, including Excel, CSV, PDF, Html or Word. This can be done by right-clicking within a table and selecting the *Export View To* to open the options.

Exporting Table Contents to an External File

Table Splitting

The table may be split so that the user may easily view multiple portions of a large table while editing. This can be done by right-clicking within the table and selecting the *Split* option. The table will be split into two views of the same data with unique scroll bars at the bottom of each view. The sections of the table can also be resized by moving the splitter bar in the center.

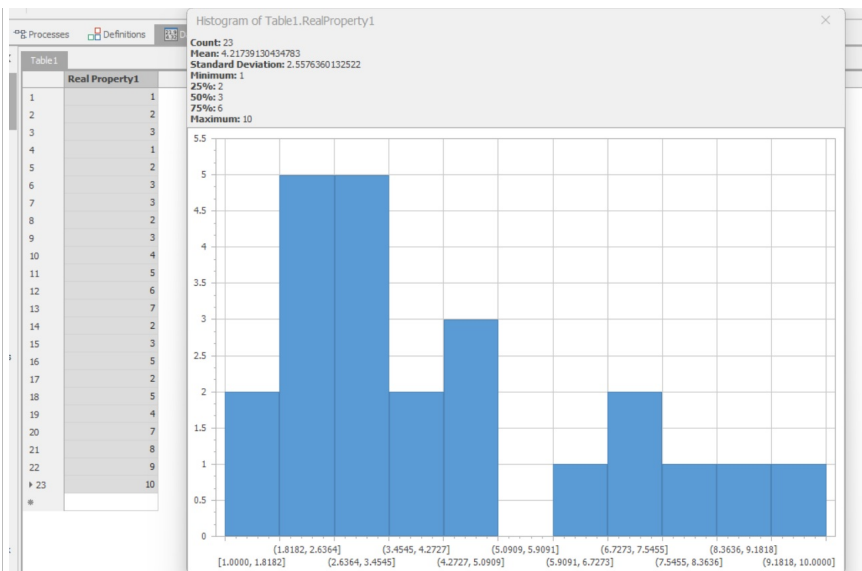
Splitting a Table

Table Filtering

Using the *Category* property of a Data Table, you can filter which Data Tables are currently visible.

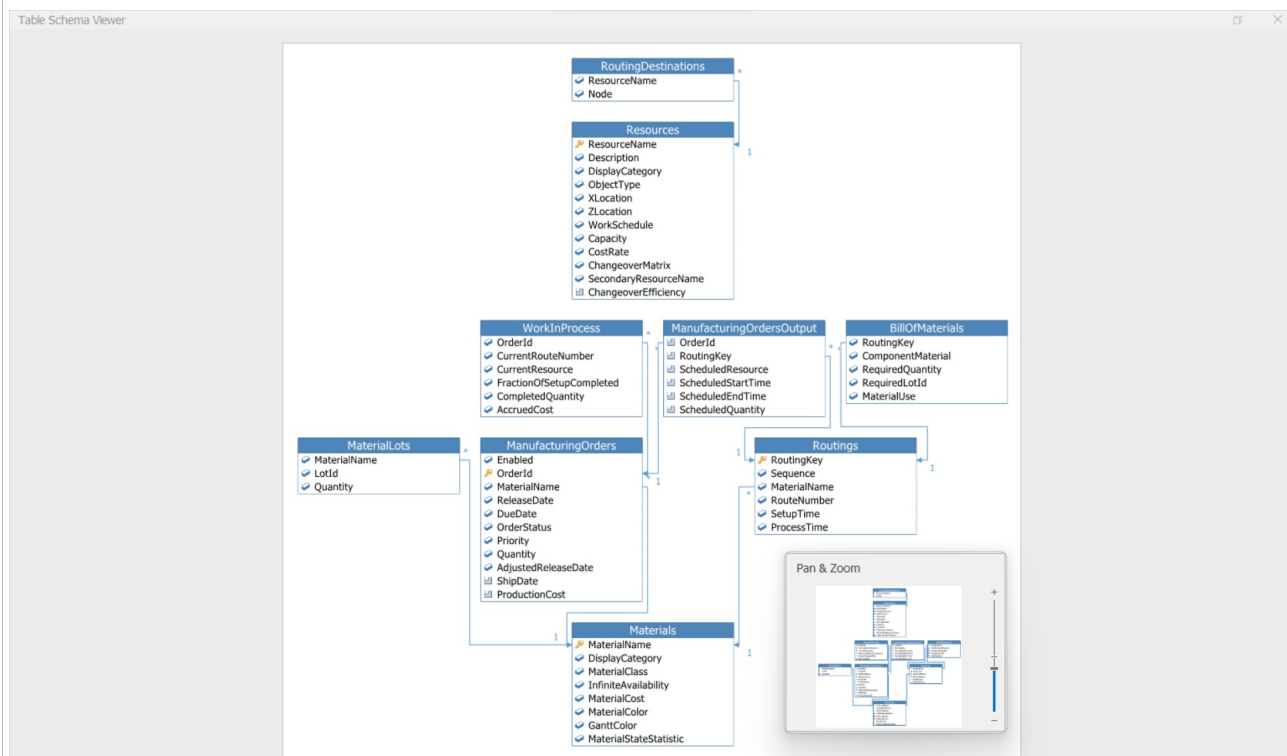
Histogram

Selecting View Histogram will open up a histogram of the selected column in a data table in a separate window.



Show Table Schema Viewer

The Show Table Schema Viewer button displays a diagram showing relationships between tables.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Data Tables

Data Tables

Tables are used to hold model data that may be referenced by individual entities and/or tokens. The data columns in the table can be any of the property types provided by Simio including expressions, object references, class types, etc. Each entity/token in the model can then reference a specific row of data in the table.

Each entity/token can set an active row of the table using the SetRow step in the Processes Window. The following functions can be used with data tables:

TableName.PropertyName: references a property value in the active row.

TableName[RowNumber].PropertyName: references a property value in any row.

TableName[RowNumber, ColumnNumber]: references a property value in any row and column.

TableName.PropertyName.RandomRow: returns a random row index using the values in the property as weights (e.g. product mix). This function can be used with property and state table columns.

TableName.PropertyName.RandomValue: returns the value of the property for a randomly selected row in the table, where each row has an equal probability of being selected. This function can be used with property and state table columns. See below section on this help page titled Selecting Random Values from a Table for more information.

TableName.KeyColumnName.RowForKey(keyValue): returns the row index of the row for which the specified KeyColumnName has the specified keyValue (or zero if no row is found). The keyValue may be an expression.

TableName.AvailableRowCount: returns the number of rows in the given table.

TableName.ColumnForName(columnName): returns the one-based index of the column in the table with the specified name, or 0 if the column is not found.

MyObject1.TableName.*: use the TableName Data Table inside of the MyObject called MyObject1.

MyObjectReferenceState.MyObject.TableName.*: use the TableName Data Table inside of the MyObject assigned to the MyObjectReferenceState.

*Note: Start with the Object reference to access the Data Table information inside that object. These table references starting with the Object reference can be used in conjunction with any of the above Data Table functions.

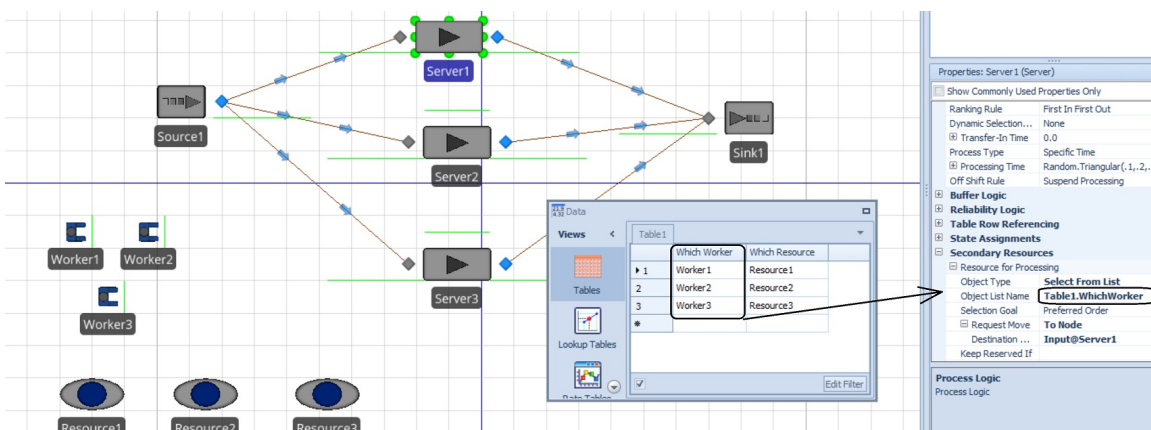
IMPORTANT NOTE: The above functions can be used with standard or relational tables. With relational tables, the RowNumber referenced above will reference the relational (not absolute) row in the table for that entity/token.

NOTE: A standard Data Table can be changed into a [Sequence Table](#) by adding or changing an existing column to a 'Sequence Destination' type of property.

Data import and export can be used for both Data Tables and Sequence Tables. Data Table import/export supports all field types as strings. All errors are noted after the import is complete. You may wish to start with designing your table and using the Export button to get the proper format. You can then import any data into your table. If the data table already has some data within it, an import will simply add file contents at the bottom of the table.

Using a Table Column as a List

Prior to Simio sprint 156, lists of objects, such as Transporter or Object Lists, could only be specified within the Definitions / Lists area by defining a List of those objects. Starting in Simio sprint 156, the Seize, Move, Route and Release steps (not all steps) allow for specifying the list members within a table and referencing the TableName.ColumnName within the Object List Name or Node List Name properties. Note that the user has to manually add 'Table.ObjectColumn' to the List Name property because it does not appear in the dropdown or right click table reference. Once specified, this then allows table referencing of list members to be done within the Secondary Resources section of many objects, as well as the Resource Requirements section of task sequence type processing. These lists can be created in relation table structures. Routing Logic within object TransferNodes may also include the table type lists. Please note that this also includes the use of table state columns (Professional and RPS editions) for the Resource, Transport, and Node List which, combined with the [Output Tables](#) and the [RemoveRows](#) step, allows for dynamic runtime building of lists. See the [SimBits UsingRelationalTablesToDefineNodeLists](#), [UsingRelationalTablesToDefineTaskResourceLists](#), and [ComplexWorkerPool](#) for more examples.



Selecting Random Values from a Table

Many times, the data that is available from a system is actual real data, which is then read into an input processor/analyzer for fitting the best distribution to the data. While Simio supports many distribution types, we also support the method of utilizing the information directly from the actual data. The 'RandomValue' table function below allows for data to be selected directly from the table.

TableName.PropertyName.RandomValue: returns the value of the property for a randomly selected row in the table, where each row has an equal probability of being selected. This function can be used with property and state table columns.

Note that if the table has primary key/foreign key relationships, and the active token or object is referencing a row selection set in the table, then this function will return the value for a row randomly selected from the related rows only. Otherwise, the row will be randomly selected from all rows in the table.

If randomness is disabled in a model and the 'RandomValue' function is utilized, the following behavior will be exhibited. If the table property (i.e., table column) is a numeric property, then the 'RandomValue' function will return the average value of all rows for the property. If the table property is non-numeric (e.g., a column of strings or object references), then the function will return the value of the middle-ish row in the table (i.e., the row value corresponding to .5 cumulative probability).

The 'RandomValue' table property function returns a value equivalent to the following expression:
`TableName[Math.Ceiling(Random.Uniform(0,TableName.AvailableRowCount))].PropertyName`. Therefore, if a user is interested in the application of this function but wants to use a different random number stream value than Simio's default, the above alternative expression approach may be taken and then the optional stream argument for the Random.Uniform function specified, such as `TableName[Math.Ceiling(Random.Uniform(0,TableName.AvailableRowCount, Stream))].PropertyName`.

Data Table Properties

Listed below are the properties of a **Data Table**:

Property	Description
Time-Indexed	If a table is time-indexed, then simulation time ranges will be mapped to the rows in the table using the specified Starting Time and Interval Size values. During a simulation run, the <code>TableName.TimeIndexedRow</code> and <code>TableName.PropertyName.TimeIndexedValue</code> functions may be used to return information for the row in the table that corresponds to the current simulation time.
Starting Time Type	Specifies how to derive the starting date time used to anchor the time intervals spanned by the rows in the table to the simulation calendar. 'SameAsRun' indicates to use the same starting date time as the model. 'Specific' indicates to use a specific date time.
Starting Time	The starting date time used to anchor the time intervals spanned by the rows in the table to the simulation calendar.

Interval Size	The interval of time spanned by each row in the table.
On Interval Process	Specifies a process that will run at the start of each time interval.

Within a time-indexed table, you can then have a column(s) of values within the table and retrieve a single value from the column, not by direct indexing, but indirectly, based on the current simulation time. The values within the table can be accessed using a function for time-indexed tables:

TableName.PropertyName.TimeIndexedValue: returns the value of the property in the row that corresponds to the current simulation time. If the current time is out of range of the table's rows, then this will return either the first row's or last row's value for the requested property.

You may also get access to the row number, based on the current simulation time, to be used with a standard table function, such as `TableName[RowNumber].PropertyName`. This row number can be accessed using the function:

TableName.TimeIndexedRow: returns the row index that corresponds to the current simulation time. Note that if the current simulation time is before or after the time ranges mapped to the table rows, this should return 1 (the index of the first row, if current time is too early), or N (the index of the last row, if the current simulation time is too late).

Table1

	Processing Time
1	10
2	15
3	20
4	25
5	30
•	

Navigation: Model

MySimioProject

ModelEntity

Model

Properties: Table1 (Table)

Show Commonly Used Properties Only

Advanced Options

Time Indexed

True

Starting Time

9/16/2013

Interval Size

1

Units

Hours

On Interval Process

Process1

General

Name

Table1

Description

Advanced Options

A table contains rows of data which can be used in the simulation.

The *On Interval Process* is an optional process that can be specified that will run at the start of each time interval. Note that this process runs "early". That is, it is scheduled on the event calendar at a slightly higher priority than most other things, so you use it to set things up for the next time interval, supposedly before anything else occurs that would be affected.

In the picture above, the value of `Table1.ProcessingTime.TimeIndexedValue` will be 10 for the first hour. At the start of the first hour, the *On Interval Process* named 'Process1' will be run. From then on, every 1 hour, the time indexed value within the table that is referenced will be updated to the next row and 'Process1' will be run.

Table Property Referencing

A Table property is defined within the Definitions window, Properties panel of an object. This table property can be used within the Processes window steps, such as `SetRow`, `AddRow` and `Search`.

Functions similar to those listed above can be used with a table property, however the row and column specified must resolve to numeric values. That is, the column name cannot be used within the table property reference, only the relative column number.

TableProperty[RowNumber, ColumnNumber]: returns a value from a row, column location of a table pointed to by a table property.

TableProperty.AvailableRowCount: returns the number of rows for a table pointed to by a table property.

TableProperty.TimeIndexedRow - returns the time indexed row of a table pointed to by a table property.

Some examples of using these table properties include the following:

`TableProperty[13, 4]` returns the value at row 3, column 4 within the table referenced in `TableProperty1`.

`MyFixed1.MyTableProp[3, 4]` returns the value of row 3, column 4 within the table referenced in `MyTableProp` within some instance of an object named `MyFixed1`.

NOTE: Reals and Expressions -- Unit Types

When using a Standard Property of type Real or Expression, the Unit Type may be specified as a Time unit. When the time unit is specified within the table, that time unit is used for any calculations using the table column. For example, if `Table1` has a real property `ProcessTimes` that is specified as 'minutes' in the Unit Type property, and the table is referenced `'Table1.ProcessTimes'` within a delay, then the time *Units* specific to the delay property are not shown and the time units of the table column (minutes) are used. Likewise, the same table entry may be referenced within an expression in a time delay, such as `'Table1.ProcessTimes * 0.5'`. In that case, the *Units* property for the expression is displayed (and can be changed), but will **NOT** be used, as the table reference includes the units (minutes in this case) to be utilized. Within the model, multiple table entries may be used in an expression with varying time units and Simio converts them to the correct base time unit.

Data Tables in Other Objects

Starting in Simio 261, a top-level object can reference a sub-level object's Data Table. Data Tables can remain in a custom object's definition and still be accessed outside of its definition. This only works in the object hierarchy where the object above can reference information from an object below. For example, the top-level `Model` can reference `Model`'s instance of `SubModel` and get `SubModel`'s table information. Similar to how the top-level model could check a `SubModel`'s `State Variable` or other definition information.

The Data Table reference from another object could look like this `'MyObject1.MyTable.PropertyName'`, where `MyTable` is a table inside the instance of `MyObject` called `MyObject1` and the `PropertyName` in `MyTable` is a property referenced by an active row reference.

`MyObject1`'s definition holds the Data Table. As `MyObject` is instantiated into a `Model`, so that there are `MyObject1` and `MyObject2`, each table is unique per instance. Row 1 could be consider in `MyObject1.MyTable.PropertyName` while Row 25 is considered in `MyObject2.MyTable.PropertyName`.

These Object table references can be used the same as any other table references. You may need to manually type these object table references as Simio might not automatically fill in this information in all places.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Relational Tables

Relational Tables

As mentioned in the [Tables](#) page, a foreign key may be used to create a table relation. This approach allows exploitation of relational data design to simplify Simio tables.

A Foreign Key reference is used to relate data from multiple data and/or sequence tables. When a Foreign Key Property is added, the Table Key property for that column is used to make a row in this table link directly to another row in the table specified by the Table Key (using the Table.ColumnName reference). The Set Column As Key button allows the user to indicate that the highlighted column may be referenced by another data and/or sequence table. This button makes the current column a Key for this table (you may have multiple columns that are Key in a single table). This column's data can then be referenced by Foreign Key properties in other tables, which will link the rows in those tables to a row in this table. The Set Column as Key and Foreign Key Property are used together for relational type data tables.

Relational tables include Master-Detail, which allows the relationships between various tables to be visually seen. For those tables that have a column that is designated as 'Set Column As Key', users can see a "detail" view, which is a collection of rows pointing to a specific keyed row in a table. The small '+' sign in front of the row indicates that the detail view is available. When the '+' is pressed, portions of the related table, those specific to the entry that was expanded, will be shown.

Master-Detail shown within Relational Data Tables

The screenshot displays the Simio software interface with the 'Service Times' table selected. The table has three columns: 'Type Of Service', 'Service Time', and 'Car Color'. The rows are 'Brakes', 'Oil', and 'Tires'. The 'Oil' row is highlighted, and a detail view is expanded for it. The detail view shows 'Arrivals' with columns 'Arrival Time' and 'Type Of Service', containing two rows for 'Oil' with arrival times 0.5 and 0.75. A red arrow points from the 'Oil' row in the main table to the detail view.

Type Of Service	Service Time	Car Color
Brakes	Random.Triangular(1.5, 2, 2.5)	Green
Oil	1	Blue
Tires	Random.Triangular(.5, 1, 1.2)	Red

Arrival Time	Type Of Service
0.5	Oil
0.75	Oil

For example, in the screen shot shown above, the Type of Service is defined as a Key Column. Therefore, this column can be used within another table as a reference (i.e., within the Arrivals table, Type of Service column is set as a Foreign Key type of property and thus references the Type of Service column within the Service Times table). Note that the name of a Foreign Key column does not have to be the same as the Key Column name.

Table based lists can also be defined using relational tables. Examples can be found in the SimBits [UsingRelationalTablesToDefineNodeLists](#) and [UsingRelationalTablesToDefineTaskResourceLists](#). For other SimBits that use relational tables, see [UsingRelationalTables](#), [ServersUsingTaskSequenceWithDataTablesJobShop](#), and others (use the

Sample SimBits Solutions to search 'relational').

*IMPORTANT NOTE: The functions listed on the [Data Table](#) page can be used with standard or relational tables. With relational tables, the RowNumber referenced will reference the relational (not absolute) row in the table for that entity/token.

The detail view can be edited just like the main view (insert / remove rows, edit rows, copy/paste, etc.). Additionally, a keyboard shortcut (Ctrl+Shift+'+') can be used to insert rows into any data grid, such as tables or lists.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Sequence Tables

Sequence Tables

A Sequence Table is a data table that has a destination column that is used to specify a sequence of destinations for an entity (e.g. the job routing sequence in a job shop). You can also add additional data columns to the sequence table and reference these values like any other table (e.g. TableName.PropertyName).

	Sequence B	Sequence A	Sequence C	Job Table
	Destination	Process Time		
▶ 1	Input@Server3	Random.Triangular(0.5,0.8,1.2)		
2	Input@Server2	1.5		
3	Input@Server1	1		
4	Input@Sink1	0.0		
*				

A standard Data Table can be changed into a Sequence Table by either adding a 'Sequence Destination' type of standard property, or using the Change Type option to change an existing column to 'Sequence Destination', as shown below.

Schema

Property

Real
Integer
Boolean
Expression
DateTime
String
Sequence Number
List
Enumeration
Event
Rate Table
Table
Sequence Table
State
Schedule
Day Pattern
Selection Rule
Steering Behavior
Changeover Matrix
Task Dependency
Replenishment Policy
Sequence Destination

Move Left
Move Right
Change Type
Remove

Standard Property

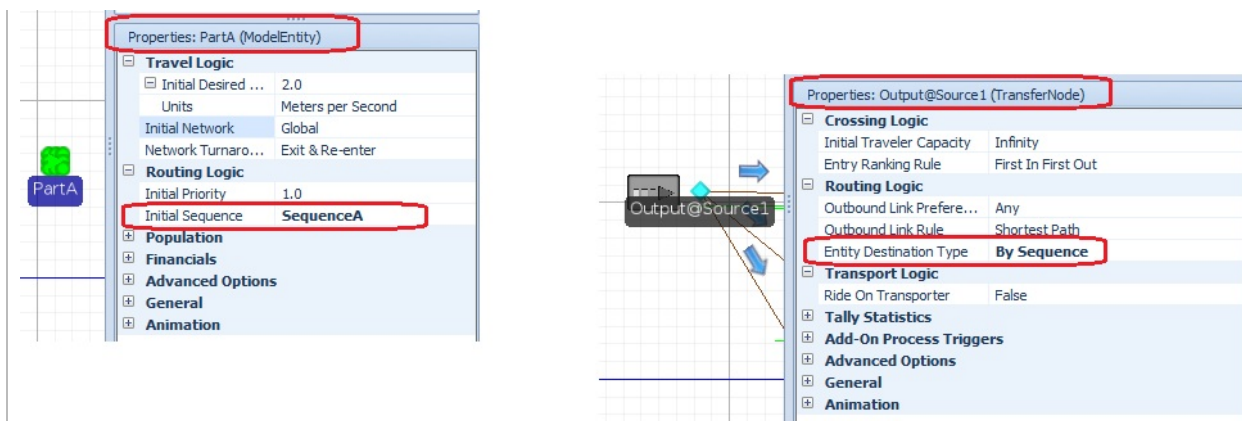
Element Reference

Object Reference

Foreign Key

Real
Integer
Boolean
Expression
DateTime
String
Sequence Number
List
Enumeration
Event
Rate Table
Table
Sequence Table
State
Schedule
Day Pattern
Selection Rule
Steering Behavior
Changeover Matrix
Task Dependency
Replenishment Policy
Sequence Destination
Color

The sequence table that an entity follows can be initially set as the *Initial Sequence* property of the entity instance. The routing logic for an entity can specify that the entity destination is to be set "By Sequence". This may be specified in the *Entity Destination Type* for the TransferNode or *Destination Type* of the SetNode step.



When setting the next destination "By Sequence", the next table row in the entities' sequence table is made active and the entities' destination is set to the destination specified in that row. The active row in the sequence table is automatically incremented each time the destination is set "By Sequence".

A destination can be any node in your model. By default, the *Accept Any Node* property on this column is 'True' so that any node can be selected. If you want to only select from object's associated input nodes, you can change the *Accept Any Node* property on this column is 'False'. When this property is 'False', if an object (e.g. a Server) has a single associated input node then the destination can be specified in shortened form as Server (instead of Input@Server). This is convenient for routing between the objects in your model. However, if the object has multiple input names (e.g. Combiner), then the input nodes must be fully qualified as NodeName@ObjectName (e.g. ParentInput@Combiner and MemberInput@Combiner). By default only the input nodes for objects are in the list of destinations. If you want to set a destination to another node in the model, click on the Destination column header and change the *Accepts Any Node* property to "True".

Data import and export can be used for both Data Tables and Sequence Tables. Sequence Table import/export supports all field types as strings. All errors are noted after the import is complete. You may wish to start with designing your table and using the Export button to get the proper format. You can then import any data into your table. If the data table already has some data within it, an import will simply add file contents at the bottom of the table.

Changing an Entity Sequence or Active Row

The entity has an initial sequence that is assigned with the *Initial Sequence* property within the ModelEntity. The sequence table associated with an entity can be changed by using the *SetRow* step. For example, if you have two sequence tables and the entity's original table is Table1, you can change it during the model logic to Table2. Keep in mind that each routing 'By Sequence' will update the row in the table by one. Therefore, if you assign a new table association after processing at a Server, for example, you may need to initially set the row to '0' such that when the entity leaves the Server 'By Sequence' it updates the table association to row 1 and sends the entity to the first location in the sequence table.

The *SetRow* step can also be used to change the entity's active row within a sequence table. Changing the active row may be useful for rework, for example, to route an entity back through a given set of steps.

Sequence Functions

There are a number of entity sequence related functions that can be used during the simulation run. These functions can be found on the [Functions, States and Events for Entity Objects](#) page.

Both BasicNode and TransferNode objects have a *Sequence Expected Operation Time* property. This property is used to estimate an expected operation time if the node is in the destination sequence of an entity's assigned sequence table.

For Server and Separator objects, the *Sequence Expected Operation Time* for the 'Input' node is the *Processing Time*. For the Combiner object, the *Sequence Expected Operation Time* for the 'ParentInput' node is the *Processing Time*. The *Sequence Expected Operation Time* property value for the 'Input' node of the Workstation (Deprecated) is the *Setup Time* (if specific and thus sequence independent) + *Processing Time* + *Teardown Time*. Any sequence dependent setup time estimates are not included by default in the expected operation time at a workstation.

The *Sequence Expected Operation Time* is used for [Dynamic Selection Rules](#) like 'Least Slack Time Remaining'.

Note that when estimating the expected operation time for a sequence jobstep, the expression evaluation is always deterministic. Thus, if a random distribution is specified, then the expected value for that random distribution will be assumed. The expected operation times for the job steps in an entity's sequence will be evaluated once on the first function call that requires the information. For performance efficiency reasons, those estimates will then be stored and simply reused on any subsequent function calls requiring the information.

Examples

For examples of using sequences, please refer to the SimBits [EntityFollowsSequence](#), [EntityFollowsSequenceMultiple](#), [EntityFollowsSequenceWithTable](#), and [SortingConveyorSystem](#).

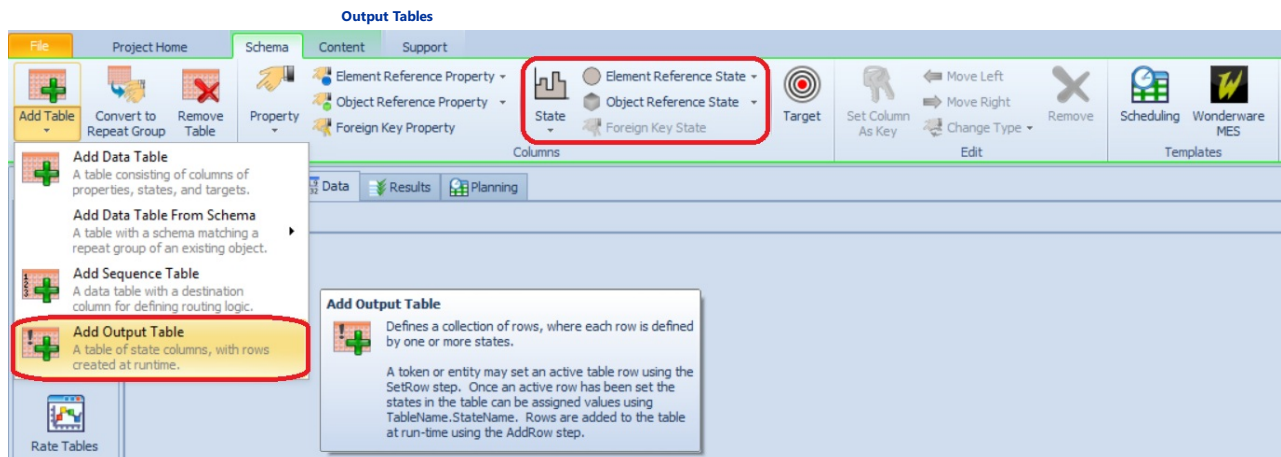
Output Tables

Output Tables

Output tables are available to users with Simio Professional or RPS Edition.

Output tables are similar to Data tables in that they define a collection of rows. Within an Output table, however, the columns are all specified as State type columns - that is, the values within the table will be written to the table during or after the simulation run. A token or entity may set an active table row using the SetRow step. Once an active row has been set, the states in the table can be assigned values using TableName.StateColumnName. Rows are added to the table at run-time using the AddRow step. The RemoveRows step may be used to remove a single row or all rows from the output table. More information on adding states can be found on the [State Columns](#) page.

An Output table may have a state column that is a designated as key column, by selecting the Set Column As Key for a particular column. The AddRow step has a Key Column Value property that should be set to a unique key value for the row. This particular feature is for RPS licensing only.



Output Table Properties

Listed below are the properties of an **Output Table**:

Property	Description
Time-Indexed	If a table is time-indexed, then simulation time ranges will be mapped to the rows in the table using the specified Starting Time and Interval Size values. During a simulation run, the TableName.TimeIndexedRow and TableName.PropertyName.TimeIndexedValue functions may be used to return information for the row in the table that corresponds to the current simulation time.
Starting Time	The starting date time used to anchor the time intervals spanned by the rows in the table to the simulation calendar.
Interval Size	The interval of time spanned by each row in the table.
On Interval Process	Specifies a process that will run at the start of each time interval.
Keep Values Between Runs (RPS Only)	Indicates whether to keep the output table's state values from the previous run if a new run is started using the RestartRun step.

For more information on Time-Indexed tables, see the Table Properties section within the [Data Table](#) page.

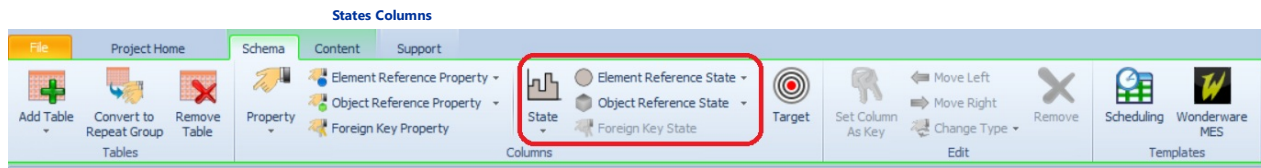
Individual [State Columns](#) within a [Data Table](#) also include the *Keep Values Between Runs* property on the state column. Note that state columns within Output Tables DO NOT have the option individually, this property refers to the entire table. Output Tables either keep the entire table or don't. This options only is available for RPS licenses and works across interactive, plan, and experiment runs. See the [Single-Pass vs Multi-Pass Simulation Approach](#) page for more details.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

State Columns

States within Schema Ribbon



In addition to property type columns from the Schema ribbon, the user may also define a column as a table state. Table states are added by utilizing the states section the Schema ribbon and may be discrete or continuous. The Move Left and Move Right buttons allow users to move a particular column for organizational purposes. As with other table columns, there are various types, including:

- **Real** - The state representing a real numeric value that may change by assignment logic at discrete times during a model run.
- **Integer** - The state variable representing an integer numeric value that may change by assignment logic at discrete times during a model run.
- **Boolean** - The state variable representing a logical or Boolean (true or false) value that may change by assignment logic at discrete times during a model run.
- **DateTime** - The state variable representing a datetime value that may change by assignment logic at discrete times during a model run.
- **String** - The state variable representing a string value that may change by assignment logic at discrete times during a model run. For more information, see String States in the Simio on-line help.
- **List** - The state variable representing an integer variable with a discrete set of possible values from 0 to N. A zero-based indexed string list is used to define the possible integer values for the state. A list state's value may be changed by assignment logic at discrete times during a model run, and the state will automatically collect and report time-persistent statistics for each of its state values. For more information, see List States in the Simio on-line help.
- **List Reference** - The state variable defining a list reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to a list reference state using an Assign step in a process. The expression used as the value for the assignment must return a list reference, for example from an Object List Property or another List State Reference.
- **Level** - The continuous-change state variable defining a numeric variable that may change continuously over time based on the current value of its rate. An automatic rate variable, based on the LevelStateName.Rate can be used to modify the rate of change throughout the simulation run. The state's rate parameter is a discrete-change value that may be changed by assignment logic at discrete times during a model run.
- **Level with Acceleration** - The continuous-change state variable defining a numeric variable that may change continuously over time based on the current value of its rate and acceleration. The state's rate, acceleration and acceleration duration parameters are discrete-change values that may be changed by assignment logic at discrete times during a model run.
- **Element Reference** - The state variable defining an element reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to an element reference state using an Assign step in a process. The expression used as the value for that assignment must return an element reference. Element references include Element, Batch Logic, Changeover Logic, Container, Cost Center, Failure, Inventory, Material, Monitor, Network, Output Statistic, Process, Regulator, Routing Group, State Statistic, Station, Storage, Tally Statistic, Task Sequence and Timer.
- **Object Reference** - The state variable defining an object reference variable that may be changed by assignment logic at discrete times during a model run. You can assign a value to an object reference state using an Assign step in a process. The expression used as the value for the assignment must return an object reference. Object references include Object, Fixed, Link, Node, Entity and Transporter.
- **Foreign Key State** - Adds a Foreign Key state, which is used to make a row in this output table link directly to another row in some data table. The value of the state cannot be set via the Assign step, but instead is set to the active row of the data table at the point in time when the AddRow step is executed on this output table. When the row in this output table is referenced using the SetRow step, or the AddRow step, the data in this row and the row it points to will be immediately available.

Unlike other column types, however, the table state column values are not input by the user at the start of the simulation run. These values are assigned during the simulation run by an Assign step, based on the current row of the associated object. The table name and column name are used in the assignment, similar to referencing other table columns. For example, if the table name is ManufacturingOrders and the table state name is ShipDate, the assignment would be made to the State Variable Name 'ManufacturingOrders.ShipDate' as shown below. The actual row that the New Value of 'Run.TimeNow' is entered into is based on the row that is associated with the entity.

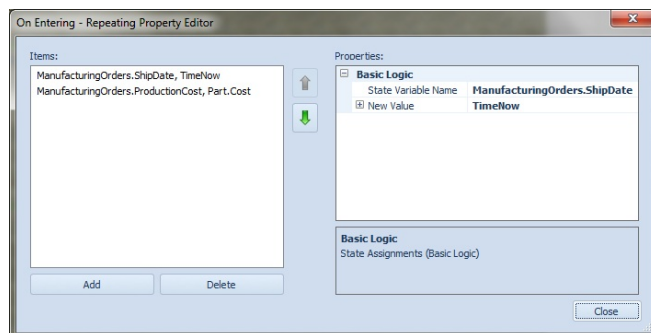


Table states may also be utilized in an expression in the same way. For example, you may wish to continually add to a table state, in which case, the assignment may be State Variable Name of 'ManufacturingOrders.NumberOfDefects' and the New Value is 'ManufacturingOrders.NumberOfDefects + 1'.

When you add a table state to the table, the entire column will be gray in color, as seen below. This signifies that it is a table state column and not a regular column that requires entries.

Table State 'ShipDate' Column

Manufacturing Orders							
Order ID	Part Type	Release Date	Due Date	Order Status	Priority	Ship Date	
Order01	A	10/3/2011 12:00:00 AM	10/12/2011 11:00:00 PM	New	1		
Order02	A	10/3/2011 12:00:00 AM	10/14/2011 9:00:00 PM	New	1		
Order03	B	10/4/2011 12:00:00 AM	10/13/2011 4:00:00 PM	New	1		
Order04	B	10/4/2011 12:00:00 AM	10/14/2011 2:00:00 PM	New	1		
Order05	A	10/5/2011 12:00:00 AM	10/19/2011 9:00:00 PM	New	1		
Order06	C	10/6/2011 12:00:00 AM	10/12/2011 12:00:00 AM	New	1		
Order07	A	10/7/2011 12:00:00 AM	10/19/2011 4:00:00 PM	New	1		
Order08	C	10/8/2011 12:00:00 AM	10/13/2011 10:00:00 PM	New	1		
Order09	B	10/9/2011 12:00:00 AM	10/19/2011 4:00:00 PM	New	1		
OrderWIP1	A	10/3/2011 12:00:00 AM	10/10/2011 4:00:00 PM	WIP	1		
OrderWIP2	A	10/3/2011 12:00:00 AM	10/5/2011 4:00:00 PM	WIP	1		
OrderWIP3	B	10/3/2011 12:00:00 AM	10/7/2011 4:00:00 PM	WIP	1		
OrderWIP4	B	10/3/2011 12:00:00 AM	10/5/2011 4:00:00 PM	WIP	1		
OrderWIP5	C	10/3/2011 12:00:00 AM	10/6/2011 4:00:00 PM	WIP	1		
OrderWIP6	C	10/3/2011 12:00:00 AM	10/3/2011 4:00:00 PM	WIP	1		
OrderWIP7	C	10/3/2011 12:00:00 AM	10/3/2011 4:00:00 PM	WIP	1		

With all of the table state types, there are various properties, such as initial state value, unit type (some) and dimension (for some). When you specify a *Unit Type*, the state column will display units following the model's Unit Settings.

The model's Unit Settings can be changed from the Facility's Run ribbon.

Once the simulation model is running, values will be calculated for the table states and shown in the appropriate columns. When assigning values to these columns, you can provide a specific unit that may be different from the column's unit. However, the value will be converted to match the display units for the state column. It will not reassign the display units for the column.

Similar to standard type table columns, the state type columns have a *Visible* property to determine whether the state column is visible or not visible to a Scheduler type user in Operational Planning mode. This includes visibility in both the Tables and Gantt properties shown on the right when a Gantt chart item is selected.

Table State Properties - Visibility

Properties: RealState1 (Real State Variable)

- Value**
 - Dimension Type: Scalar
 - Unit Type: Unspecified
 - Initial State Value: 0
 - Auto Reset When Statistics Cleared: False
 - Display Format:
- Advanced Options**
 - Unit Type Property:
 - Keep Values On Run Restart: False
- Operational Planning**
 - Display Name: RealState1
 - Tables**
 - Visible**: True
 - Category Name:
- General**
 - Name: RealState1
 - Description:

Visible
If true, then the state is visible in the Operational Planning data table.

Multi-Pass Runs and RestartRun Step

Individual state columns within a *Data Table* also include the *Keep Values Between Runs* property on the state column. This property allows the user to keep the state's value from the previous run if a new run is started using the RestartRun step. This is the case for table states that are not an object or element reference state type. For example, within a Manufacturing

Orders table (see Sched* Examples), there may be a state column that stores the order's ActualReleaseDate or starting processing time (as opposed to ReleaseDate) that is updated within a multi-pass run based on when the order is actually started. This field can then be used in a second run ([RestartRun step](#)) as the starting time for the order. See the [Single-Pass vs Multi-Pass Simulation Approach](#) page for more details.

Note that state columns within [Output Tables](#) DO NOT have the option individually, this property refers to the entire table. Output Tables either keep the entire table or don't.

The *Keep Values Between Runs* property is only available for RPS licenses and works across interactive, plan, and experiment runs.

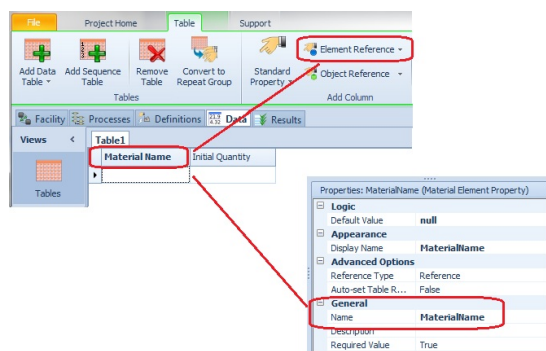
[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

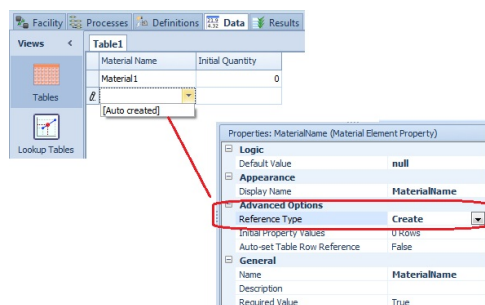
Table-Based Elements (Auto-Create)

Element Objects

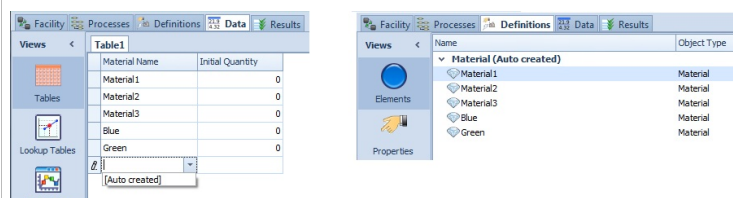
When a column within a sequence or data table is of the type Element Reference, the user will get the following options within the property window:



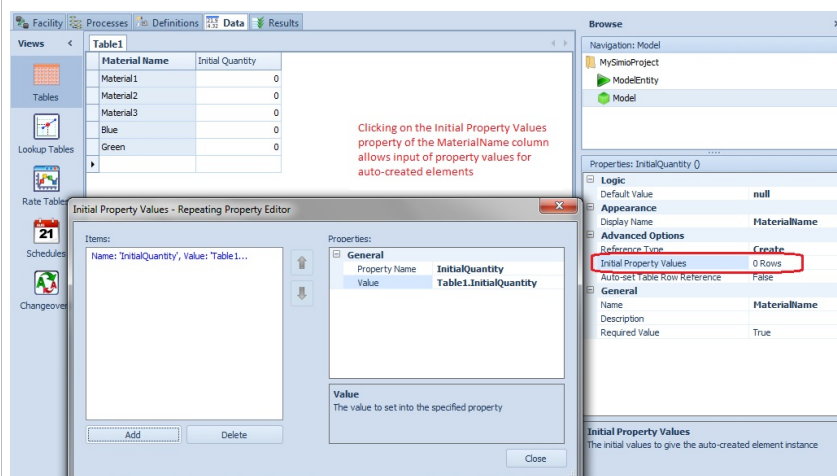
By default, any Element Object columns will have a *Reference Type* (Advanced Options) of 'Reference' to simply reference a previously defined element. However, if the *Reference Type* is changed to 'Create', then any entries into that column will automatically create a new element of the specified type (in the example above and below, a Material element is used as an example).



When an Element is automatically created, you will see it located within the Definitions window, Elements panel. It will be clearly marked as auto-created and it cannot be deleted. Any edits to the element *Name* within either the data table or the element panel window will be reflected in each other.



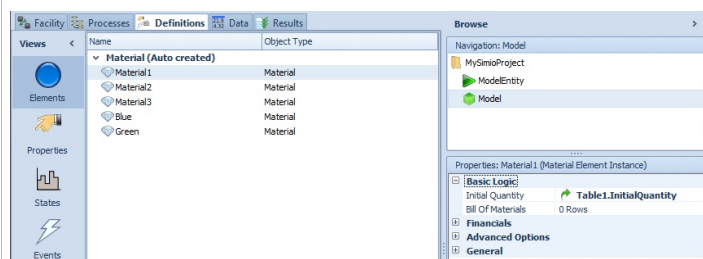
In addition to the element itself being created, any properties of that element may also then be referenced and created by a column within the table. This is done with the *Initial Property Values* repeatable property within the Element Reference property window, as shown below. For each given Element type, a different set of *Property Name* properties are available from the list. In this example, the 'InitialQuantity' of a material is input. Within the *Value* property an expression can be entered which will be fed into the property information within the Element itself. In this example, a table reference is used, 'Table1.InitialQuantity'.



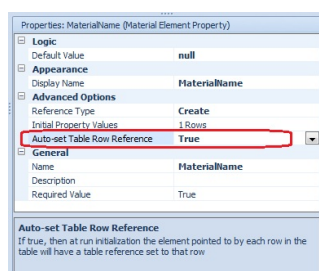
Clicking on the Initial Property Values property of the MaterialName column allows input of property values for auto-created elements.

The value of the property specified in the Value field will then get fed into the appropriate property within the Element itself, as shown below. Likewise, the *Value* reference (above) can also be a simple integer value, such as '3' that will similarly

be fed into the correct property value within the Element itself.



Finally, it is important to note that if an auto-created Element's property value is linked to a table (as it is in the example above, using 'Table1.InitialQuantity'), the *Auto-set Table Row Reference* property for the Element in the table must be set to 'True'. This is shown in the below diagram.



The *Auto-set Table Row Reference* property, if true, then at run initialization, the Element pointed to by each row in the table will have a table reference set to that row. This will allow an entry in an Element column to 'correspond' to the same row entry in another column, which is essential for defining Element properties.

Some of the commonly auto-created elements include [TallyStatistic](#), [Material](#), [RoutingGroup](#) and [Monitor](#).

Note: Auto-created elements do not appear in the expression editor, however you can manually type them in.

Using Data Table Schemas and Initial Property Values

Data Table Schemas are a quick way to create a Data Table containing all the properties related to an object, element, or repeat group. See the page [Data Table From Schema](#) for more information on table schemas.

When created via Add Data Tables From Schema (Data tab > Tables view > Schema ribbon > Add Table, more), the *Initial Property Values* (used for *Reference Type* = 'Create'), will automatically be filled in so that each property corresponds with its column in the Data Table. If the Parent Table was created at the same time as any Repeat Group Tables, the Repeat Group Properties will automatically be included in the *Initial Property Values*. If these tables were not all created at the same time, then the Repeat Group Properties will not automatically appear in the *Initial Property Values*, nor will the Element Repeat Group's property names appear in the drop-down of *Property Name* in *Initial Property Values*.

For example, the Material Table will have the *Initial Property Values* set for the *Bill Of Materials* repeat group if the tables (i.e., Material and Material.BillOfMaterials) are created in the same command (i.e., both selected at the same time). But, if not created at the same time or created manually, the property names may not show in the drop-down of the *Property Name*. Items for *Bill Of Materials* could be manually typed into the *Property Name* field. The values for the *Bill Of Materials* repeat group are as follows:

- BillOfMaterials
- BillOfMaterials.ComponentMaterialName
- BillOfMaterials.ComponentQuantity

While these properties are not shown in the drop-down, they can be inferred via the Material Element *Bill Of Materials* repeat group.

The above also applies to *Material Initial Quantities (More)* repeat group, with property names as follows

- InitialQuantities
- InitialQuantities.InitialQuantitiesInitialQuantity
- InitialQuantities.InitialQuantitiesLotID

This behavior may also apply to other repeat groups.

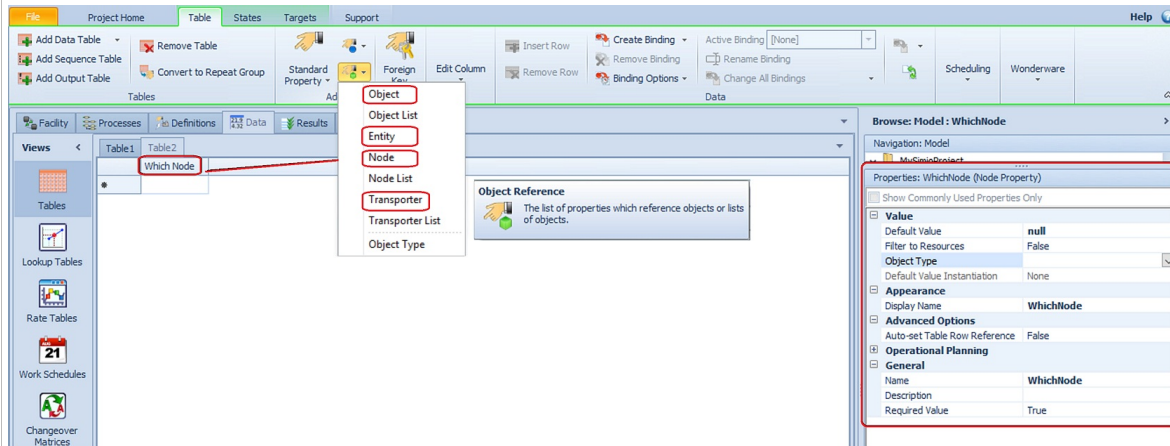
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

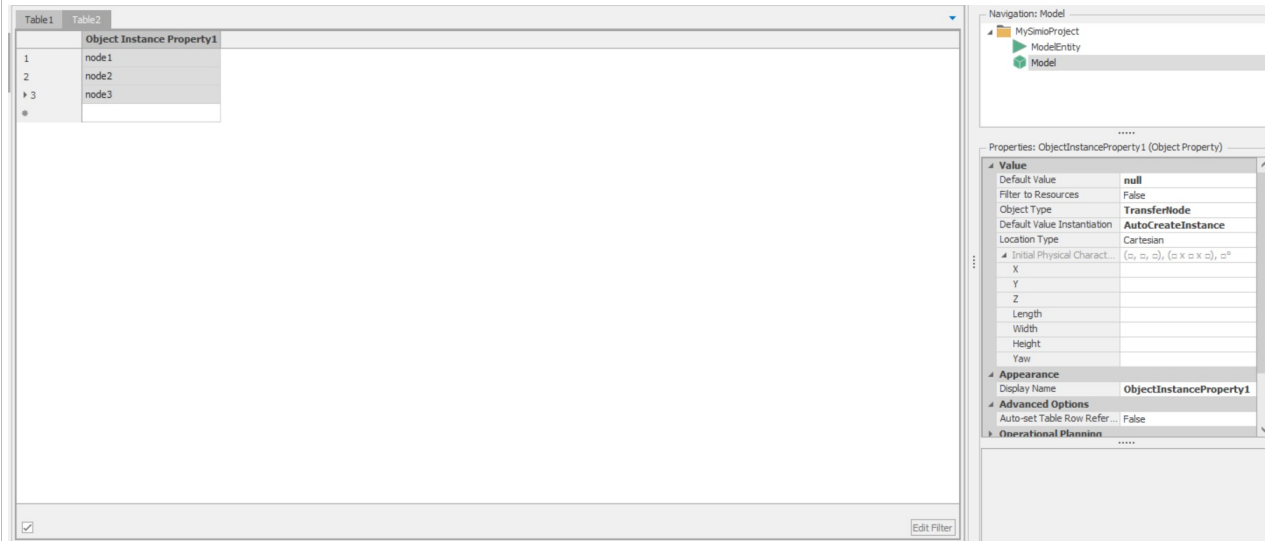
Table-Based Objects (Auto-Create)

Auto-Created Objects in the Facility Window

When a column within a sequence or data table is of the type Object Reference for those circled (not lists), the user will get the following options within the property window:

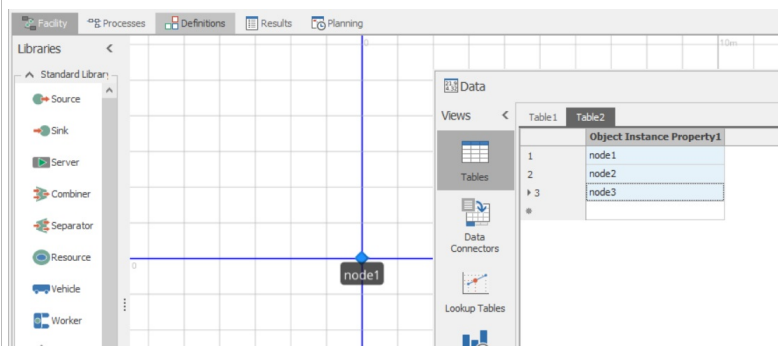


By default, any Object Reference columns will have a *Object Type* of '' (blank) meaning any object of that given type, as well as a *Default Value* of 'None'. However, if the *Object Type* is specified as a particular object, and the *Default Value* is 'AutoCreateInstance', then any entries into that column will automatically create a new object of that Object Type within the Facility window.



Auto-Created Fixed, Node and Transporter Type Objects

When an Object is automatically created, you will see it located within the Facility window. By default, the location of a fixed, node or transporter type object will be the *Initial Physical Characteristics* grouping of properties, initially set to 0,0,0 (X,Y,Z locations). Objects can be moved to the desired location. The *Location Type* property allows users to select between 'Cartesian' and 'Geographic' type locations. For example, selecting the default value of 'Cartesian' provides expression properties for X, Y, and Z. Selecting the *Location Type* of 'Geographic' provides expressions for Latitude (-90.0 to +90.0), Longitude (-180.0 to +180.0), and Elevation (in meters). These coordinate values can optionally be specified as table columns, and will take their values from the corresponding row in the table, as seen in later examples. Yaw (orientation) can also be specified in this property column. Auto-creating links is slightly different and is discussed later below.



Object Location, Orientation, and Size, and Symbol Index Specified Within Table

Alternatively, instead of simply moving an object once it is placed, you can utilize the *Initial Physical Characteristics* properties to indicate the specific location. If you simply have a single row and auto-create one object, then using static numeric values for X, Y, Z, and Yaw will place the object in the specific location with a specific orientation. Setting a specific Length, Width, or Height of an object will alter the physical characteristics of that object. If you specify a Length for a link, that value will be bound as the *Logical Length* property and *Drawn to Scale* will be 'False'. Note that the X, Y, Z, and Yaw properties can also reference the name of a different numeric (Real or Integer) column in the same table. If the data is a column reference, changes in the data in the table will update the object's location in the Facility window, and an object's location change in the Facility window will update the data in the table. See the below TransferNode location values are based on the NodeX, NodeY and NodeZ column values.

Setting the *Symbol Index* sets an index into the auto-created object's associated symbol list that indicates which symbol to

display. If the object's *Current Symbol Index* expression property is left unspecified and its *Random Symbol* property is 'False', then this will stay as the object's symbol during a simulation run.

The screenshot displays a 3D environment with three server objects labeled 'server1', 'server2', and 'server3'. A data table is visible with columns: Object Instance Property, X, Y, Z, Length, Width, Height. The 'Object Instance Property' column contains 'server1', 'server2', and 'server3'. A properties panel on the right shows settings for 'ObjectInstanceProperty1' (Object Property). The 'Value' section includes: Default Value (null), Filter to Resources (False), Object Type (Source), Default Value Instantiation (AutoCreateInstance), Location Type (Cartesian), and Initial Physical Characteristics (X, Y, Z, Length, Width, Height, Yaw). The 'Appearance' section shows Display Name (ObjectInstanceProperty1). The 'Advanced Options' section shows Auto-set Table Row Reference (False). The 'Operational Planning' section is also visible.

Fixed Object's Associated Node Locations Specified Within Table

On an Object Instance Property column, when an object is being auto-created, the Initial Physical Characteristics can be specified for the object. In this section, there is an option to set the *Associated Node Locations*. This is a list of Nodes and their location information allowing the user to control where the object's nodes will be created. The *Node Name* must be the exact name of the node in the object's definition. For example, for a Server, you might put "Input" or "Output" as those are the Node Names for the associated Basic Node and Transfer Node attached to a Server Object. If specifying the *Node Name* in a table, the column Type should be a String type, not a Node Object Reference column. The *Node Name* is the string name of the node.

The screenshot shows a data table with columns: Object Instance Property, X, Y, Z, In Node, In X, In Y, In Z, Out Node, Out X, Out Y, Out Z. The 'Object Instance Property' column contains 'A' and 'B'. A properties panel on the right shows settings for 'ObjectInstanceProperty1' (Object Property). The 'Value' section includes: Default Value (null), Filter to Resources (False), Object Type (Server), Default Value Instantiation (AutoCreateInstance), Symbol Index, Location Type (Cartesian), and Initial Physical Characteristics (X, Y, Z, Length, Width, Height, Yaw). The 'Appearance' section shows Display Name (ObjectInstanceProperty1). The 'Advanced Options' section shows Auto-set Table Row Reference (True). The 'Operational Planning' section is also visible. A dialog box titled 'Auto-Created Object Associated Node Placement - Repeat Group Editor' is open, showing a list of items and a properties panel.

This properties can point to rows in a table. This information can live in the same table the auto-created object lives, or in a table related with a Key Relationship. The auto-created object must have the Key and the Node Locations can live in a table with a Foreign Key.

The screenshot shows a data table with columns: Object Instance Property, X, Y, Z. The 'Object Instance Property' column contains 'A' and 'B'. A properties panel on the right shows settings for 'ObjectInstanceProperty1' (Object Property). The 'Value' section includes: Default Value (null), Filter to Resources (False), Object Type (Server), Default Value Instantiation (AutoCreateInstance), Symbol Index, Location Type (Cartesian), and Initial Physical Characteristics (X, Y, Z, Length, Width, Height, Yaw). The 'Appearance' section shows Display Name (ObjectInstanceProperty1). The 'Advanced Options' section shows Auto-set Table Row Reference (False). The 'Operational Planning' section is also visible. A dialog box titled 'Auto-Created Object Associated Node Placement - Repeat Group Editor' is open, showing a list of items and a properties panel.

Object Type Specified Within Table

As was shown above, the Object Reference has a property named *Object Type* that can be set to the type of object. If a specific type is designated, that type will be used for the entire column. However, the *Object Type* property of an Object Reference column can be set to look at an Object Type column. Referring to the example below, an Object Reference column of 'Object Type' has been added to the table and named 'WhatKind'. The values available within this column are based on a list of objects (excluding links) from the Project and any loaded libraries. As shown, the WhatKind column contains many different types of objects, including Resource, Server, Separator, and so on. The column to the right, named 'ObjectName', is an object property column (similar to the example above where WhichNode was a node property column). This 'ObjectName' property then references the 'WhatKind' column in its *Object Type* property, as shown in red on the right side.

The screenshot shows the 'Table1' data table with the following data:

What Kind	Object Name	x	y	z
Resource	Resource1	0	0	0
Server	Server1	10	0	0
Separator	Separator1	20	0	0
Worker	Worker1	30	0	0
Vehicle	Vehicle1	40	0	0
Resource	Resource2	50	0	0
Separator	Separator2	60	0	0
TransferNode	TransferNode1	70	0	0
TransferNode	TransferNode2	80	0	0

The 'Properties: ObjectName (Object Property)' window shows the following settings:

- Value: null
- Filter to Resources: False
- Object Type: WhatKind
- Default Value Instantiation: AutoCreateInstance
- Location Type: Cartesian
- Initial Object Offset: x,y,z
- X: 0, Y: 0, Z: 0
- Display Name: ObjectName
- Advanced Options: Auto-set Table Row Reference: False
- Operational Planning: General
- Name: ObjectName

The resulting Facility window objects that are created from the Table1 discussed above can be seen below. Note that changing the object type ('WhatKind' column), object reference ('ObjectName' column) or location (X, Y, Z columns) within the table will change the type, name or location in the Facility and vice versa. Any changes in the Facility window of objects that were auto-created will change the corresponding table column values.

The Facility window displays a layout of objects corresponding to the data in Table1. The objects are arranged in a horizontal line, with their positions determined by the 'x' column values. The 'Table1' data table is shown below the Facility window.

What Kind	Object Name	x	y	z
Resource	Resource1	0	0	0
Server	Server1	10	0	0
Separator	Separator1	20	0	0
Worker	Worker1	30	0	0
Vehicle	Vehicle1	40	0	0
Resource	Resource2	50	0	0
Separator	Separator2	60	0	0
TransferNode	TransferNode1	70	0	0
TransferNode	TransferNode2	80	0	0

Auto-Created Link Objects

Similar to auto-creating fixed, node or transporter type objects, table data can be used to auto-create links between nodes in the Facility window. This is typically done within a separate table from the other fixed object creation, as the data required is different.

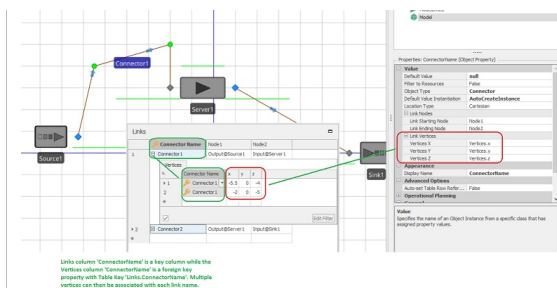
When an object's *Default Value Instantiation* is 'AutoCreateInstance' and the *Object Type* is a link type object, the *Link Starting Node* and *Link Ending Node* properties should reference the associated Node property column. In the example below, two connectors are automatically created using the 'Node1' and 'Node2' node property columns, which are referenced in the Object property column (ConnectorName).

The Facility window shows a link between two nodes, 'Node1' and 'Node2'. The 'Links' table is shown below the Facility window.

Connector Name	Node1	Node2
1	Node1	Node2
2	Node1	Node2

The above shows a simple link between two nodes with no vertices within the link itself. Relational tables can be used to generate vertices within the auto-created links. Note that the Link Vertices are referenced within the object column's properties (either Cartesian - Vertices X, Vertices Y and Vertices Z or Geographical - Vertices Latitude, Vertices Longitude and Vertices Elevation).

First, the object property within the Links table (ConnectorName in this example) should be specified as a key column by using the Set Column As Key button. An additional table should be added (named Vertices in the example below) that can reference the link object name key column using a Foreign Key Reference property within the table. Real type properties can be used within this relational table to reference the one or more vertices locations.



The link's Type can be specified in a table. When referencing the Link's Type in a table, the link will show unidirectional arrows during design time, but during runtime the arrows for bidirectional links will change to bidirectional arrows.

Creating and Updating Table-Based Objects

See the [Creating New Objects](#) page, specifically the section titled "Creating an Object by Subclassing Another Object Including Default Data Using 'Create Object From This' Option" for information on creating new objects within a project. These objects, populated with default data from the same or other tables, can also be referenced from an Object Type column in the table.

Finally, it is important to note that if an auto-created object's row in the table is deleted, the object will be deleted from the Facility window as well. Objects in the Facility window that are auto-created from a table cannot be deleted from the Facility window itself, but must be deleted by removing the corresponding row within the appropriate data table.

Note: Auto-created objects do not appear in the expression editor, however you can manually type them in.

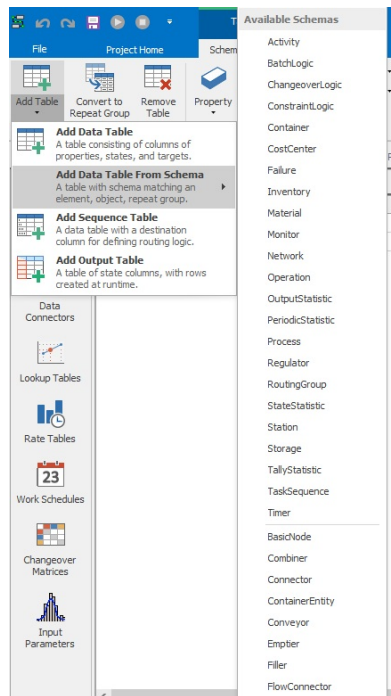
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Data Table From Schema

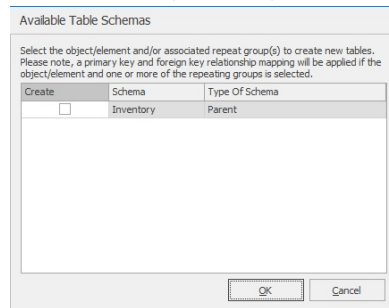
Data Table From Schema

Within the Data tab, selecting the Table panel allows a user to add a data table from a schema. Selecting this allows a table to be created based on the schema for elements, objects, and/or repeat groups.



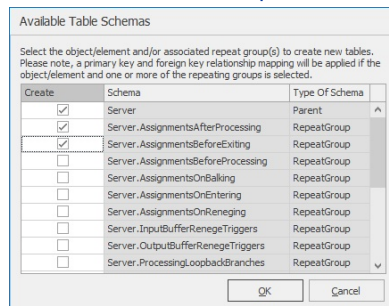
Element schema table options will include the parent type and all of its repeat groups.

Inventory Schema Example





Object schema table options will include the parent type and all of its repeat groups.

Server Schema Example



Parent tables will have the Primary Key column set up along with all the schema columns.

Primary Key column within an Inventory schema

Inventories														Table 1	Table 2	Table 3	Table 4										
		Inventory Name1	Report Statistics	Material Name	Site Object Name	Initial Quantity	Review Period	Review Timer Name	Replenishment Policy	On Replenishment Order Process	Allocation Ranking Rule	Allocation Ranking Expression	Assume Infinite Availability If	All													
																											

Properties: InventoryName1 (Inventory Element Property)

Value	
Default Value	null
Appearance	
Display Name	InventoryName1
Advanced Options	
Reference Type	Reference
Auto-set Table Row Reference	False
Operational Planning	
General	
Name	InventoryName1
Description	
Required Value	True

Repeat group tables that are selected along with the parent table will have the appropriate Foreign Key column in addition to the schema columns.

Inventories	Table1	Table2	Table3	Table4
	ServerName1_1	State Variable Name	New Value 0	

Properties: ServerName1_1 (Foreign Key Property)

Value	
Default Value	
Table Key	Table1.ServerName1
Appearance	
Display Name	ServerName1_1
Operational Planning	
General	
Name	ServerName1_1
Description	
Required Value	True

Selecting a Repeat Group without a parent table will create a table without a Foreign Key column (i.e. Material.BillOfMaterials).

Inventories	Table1	Table2	Table3	Table4	Table5	Table6
	Component Material Name	Component Quantity				

Dropdown options for the Primary Key will show appropriate values based on the type.

Inventories	Table1	Table2	Table3	Table4	Table5	Table6	Table7
	Timer Name1	Report Statistics	Interval Type	Time Offset (hours)	Time Interval (hours)		

null
 Timer1
 Inventories.ReviewTimerName
 Table4.ReviewTimerName
 ParentCostCenter

Additionally, setting the *Reference Type* to 'Create' will auto-populate the *Initial Property Values*.

Initial Property Values - Repeating Property Editor

Items:	
Name: 'ReportStatistics', Value: 'Table8.ReportStatistics'	
Name: 'LocationBasedInventory', Value: 'Table8'	
Name: 'InitialQuantity', Value: 'Table8.InitialQuantity'	
Name: 'ReviewTimerName', Value: 'Table8.ReviewTimerName'	
Name: 'ReplenishmentPolicy', Value: 'Table8.ReplenishmentPolicy'	
Name: 'OnReplenishmentOrderProcess', Value: 'Table8.OnReplenishmentOrderProcess'	
Name: 'CostPerUnit', Value: 'Table8.CostPerUnit'	
Name: 'AllocationRankingRule', Value: 'Table8.AllocationRankingRule'	
Name: 'AllocationRankingExpression', Value: 'Table8.AllocationRankingExpression'	
Name: 'AssumeInfiniteAvailabilityCondition', Value: 'Table8.AssumeInfiniteAvailabilityCondition'	
Name: 'AllowPartialAllocationCondition', Value: 'Table8.AllowPartialAllocationCondition'	
Name: 'AllowBackorderPolicy', Value: 'Table8.AllowBackorderPolicy'	
Name: 'BackConditionOnProbability', Value: 'Table8.BackConditionOnProbability'	
Name: 'OnBalkedAtBackorderProcess', Value: 'Table8.OnBalkedAtBackorderProcess'	
Name: 'BackorderMaxWaitTime', Value: 'Table8.BackorderMaxWaitTime'	
Name: 'OnRenegedBackorderProcess', Value: 'Table8.OnRenegedBackorderProcess'	
Name: 'LogMaterialUsage', Value: 'Table8.LogMaterialUsage'	
Name: 'BillOfMaterials', Value: 'Table8.BillOfMaterials'	
Name: 'BillOfMaterials[0].ComponentMaterialName', Value: 'Table8.BillOfMaterials[0].ComponentMaterialName'	
Name: 'BillOfMaterials[0].ComponentQuantity', Value: 'Table8.BillOfMaterials[0].ComponentQuantity'	

Add Delete

Properties:

General	
Property Name	ReportStatistics
Value	Table8.ReportStatistics

General

Properties: MaterialName1 (Material Element Property)

Value	
Default Value	null
Appearance	
Display Name	MaterialName1
Advanced Options	
Reference Type	Create
Initial Property Values	21 Rows
Auto-set Table Row Reference	False
Operational Planning	
General	
Name	MaterialName1
Description	
Required Value	True

Reference Type

If set to 'Create', then an element will be auto-created for the proper instance, and changes to the property value will rename the element set to 'Reference', then changes to the property value will change the reference to the element with that name.

When the Parent Table Schema was created, if the associated RepeatGroup Tables were also created at the same time, then the *Initial Property Values* will include the Repeat Group Table information. For more information see the [Table-Based Elements \(Auto-Create\)](#) page.

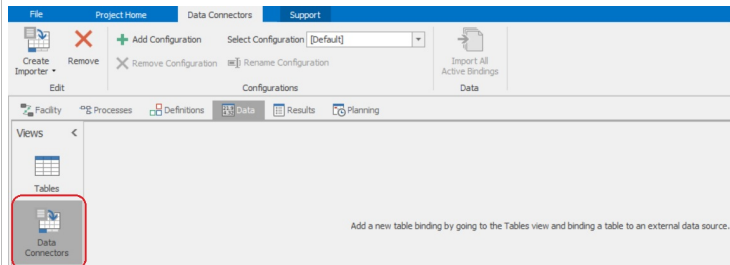
DataConnectors

The Data Connectors view within the Data ribbon displays two types of table bindings for importing data. Prior to Simio Sprint 202, all data binding was done through the use of what we now call 'Classic' bindings. That is, using the Create Binding option on the Content ribbon to bind to a Database, CSV file, Excel, Wonderware MES or XML Transformation. For models built prior to Simio Sprint 202 that include bound tables, these classic bindings will be seen in the Data Connectors area. From Simio Sprint 202 on, the Data Connectors view allows users to Create Importers which can then be used to bind multiple tables using a similar data import structure. This type of data importing is described within this help section. Creating Data Importers through the Data Connectors view is available with all Simio licenses. Later in this section, the Create Exporters functionality, which is available for RPS licenses only, is discussed.

Data Importers

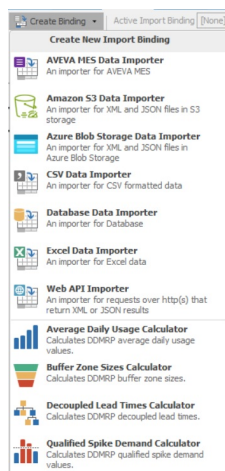
Creating and Defining a Data Importer

To define a data connector, first click on the Data Connectors button within the Data view. Data tables do not have to exist first before data importers can be defined.



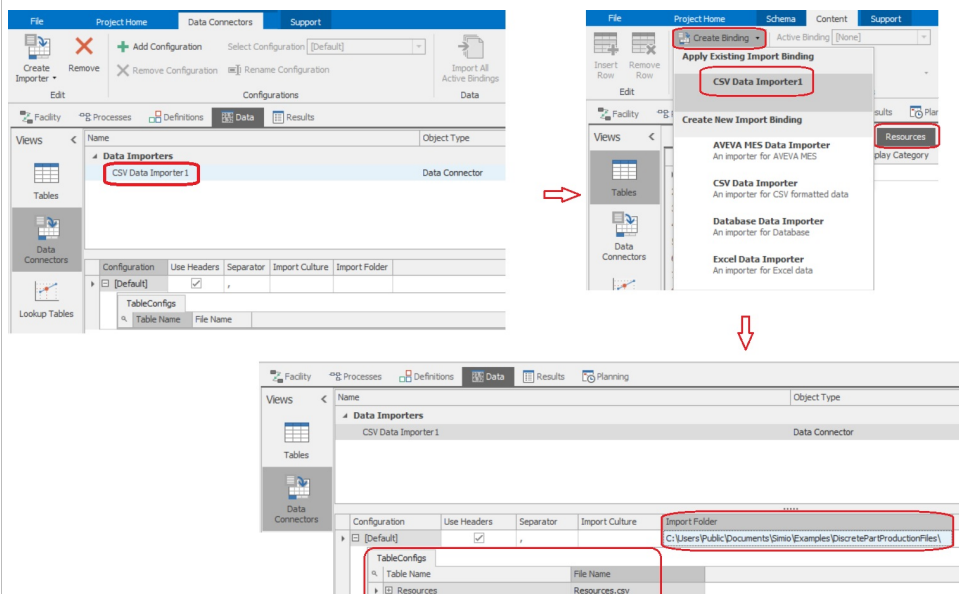
Once in the Data Connectors view, the Create Importer drop-down allows for several different types of importers, such as AVEVA MES Data Importer, Amazon S3 Data Importer, Azure Blob Storage Data Importer, CSV Data Importer, Database Data Importer, Excel Data Importer, Web API Importer, Average Daily Usage Calculator (RPS Only), Buffer Zone Sizes Calculator (RPS Only), Decoupled Lead Times Calculator (RPS Only), Qualified Spike Demand Calculator (RPS Only), Excel Data Importer, or WebAPI Importer, as shown below.

Note: The AVEVA MES Data Importer requires Simio 32bit to work.



Once a data importer is created, the configuration characteristics for the importer can be defined. The configuration will include a configuration name (initially Default), as well as characteristics based on the data importer type. For example, for a CSV Data Importer, the *Use Headers* boolean, *Separator*, *Import Culture* and *Import Folder* may be specified. The *Table Name* properties in the TableConfigs tab will ONLY appear once a data table has applied the data importer - this is done through the *Create Binding* option on the Content ribbon in the Data view.

When one or more data importers have been defined, the importer will be displayed within the Data tables. *Create Binding* list under the Apply Existing Data Import Binding section. By applying an existing import binding (in this example, CSV Data Importer1), the data table name will then automatically appear within the contents of the configuration for that data importer under the TableConfigs tab. In the example below, the Resources data table has the CSV Data Importer1 applied and thus, 'Resources' appears as the *Table Name* under the (Default) configuration (using the + to expand). The associated *File Name* property can then be specified with the table, in this case 'Resources.csv' will be used for the Resources table. The *Import Folder* property will indicate the location for all tables within this configuration.



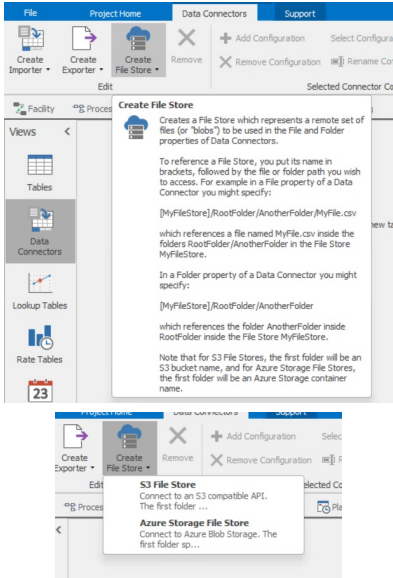
Note that if no *Import Folder* property is specified, the folder and file name may be defined within the *File Name* property for the given data table.

Configuration	Use Headers	Separator	Import Culture	Import Folder
Default	<input checked="" type="checkbox"/>	,		
TableConfigs				
Table Name	File Name			
Resources	C:\Users\Public\Documents\Simio\Examples\DiscretePartProductionFiles\Resources.csv			

File Stores

A File Store represents a remote set of files or "blobs" to be used in the *File* and *Folder* properties of Data Connectors. To reference a File Store, you put its name in brackets, followed by the file or folder path you wish to access. For example, in a *File* property of a Data Connector you might specify: [MyFileStore]/RootFolder/AnotherFolder/MyFile.csv which references a file named MyFile.csv inside the folders RootFolder/AnotherFolder in the File Store MyFileStore. In a *Folder* property of a Data Connector you might specify: [MyFileStore]/RootFolder/AnotherFolder which references the folder AnotherFolder inside RootFolder inside the File Store MyFileStore.

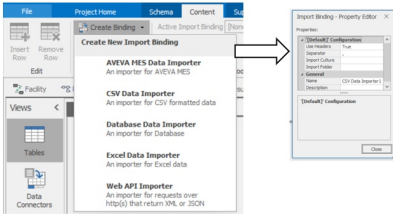
Note: For S3 File Stores, the first folder will be an S3 bucket name, and for Azure Storage File Stores, the first folder will be an Azure Storage container name.



Content Ribbon - Creating a Data Binding

A user can also create a data binding directly through the Content ribbon, Create Binding option (Tables view once a Data Table exists). If there are no existing import bindings, the user will be prompted to create an import binder, similar to the option within the Data Connectors window. See the [Importing and Binding to Tables](#) page for more information.

'Create Binding' Option to Create New Import Binding



Example - SchedulingDiscretePartsProduction

As shown below, multiple tables can have the same data importer applied. Within the Data tab view of each table, the Active Binding displays the data importer (red) and the import folder and file name are displayed with the table (green).

The screenshot shows the Simio software interface with the 'Data Connectors' tab selected. The 'Data Importers' section is expanded, showing 'CSV Data Importer 1'. The 'TableConfigs' table lists various data sources like Resources, RoutingDestinations, Materials, etc. A red box highlights the 'Active Binding' dropdown in the 'Data Tools' section, which is set to 'CSV Data Importer 1'. A green arrow points from this dropdown to the 'Import Folder' field in the 'TableConfigs' table, which contains the path 'C:\Users\Public\Documents\Simio\Examples\DiscretePartProductionFiles\'. A red arrow points from the 'TableConfigs' table to the 'Data Tools' section.

Connection Strings

If a Data Connector has been set up to a database, you will need to specify a connection string to that database.

An SQL database connection string can follow the following structure:

Server=servername;Database=databasename;uid=username;pwd=password. Note: For MySQL, you will need to specify the table using 'schema.tablename'.

An Access database connection string can follow the following structure: Provider=Microsoft.ACE.OLEDB.12.0;Data Source=path\to\database.

An Oracle database connection string can follow the following structure: User ID=system;Password=root;Data Source=

(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=HostValue)(PORT=PortValue))(CONNECT_DATA=(SID=XE)));

Adding and Renaming Configurations

A single Data Importer may consist of multiple 'configurations', that is, import locations and data files for the same importer. The Add Configuration button on the ribbon will prompt the user for a configuration name and add the configuration to the importer (setting it to the active configuration as shown in the Select Configuration property). A configuration only needs to have values set for those settings that need to be different from the [Default] configuration. When a configuration is set as the active configuration, it will use the value for a given setting from the [Default] configuration if it doesn't have a value set. For example, the [Default] configuration may provide a file name to use, as well as several other settings. A new configuration then may choose to only provide a different file name, because it wants to use the values for all other settings from the [Default] configuration without changes.

Any data table names that have the data importer applied will automatically appear in the TableConfigs tab with the Table Name. The user may then use a different Import Folder and/or File Name for each table. The Select Configuration property on the ribbon determines the 'active' configuration that is used for importing. Configurations (except for the [Default] configuration) can be renamed by selecting the configuration and using the Rename Configuration button.

Note that in the below example, the Config2 configuration automatically will use the [Default] configuration settings for Use Headers, Separator and Input Culture properties, as they are blank and don't override the [Default] configuration. The Import Folder property, however, will override the [Default] setting to be the 'C:\Program Files\Simio\LLC\Simio\Examples\DiscretePartProductionFiles_Config2\'. directory.

The screenshot shows the Simio software interface with the 'Data Connectors' tab selected. The 'Data Importers' section is expanded, showing 'CSV Data Importer 1'. The 'TableConfigs' table lists various data sources like Resources, RoutingDestinations, Materials, etc. A red box highlights the 'Active Binding' dropdown in the 'Data Tools' section, which is set to 'Config2'. A green arrow points from this dropdown to the 'Import Folder' field in the 'TableConfigs' table, which contains the path 'C:\Users\Public\Documents\Simio\Examples\DiscretePartProductionFiles_Config2\'. A red arrow points from the 'TableConfigs' table to the 'Data Tools' section.

Experimentation and Data Connectors

Each data connector can have multiple configurations, as discussed above. For each table that has a data connector import binding with 2 or more configurations, there is a column within the Experiment Design view of an experiment. The header of a column is the Data Importer name and the cell values are a drop down of the configurations for that data importer. The default value of the cell for a new scenario will be the currently active databinding for that data importer. When using the OptQuest add-in, Simio also uses the default active binding for each table, which is set via the data table's 'Active Import Binding' or 'Active Export Binding'.

Scenario	Replications	Importer	Configurations
Scenario1	Required	Completed	CSV Data Importer 1
Scenario2	10	0 of 10	Testing
Scenario3	10	0 of 10	Testing
Scenario4	10	0 of 10	Testing

Token Replacements for Data Connector Property Values

When specifying values for data connector properties, you can use a special *Token Replacement* syntax to provide values from other places. The supported token formats are:

- `$(property:PropertyName)` - Replaced with the string value of the given *PropertyName*
- `$(datetime:regionnow:DateTimeFormat)` - Replaced with the local wall clock time (or current time in the time zone specified for the 'Use Current...' model start time type) formatted with the given *DateTimeFormat*. Valid format strings can be found for [standard formats](#) and for [custom formats](#).
- `$(datetime:regionnow:DateTimeFormat:HourOffset)` - Same as above, except the datetime value is offset by some real number of hours specified by *HourOffset*

As an example: A model with a property called 'MyProperty' with string value 'Blue', where current UTC date time was February 3, 2020 7:00:00 PM, and model time zone set to 'Eastern Standard Time' then a data connector property value of:

`$(property:MyProperty)_$(datetime:regionnow:yyyyMMdd-HH:3)`

Will result in the value:

Blue_20200203-17-00

For the above date time calculation, first 3 hours was added to the UTC time (making it Feb 3, 2020 10:00 pm), then that time was converted to Eastern Standard Time (Feb 3, 2020 5:00 pm). Using the format string 'HH' formatted the hours in 24 hour format, thus resulting in '17' instead of '5'.

You can use the *Import DateTimeFormat* property of a *DateTime* Property column to specify the regional format to align with your locale. This property should contain a format specifier character (see [.NET Standard Date and Time Format Strings](#))

Database Importer

Connector Level Properties

Data Provider - Database provider...By default, SqlClient Data Provider and OracleClient Data Provider are installed part of Windows. MySQL Data Provider and SAP HANA Data Provider will be available after their client providers are installed. Although OracleClient Data Provider is listed, the Oracle .NET Provider need to be installed and configured before it is available.

Connection String - Connection String of the data provider. Here is a SQL Server connection string example is "Server=;SQLExpress;Database=SimioSDPP;Trusted_Connection=True". Oracle connection string example is "User Id=system/Password=root@Data Source=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=10.74.64.48)(PORT=1521)))CONNECT_DATA=(SID=XE)";. MySQL connection string example is "Server=localhost; Port=3306; Database=simiodemo; Uid=root; Pwd=root". SAP HANA connection string example is "Server=192.168.56.101:39015;User ID=system;Password=xyz". Ole Db connection string is "Data Source=myServerAddress;Initial Catalog=myDatabase;Integrated Security=SSPI";.

Connection Time Out - Number of seconds before database connection times out.

Table Level Properties

Data Object - If IsStoredProcedure==false and IsSQLStatement==false, this is the table name or view. If IsStoredProcedure==true and IsSQLStatement==false, this is the stored procedure name. If IsSQLStatement==true, this is an SQL Statement.

Is Stored Procedure - determines if Data Object is a stored procedure. If IsStoredProcedure==true and IsSQLStatement==false, the Data Object is a stored procedure name.

Is SQL Statement - determines if Data Object is a SQL Statement. IsSQLStatement==true, the Data Object is a SQL Statement.

Column Level Properties

Source Column Name - By default, data result column name must match Simio table column. If Source Column Name is defined, this is the column name of the data result that will be mapped to the Simio table column.

Web API Importer Properties

Connector Level Properties

Listed below are the properties of a **Web API Importer**:

Property	Description
URL	The HTTP method to use
Message	The message to post if Method is set to POST. Can contain tokens (in the form \$(name)) to be later replaced by per-table settings.
Headers	List of header name-value to send along with the web request. If the request returns XML, set Content-Type to application/xml here.
Authentication Type	Indicates what the of authentication should be used for this request. None - No authentication (an anonymous request). Basic - Basic authentication sends a Base64-encoded string that contains a user name and password for the client. Base64 is not a form of encryption and should be considered the same as sending the user name and password in clear text. Negotiate - Negotiates with server to select either Kerberos or NTLM. Kerberos - Used for Windows credentials. More secure than NTLM. NTLM - Used for Windows credentials. More secure than Digest. Digest - Used for Windows credentials. CurrentUser - Uses the credentials of the user account running the application. MSAL - Use account used to signed-in in Windows(WAM). MSAL - Use one of the Accounts known by Windows(WAM). MSAL - Use any account(Azure AD). MSAL - Authenticate the user silently using username and password.
User Name	Optional user name when creating credentials for the web request.
Password	Optional password used when creating credentials for the web request.
Domain	Optional domain used when creating credentials for the web request.
MSAL Parameters	MSAL Parameters
Bearer Token Form Parameters	Bearer Token Form Parameters
Bearer Token Token Replacements	Bearer Token Token Replacements
BearerTokenHeaders	List of bearer token header name-values to send along with the web request. If the request returns XML, set ContentType to application/xml here.
Bearer Token Result Name	Bearer Token Result Name
Bearer Token Refresh Interval (minutes)	Bearer Token Refresh Interval in minutes
Method	The HTTP method to use
Number Of Records Parameter	NumberOfRecordsParameter is used to specify the number of records to return for each Web API call. The parameter must also be specified in the URL. Defining this parameter along with the StartOffsetParameter will enable the importer to page during the importing of data. The NumberOfRecordParameter that is defined in the URL is used to determine the number of rows to retrieve during each call. This property only defines what parameter to look for.
Starting Offset Parameter	StartOffsetParameter is used to specify the number of records to skip. This parameter must also be specified in the URL. Defining this parameter along with the NumberOfRecordsParameter will enable the importer to page during the importing of data. The StartOffsetParameter is used as a place holder of the current set of paged rows. This property only defines what parameter to look for. The value for the parameter is defined in the URL and typically should be set to zero. The import will increment this parameter for each call based on the NumberOfRecordsParameter value. The importer will keep incrementing this parameter and calling end point until no records are returned. When no records are returned, this signals that all the data has been imported.

Table Level Properties

Method - Override Method Used For Import...To use Connector Property, set to "[DEFAULT]"

FormParameters - Form parameters are used with Content-Type= multipart/form-data and application/x-www-form-urlencoded. Simio Portal uses form-data (Content-Type= multipart/form-data) and SAP uses application/x-www-form-urlencoded.

TokenReplacements - Values used for tokens specified in either the URL or Message of the Data Connector. The values here can themselves contains tokens of the form \$(colName) to use column values by column name.

Stylesheet – XSLT 1.0 Stylesheet used to transport source data into Simio table.

ElementNameForCount – Used to count number of rows in response. This is helpful when using paging and filtering data using the stylesheet. The stylesheet might filter the response to zero row, but the ElementNameForCount will be used to determine if whether there are more rows to import.

RequestDebugFileFolder – Used to test importer off-line. If this folder name is populated, the importer will pull messages from this folder instead of calling the URL.

ResponseDebugFileFolder – Used to see the raw response from the URL before the response is processed. This is used in troubleshooting.

Database Exporter

Connector Level Properties

Data Provider – Database provider...By default, SqlClient Data Provider, OracleClient Data Provider, OleDb Data Provider and Odbc Data Provider are installed part of Windows. MySQL Data Provider and SAP HANA Data Provider will be available after their client providers are installed. Although OracleClient Data Provider is listed, the Oracle .NET Provider need to be installed and configured before it is available.

Connection String – Connection String of the data provider. Here is a SQL Server connection string example is "Server=\\SQLExpress;Database=SimioSDPP;Trusted_Connection=True". Oracle connection string example is "User Id=system;Password=root;Data Source=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=10.74.64.48)(PORT=1521))(CONNECT_DATA=(SID=XE)))"; MySQL connection string example is "Server=localhost;Port=3306;Database=simiodemo;Uid=root;Pwd=root". SAP HANA connection string example is "Server=192.168.56.101:39015;UserD=system;Password=xyz". Ole Db connection string is "Data Source=myServerAddress;Initial Catalog=myDataBase;Integrated Security=SSPI"; ODBC connection string example is "DSN=ABCD"

Connection Time Out – Number of seconds before database connection times out.

Provider Settings – Data provide specific options. Currently, "OracleDateFormat" is the only provider setting. This is the DateTime format used to save to the database (defaulted to yyyy-MM-dd HH24:miss).

Date Time Format – DateTimeFormat of the source data...This is the local format used to read the datetime as a localized datetime.

Web API Exporter Properties

Connector Level Properties

Listed below are the properties of a **Web API Exporter**.

Property	Description
URL	The URL of the http(s) request. Can contain tokens (in the form \$(name) or of the form \$(colName) to use column values by column name) to be later replaced by per-table settings.
Message	The message to post if Method is set to POST. Can replace tokens (in the form \$(name) or of the form \$(colName) to use column values by column name) to be later replaced by per-table settings.
Headers	List of header name-value to send along with the web request. If the request returns XML, set Content-Type to application/xml here.
FetchHeaders	List of fetch header name-values to send along with the web request (only used with PATCH and PUT web request methods). If the request returns XML, set Content-Type to application/xml here.
Authentication Type	Indicates what the of authentication should be used for this request. None - No authentication (an anonymous request), Basic - Basic authentication sends a Base64-encoded string that contains a user name and password for the client. Base64 is not a form of encryption and should be considered the same as sending the user name and password in clear text. Negotiate - Negotiates with server to select either Kerberos or NTLM. Kerberos - Used for Windows credentials. More secure than NTLM. NTLM - Used for Windows credentials. More secure than Digest. Digest - Used for Windows credentials. CurrentUser - Uses the credentials of the user account running the application. MSAL - Use account used to signed-in in Windows(WAM). MSAL - Use one of the Accounts known by Windows(WAM). MSAL - Use any account(Azure AD). MSAL - Authenticate the user silently using username and password.
User Name	Optional user name when creating credentials for the web request.
Password	Optional password used when creating credentials for the web request.
Domain	Optional domain used when creating credentials for the web request.
MSAL Parameters	The valid parameters used as part of the authorization process are clientid, tenant, instance and scopes.
Bearer Token Form Parameters	When using a body type of 'form-data' or 'x-www-form-urlencoded', these are the parameters that are contained in the body of the message.
Bearer Token Token Replacements	Bearer Token Token Replacements
BearerTokenHeaders	List of bearer token header name-values to send along with the web request. If the request returns XML, set ContentType to application/xml here.
Bearer Token Result Name	Name of the element (or original JSON id) in an XML reponse that contains a bearer token.
Bearer Token Refresh Interval (minutes)	Bearer Token Refresh Interval in minutes
Export Start Time String	Used to specify the export start time based on the models regional setting. If left blank or valid, the time of the computer will be used.
Method	The HTTP method to use
Number Of Rows Per Call	Defines number of rows from the table that should be sent per web API call. If this number is less that the number or rows in the table, the export will call the web API multiple time. By default, this value should be set to 1. If set to one, a Row Message and Row Delimiter are not needed. The column mapping can happen within the Message. The Row Message can still be use if the value is set to 1, but the Row Delimiter will not be used since a message will be sent after each row is read.
Row Message	Message that will be added per row. The column mappings will typically be defined within the Row Message.
Row Delimiter	Text that will separate each row. For a JSON message, the delimiter is typically a comma. For an XML message, there typically will not be a row delimiter.

Table Level Properties

Method – Override Method Used For Export....To use Connector Property, set to "[DEFAULT]"

FormParameters – Form parameters are used with Content-Type=multipart/form-data and application/x-www-form-urlencoded. Simio Portal uses form-data (Content-Type=multipart/form-data) and SAP uses application/x-www-form-urlencoded.

TokenReplacements – Values used for tokens specified in either the URL or Message of the Data Connector. The values here can themselves contains tokens of the form \$(colName) to use column values by colum name.

FetchTokenReplacements – Token Replacements Used In Fetch Token Retrieval.

StatusFileName – Path and File Name of Export Status File. If left blank, no status file will be generated.

StatusDelimiter – Column Delimiter in Status File.

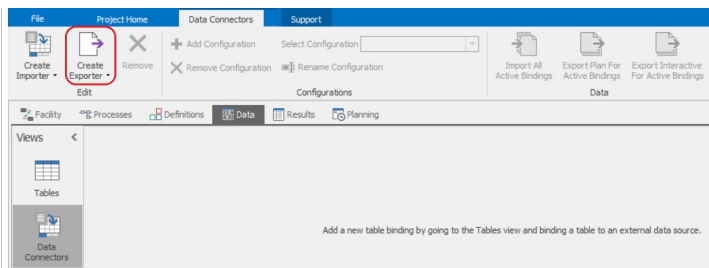
StatusSeverity – Status Severity Level. The options are "ALL" or "ERRORSANDWARNINGS". If "ALL", provide status for every export message. If "ERRORSANDWARNINGS", only provide status for errors and warnings.

ResponseDebugFileFolder – Used for debugging. If populated, this will save the response from the request into this folder.

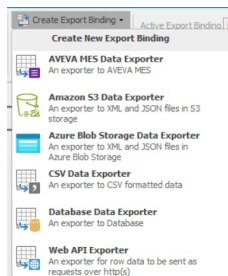
Data Exporters - Professional and RPS Edition only

Creating and Defining a Data Exporter

Creating a data exporter within RPS is similar to the above described data importer. To define a data connector, first click on the Data Connectors button within the Data view. Data tables do not have to exist first before the data exporters can be defined.

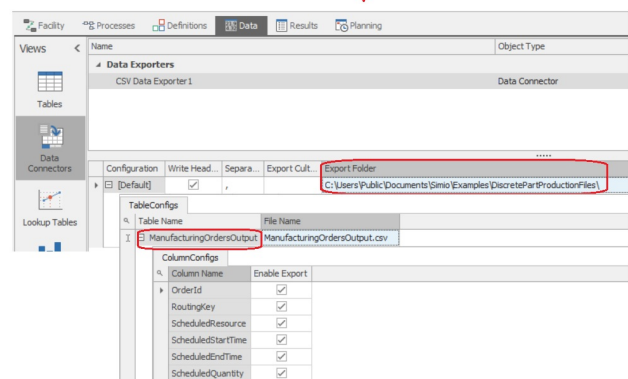
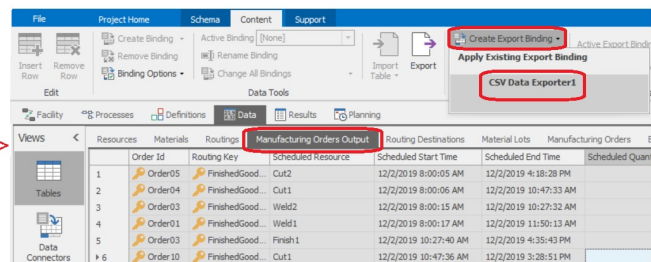
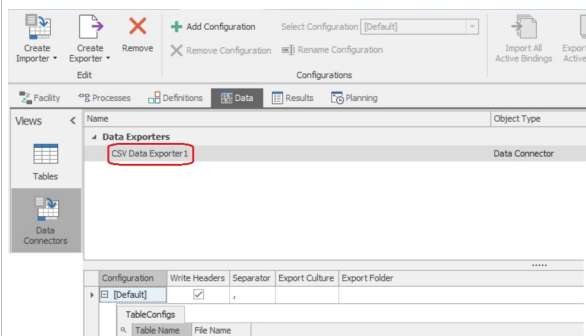


Once in the Data Connectors view, the Create Exporter drop-down allows for several different types of exporters, such as AVEVA MES Data Exporter, Amazon S3 Data Exporter, Azure Blob Storage Data Exporter, CSV Data Exporter and Database Data Exporter, and Web API Exporter as shown below.



Once a data exporter is created, the configuration characteristics for the exporter can be defined. The configuration will include a configuration name (initially [Default]), as well as characteristics based on the data exporter type. For example, for a CSV Data Exporter, the **Write Headers** boolean, **Separator**, **Export Culture** and **Export Folder** may be specified. The **Table Name** properties in the **TableConfigs** tab will ONLY appear once a data table has applied the data exporter - this is done through the **Create Export Binding** option on the Content ribbon in the Data view.

When one or more data exporters have been defined, the exporter will be displayed within the Data tables. **Create Export Binding** list. By applying an existing export binding (in this example, CSV Data Exporter1), the data table name will then automatically appear within the contents of the configuration for that data exporter under the **TableConfigs** tab. In the example below, the **ManufacturingOrdersOutput** data table has the CSV Data Exporter1 applied and thus, 'ManufacturingOrdersOutput' appears as the **Table Name** under the [Default] configuration (using the '+' to expand). The associated **File Name** property can then be specified with the table, in this case 'ManufacturingOrdersOutput.csv' will be used for the **ManufacturingOrdersOutput** table. The **Export Folder** property will indicate the location for all tables within this configuration. Expanding the data table name (**ManufacturingOrdersOutput**) displays the various columns within the table. An **Enable Export** boolean may be used to specify whether a column should be exported.



Note that if no **Export Folder** property is specified, the folder and file name may be defined within the **File Name** property for the given data table. See the above data importer example, which shows this case with data importers.

Targets within data tables will also be exported with data tables. When a model is run in interactive mode, the **Value** and **Status** of a Target will be exported. When running in planning mode, the export will include options for **Value**, **Status**, **Expected** and **Within Bounds Probability**. For each target, the **Enable Export** button boolean can be toggled within the Data Connector view, as seen above, or within the Target properties of the data table.

Export Options

When using the Data Exporters, the user has several options for exporting, shown below, which are available through the Export Options on the Content ribbon in Data view.

Export At End Of Plan Run - Export the data at the end of a plan run.

Export At End Of Risk Analysis Run - Export the data at the end of a risk analysis run.

Export At End Of Experiment Replication Run - Export the data at the end of each experiment replication run.

Export At Project Save - Export the data at project save.

Export Plan At Project Save - Export the plan data at project save.

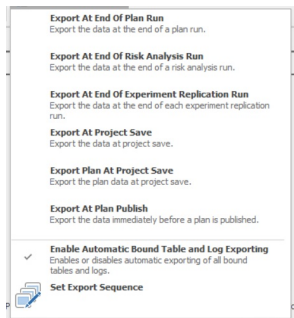
Export At Plan Publish - Export the data immediately before a plan is published.

Enable Automatic Bound Table and Log Exporting - Enables or disables automatic exporting of all bound tables and logs.

NOTE: 'Export At End Of Experiment Replication Run' only works with Web API and Database Exporters

NOTE: If user has both 'Export at End of Plan Run' and 'Export at End of Risk Analysis Run' turned on, and then selects the Analyze Risk button via the Operational Planning ribbon, the user will get two (2) exports, since via the ribbon command, we run the plan first and then run risk analysis.

Set Export Sequence allows you to specify the order that the table exports occur in when there are multiple data tables with export bindings in the model. This is particularly useful when certain data tables are dependent on the exportation of other data tables. To change the order, use the up and down arrows. The data tables will export from top to bottom.



As shown above, the Export button also allows the user to simply export the current table or all bound tables.
Database Data Exporter Data Export Types

Drop Create And Repopulate - Drop existing table if it exists, then create new database table and add data. The SQL Drop statement is used to remove a table definition (i.e., columns and table structure) and all the data.

Truncate And Repopulate - Create database table (if it does not exist), then truncate data and add data. The Truncate SQL command is used to delete table data but preserve the table definition and structure.

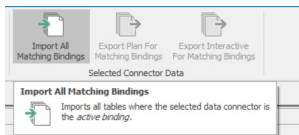
Update And Insert - Create database table data (if it does not exist). If data row already exists based on primary keys, update. If data row does not exist, add based on primary key. Note, the data table must have a primary key defined for this to function.

Update Insert And Delete - Create database table data (if it does not exist). If data row already exists based on primary keys, update. If data row does not exist, add data row and data to database. If export does not contain the row, but the database table does, delete the database table data row based on primary key. Note, the data table must have a primary key defined for this to function.

Insert - Create database table (if it does not exist) and insert data based on primary key. Note, the data table must have a primary key defined for this to function.

Import / Export by Data Connector

In the Data Connectors view, users can select a Data Connector and run and import/export for every table/log that has that particular Data Connector set as the Active Binding.

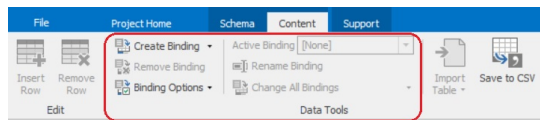


Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

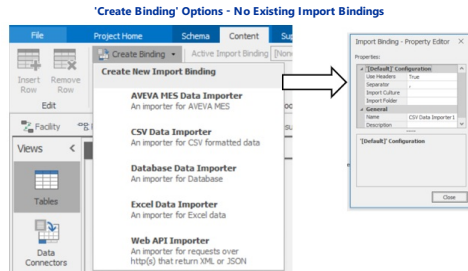
Importing and Binding To Tables

Once a table has been added to the Data window, the Content ribbon can be used for importing external data by applying existing or creating new Data Connectors. The user can import, export and bind to data tables to utilize data external to Simio. See the [Data Connectors](#) page for more information on data importers and configurations for data tables.



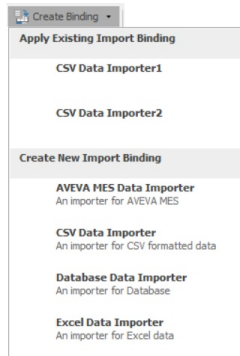
Content Ribbon - Creating a Data Binding

If there are no existing import bindings defined through the Data Connectors view, the user will be prompted to create an import binding by selecting one of the data importer options. Then, using the Import Binder property editor dialog, the user can provide information about the data connector, similar to defining it through the Data Connectors window.



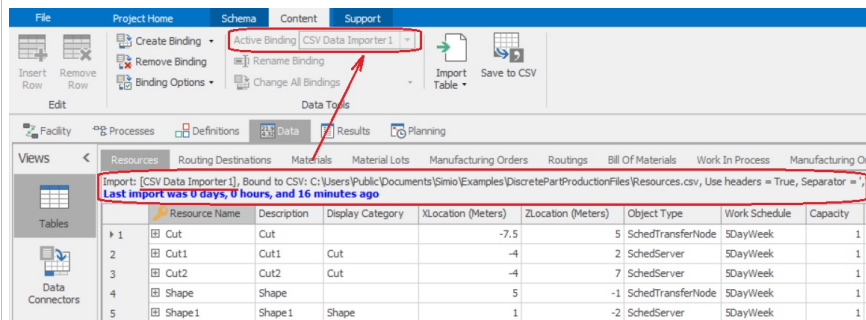
If there are existing data bindings, the user will be prompted to either select an existing import binding or create a new import binding.

'Create Binding' Options - Existing Import Bindings



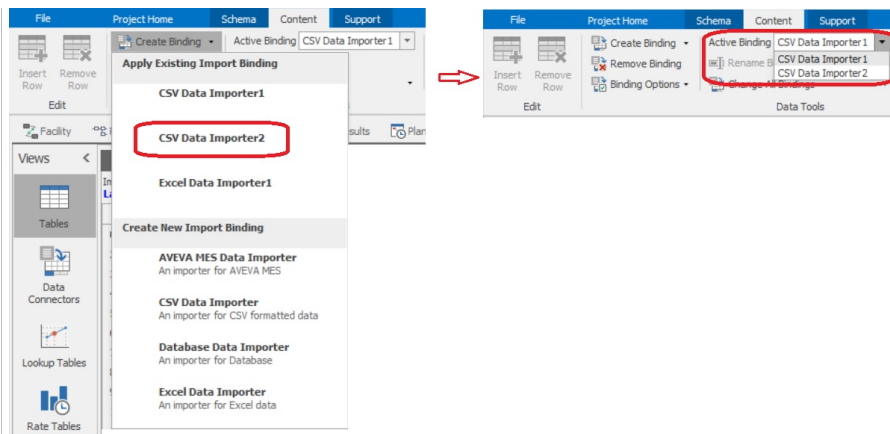
Active Data Binding

When a data table is first associated with a data import binding, the Active Binding property on the Content ribbon displays the import binding name, as shown below. Additionally, the data importer name, as well as the import folder, file name and other properties are displayed above the table. Note that if the data binding has only one configuration [Default], the data importer name, displayed above the table, will be shown as it is below. If the data importer has multiple configurations (see the [Data Connectors](#) page), the selected configuration will also be shown next to the data importer name.



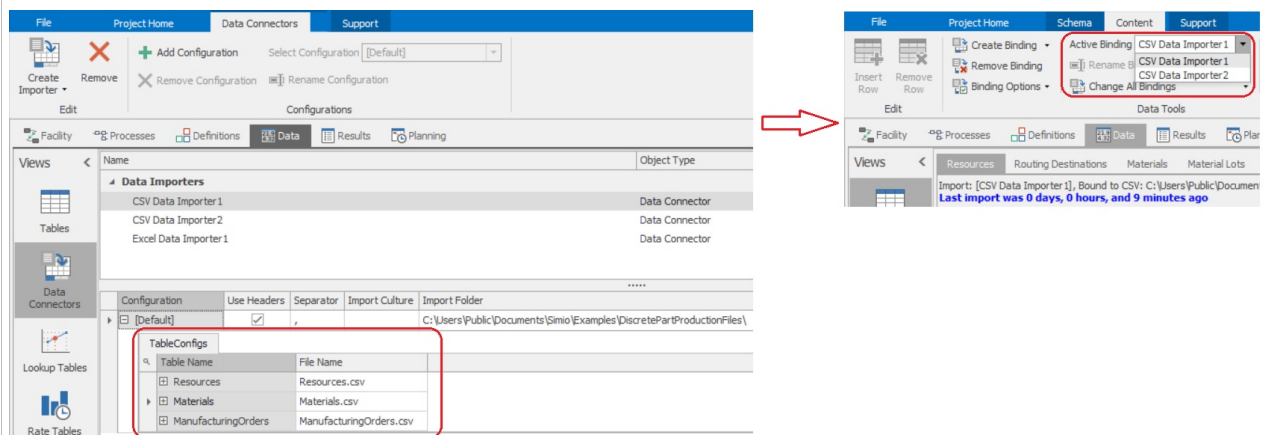
Multiple Data Importers Per Table

The Create Binding option can be used to associate more than one data importer with the same table. This allows users to have multiple data importers (and associated files) that can be selected through the Active Bindings property. Note that this is different than having multiple 'configurations' of the same data binding, as configurations can be used within experimentation for a specified data import binding (see [Data Connectors](#) for more information).

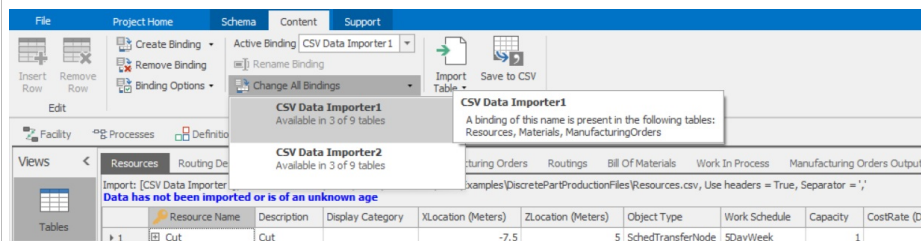


Change All Bindings

Many times it may be useful to use the same import binding across multiple tables, where the file names are specified within the TableConfigs section of the data importer. For example, perhaps there is an initial set of data that the model uses, as noted under the TableConfigs area of 'CSV Data Importer1', as shown below. Although all of the associated tables may have the same data importer(s), they may not all be the active binding for each table. **IMPORTANT NOTE:** When switching between active bindings for a particular table, the data will NOT automatically update within the table view. The Import button should be used to import the data if the Binding Option is 'Manual'. If the Binding Option is set to 'Automatic', then once the model begins running, the table will data will be updated.

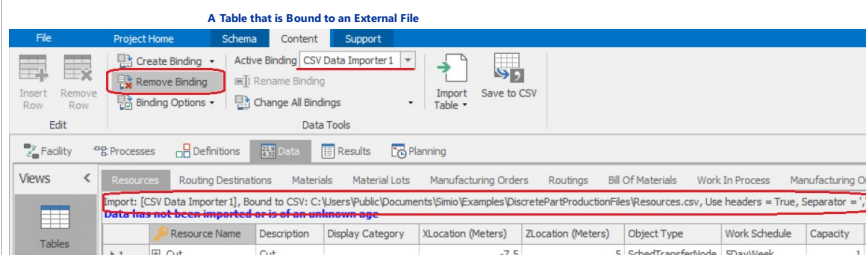


When more than one data table has the same import binding specified, the *Change All Bindings* option becomes visible. Simio notes within each of those similar named bindings which tables contain the particular name, as shown below. The *Change All Binding* option then makes it easy for users to switch all active data table bindings simultaneously to a particular named set.



Removing Data Bindings

Within a model and particular table in the Data window, if the table has an *Active Binding* property specified, this indicates that the table has been bound to an external file. If a selected table has one or more import binding, the *Remove Binding* button is active so that the binding can be removed. Simio provides the name of the file and the location in the space above the data, in order to direct users to the external file that is currently bound to that table. The *Remove Binding* button on the Table ribbon will remove the binding for the currently selected table's *Active Binding*. Holding down the Shift key while pressing the *Remove Binding* button will prompt the user whether they'd like to unbind all the *Active Binding* importers in the model. Note that this currently removes all active bindings only (even if the active binding is a different binder in each table).

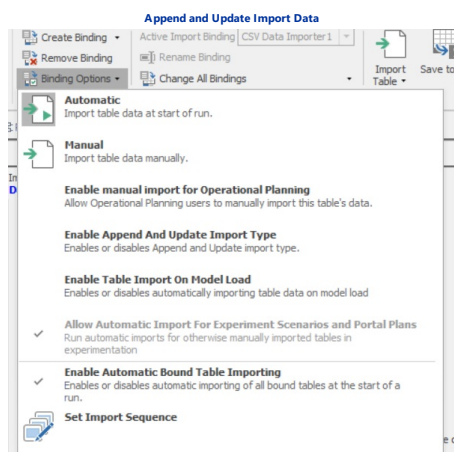


Binding Options

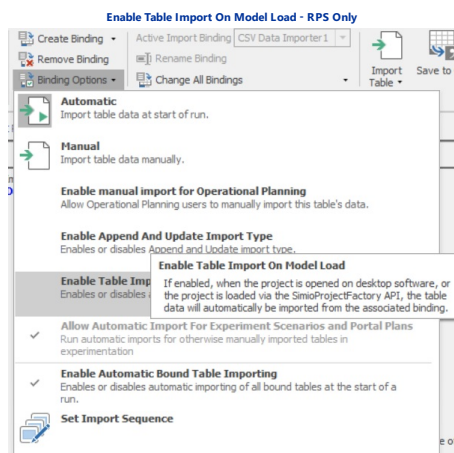
If a table has an associated data connector, the information within the Binding Options list is made available. *Automatic* will import the table data at the start of each simulation run, while *Manual* will allow for importing the table data manually. The *Automatic* option has the benefit of allowing the user to change the contents of the table without having to do an import each time the data has been changed.

Within the Binding Options pull down, users may choose to append to binded table data. The *Enable Append and Update Import Type* option, if enabled, will append imported data to the existing data table. If a record already exists then it will be updated. This updating is based on the primary key column (if any) in the table. This option allows the table's data to be manually edited even if the binding is set to 'Automatic'. If this feature is disabled (which is the default setting), the data in

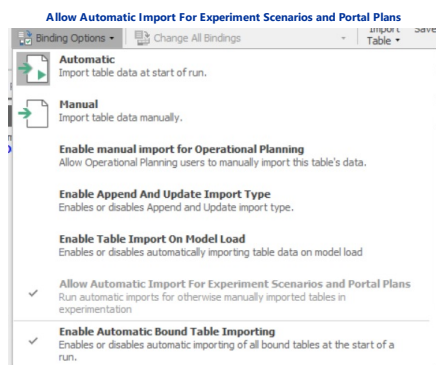
the data table will be replaced by the imported data.



Also within the Binding Options, there is an option to enable table importing when the simulation model is loaded. This is available in Simio RPS.



Allow Automatic Import For Experiment Scenarios and Portal Plans allows you to run a Plan on Portal and an Experiment on Desktop and Portal and have the tables imported although the binding is set to manual.



There is an option to enable or disable automatic bound table importing at the start of a simulation run. Temporarily disabling the bound table importing may be useful during model development and testing if the user wishes to prevent input data from changing or if the data sources are not currently accessible. NOTE: This red message will appear only when the *Automatic* binding option is active.

Note: You can manually edit bound tables by either setting the Binding Option to 'Manual' or by setting the Binding Option to 'Automatic' and disabling 'Enable Automatic Bound Table Importing'.

Enable or Disable Automatic Bound Table Importing

Finally, there is an option to define the sequence that table imports happen. The Set Import Sequence window allows you to specify the order that table imports occur in when there are multiple data tables with bindings in the model. This is particularly useful for when certain data tables are dependent on the importation of other data tables.

To change the order, use the up and down arrows. The data table at the top of the Items list will import first.

Note: If a table fails to import, the sequence of imports will stop.

Importing and Exporting Data into Tables

The user can **export** the table to a comma separated value (CSV) file by clicking on the Export icon in the ribbon menu. This is particularly useful to create a table in the proper format if you wish to add your table data externally (e.g. in Excel) and later import it into Simio. The user can also **import** the data from a database file, comma separated value (csv) file or from a Microsoft Excel file into the table by clicking on the Import icon in the ribbon menu. Users can use the Import option to import the current table's data into the table or may select Import All to import each bound table's data from its data source. As was mentioned above, the Import option is only available once a file is bound to the table. It is recommended that the initial table be generated by first exporting an existing (possibly blank) table from Simio.

When performing an import from Excel, it will always assume the first row selected is a column name. Also, the import logic always tries to match up already existing column names in the table with column names from the data, and if it cannot match a column from the data up with a column in the table (and the table actually has columns already in it), it throws the data away. If you run into this problem, delete all the columns in the table and do the import again with a blank Simio table.

Data can be copied out of Excel and pasted into a Simio table. Copy the data from Excel, click on the appropriate row and column of the Simio table and click Ctrl-V to paste the data into the Simio table.

Within the data tables, when a table is bound to a file, a message will be shown displaying the time of the last import (X days, Y hours and Z minutes ago). If the data table is bound to a file, but data not yet imported, that will be noted as well.

Importing Data - Time of Import

Resources	Routing Destinations	Materials	Material Lots	Manufacturing Orders	Routings	Bill Of Materials	Work In Process	Manufacturing Orders Output
Bound to CSV: C:\Users\Public\Documents\Simio\Examples\DiscretePartProductionFiles\Resources.csv, Use headers = True, Separator = ','								
Last import was 0 days, 0 hours, and 29 minutes ago								
1	Cut	Cut	-8	5	SchedTransferNode	Standard/Week		0
2	Cut1	Cut1	-4	2	SchedServer	Standard/Week		20

Resources	Routing Destinations	Materials	Material Lots	Manufacturing Orders	Routings	Bill Of Materials	Work In Process	Manufacturing Orders Output
Bound to CSV: C:\Users\Public\Documents\Simio\Examples\DiscretePartProductionFiles\RoutingDestinations.csv, Use headers = True, Separator = ','								
Data has not been imported or is of an unknown age								
	Resource Name	Node						

See the [functions](#) that can be used with Tables.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Data and Sequence Tables. Please refer to the [User Interface](#) for more details.

Clearing Data from Table(s)

You can choose to quickly clear all of the data out of tables using the Clear dropdown on the Data Content ribbon. Clear Current Table will clear all rows of data in the table you currently have selected. Clear All Tables will clear all rows of data from all data tables. Clear All Tables With Active Binding will clear all rows of data from the data tables with the same binding. Clear All Tables in Category will clear all rows of data from data tables that have the same *Category* property.

Behavior of a Bound Table in a Child Model

If a model that contains a table that is bound to an external source and that model becomes a child model (i.e. it is used as an object inside another model), then the behavior of binding works different depending on whether or not that child model is set as a Runnable model. A model is a runnable model if it's *Runnable* property is set to 'True'. This property can be found, along with other model properties, by right clicking on the name of the model in the Navigation window and selecting Properties.

If the *Runnable* property is set to 'False' then if this model is placed inside another model, the top level model will automatically recognize changes made in the source file (.csv, excel, etc) if the binding option is set to Automatic.

If the *Runnable* property is set to 'True' then if this model is placed inside another model, the top level model will not recognize changes made in the source file. The user must open the child model and either run the model to trigger an automatic import or perform a manual import so that the new data from the source file is now loaded into the data tables.

Note: The above statements apply to the scenario where the child model definition is in the same project as the parent model. If the child model definition is located in a separate project, the top level model will never automatically recognize changes in a data source that is bound to a child model. The user must open the child model, re-import the data and save the child model. When the parent model is then opened, it should prompt the user that changes have been made to the child definition and the user has the option to accept these changes.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Lookup Tables

Lookup Tables

By clicking on the Data tab and selecting the Lookup Tables panel, users may add a table to model situations where a value (e.g. processing time) is dependent on some other value (e.g. number of completed cycles). Lookup tables return a function value based on a lookup value.

The value of a lookup can be specified in an expression using the format **Name[X_Expression]**, where Name is the name of the lookup table, and X_Expression (specified as any valid expression) is the independent index value into the lookup. For example `CycleTime(NumberBusy)` returns the value from the lookup table named CycleTime based on the current value of the NumberBusy state. The lookup value is computed using linear interpolation between the defined values. If the index is out of the defined range then the closest endpoint (first/last) point in the range is returned.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Lookup Tables. Please refer to the [User Interface](#) for more details.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Rate Tables

Rate Tables

The Rate Tables panel in the Data window allows the user to model situations where an arrival/event rate varies discretely over time. The number of fixed rate periods (*Number of Intervals* property) and the length of each rate period (*Interval Size* property) are specified in the Rate table. The rate pattern automatically repeats during the running of the simulation. Please note that the *Starting Offset* is the offset from when the model starts or table repeats. It is not the time of day.

A Rate table is used by the Source object / Timer element to generate entities / events with a time-varying rate. Internally, a non-stationary Exponential distribution is used to calculate the rates. The rate units for the Poisson arrival process is Arrivals per Hour, regardless of the time units specified for the intervals of the rate table.

In order to use a Rate Table with the standard Source object, set the *Arrival Mode* property in the Source to 'Time Varying Arrival Rate' and then select the appropriate Rate Table for the *Rate Table* property. The Source also includes a *Rate Scale Factor* property that can be used to easily modify the values within the table by a given factor instead of changing the values separately. For example, to increase the Rate Table values by 50%, simply specify the *Rate Scale Factor* within the Source to '1.5'.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Rate Tables. Please refer to the [User Interface](#) for more details.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Work Schedules

Work Schedules

Schedules are used to model situations where the capacity of an object varies over time. An object (e.g., Server or Worker) can have an associated schedule that automatically changes its capacity over time. Schedules can be defined as either Pattern Based or Table Based Work Schedules. Table Based Work Schedules can be used for easily reading in schedule data from an external data file.

The work schedule assigned to a particular resource can be changed during the simulation run using the [SetWorkSchedule](#) step. The function `CurrentWorkSchedule.Name` can be used with a resource to return the currently assigned work schedule.

Pattern Based Work Schedules

A Pattern Based Work Schedule is comprised of a repeating base pattern called a Work Schedule with superimposed exceptions (e.g. holidays or planned maintenance). Pattern Based Work Schedules are defined within the Data tab, Schedules panel of a model by selecting the Pattern Based tab. Selecting either the Pattern Based Work Schedule or Day Pattern button on the Schedule ribbon will open the Pattern Based tab.

A Pattern Based Work Schedule is defined by specifying the number of repeating days within the schedule and the associated Day Pattern for each day. Multiple work schedules can be defined within the Work Schedules section.

Pattern Based Work Schedule Window

File

Project Home

Schedule

Support

21

Pattern Based Work Schedule

Day Pattern

2

Table Based Work Schedule

X

Remove

Create

Edit

Facility

Processes

Definitions

2194.32Data

Results

Planning

Views

Pattern Based

Table Based

Tables

Lookup Tables

Rate Tables

Schedules

Changeovers

Input Parameters

Work Schedules

Name	Start Date	Description	Days	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
StandardWeek	1/3/2011	Standard Work Week Schedule	7	StandardDay	StandardDay	StandardDay	StandardDay	StandardDay		
*										

Day Patterns

Name	Description
StandardDay	Standard 8-5 Work Day
*	

A Day Pattern consists of a single or multiple Work Periods that define on shift and off shift periods with a starting time, duration and ending time. Any periods that are not defined are considered to be off shift, or have a capacity *Value* of '0'. A cost multiplier may be defined for the given work period as well. An on shift Work Period entry has a value that defines the capacity of the object that is following this schedule. Multiple Day Patterns can be defined within the Day Patterns tab.

Day Patterns / Work Periods window

Day Patterns

Name	Description
StandardDay	Standard 8-5 work day

Work Periods

Start Time	Duration	End Time	Value	Cost Multiplier	Description
8:00 AM	4 hours	12:00 PM	1	1	
1:00 PM	4 hours	5:00 PM	1	1	
*					

Multiple Day Patterns can be defined. Click on the '+' to expand the Day Pattern into Work Periods

Each Day Pattern then has one or more Work Periods that specify the starting / ending times and capacity 'Value'

Exceptions override the repeating work schedule for the duration of the exception. Exceptions can be defined within the Work Day Exceptions tab and within the Work Period Exceptions tab, both from the Work Schedule tab. A Work Day exception may be used for holidays or single days and is defined by a single calendar day and an alternative Day Pattern for that calendar day. A Work Period exception may be used for overtime, planned maintenance, vacation periods, etc., and is defined by starting day/time and ending day/time. Within the Work Period exceptions, the capacity *Value* can be specified.

Work Day Exceptions and Work Period Exceptions

Pattern Based

Table Based

Work Schedules

Name	Start Date	Description	Days	Monday	Tuesday
StandardWeek	1/3/2011	Typical Work Week	7	StandardDay	StandardDay

Work Day Exceptions

Work Period Exceptions

Date	Day Pattern Name	Description
2/25/2011	HalfDay	Half-Day off
4/8/2011	HalfDay	Half-Day off
*		

Work Day Exceptions allow users to specify an alternative Day Pattern for a particular day(s).

Pattern Based

Table Based

Work Schedules

Name	Start Date	Description	Days	Monday	Tuesday	Wednesday	Thursday
StandardWeek	1/3/2011	Typical Work Week	7	StandardDay	StandardDay	StandardDay	StandardDay

Work Day Exceptions

Work Period Exceptions

Start	End	Value	Cost Multiplier	Description
7/11/2011 12:00:00 AM	7/15/2011 12:00:00 AM	0	1	Plant Shutdown
8/15/2011 12:00:00 AM	8/16/2011 12:00:00 AM	2	1.5	Overtime
@				

Work Period Exceptions allow users to specify a period of time in which the capacity Value overrides the regular Work Schedule.

Creating a Work Schedule

- Click on the Data window tab and click on the Schedules panel.
- By default, a work schedule named 'Standard Week' is already available for use or modification. To add a new schedule, you can either press the Pattern Based Work Schedule button on the Schedule ribbon or simply enter a new work schedule *Name*.
- The starting date of the schedule should be entered in the Start Date property in date format.
- Specify how many days are in the schedule by inserting a number between 1 and 28 in the *Days* property. In other words, how many days until this schedule repeats itself. If any schedule has more days than another schedule, the extra days for the particular schedule will be grayed and not editable. If the number of Days is '7', days will be shown as days of the week, based on the Start Date specified.
- Select a Day Pattern name for each day specified.

Important Note regarding Start Date: The Start Date is the date on which a repeating cycle starts. That repeating cycle then repeats infinitely in both directions (e.g., backwards in time as well). To override that behavior and prevent immediate availability in a schedule with a future start date, you would need to add an exception to prevent that interim availability.

Creating a Day Pattern with Work Periods

- Day Patterns are defined within the Day Patterns section of the Schedules window for Pattern Based Work Schedules.

A default Day Pattern named 'StandardDay' has been provided. To add a new day pattern, you can either press the Day Pattern button on the Schedule ribbon or simply enter a new day pattern *Name*.

- To enter values into the Day Pattern, press the '+' next to the name and the Work Periods will open.
- With the Work Periods, enter the Start Time. You may then enter either the Duration (and the End Time will be determined) or the End Time (and the duration will be calculated).
- The Value field is used to determine the capacity of the object using the schedule. See notes below on limitations for Worker schedules.
- The Cost Multiplier property is applied to the idle and usage cost rates of a resource following the work schedule. This may be useful for specifying overtime costing.
- Multiple Work Period rows may be added for various capacities throughout the day. If there is a time period where no information is specified, Simio assumes the capacity to be '0'.

Important Note regarding Schedules and Worker object: The Value field for a worker should either be '1' (indicating On Shift) or '0' (indicating Off Shift), as the capacity of the Worker object is defined within the object itself using the *Initial Number In System* property. The schedule for the Worker should be used only to define On Shift or Off Shift times, not capacity values.

Adding a Work Day or Work Period Exception to the Schedule

- Within the Work Schedules tab, select the Work Schedule that will have Exceptions
- Click on the '+' next to the Work Schedule.
- For a Work Day exception, select the Date for the exception and specify a Day Pattern that should be used for this date. When this date is encountered, Simio will use the exception Day Pattern instead of the Day Pattern specified in the main Work Schedule.
- For a Work Period exception, enter the Start column and select the date and time of the exception. Within the End column, enter the date and time that the exception will end. For that same exception, enter the capacity Value that will be used instead of the value specified in the associated Work Schedule's Day Pattern. A cost multiplier may also be specified with the work period exception.

An experiment can be run that uses a Schedule as an input into the model. This allows for the user to experiment with how different work schedules can affect their key performance metrics. To experiment with a schedule, create a Schedule type property on the model, from the Definitions window. From within the Facility window, go to the object that will be following the work schedules and set the *Work Schedule* property of that object instance to reference the new Schedule property that you created above. From the Experiment design window, you can input the name of a Work Schedule into different scenarios on an Experiment.

Additionally, the *Value* and *Cost Multiplier* properties of a schedule and/or exception can be expressions. This includes the ability to utilize a property value (for experimentation) within a *Value* or *Cost Multiplier* field. For example, the *Value* property may be set to a property named 'Value1' for a given exception, where the 'Value1' property can be used within the experiment to test various capacities for a given Resource.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Pattern Based Work Schedules. Please refer to the [User Interface](#) for more details.

Table Based Work Schedules

A Table Based Work Schedule is defined within a table (or group of relational tables) with properties for the schedule referencing a *Table Name*, *Start Date Time*, *End Date Time*, *Value* and *Cost Multiplier*. Table Based Work Schedules are defined within the Data tab, Schedules panel of a model by selecting the Table Based tab. Selecting the Table Based Work Schedule button on the Schedule ribbon will also open the Table Based tab.

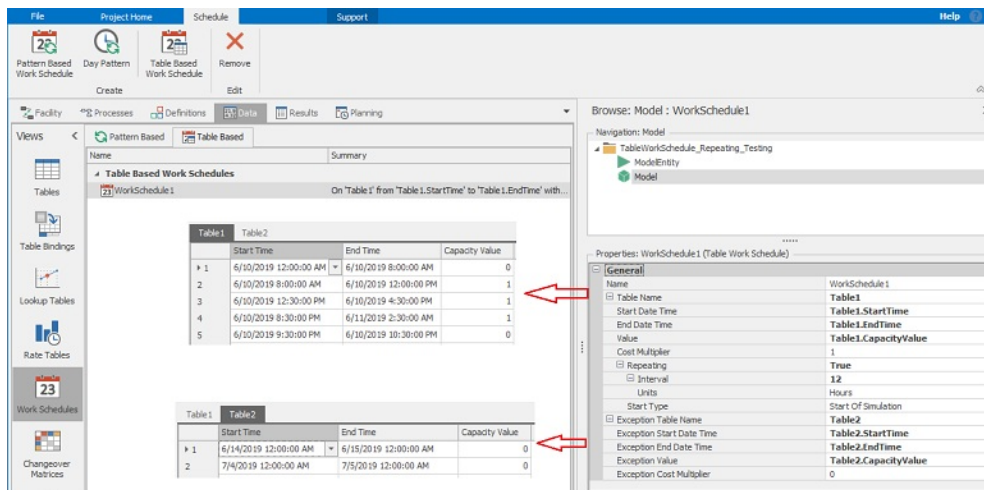
Table Based Work Schedules may repeat, by indicating 'True' in the *Repeating* property for the table. The schedule may repeat given the specified interval (in minutes, hours, days, etc.). Users may also specify a *Start Type*, specified as either the 'Start Of Simulation', 'Offset From Midnight Sunday' (with *Start Offset Hours*) or 'Specific Date Time'. If the *Start Type* is 'Specific Date Time', then the user can specify the *Schedule Start Date Time* as well as the *Offset Calculation Type* which determines how the *Schedule Start Date Time* relates to the *Start Date Time* and *End Date Time*.

When using the 'Start Of Simulation', Simio will use the *Start Date Time* and *End Date Time* as offsets relative to the simulation start time into each interval rather than absolute date times. It's important to note that if the table schedule *Start Date Time* is AFTER the simulation start date, Simio will assume 0 capacity (no schedule specified in table) for the first X amount of time from simulation start date through schedule start date, then repeating after *Interval* time. As an example, if the *Start Type* is 'Start Of Simulation', let's say the simulation start date is 6/09/19 and the schedule starts 6/10/19 and contains a single day of schedule/capacity times. If the repeatable interval is 1 day, it will include 6/09/19 of 0 capacity and then repeat, thus the 6/10/19 schedule will not be realized. If the repeatable interval is 2 days, it will include 6/09/19 of 0 capacity, then 6/10/19 schedule from table and then repeat.

A Table Based Work Schedule may also have an associated Exception Table. If an *Exception Table Name* is specified, properties for the associated start time, end time, value and cost multiplier for that exception table become available. Users may wish to think of an Exception table as a 'Global' table, as it can be referenced from multiple Table Based Work Schedules and may be used to specify holidays or vacation days. It's important to note that Exceptions table data will take precedence over the Table Based Work Schedule.

Work Day Exceptions and Work Period Exceptions can still be defined within the object itself when a Table Based Work Schedule is used. These Exceptions specified on the object itself will take precedence over both the Table Based Work Schedule and the Exceptions Table data.

Table Based Work Schedule Window



IMPORTANT NOTE: Any expressions specified in the *Value* or *Cost Multiplier* properties for Table Based Work Schedules are evaluated at the start of the simulation run only and are NOT dynamically evaluated as the simulation progresses. This also means that the functions listed above for the Pattern Based Work Schedules cannot be used for Table Based Work Schedules.

Listed below are the Properties of **Table Based Work Schedules**:

Property	Description
Name	The name of the work schedule.
Table Name	The name of the table used for this work schedule.
Start Date Time	An expression that resolves to the start of a work period for a particular row in the referenced table.
End Date Time	An expression that resolves to the end of a work period for a particular row in the referenced table.
Value	An expression that resolves to the value of a work period for a particular row in the referenced table.
Cost Multiplier	An expression that resolves to the cost multiplier of a work period for a particular row in the referenced table.
Repeating	If true, the schedule repeats given the specified interval, using the Start Date Time and End Date Time as offsets relative to simulation start time into each interval, rather than absolute date times.
Interval	Integer value that specifies how long each repeated iteration of the schedule should last.
Start Type	Indicates where the repeating schedule will start from (or be "anchored" to). By default, the schedule will start from the simulation start date. It can, however, instead be started from some offset from midnight on the Sunday before the start of the simulation or started from a specific date and time.
Start Offset Hours	The offset (in hours) from midnight on the first Sunday before the start of the simulation to start the repeating schedule.
Schedule Start Date Time	The exact date and time from which to start the repeating schedule.
Offset Calculation Type	<p>Determines how Start Date Time and End Date Time are interpreted with relation to the Schedule Start Date Time.</p> <p>Relative interprets the table values mapped to the Start Date Time and End Date Time expressions as relative time offsets from the Schedule Start Date Time.</p> <p>Absolute interprets the table values mapped to the Start Date Time and End Date Time expressions as absolute date times, which are then used to determine the actual offsets from the Schedule Start Date Time.</p> <p>Default uses the Absolute offset calculation type if the Start Date Time and End Date Time expressions are DateTime table column references. Otherwise, it uses the Relative offset calculation type.</p>

Exception Table Name	The name of the table used for exceptions to this work schedule.
Exception Start Date Time	An expression that resolves to the start of an exception period for a particular row in the referenced table.
Exception End Date Time	An expression that resolves to the end of an exception period for a particular row in the referenced table.
Exception Value	An expression that resolves to the value of an exception period for a particular row in the referenced table.
Exception Cost Multiplier	An expression that resolves to the cost multiplier of an exception period for a particular row in the referenced table.

Work Schedule Functions

Functions Available for Work Schedules

Listed below are the Functions of **Schedules**:

Function	Description
Value(dateTime)	Returns the schedule's value for the specified numeric datetime (simulation time).
NextValue(dateTime)	Returns the schedule's next value if at the specified numeric datetime (simulation time).
TimeOfNextValue(dateTime)	Returns the simulation time (in hours) of the schedule's next value change if at the specified numeric datetime (simulation time).
TimeUntilNextValue(dateTime)	Returns the time duration remaining (in hours) until the schedule's next value change if at the specified numeric datetime (simulation time).
AverageValue(fromDateTime,toDateTime)	Returns the average value over a specified time range.
MinimumValue(fromDateTime,toDateTime)	Returns the minimum value over a specified time range.
MaximumValue(fromDateTime,toDateTime)	Returns the maximum value over a specified time range.

These are available from any schedule reference, including ScheduleName.*, SchedulePropertyName.Schedule.* (e.g., ResourceObjectName.WorkSchedule.Schedule.* if the resource is following a work schedule), or TableName.SchedulePropertyName.Schedule.*. If the work schedule for a resource is changed during the simulation run using the [SetWorkSchedule](#) step, the CurrentWorkSchedule.* can also be used with the above functions.

Work Schedule Comparisons

Note: When comparing schedule references for equality, the equality will always be made with regards to the base schedule. For example, a comparison of 'Resource1.CurrentWorkSchedule == Schedule1', will use the base schedule returned from the CurrentWorkSchedule function and will ignore any Work Schedule Exceptions defined on Resource1.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Changeover Matrices

Changeovers

Changeover matrices are used to model situations where a setup time is sequence dependent. The setup time is the matrix value indexed in row/column by a list value such as color, size, part family, etc. The row is selected based on this value for the previous entity, and the column is selected based on this value for the current entity. The [ChangeoverLogic](#) element is used in conjunction with this matrix.

When using Changeovers, a string list must be defined within both the model, as well as the model entity. Lists are specified within the Definitions tab, Lists panel. See the help section on [Lists](#) for more information on String lists.

Examples of using a changeover matrix are found in the SimBits [ServersUsingTaskSequenceWithSequenceDependentSetups](#) and [ServersUsingTaskSequenceWithDataTables_SequenceDependentSetups](#).

Conceptual Example of Changeover Matrix

From/To	Small	Medium	Large
Small	0	11.4	14.5
Medium	2	0	16.3
Large	45	27.3	0

The changeover matrix data is also accessible in an expression. A changeover matrix value may now be referenced in any expression using the syntax **MatrixName[*row,column*]**, where the row and column subscripts are numeric constants or expressions that return zero-based row and column indexes into the matrix. When a changeover matrix has been added to a model, the available changeover matrix syntax can be viewed within the expression builder.

The name of a changeover matrix can also be accessed within an object property (repeating or non-repeating). The object's process logic is then able to get to the matrix data through the property reference. This is done by adding a Changeover Matrix type property to the model by going to Definitions / Properties / Standard Property and adding a property. Then, in the process logic of the object that has the property, the syntax for **ChangeOverMatrixPropertyName.Matrix[*row,column*]** or **RepeatGroupPropertyName.ChangeOverMatrixPropertyName.Matrix[*row,column*]** may be used to access a value at a specified row and column index location in the changeover matrix that is being referenced. **ChangeoverMatrixPropertyName.Matrix.Name** can be used to get the string name of the changeover matrix that has been specified using the property.

NOTE: Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste) is supported for Changeovers. Please refer to the [User Interface](#) for more details.

Input Parameters

Input Parameters

Input Parameters are generally used to define named parameters which can then be referenced anywhere within a model. For example, the user might define Input Parameters named *InterarrivalTime* and *ProcessingTime*, and then reference these names on the Source and Server objects in the model. There are several advantages in using Input Parameters in place of directly entering these expressions in the model.

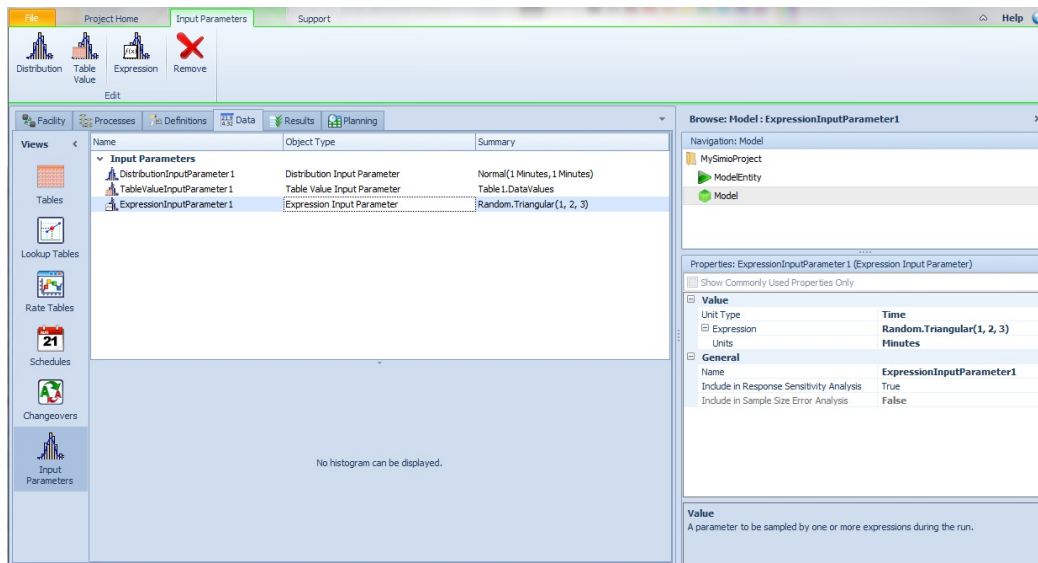
- Input Parameters can be shared across multiple objects; e.g. if five servers all share the same Process Time, they can all reference the same Input Parameter.
- It is more convenient to enter distributions using Input Parameters since the distribution parameters are individually defined as properties of the Input Parameter and a histogram is based on 10,000 samples is provided to show the shape of the distribution for the parameters.
- Input Parameters enable both Response Sensitivity and Sample Size Error analysis.

Input Parameters are defined by selecting the Data window tab, and then selecting Input Parameters on the left panel. There are three different types of Input Parameters – Distributions, Table Values, and Expressions.

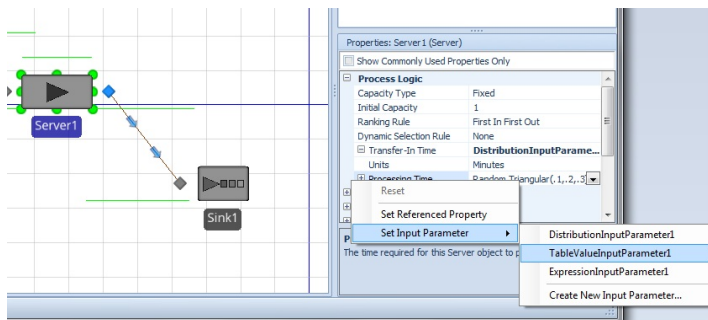
A **Distribution** is used to specify a random distribution. The parameters for the distribution are specified in the property grid, and the histogram based on 10,000 samples of that distribution is displayed in the bottom panel of the window. In addition to the distribution parameters, the user may specify if this Input Parameter is to be active in the Response Sensitivity analysis or included in the Sample Size Error analysis. If it is included in the Sample Size Error analysis, the number of real-world observations that were used in fitting this distribution are specified.

The second type of Input Parameter is a **Table Value** and is used whenever the actual observational data is randomly sampled in place of fitting the data to a distribution and then sampling from that distribution. This is typically the preferred approach when actual data is available, given there is confidence that the data is a representative sample and/or no additional information about the process is available. In this case the table column name that holds the real-world observations is specified. Additionally, there is an option to *Ignore Zero Values* within the data table. If this value is 'True' and a '0' is sampled from the data table, a new data value from the table will be selected. *Starting Index* and *Ending Index* may also be used to indicate a starting row and/or ending row to use when taking the sample. If not specified, these default to the starting and ending rows in the table.

The third type of Input Parameter is an **Expression**. In this case, any mathematical expression, including random distributions and model properties can be specified. Although this is the most general form of the Input Parameter, one limitation is that it cannot be used in the Sample Size Error analysis.



The input parameter name is then referenced within an object's property field in a model, similar to the way that states and property values are used. Right-clicking on a object's property value in the Facility window also allows the user to create a new input parameter or select from the already existing input parameters. Alternatively, the input parameter name can be used by itself or within an expression in any object's property field that accepts an expression.



The input parameters are unique, in that they are also used within the Experiments defined for a model. The Input Analysis tab within an Experiment will display all input parameters and associated scenario results. See the [Response Sensitivity](#) and [Sample Size Error Analysis](#) topic for additional information.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

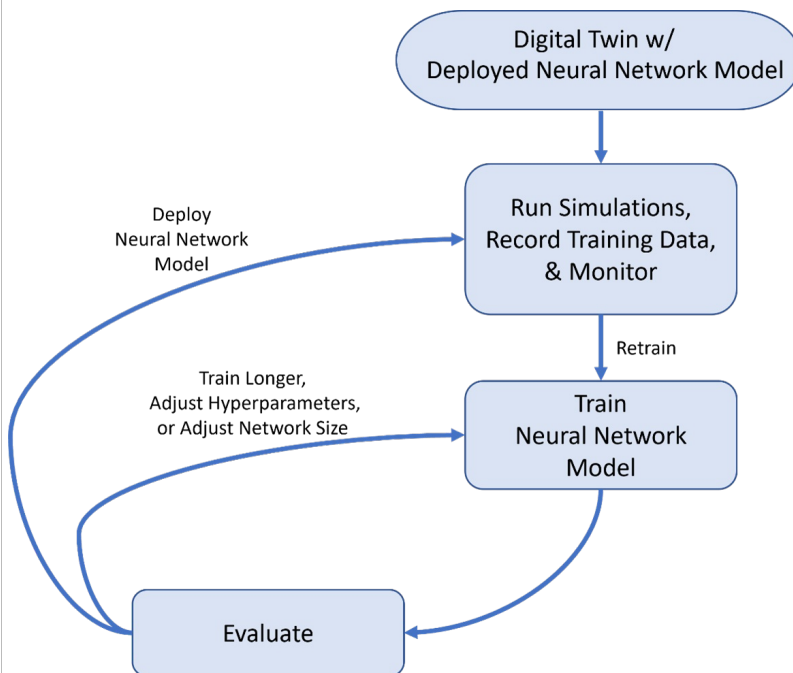
Neural Network Models

The Neural Network Models view within the Data ribbon allows you to use Simio's machine learning features. Neural networks are a subset of machine algorithms commonly used in a variety of applications including regression models for predicting values, analyzing text/speech, recognizing/classifying images, anomaly detection, and pattern recognition. The Simio implementation of neural networks focuses on the first application, regression models for predicting values. These models are often feedforward, where data moves from the input neurons of the network to the output neurons and does not loop back to the input neurons. Currently, Simio supports training feedforward neural networks with one or more inputs and one output. We add a neural network to the model to make the decisions, and then we train the neural network using simulation data so that the model gets better and better at making the correct decision over time.

Note that our neural network implementation was designed and implemented to improve the machine learning workflow when used jointly with discrete event simulation. Therefore, some terminology might be used differently to complement the strengths of discrete event simulation. For instance, the term "untrained model" generally indicates a model whose trainable parameters have been initialized but that has not been trained. However, in Simio, an untrained model has a set number of inputs and one output, but all other trainable parameters such as the weights and biases have not yet been initialized. This is because a neural network which has been initialized but is untrained will not produce outputs which are generally useful in a simulation model. Instead, Simio uses the Untrained Predicted Value Expression property of the Neural Network Element to generate usable outputs from the Neural Network Element before when a Neural Network Model is not trained.

Additional Use Cases

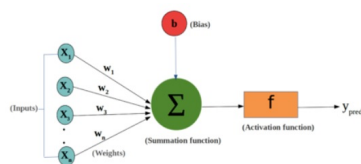
In addition to training Neural Network Models, Simio also includes robust data collection features which can be used to generate synthetic training data for training machine learning models. This data can be used to train neural network models within Simio or exported to a third-party program to train other types of machine learning models. These data collection support a data-driven machine learning workflow, where data is generated and used to train a model, then a model is evaluated and tuned before implemented in a production environment. This addresses one of the largest challenges in the application of machine learning: acquiring sufficient high-quality training data. Simio's machine learning data collection features address this by equipping high-fidelity simulation models to generate an unlimited quantity of synthetic training data.



For more information and a Neural Network model example, visit the [Neural Network Models - Discussion and Examples](#) page.

Technical Information

In simple terms, a neural network maps a set of numeric inputs into a set of one or more outputs. In its most basic form, we have a single node called a Perceptron (also called a neuron), with a set of inputs, X 's, a weight w assigned to each input, a bias assigned to the node, an activation function, $f()$, following the node, and a single output, Y , that is computed from the inputs, weights, bias, and activation function. The output is a predicted value that the Perceptron is making based on the current set of inputs. *Weights and Bias* are the learnable parameters of a neural network. When inputs are passed into a node, they are multiplied by their respective weights and then passed along with the bias into a summation and then activation function to compute the node's predicted output. The following depicts a Perceptron.



Simio Concept: Neural Network Modes and Neural Network Elements

Simio's neural network features are split into two levels: Neural Network Elements and Neural Network Models. A Neural Network Model represents a trained, neural network regression model for inference. It can be a feedforward model that may be created and trained directly in Simio (no code or 3rd party framework required) or an imported, arbitrary ONNX model. If simulation runs are generating synthetic training data, the Neural Network Model definition serves as the training data repository. Recorded training data may be exported to CSV. Neural Network Models can be accessed in the Neural Network Models view of the Data tab.

A Neural Network Element is used to integrate a Neural Network Model definition into simulation logic. This element defines simulation expression mappings for the neural network model's inputs and "actual" output, defines simulation event-based triggers for training data collection, and provides a *PredictedValue* function that may be used in any simulation expression for inference purposes.

A Neural Network Element references a Neural Network Model with its *Neural Network Model Name* property. A Neural Network Model can be referenced by any number of Neural Network Elements, allowing a single Neural Network Model to be applied to multiple use-case in a single simulation model.

Neural Network Training: Data Splitting

A Neural Network Model is trained using its recorded training data. Machine learning practitioners recommend splitting available training data into three distinct datasets: a training dataset, a validation dataset, and a test dataset. The training dataset is the primary dataset used to train the model. The records from the training dataset are provided to the model directly and used to refine the model's performance. The validation dataset is used to provide an unbiased estimate of the model's performance, and to automatically halt training when performance stops improving. This dataset drives the "tuning" of model hyperparameters. The test dataset is used to provide a final unbiased estimate of the model's performance. The test dataset is important because the process of model "tuning" to perform better on the validation dataset can cause overfitting. Simio will split the training data automatically using parameters defined in the neural network training dialog.

The Open Neural Network Exchange (ONNX) Format and ONNX Runtime

The Open Neural Network Exchange or ONNX format is an open source, standardized file format for machine learning models that enables models to be transferred between AI frameworks. This format uses a protocol buffer (protobuf)-based data format. Protobuf is a compact data serialization format developed by Google that involves specifying a human-readable schema in the form of a proto file. The proto file defines how the protocol buffer data is structured and therefore how it should be parsed. A machine learning model saved in the ONNX format can be loaded and used by the ONNX

runtime engine, a cross-platform, performance focused inference engine for ONNX developed by Microsoft, which is widely supported by many AI tools and frameworks. There are ONNX runtime packages for popular programming languages including C#/C++, Java, and Python, allowing models to be transferred between projects in different languages.

Using the Export Model to ONNX button, you can export the selected neural network model to an ONNX file which can be saved and then imported into a third party software that supports the ONNX format.

Other Terminology: Weights and Biases, Supervised Learning, Regression

A neuron, sometimes referred to as a node, is a sub-element of a neural network model. A neural network may contain an arbitrary number of neurons spread among an arbitrary number of layers. In Simio's neural network trainer, the number of neurons in each layer is a hyperparameter which can be manipulated to tune the neural network model to improve its accuracy.

Weights and biases are the learnable parameters of a neural network. When inputs are passed into a neuron, they are multiplied by their respective weights and then passed along with the bias into a summation and then activation function to compute the neuron's predicted output.

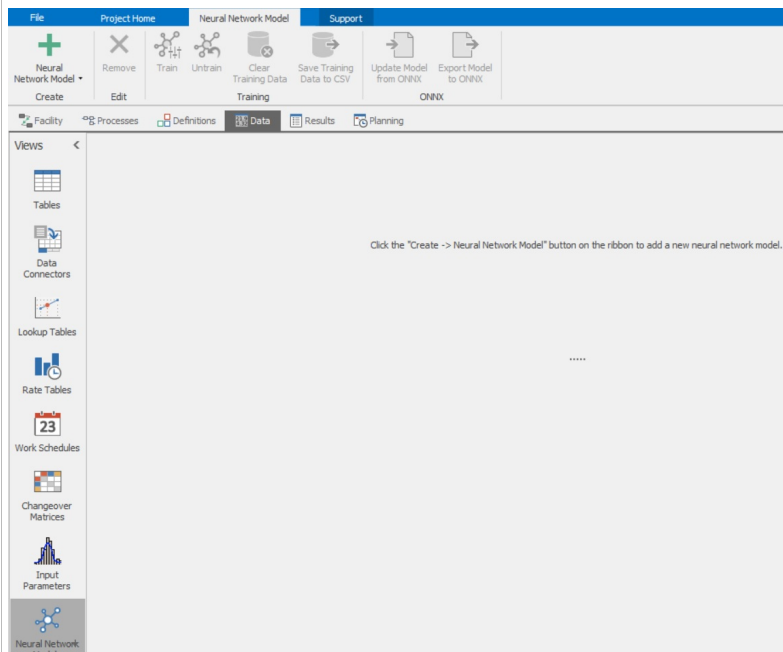
Regression is a technique that uses a function approximation to predict a numerical output given a set of numerical inputs. Although neural networks are widely known for use in modeling complex problems such as image recognition, they are also suitable for regression problems.

Neural Network Weight and Bias Initialization

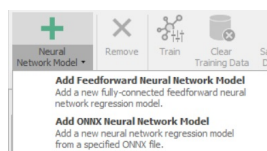
The weights in neural network models created by Simio are initialized according to the Xavier initialization method defined in the paper "Understanding the difficulty of training deep feedforward neural networks" by Xavier Glorot and Yoshua Bengio. With this approach, weights are initialized according to a uniform distribution between the range $[-1/\sqrt{n}, 1/\sqrt{n}]$ and $1/\sqrt{n}$, where n is the number of inputs to the neuron. This method randomly initializes the weights in the neural network such that the mean and the variance of the activations is approximately zero, avoiding vanishing or exploding gradients. Biases are initialized to equal zero.

Adding a Neural Network

The Neural Network Models view is located on the Data tab.



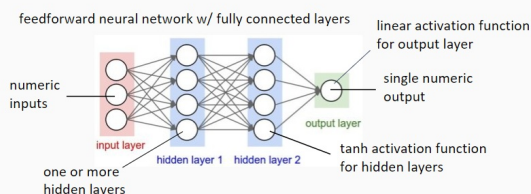
Clicking the Neural Network Model dropdown will allow you to create either a Feedforward Neural Network Model or an ONNX Neural Network Model.



A *Feedforward Neural Network* is the first and simplest form of neural networks. It is an artificial neural network in which the data moves in only one direction from the input layer to the output layer with no loopbacks.

An *ONNX Neural Network* is an open, standardized file format for representing machine learning models that was introduced by Microsoft and Facebook in 2017. It is now widely supported and can be found in many AI tools and frameworks (e.g., MATLAB, PyTorch, TensorFlow, and MyCaffe). The ONNX runtime engine is a Microsoft developed performance focused inference engine that is used in high-scale services such as Bing and Office.

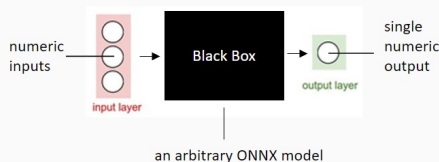
Feedforward Neural Network Model



Created directly in Simio.

May be trained using Simio's built-in Neural Network Trainer.

ONNX Neural Network Model



Or

An imported ONNX model with numeric inputs and a single numeric output.

May only be trained using a 3rd party deep learning framework, then deployed to Simio for inference.

The Feedforward Neural Network Model can have any number of hidden layers and nodes and is created directly in Simio or may be an imported ONNX model if the model is a standard feedforward network design with tanh activation functions for the hidden layers, and a linear activation function for a single output layer. These standard networks can be trained using Simio's built-in Neural Network Trainer. An ONNX Neural Network Model can be any ONNX model that has ONNX runtime support and numeric inputs and a single numeric output that is created by a 3rd party tool. These must be trained using a 3rd party deep learning framework, then deployed to Simio for inference. However, Simio may be used to provide the synthetic data to the 3rd party tool for training.

Neural Network Definitions are added/edited in the Neural Network Models view of the Data window. For each Neural Network Definition in this view a summary of the recorded training data is provided. This summary includes each training record comprised of the input values that were used to generate the predicted value, the predicted value, and the actual value. The view also provides a histogram of the prediction errors (actual value - predicted value) for each observation. In a well-trained model this error should be symmetric around 0 and narrow in width.

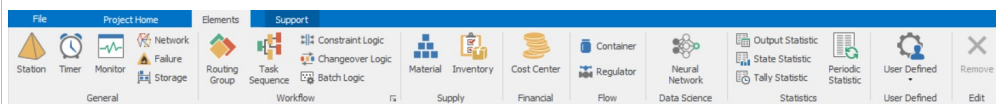
Neural Network Properties

Listed below are the properties of **FeedForward Neural Network Model**:

Property	Description
Number Input Nodes	The number of nodes in the neural network's input layer.
Force Negative Outputs To Zero	Indicates whether to force negative output values from the neural network model to zero.
Current Records Count	The current number of records in the neural network model's training data repository.
Automatic Data Recording	Indicates whether to automatically record training data for the neural network model if running a simulation in interactive or operational planning mode. For more information about defining simulation event-driven triggers to record synthetic training data, refer to the Neural Network element in Definitions, Elements, Data Science.
Maximum Records Limit	The maximum number of records that may be stored in the neural network model's training data repository if automatic data recording is enabled. The oldest records will be replaced by new ones when the specified maximum records limit is reached.
Name	The name of this instance.
Description	Description text for this instance.
Is Trained	Indicates whether the neural network model has a defined computational graph that returns a predicted output value given a set of input values.
Last Update Time (UTC)	The date and time of the last update to the neural network model if it is currently trained, expressed as the Coordinated Universal Time (UTC).

Neural Network Element

A Neural Network Element may be used to integrate a neural network regression model into simulation logic.

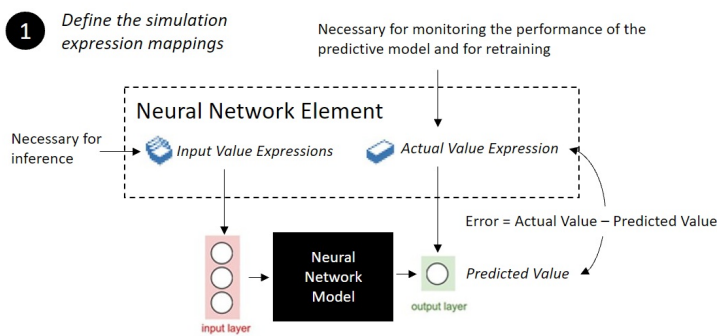


Listed below are the properties of **Neural Network Element**:

Property	Description
Neural Network Model Name	The name of the associated neural network model. For more information on creating and training a neural network regression model, refer to the Data, Neural Network Models view.
Input Value Expressions	The expressions used to get a set of input values for the associated neural network model. Note that the order of these expressions should always match the order of the input values provided during the training of the model.
Untrained Predicted Value Expression	The expression used as a substitute to return a predicted output value if the associated neural network model is not trained.
Save Inputs Triggers	Optional event-driven triggers that will save a set of input values for the associated neural network model. The Training Key Expression is used to uniquely identify the saved input data. Note that synthetic training data recording for a neural network model can be enabled or disabled globally by going to Data, Neural Network Models and setting the model's Training Data Repository, Data Recording Enabled property to True or False.
Save Actual Triggers	Optional event-driven triggers that will save an actual observed value for the associated neural network model. The Training Key Expression is used to pair the actual observed value with a previously saved set of input values so that a new record may be added to the neural network model's training data repository. Note that synthetic training data recording for a neural network model can be enabled or disabled globally by going to Data, Neural Network Models and setting the model's Training Data Repository, Data Recording Enabled property to True or False.
Actual Value Expression	The expression used to record an actual observed value for the associated neural network model when a Save Actual trigger occurs.
Training Key Expression	The expression used to return a key for pairing a saved set of input values with a saved actual observed value so that a new record may be added to the associated neural network model's training data repository. Can be any expression that evaluates to a string value. If left unspecified, defaults to the string identifier name of the object associated with the Save Inputs or Save Actual triggering event occurrence.
Name	The name of this instance.
Description	Description text for this instance.
Public	Indicates if this instance is publicly accessible outside of its containing object.
Report Statistic	Indicates whether statistics are to be automatically reported for this object or element.

The following picture illustrates the process of how you can integrate a Neural Network Regression Model into simulation logic with the Neural Network Element.

Integrating a Neural Network Regression Model Into Simulation Logic Using a Neural Network Element



TECH NOTE

If an ONNX Neural Network Model, then the vector of simulation input values is mapped to the input tensor of the ONNX model in row-major order. If the ONNX model has multiple input tensors, then those tensors are mapped in the order that they are defined.

3 Use the Neural Network element's Predicted Value function in a simulation expression

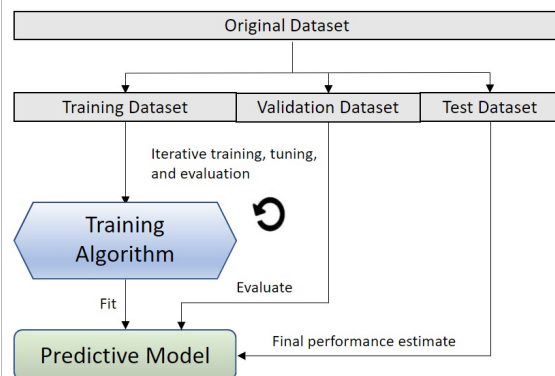
Expression **NeuralNetwork1.PredictedValue(Entity)**

Neural Network Trainer

In Simio, a trained feedforward neural network model is one that has a defined computational graph (configured using the Neural Network Trainer).

A neural network model is trained using its recorded training data. Machine learning practitioners recommend splitting available training data into three distinct datasets:

- **Training Dataset:** This dataset is used to fit the model.
- **Validation Dataset:** This dataset is used to provide an unbiased estimate of the model's performance, and to automatically halt training when generalization, or the model's ability to adapt properly to new, previously unseen data, stops improving. This dataset drives the "tuning" of model hyperparameters. A hyperparameter is a model parameter that can be explicitly specified by the practitioner, as opposed to parameters that are "learned" via training (such as the model's weights and bias values). Learning rate, momentum, and the network's hidden layer sizes are examples of hyperparameters.
- **Test Dataset:** This dataset is used to provide a final unbiased estimate of the model's performance. The test dataset is important because the process of model "tuning" to perform better on the validation dataset can cause a form of overfitting.



	Purpose	Yield	Used for Model training	Used for Parameter tuning
Train Data	To learn patterns from the data.	A model that makes near-expected predictions	Yes	Yes
Validation Data	To understand model behaviour and generalizability on unseen data.	Insights on how to tune your model.	No	Yes
Test Data	To understand how the model would perform in real world scenario.	A completely unbiased estimate of model performance.	No	No

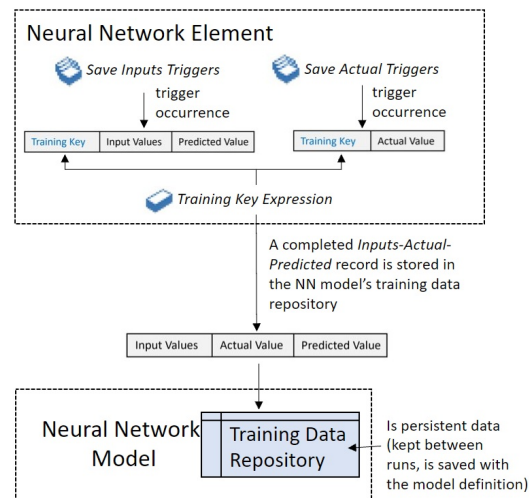
Neural Network Training: The Stochastic Gradient Descent with Momentum Training Algorithm

Simio's Neural Network Trainer uses the Stochastic Gradient Descent algorithm to train a neural network model. This form of training algorithm iterates through all records in the training dataset one record at a time and in a random order, making incremental adjustments to the neural network's weights and bias parameters to reduce the prediction error. The term Epoch is used in machine learning to refer to a single pass by the training algorithm through the full training dataset. A training run is comprised of some number of Epochs, where the training records are re-shuffled at the beginning of each Epoch.

Learning Rate and Momentum are two hyperparameters used to tune the steps the stochastic gradient descent algorithm takes as it incrementally reduces the prediction error. The learning rate directly controls the size of the steps. A larger learning rate will cause the neural network to converge on a solution in fewer epochs, but sometimes at the cost of a suboptimal set of final weights. The momentum term controls the influence of the previous weight update on the next weight update. Momentum can help accelerate gradient descent optimization, leading to faster convergence. However, specifying large momentum values can cause the algorithm to fail to converge.

The cost function used by Simio's Stochastic Gradient Descent method is the mean square error (MSE), which is defined as follows:

2 Define the simulation event-based triggers for training data collection



$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

MSE = mean squared error

n = number of data points

Y_i = observed values

Ŷ_i = predicted values

In other words, this is the average of the squared difference between the predicted and actual observations across all training records. The training algorithm adjusts the weights and biases to reduce this value at each step. Although it generally makes progress, it can also have setbacks with increases in the MSE, particularly with larger values of Learning Rate and Momentum.

Neural Network Training: Early Stopping

A major challenge in training a neural network model is how long to train. Too many epochs can lead to overfitting the model on the training dataset, whereas too few may result in an underfit model. A compromise is to train on the training dataset but to stop training at the point when performance on the validation dataset starts to degrade (i.e., validation error starts to increase) as illustrated below:

This approach to training neural networks is called "early stopping". In the simplest case, training is stopped as soon as the mean squared error on the validation dataset is greater than the error at the end of the prior training epoch. However, because the training of a neural network is stochastic and the validation error may fluctuate many times, some delay or "patience" in stopping is almost always a good idea. Simio's Neural Network Trainer provides a *Validation Patience* setting that indicates the number of consecutive epochs that the mean squared error on the validation dataset can be greater than or equal to the error at the end of the prior epoch before training is automatically stopped.

Listed below are the properties of **Neural Network Trainer**:

Property	Description
Total Records Available	The total number of records in the neural network model's training data repository.
Validation Holdout Percentage	The percentage of the total records available to hold out for the validation dataset. This dataset is used during training to provide an unbiased estimate of the neural network model's performance and to automatically halt training when generalization stops improving (see Early Stopping settings).
Test Holdout Percentage	The percentage of the total records available to hold out for the test dataset. This dataset has no effect on training and is used to provide a final unbiased estimate of the neural network model's performance.
Training Dataset Size	The number of records in the training dataset. This dataset is used to fit the neural network model.
Validation Dataset Size	The number of records in the validation dataset. This dataset is used during training to provide an unbiased estimate of the neural network model's performance and to automatically halt training when generalization stops improving (see Early Stopping settings).
Test Dataset Size	The number of records in the test dataset. This dataset has no effect on training and is used to provide a final unbiased estimate of the neural network model's performance.
Number Hidden Nodes	The number of nodes in the neural network model's hidden layer(s). Specify integers separated by commas if multiple hidden layers. For example, enter 20,10 if 20 nodes in the first hidden layer and 10 nodes in the second hidden layer.
Max Epochs	The maximum number of passes through the entire training dataset. Training will be automatically stopped if this limit is reached.
Optimizer Type	The optimizer used to fit the neural network model to the training dataset. Adam - Use the Adam optimizer. SGDM - Use the stochastic gradient descent with momentum (SGDM) optimizer.
Learning Rate	A small positive value less than or equal to 1 that controls how much the neural network model's learnable weight and bias values are adjusted each time a training record is processed during stochastic gradient descent optimization.
Beta1 Parameter	A value in the range [0,1) that controls the decay rate of the exponential moving averages of the error gradients.
Beta2 Parameter	A value in the range [0,1) that controls the decay rate of the exponential moving averages of the squared error gradients.
Momentum	A value between 0 and 1 that controls the contribution of the previous weight update on the current weight update during stochastic gradient descent optimization. A value of 0 means no contribution from the previous weight update whereas a value of 1 means maximal contribution from the previous update.
Early Stopping Min Delta	The minimum decrease in the validation dataset's mean squared error after completing an epoch for the error reduction to be considered an improvement.
Early Stopping Patience	Indicates the number of consecutive epochs that the mean squared error performance on the validation dataset can stop improving before training is automatically stopped.
Skip First Epochs	The number of initial epochs to exclude from displaying in the training progress plot.

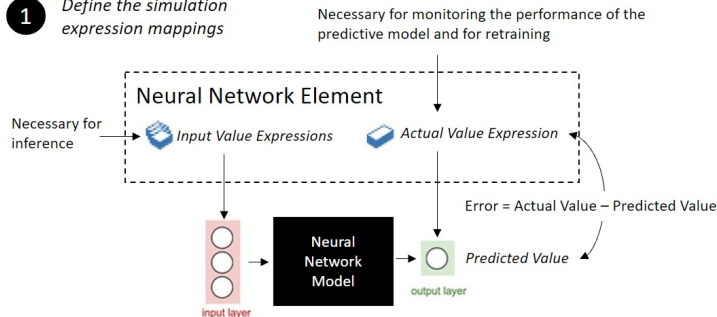
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Neural Network Models - Discussion and Examples

Integrating a Neural Network Regression Model Into Simulation Logic Using a Neural Network Element

1 Define the simulation expression mappings



If an ONNX Neural Network Model, then the vector of simulation input values is mapped to the input tensor of the ONNX model in row-major order. If the ONNX model has multiple input tensors, then those tensors are mapped in the order that they are defined.

3 Use the Neural Network element's Predicted Value function in a simulation expression

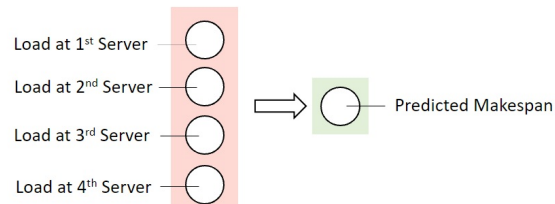
Expression **NeuralNetwork1.PredictedValue(Entity)**

1 Create a feedforward neural network model for predicting the makespan of an order entity through either of the two alternative production lines.

Properties: MakespanPredictor (Feedforward Neural Network Model)

- IO
 - Number Input Nodes: 4
 - Force Negative Outputs To Zero: True
- Training Data Collection
 - Record Training Data: True
 - Maximum Training Record Limit: 1
- General
 - Name: MakespanPredictor

For the purposes of this example, assume the predicted makespan value depends only on the current loads at each of the 4 servers in the production line:



However, note that the input variables affecting a makespan estimation such as this may be much more complex, and include factors such as the current product mix, labor headcounts, inventory levels, scrap or rework rates, etc.

2 Add a Neural Network element for predicting the makespan of an order entity through Production Line 1.

Properties: Line1MakespanPredictor (NeuralNetwork)

- Basic Logic
 - Neural Network Model Name: MakespanPredictor
- Expression Mappings
 - Input Value Expressions: 4 Rows
 - Actual Value Expression: Timeflow - ModelEntity.OrderStartTime
- Training Data Collection
 - Save Inputs Triggers: 1 Row
 - Save Actual Triggers: 1 Row
 - Training Key Expression:
- General
 - Name: Line1MakespanPredictor

Enter the simulation expressions returning the current loads at each of the 4 servers in Production Line 1.

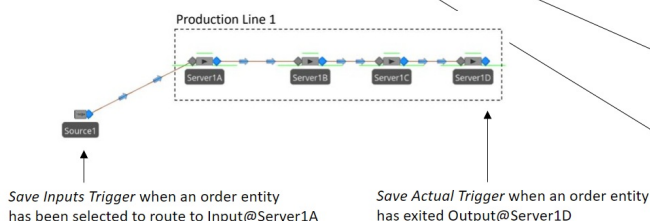
Input Value Expressions - Repeating Property Editor

Items:

- Input@Server1A.AssociatedStationLoad
- Input@Server1B.AssociatedStationLoad
- Input@Server1C.AssociatedStationLoad
- Input@Server1D.AssociatedStationLoad

Properties:

- Expression Mapping
 - Input Value Expression: Input@Server1A.AssociatedStationLoad
- Input Value Expression:



Properties:

- Save Inputs Trigger
 - Triggering Event Name: Output@Source1.RouteRequestSelected
 - Condition: Node.Is.Input@Server1A
- Save Actual Trigger
 - Triggering Event Name: Output@Server1D.Exited
 - Condition:

- 3 Add a Neural Network element for predicting the makespan of an order entity through Production Line 2.

Properties: Line2MakespanPredictor (NeuralNetwork)

- Basic Logic
 - Neural Network Model Name: **MakespanPredictor**
- Expression Mappings
 - Input Value Expressions: **4 Rows**
 - Actual Value Expression: **TimeNow - ModelEntity.OrderStartTime**
- Training Data Collection
 - Save Inputs Triggers: **1 Row**
 - Save Actual Triggers: **1 Row**
 - Training Key Expression
- General
 - Name: Line2MakespanPredictor

Enter the simulation expressions returning the current loads at each of the 4 servers in Production Line 2.

Input Value Expressions - Repeating Property Editor

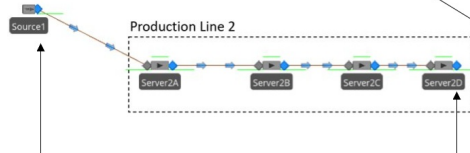
Items:

- Input@Server2A.AssociatedStationLoad
- Input@Server2B.AssociatedStationLoad
- Input@Server2C.AssociatedStationLoad
- Input@Server2D.AssociatedStationLoad

Properties:

- Expression Mapping
 - Input Value Expression: **Input@Server2A.AssociatedStationLoad**

Add Delete Close



Save Inputs Trigger when an order entity has been selected to route to Input@Server2A

Save Actual Trigger when an order entity has exited Output@Server2D

Properties:

- Save Inputs Trigger
 - Triggering Event Name: **Output@Source1.RoutingOut.RouteRequestSelected**
 - Condition: **Node.Is.Input@Server2A**
- Save Actual Trigger
 - Triggering Event Name: **Output@Server2D.Exited**
 - Condition

- 4 Create a data table that specifies the starting nodes and makespan predictors for each of the two alternative production lines.

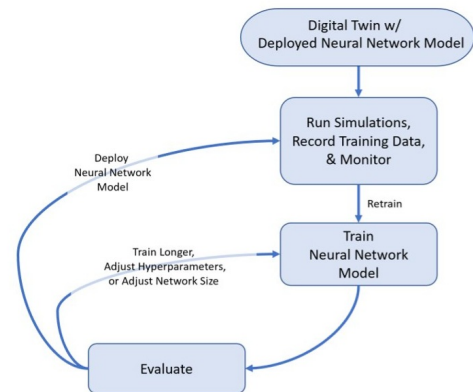
Line Selection		
	Line Starting Node	Makespan Predictor
1	Input@Server 1A	Line 1MakespanPredictor
2	Input@Server 2A	Line 2MakespanPredictor

- 5 Configure the Transfer Node object that represents the routing decision point for the order entities. Route each order entity to the starting node of the production line that has the smallest predicted makespan.

Properties: Output@Source1 (TransferNode)

- Routing Logic
 - Outbound Travel Mode: Continue
 - Outbound Link Preference: Any
 - Outbound Link Rule: Shortest Path
- Entity Destination Type: **Select From List**
 - Node List Name: **LineSelection.LineStartingNode**
 - Selection Goal: **Smallest Value**
 - Selection Expression: **LineSelection.MakespanPredictor.NeuralNetwork.PredictedValue(Entity)**

- 6 Once you have a Digital Twin of the example system with the deployed neural network model, you can improve the accuracy of the NN model's predictions by following the iterative process in the diagram below:

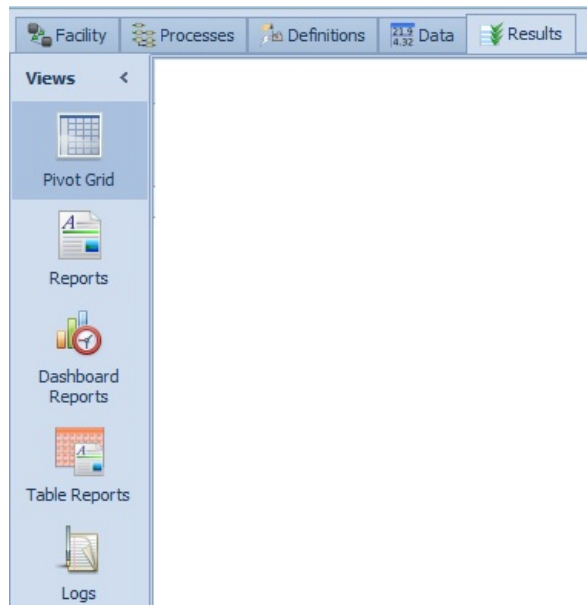


The Results Window

Results Window

The Results window allows the user to display the the results in the form of a [Pivot Grid](#), [Reports](#), [Dashboard Reports](#), [Table Reports](#) or [Logs](#), depending upon which is selected in the panel. The data from the Pivot Grid and Logs can be [exported](#) into a .csv file.

For information on how to add manual statistics to the project, see [OutputStatistic](#), [StateStatistic](#) and [TallyStatistic](#).



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Pivot Grid

Pivot Grid

The Pivot Grid displays result data from a simulation run(s). It lets the user interactively analyze the results in the form of a dynamic rotatable table. You can drag and drop columns in the table to rotate or “pivot” the data. You can also filter and sort the information that is shown. This is similar to features found in the pivot table provided by Microsoft Excel.

The Results window displays the Pivot Grid panel by default. So when you first click the Results tab, you are viewing the Pivot Grid.

The section of the Pivot Grid ribbon labeled Unit Settings allows the user to specify the units to be displayed in the reports, pivot grid, watch window, status bar, and trace window.

The following categories help to organize the result data:

- Content – category used for statistics on the number of things inside or on something, such as NumberInSystem or NumberOnLink.
- Throughput – category used for totals on items entered/created or items exited/destroyed.
- Capacity – category used for statistics related to capacity, including number of units allocated over time, units scheduled and utilization of the object. Scheduled utilization is calculated based on the units utilized divided by the units scheduled.
- FlowTime – category used for entity time in system (population or by Sink) or entity time on link.
- HoldingTime – category used for entity time held in a station location or held in a batch.
- ResourceState - category used to show occurrences, percent and average times in each of the resource states in which an object may be.
- Costs - category used to show final values of cost for each object or cost center, in addition to Population.Cost statistics for Entity and Transporter type objects.

A Pivot Grid

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
ModelEntity	DefaultEntity	[Population]	Content	NumberInSystem	Average	2.3831
					Maximum	15.0000
			FlowTime	TimeInSystem	Average (Ho...	0.0101
					Maximum (Ho...	0.0482
					Minimum (Ho...	0.0025
			Throughput	NumberCreated	Total	5,649.0000
				NumberDestroyed	Total	5,648.0000
Path	Path1	[Travelers]	Content	NumberOnLink	Average	0.0925
					Maximum	3.0000
			FlowTime	TimeOnLink	Average (Ho...	0.0004
					Maximum (Ho...	0.0004
					Minimum (Ho...	0.0004
			Throughput	NumberEntered	Total	5,649.0000
				NumberExited	Total	5,649.0000
	Path2	[Travelers]	Content	NumberOnLink	Average	0.0866
					Maximum	1.0000
			FlowTime	TimeOnLink	Average (Ho...	0.0004
					Maximum (Ho...	0.0004
					Minimum (Ho...	0.0004
			Throughput	NumberEntered	Total	5,648.0000
				NumberExited	Total	5,648.0000
Server	Server 1	[Resource]	Capacity	ScheduledUtilization	Percent	78.3403
				UnitsAllocated	Total	5,649.0000
				UnitsScheduled	Average	1.0000
				UnitsScheduled	Maximum	1.0000
				UnitsUtilized	Average	0.7834
				UnitsUtilized	Maximum	1.0000

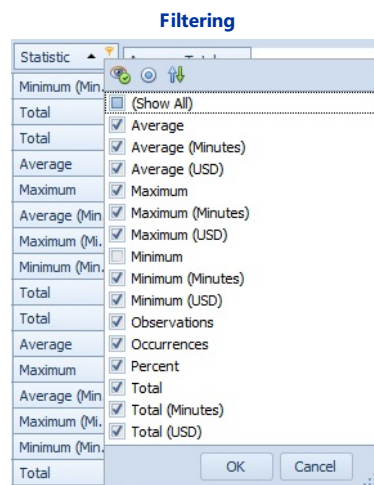
This Pivot Grid provides a very easy way to manipulate, summarize, and display data and provides a powerful mechanism for exploring your data (data mining).

The red circle above identifies the columns and their symbols. Columns may be moved around within the pivot grid. The up/down arrow may be used to sort the rows within a column into alphabetical order. Moving the mouse over a given column will display the filter button (described below). Data items may be moved to reconfigure the table.

Grouping: For example, left click and drag the leftmost column named Object Type to the top row (next to Data Source Type). You will see that the data is no longer grouped by Object Type, but rather only by Object Name. You may also drag the columns to different positions to cause the data to be grouped differently. For example, if you wanted all of your data items displayed together (e.g. all the Arrivals statistics together) you could drag the Data Item column to the far left, in front of Object Name.

Sorting: You will see a small triangle in every column heading – this determines the sorting order for that column. If you click on that triangle (or in fact almost anywhere in the heading box), it will toggle the column sort order between ascending and descending.

Filtering: You may have noticed that the Statistic heading in the Pivot table above has an extra symbol in the upper right corner – a funnel. This indicates that it is filtering the data. If you click on the funnel you will see a menu indicating all possible values in that column and you will see that Minimum has been unchecked. This means that any items labeled with Minimum have been suppressed (our default setting). If you check the Minimum or the (Show All), you will see all the minimum values will now be added to your report.

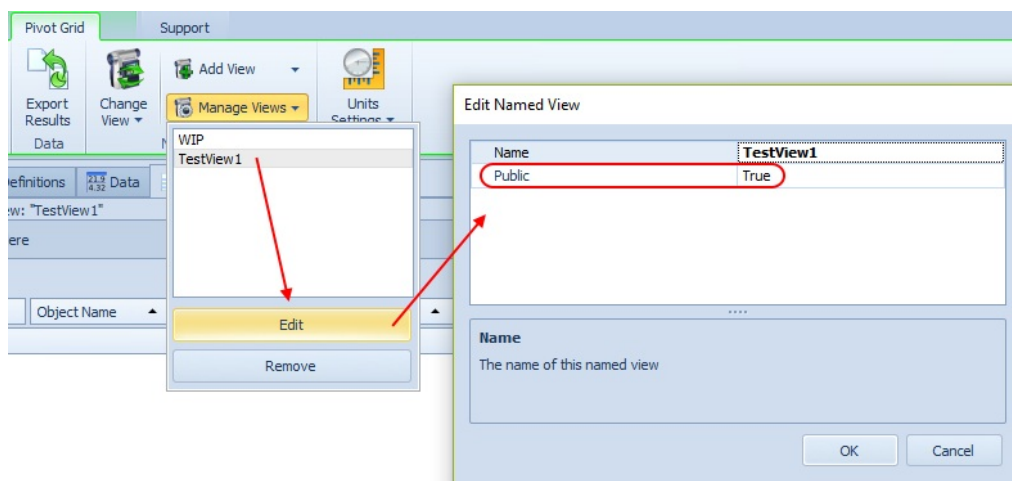


All column headings and items in the filter area have this filtering capability. For example, if you hover the mouse over the upper right corner of the Object Type box (the one you just moved to the filter area), you will see that funnel appear. Clicking on that funnel shows all object types found in the report. If you have no interest in seeing statistics on your paths for example, simply uncheck the Paths item and all of the statistics associated with paths will be suppressed.

Named Views

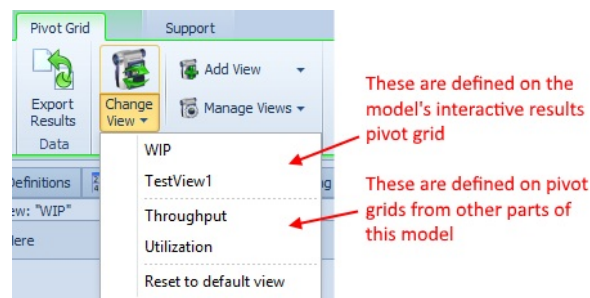
The Change View, Add View and Manage Views button allow you to record column ordering, column width, sort and filter settings and save them to a particular 'view' for later reference. Within the Change View button is also the option to reset the pivot grid to the original default view. When a named view is open, you will see *Showing Named View: "ViewName"* at the top of the grid.

Named views in our various pivot grids are "Public" by default, allowing them to be used by any other pivot grid in the model or any of its experiments:



When using named views in a pivot grid, the public ones from the other pivot grids associated with the same model are

visible. For example, when viewing the Results pivot grid for a model, note its own named views, followed by public named views from other pivot grids (including those defined by the model's Experiments, as well as Planning results in Simio RPS edition).



When applying a named view to a pivot grid, Simio will automatically hide or show certain columns as appropriate to the pivot grid. Specifically, the Minimum, Maximum, Half Width, and Std. Dev. columns will be hidden when applying an experiment's named pivot view to a model's pivot grid, while the Minimum, Maximum, and Half Width columns will automatically be shown when applying a named pivot view from a model's pivot grid to an experiment's pivot grid.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Reports

Reports

The Reports panel of the Results tab displays a traditionally formatted report that provides flexible visual formatting for inclusion in printed documents and is useful in presenting results to others. Reports are provided for both a single scenario and for comparing two scenarios. After clicking on the Results tab and Reports panel button to view the reports, select the appropriate report from the Current Report dropdown in the ribbon menu.

The [Report Designer](#) allows you to create a custom Report based on generated model data. Reports can be made using data from the Interactive Model Results (i.e. the data displayed in the Pivot Grid), the Resource Usage Log, the Resource State Log and the Constraint Log.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

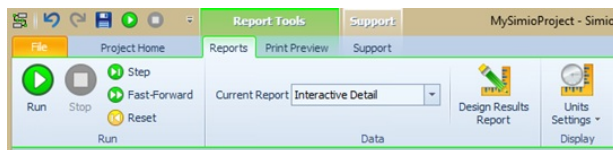
Send comments on this topic to [Support](#)

Report Designer - RPS

The Report Designer allows you to create a custom Report based on generated model data. Reports can be made using data from the Interactive Model Results (i.e. the data displayed in the Pivot Grid), the Resource Usage Log, the Resource State Log and the Constraint Log. Simio also provides access to the Report Designer through the Table Reports panel button of the Results window. Custom reports may be created using any of the data tables within a Simio model.

Custom reports can contain tables, charts, grids, pictures, text and interactive features, such as drop down selections and check boxes. There is complete documentation, including helpful videos, on how to create custom reports using this interface on the Dev Express website: <https://documentation.devexpress.com/#WindowsForms/CustomDocument8118>

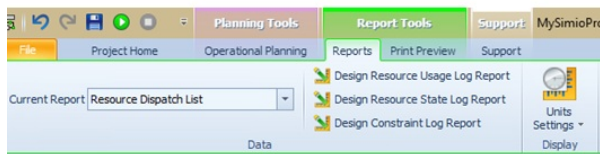
The Report Designer button can be found on the Reports ribbon, which is visible from the Results window when you view the Reports panel. Report Designer is also available within the Table Reports panel to create custom table reports from data tables in a model. The Reports ribbon is also available when viewing the Reports window of the Planning/Results panel or viewing the Reports window of an Experiment. The Report Designer button can also be found on the Table Report ribbon, which is visible from the Results window when you view the Table Reports panel.



Reports Ribbon from Results/Reports



Reports Ribbon from Experiment/Reports



Reports Ribbon from Planning/Results/Reports

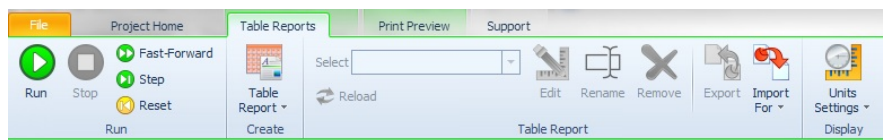
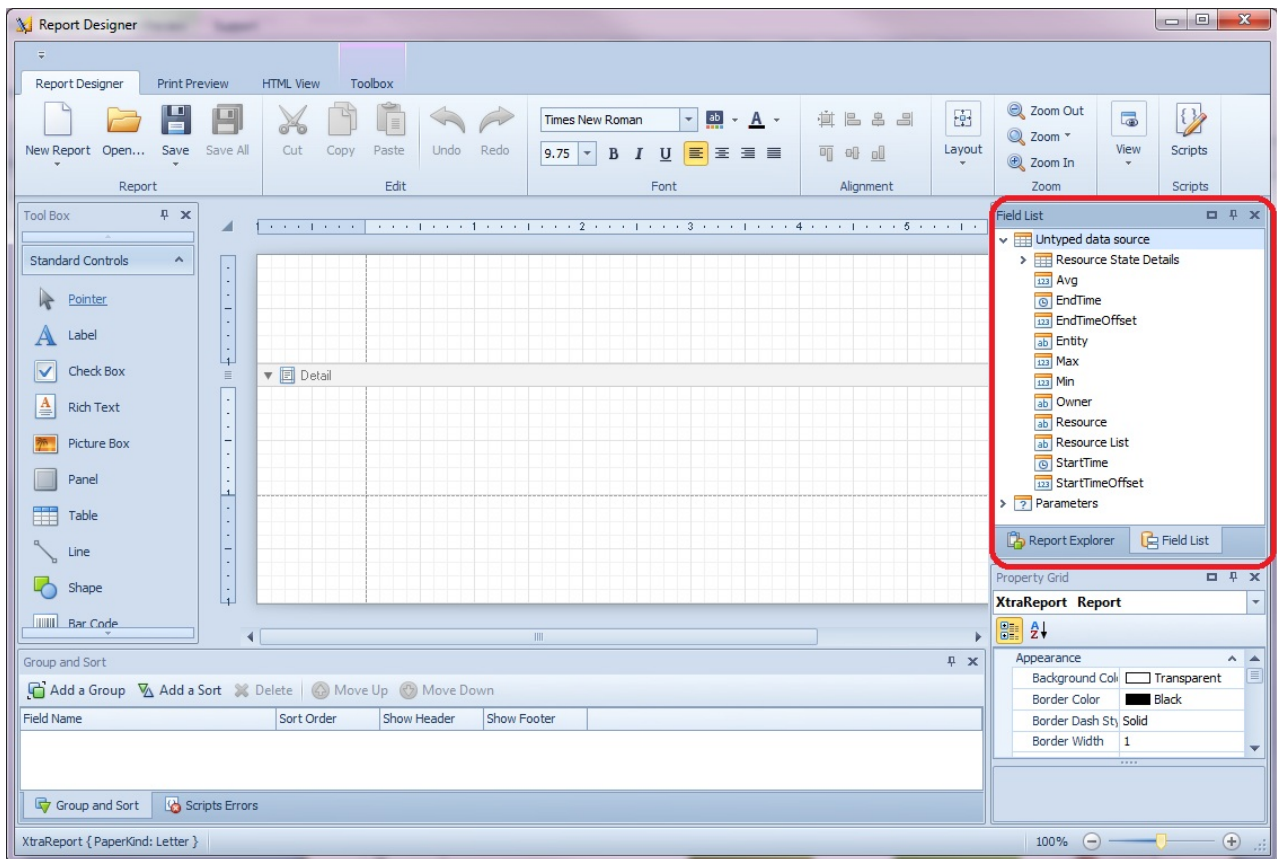
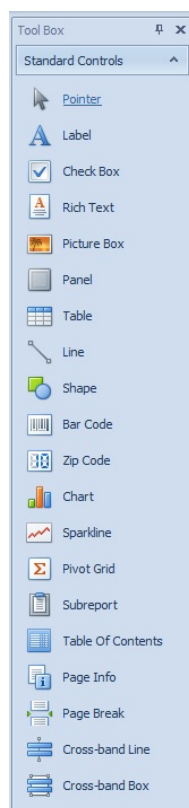


Table Reports Ribbon from Results / Table Reports

The Report Designer interface will appear in the new window. Depending on what data source you selected from the Report Designer drop down in the ribbon, the Field List will be populated with the appropriate tables and fields that are available from this data source. This is the data that can be used to create your custom report.



These are the tools that are used to design the layout of the report. They are placed into the appropriate band within the main grid area.



You can customize the layout of the report by right clicking in the main grid area and selecting Insert Band and then selecting Report Header, PageHeader, GroupHeader, ReportFooter or PageFooter. Each type of band has its own use, such as the Report Header, that whatever is placed in this band is common among all pages of the report. For example, you might put a report title here by placing a Label from the Toolbox. Or the Page Footer can be used to display page related information that is common on all pages, such as a page number.

Simio has three reports that are found on the Reports page of the Results tab. These are the **Interactive Detail** report, the **Scenario Comparison** report and the **Scenario Detail** report. If you are using the RPS version, there are more reports on

the Reports tab of the Results page. These reports are the **Resource Dispatch List**, the **Workflow Constraints Analysis** and the **Resource Utilization Summary**. All six of these reports can be found in the Simio reports folder (C:\Program Files\Simio LLC\Simio\Reports). You can open these reports in the Report Designer to see how they were created and to use these as a starting point for creating your own custom report.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

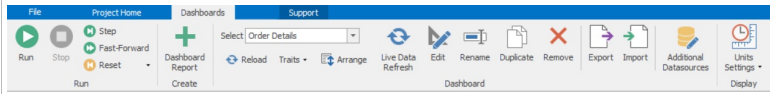
Dashboard Reports

What are Dashboard Reports

Dashboard reports allow you to display a dashboard based report of generated model data.

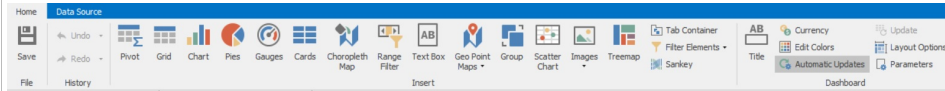
A dashboard allows you to organize and present model information that is customizable and easy to read. Model data can be displayed in many graphical forms such as grids, charts and gauges. It also allows you to customize your dashboard by adding images, text boxes and filter ranges.

The major components of the Dashboard Report tool include creating a dashboard report and selecting a dashboard report to view, edit, rename or remove. A dashboard report may also be exported or imported. The file format is XML. There is also the option to set display settings such as time, length and distance, volume and mass and weight.



Additional Datasources allows you to enable to add periodic views of different logs.

Selecting Create Dashboard Report will prompt you to enter a dashboard name. Once a name is entered, the DashboardDesignerForm window will appear.



From here, you have the option to insert various dashboard items such as a pivot grid, grid, chart, pie chart, gauge, card image, text box or range filter.

Pivot – A pivot grid displays a cross tabular report. It can be used to present multi-dimensional data in an easy to read format.

Grid – A grid shows data in a tabular form while allowing you to summarize against specific measures or calculate differences between them.

Chart – A chart visualizes data in an XY diagram, allowing you to render a wide range of diagram types such as a bar, point, area, range, bubble or financial chart.

Pies – A pie chart displays a series of pies that represent the contribution of each value to the total.

Gauges – A gauge chart is another method for displaying a series of model data. It displays two values – one with a needle and one with a marker on the scale.

Cards – Cards display the differences between two values, which can be expressed as an absolute for percent.

Choropleth Map – This item allows you to place a choropleth map dashboard item onto your dashboard.

Range Filter – A Range Filter item allows you to apply filtering to other dashboard items. It displays a chart with selection thumbs over it that allow you to filter out values displayed along the argument axis.

Text Box – A Text Box item allows you to enter text onto your dashboard. Right clicking within the Text Box item allows you to enter text.

Geo Point Map – This allows you to place a geo point map dashboard item onto your dashboard.

Group – Create a group that arranges dashboard items and allows you to manage interaction between dashboard items within and outside the group.

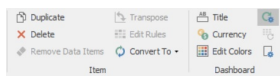
Scatter Chart – Create a scatter chart dashboard item and insert it into the dashboard.

Image – An Image item allows you to insert an image into the dashboard. Right clicking in the Image item will allow you to load or import an image.

Treemap – Create a treemap dashboard item and insert it into the dashboard.

Sankey – Create a Sankey diagram dashboard item and insert it into the dashboard. The Sankey diagram visualizes data as weighted flows or relationships between nodes.

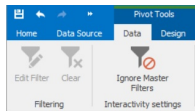
Once a dashboard item has been inserted, you have the ability to duplicate or delete an item. There is also the ability to add a dashboard title and to identify currency units to be displayed.



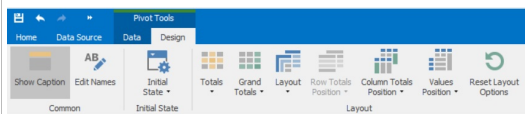
In addition, right clicking into a dashboard item will give you the option to show an item caption, duplicate or delete the item, edit the dashboard item name, edit an item filter, print preview the item or export to a PDF or image (PNG) file.

Once a dashboard item is inserted into the window, other ribbons or tools are available which are specific to the item inserted. These are briefly discussed below. By hovering over each tool within the ribbon, you will get more information on what each tool accomplishes.

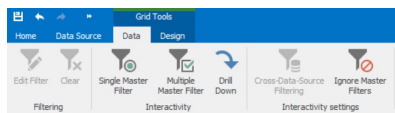
Pivot Data Tools include Edit Filter, Clear, and Ignore Master Filters.



Pivot Design Tools include Show Caption, Edit Names, Initial State, and various Layout options.



Grid Data Tools include Edit Filter, Clear, Single Master Filter, Multiple Master Filter, Drill Down, Cross-Data-Source Filtering, Ignore Master Filters.



Grid Design Tools include Show Caption, Edit Names, Horizontal Lines, Vertical Lines, Banded Rows, Merge Cells, Word Wrap, AutoFit to Contents, AutoFit to Grid, Manual.

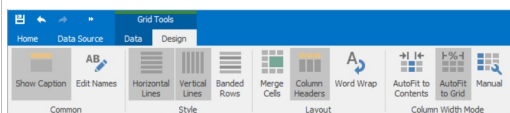


Chart Data Tools include Edit Filter, Clear, Single Master Filter, Multiple Master Filter, Drill Down, Cross-Data-Source Filtering, Ignore Master Filters, Arguments, Series, Points.

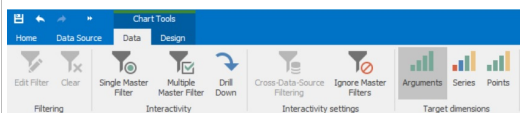
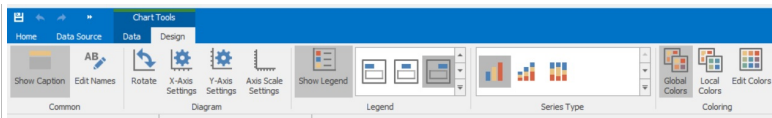
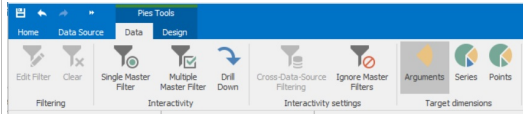


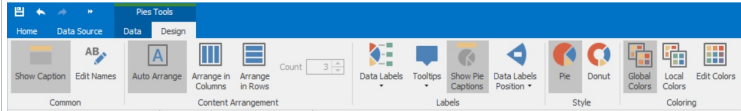
Chart Design Tools include Show Caption, Edit Names, Rotate, X-Axis Settings, Y-Axis Settings, Axis Scale Settings, Show Legend and Series Type, Global Colors, Local Colors, Edit Colors.



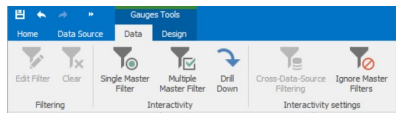
Pie Data Tools include Edit Filter, Clear, Single Master Filter, Multiple Master Filter, Drill Down, Cross-Data-Source Filtering, Ignore Master Filters, Arguments, Series, Points.



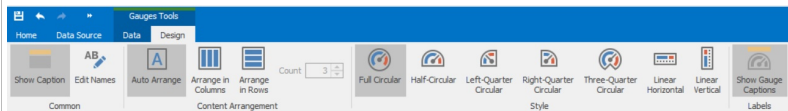
Pie Design Tools include Show Caption, Edit Names, Auto Arrange, Arrange in Columns, Arrange in Rows, Data Labels, Tooltips, Show Pie Captions, Data Labels Position, Pie or Donut style, Global Colors, Local Colors, Edit Colors.



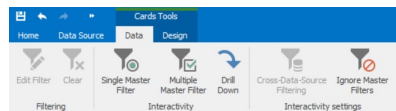
Gauge Data Tools include Edit Filter, Clear, Single Master Filter, Multiple Master Filter, Drill Down, Cross-Data-Source Filtering, Ignore Master Filters.



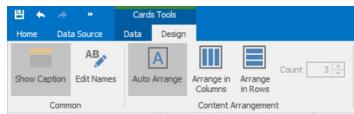
Gauges Design Tools include Show Caption, Edit Names, Auto Arrange, Arrange in Columns, Arrange in Rows, and styles such as Full Circular, Half-Circular, Left-Quarter Circular, Right-Quarter Circular, Three-Fourth Circular, Linear Horizontal, Linear Vertical and Show Gauge Captions.



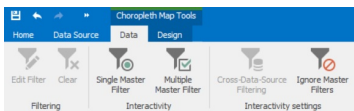
Card Data Tools include Edit Filter, Clear, Single Master Filter, Multiple Master Filter, Drill Down, Cross-Data-Source Filtering, Ignore Master Filters.



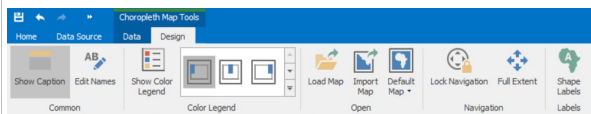
Cards Design Tools include Show Caption, Edit Names, Auto Arrange, Arrange in Columns or Arrange in Rows, Count.



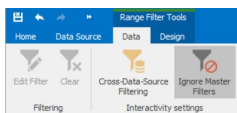
Map Data Tools include Edit Filter, Clear, Single Master Filter, Multiple Master Filter, Cross-Data-Source Filtering and Ignore Master Filters.



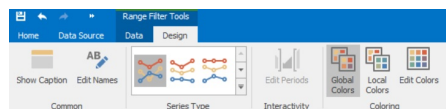
Map Design Tools include Show Caption, Edit Names, Show Color Legend, Load Map, Import Map, Default Map, Lock Navigation, Full Extent, Shape Labels.



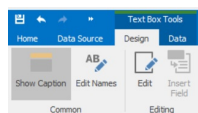
Range Filter Data Tools include Edit Filter, Clear, Cross-Data-Source Filtering, Ignore Master Filters.



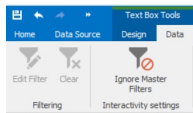
Range Filter Design Tools include Show Caption, Edit Names, allows you to change the series type to a Line, Stacked Line, Full Stacked Line, Area, Stacked Area or Full-Stacked Area, Edit Periods, Global Colors, Local Colors, Edit Colors.



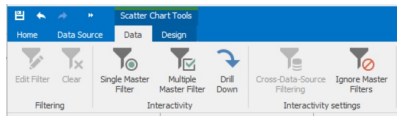
Text Box Design Tools allow you Show Caption, Edit Names as well as allowing you edit various text settings.



Text Box Data Tools includes Edit Filter, Clear, Ignore Master Filters.



Scatter Chart Data Tools include Edit Filter, Clear, Cross-Data-Source Filtering, Ignore Master Filters.



Scatter Chart Design Tools include Show Caption, Edit Names, Rotate, X-Axis Settings, Y-Axis Settings, Axis Scale Settings, Point Labels, Show Legend, Global Colors, Local Colors, Edit Colors.

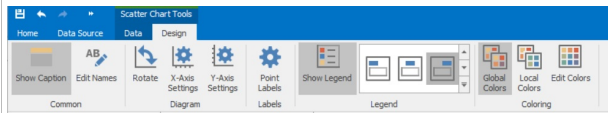
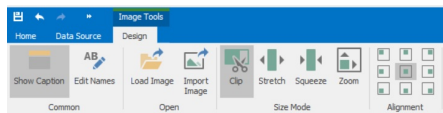
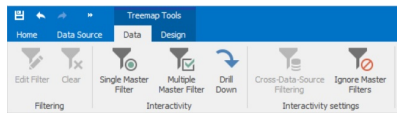


Image Design Tools include Show Caption, Edit Names and include options such as Load Image, Import Image, Clip, Stretch, Squeeze, Zoom and Alignment.



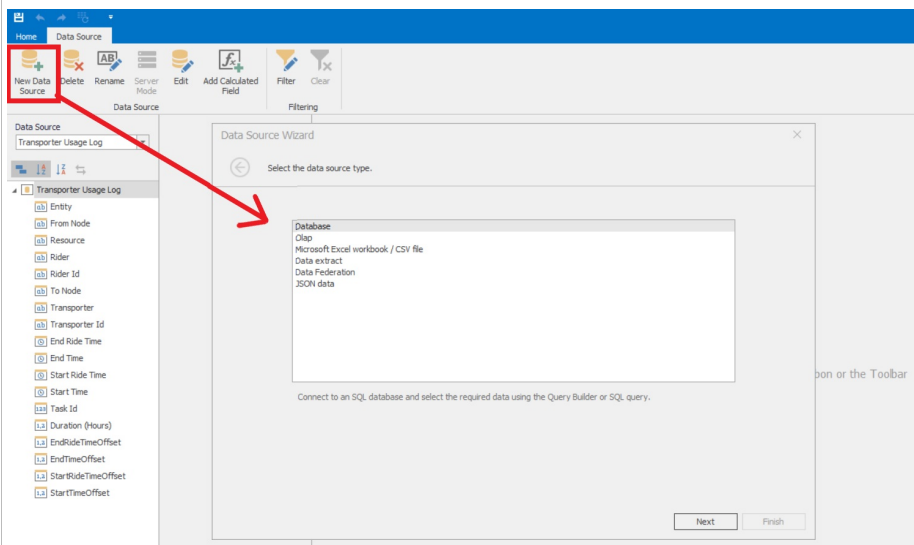
Treemap Data Tools include Edit Filter, Clear, Cross-Data-Source Filtering, Ignore Master Filters.



Treemap Design Tools include Show Caption, Edit Names, Slice and Dice, Squarified, Striped, Layout Direction, Labels, Tooltips, Global Colors, Local Colors, Edit Colors.



There is also a Data Source ribbon tab in the dashboard reports designer, so that a data source, such as a database, may be connected to retrieve additional external data for a dashboard. The New Data Source button on the Data Source ribbon will open the following dialog.



Here you can select the data source type you want to use. Once you select your data source, click Next and then you can connect to the data source and specify the import settings.

The object to include in your data source is then selected and can later be seen in the list of data logs and reports to utilize when building the dashboard report.

In earlier versions of Simio, there was the concept of a dashboard where plots, pie charts, gauges, etc. could be added to a model. This functionality is still available however it has been renamed to Console and is now available under the Definitions tab. While the items in the Console view are animated as the model runs, the new Dashboard Reports display post run data.

Dashboard Report Edition Specific Functionality

Dashboard Report functionality varies based on your license type. Below is a chart that describes the differences:

Edition/License Type	Functionality
Trial Edition	Access to the Dashboard Reports Window and ability to view dashboards. Creation and editing of dashboards are limited to state and tally observations in addition to user created tables.
Design Edition	Access to the Dashboard Reports Window and ability to view dashboards that contain state and tally observations in addition to user created tables. No editing is permitted.
Professional Edition	Access to the Dashboard Reports Window and ability to view dashboards. Creation and editing of dashboards are limited to state and tally observations in addition to user created tables.
RPS Edition	Access to the Dashboard Reports Window and ability to view dashboards. Creation and editing of dashboards are not limited.

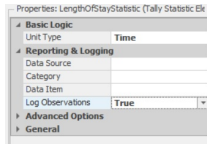
Example 1 – How to create a dashboard report with a model that contains tallies

Problem – In the HospitalEmergencyDepartment example, I want to be able to graphically display the differences between the times patients wait for rooms, times patients wait for beds and patients length of stay, both as an average as well as each data point over time.

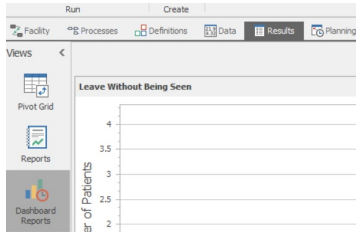
This example will illustrate how to create a simple dashboard that will display tally statistics.

Load the HospitalEmergencyDepartment SIMIO Project File located in your Examples folder.

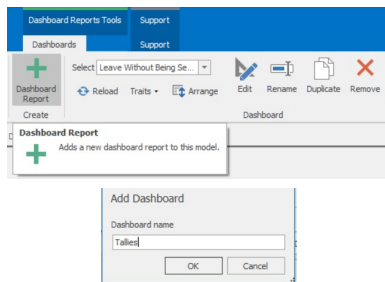
Open the Definitions window and for each TallyStatistic, under Advanced Options, set the Log Observations value to 'True' for TallyStatistic, TimeWaitedForBeds, TimeWaitedForRoom and LengthOfStayStatistic. This will create a log of the tally observation data that will be used to populate the dashboard items.



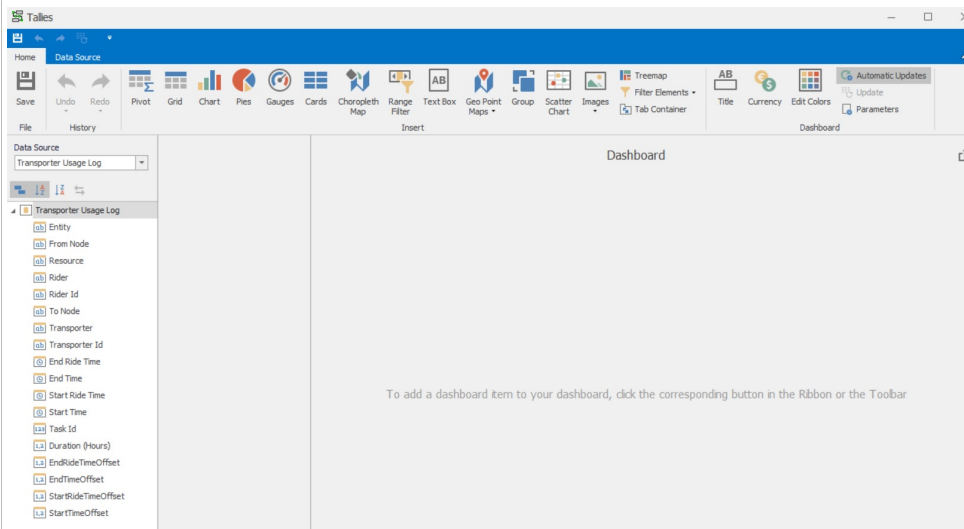
Fast forward the model to completion. Once the model run has ended, open the Results window and then select the Dashboard Reports view.



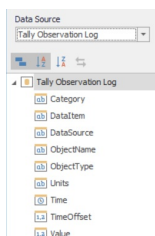
Click Dashboard Report Create in the upper left corner of the window and select a dashboard name (ex: Tallies).



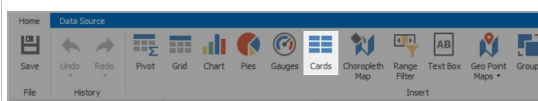
This will bring up the DashboardDesignerForm window.



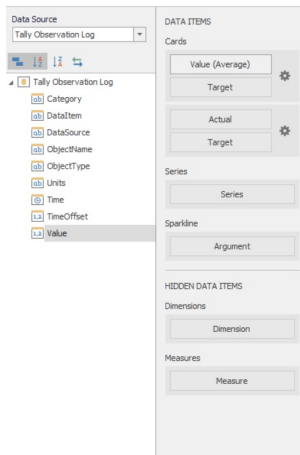
From the dropdown list in the upper left corner, select Tally Observation Log. That is the log file we will use to populate the dashboard.



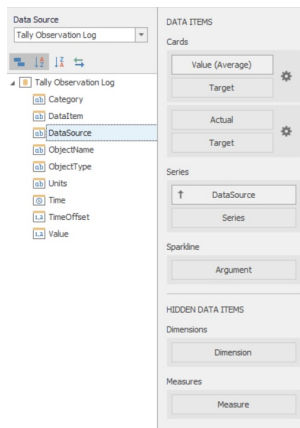
From the Dashboard ribbon click on Cards. Cards are used to display the differences between values. A Card window and corresponding data items will appear.



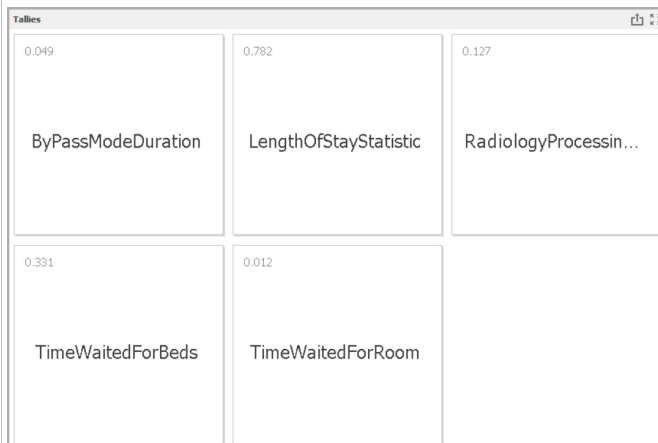
On the left side of the window, under Tally Observation Log, select the Value item and drag it onto the Actual data item box. This is the value that will be displayed on each card. Now change the modifier from Sum to Average by clicking on the arrow to the right of the data item box. This will cause the average of all Values in the log for each data source item to be displayed on each card.



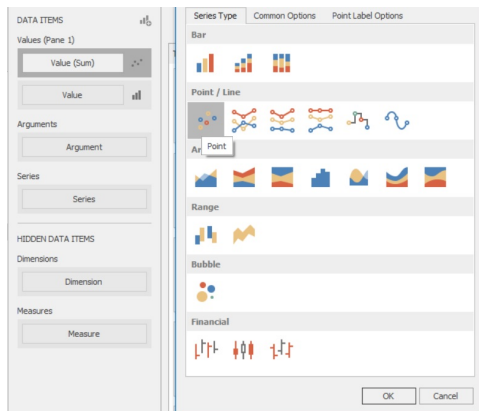
Now drag the DataSource item onto the Series data item box. This will create a card for each DataSource item in the Tally Observation Log and label each card with the name of the DataSource item. Each Card will display the average of all Values for that Tally.



Right click in the Card box and select Edit Names. Change the name from Chart 1 to Tallies. You should see something that looks like this.



On the Dashboard ribbon and select Chart. Drag Value item onto the Value data item box (keeping the modifier as Sum). This will display the sum of the Value for each Tally item. To the left of the Value(Sum) data item box, click on the icon and change the Series type to Point.



Drag the Time item to the Argument data item box and change the modifier to be Date-Hour-Minute. This will define the X-axis as a time line. Make sure the arrow next to the Time data item is pointing up so that the chart will show the correct order of the time line.

DataSource

Tally Observation Log

Category

DataItem

DataSource

ObjectName

ObjectType

Units

Time

TimeOffset

Value

DATA ITEMS

Values (Pane 1)

Value (Sum)

Value

Arguments

Time (Date-Hour-Minute)

Argument

Series

Series

HIDDEN DATA ITEMS

Dimensions

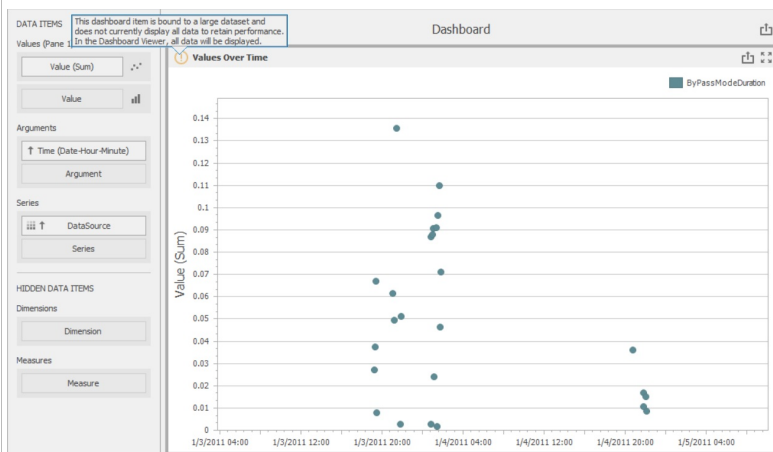
Dimension

Measures

Measure

Drag the DataSource item to Series data item box. This identifies the Tallies that are to be charted. Right click in the Chart box and select Rename to change the name from Chart 1 to Values Over Time.

You should have something that looks like this.



To see the entire Dashboard, you will need to look at the Dashboard Viewer, where you will see something that looks like this.

Values Over Time



Now click on Range Filter in the Dashboard ribbon. This will allow you to apply filtering to the dashboard items.

Drag the Value item over the Value data item box, keeping the Sum modifier. Drag the Time item over the Argument data item box, changing the modifier to Date-Hour-Minute. By identifying the Argument as Time, you will be able to filter the other items with the dashboard bases on the time range. Drag the DataSource item to the Series data item box.

DataSource

Tally Observation Log

Category

DataItem

DataSource

ObjectName

ObjectType

Units

Time

TimeOffset

Value

DATA ITEMS

Values

Value (Sum)

Value

Argument

Time (Date-Hour-Minute)

Series

DataSource

Series

HIDDEN DATA ITEMS

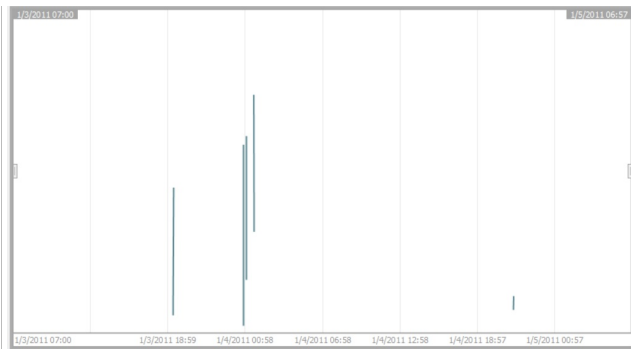
Dimensions

Dimension

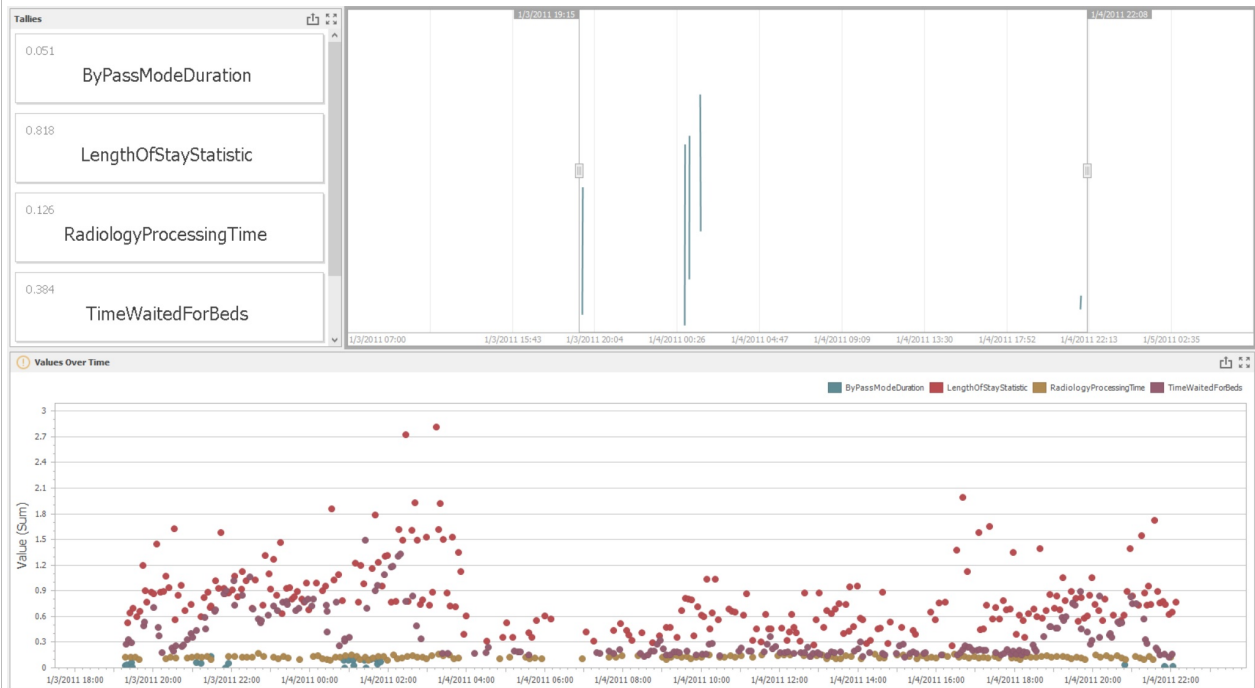
Measures

Measure

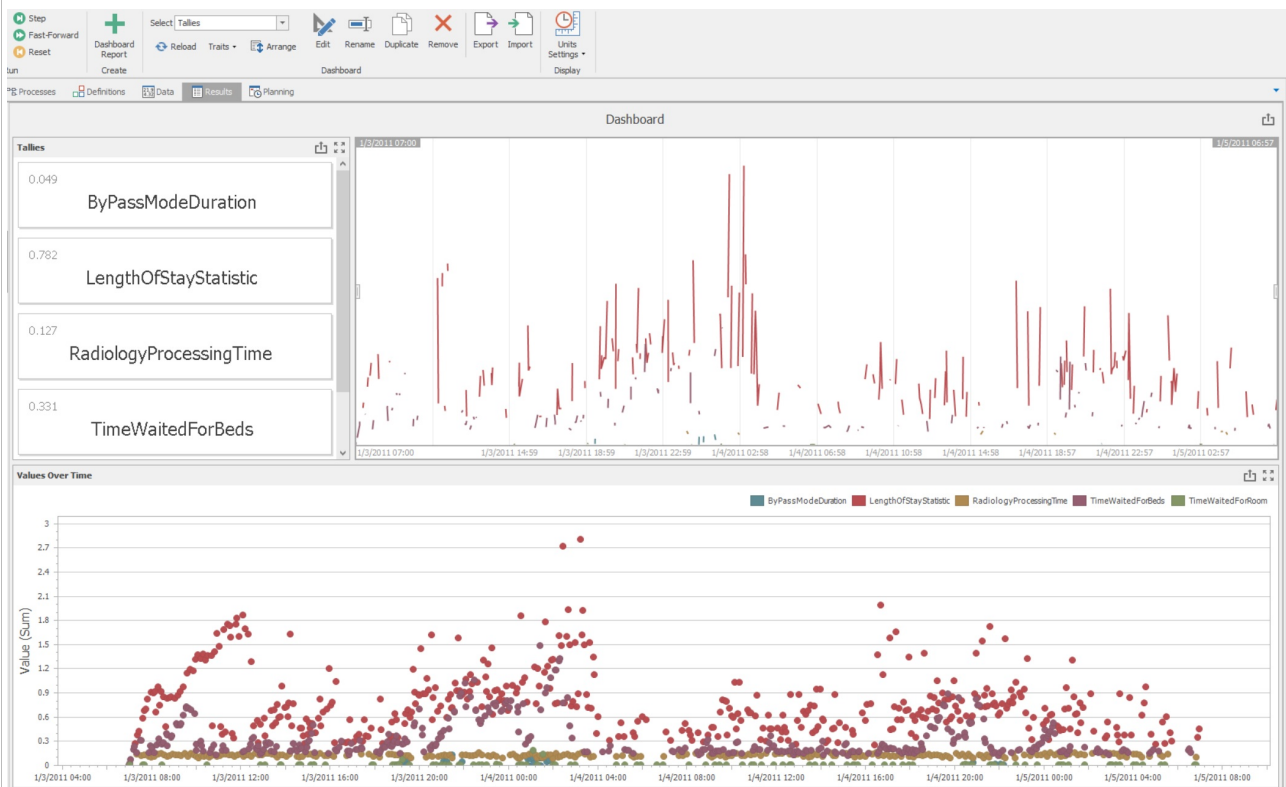
You should have something that looks like this.



Each window within the DashboardDesignerForm window can be resized and moved. Below is an example of what can be shown.



Select Save from the dashboard ribbon and close the DashboardDesignerForm window. To view a specific dashboard report, select it from the Select pull down list.



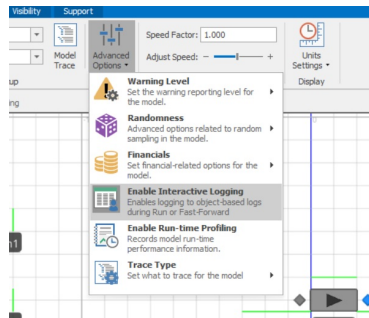
You may also want to try to use the selection thumbs on the range filter to see how the dashboard changes. Also, try using

the grid item instead of the cards to display the same results in grid fashion.

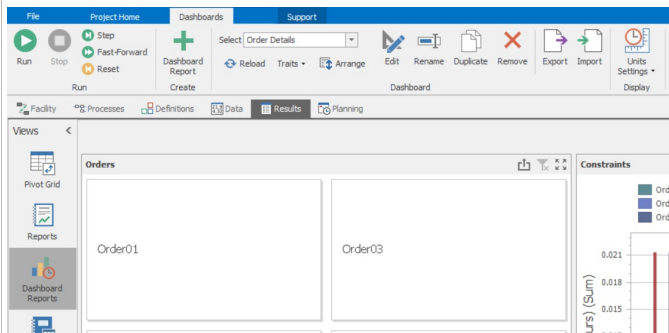
Example 2 – How to create a dashboard report with a tables

Problem – In the SchedulingDiscretePartProduction model, I want to be able to graphically display material usage over time as well as the resource states of the operator, forklift and workcenter areas as a percentage of model duration.

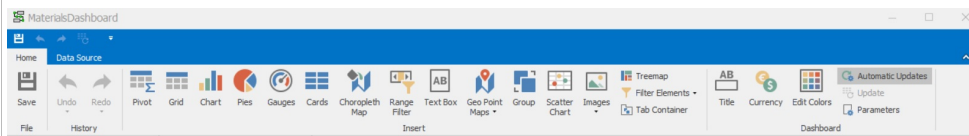
Load the SchedulingDiscretePartProduction SIMIO Project File located in your Examples folder. In the Facility window, under the Run Ribbon, under Advanced Options, make sure Enable Interactive Logging is turned on. This will ensure that log file data will be captured.



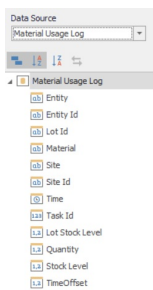
Fast forward the model to completion. Once the model run has ended, open the Results window and then select the Dashboard Reports view.



Click Dashboard Report Create in the upper left corner of the window and select a dashboard name (ex: MaterialsDashboard). This will bring up the DashboardDesignerForm window.



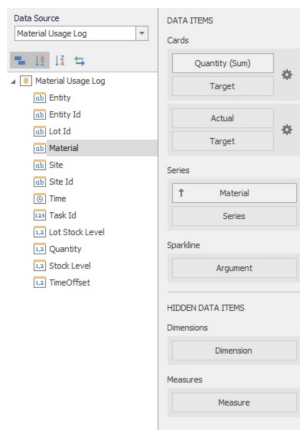
From the dropdown list in the upper left corner, select Material Usage Log. That is the log file we will use to populate the dashboard.



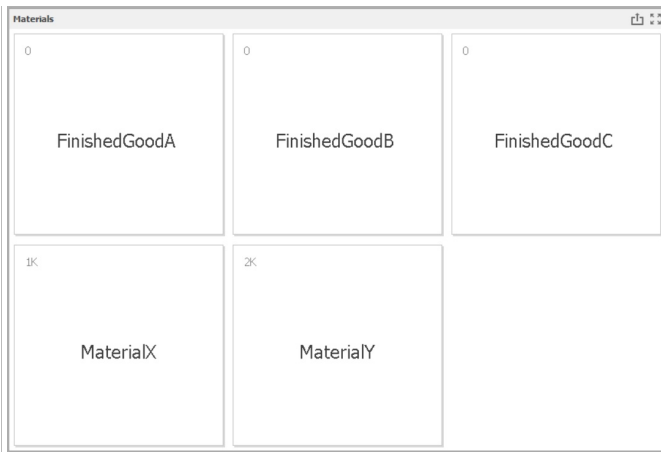
From the Dashboard ribbon click on Cards. Cards are used to display the differences between values.

On the left side of the window, under Material Usage Log, select the Quantity item and drag it onto the Actual data item box. This is the value that will be displayed on each card. This will cause the sum of all Material Quantities in the log for each data source item to be displayed on each card.

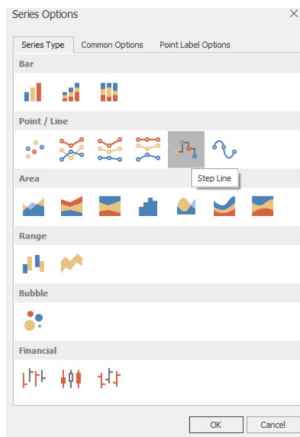
Now drag the Material item onto the Series data item box. This will create a card for each Material item in the Material Usage Log and label each card with the name of the Material item. Each Card will display the Sum of all Material Quantities.



Right click in the Card box and select Edit Names. Change the name from Chart 1 to Materials. You should see something that looks like this.



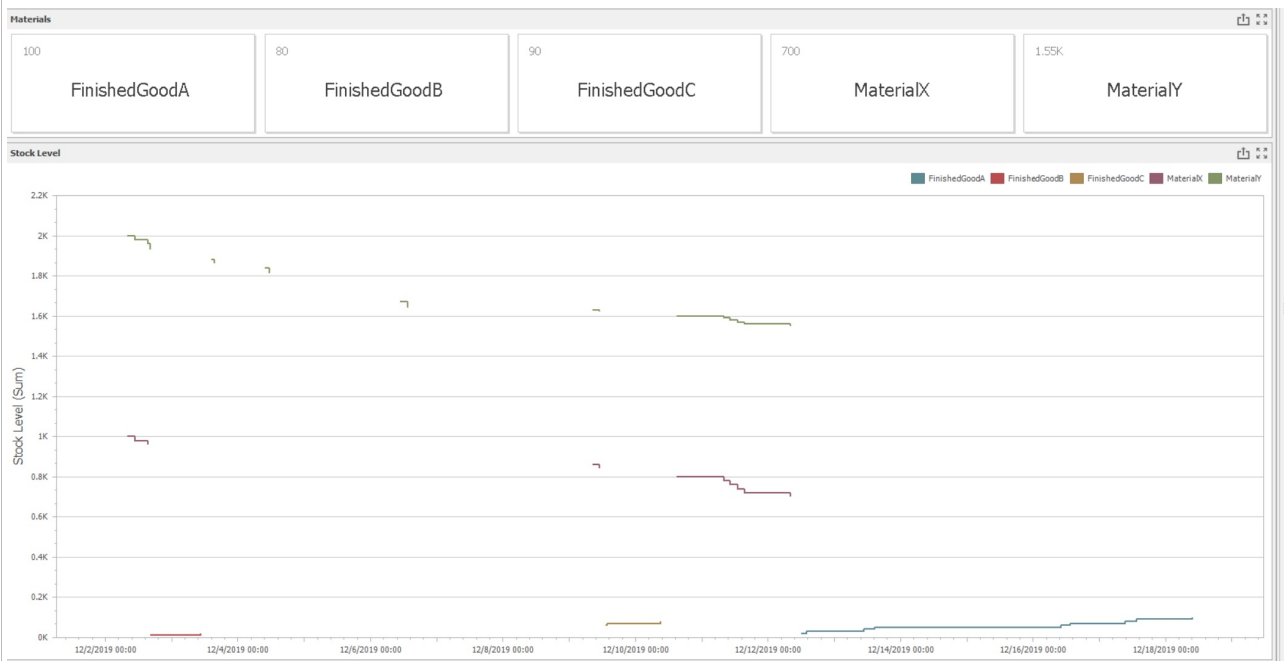
On the Dashboard ribbon and select Chart. Drag the StockLevel item onto the Value data item box (keeping the modifier as Sum). This defines the data item, in this case StockLevel, to chart. To the right of the StockLevel(Sum) data item box, change the Series type to Step Line.



Drag the Time item to the Argument data item box and change the modifier to be Date-Hour-Minute. This will define the X-axis as a time line. Make sure the arrow next to the Time data item is pointing up so that the chart will show the correct order of the time line.

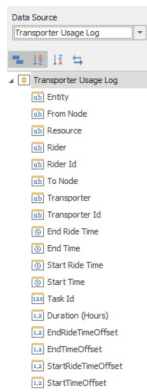
Drag the Material item to Series data item box. This defines the Materials types that will be charted. Right click in the Chart box and select Edit Names to change the name from Chart 1 to Stock Level.

Click the title bar of the Stock Level chart and move it below the Material cards. Save the dashboard. You should see something that looks like this.

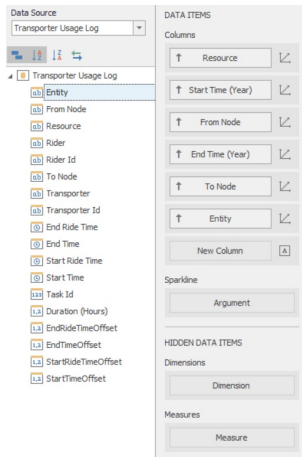


You may also want to try adding a Range Filter to the dashboard to allow you to display the data in a shorter time span.

Let's create another dashboard. Name this one Pick Ups. Make sure Transporter Usage Log appears from the pull down list on the left side of the window.



On the Dashboard ribbon click on Grid. Drag the Resource item onto the first Column data item box, followed by Start Time (Date-Hour-Minute), From Node, End Time(Date-Hour-Minute), To Node and Entity. This identifies the columns that will be displayed on the grid and their corresponding values.

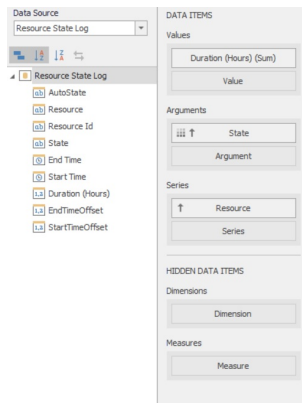


Change the log file on the left side to me Resource State Log. From the Dashboard ribbon click on Pies.

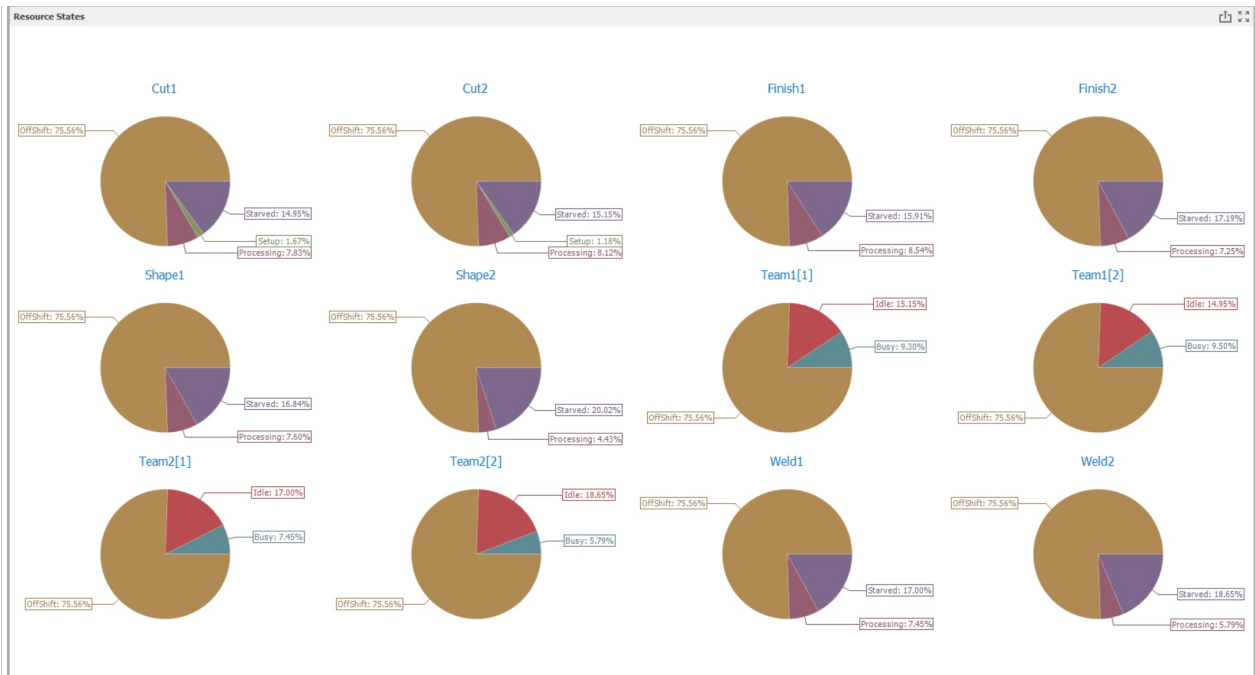
On the left side of the window, under Resource State Log, select the Duration item and drag it onto the Actual data item box. This is the value that will be displayed within each pie.

Drag the State item onto the Argument data item box. This identifies for each State, the Duration in that State will be displayed.

Drag the Resource item onto the Series data item. This will create a pie in the dashboard for each Resource item.



Right click in the Pie window and select Edit Names. Change the name from Pie 1 to Resource States. You should see something that looks like this.



You may also want to try using the Resource State Log and create a pivot table to display the actual values of the states and not just a percentage of the whole.

Experiment Dashboard Reports

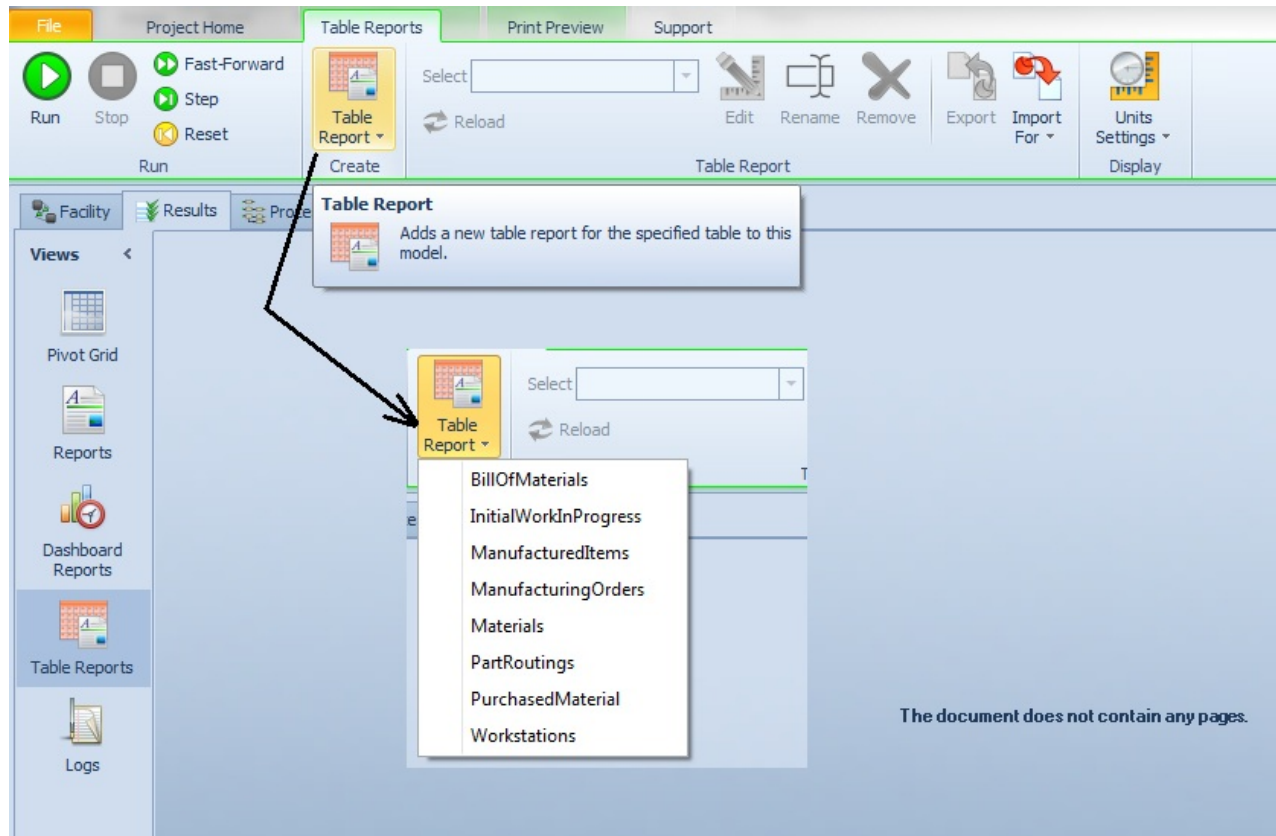
See the [Experiments](#) page, Experiment Dashboard Reports section, for more information on building a Dashboard Report within the Project Experiment.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Table Reports

The Table Reports window provides access to the [Report Designer](#) for creating reports using the data within the various Data Tables in Simio. All user licensing allows for access to previously developed table reports (Select pull-down list), however, the Create Table Report button, as displayed below, is only available to RPS users.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Logs

Logs

The Logs window includes detailed logs for the interactive run displayed in multiple tabbed windows.

For all Simio license types, this includes the **Material Usage Log**, [State Observation Log](#), [Tally Observation Log](#), Periodic Output Statistic Log, Periodic State Statistic Log, and Periodic Tally Statistic Log.

For Simio RPS Edition customers, this also includes the [Resource Usage Log](#), [Resource State Log](#), [Resource Capacity Log](#), [Constraint Log](#), [Transporter Usage Log](#), [Material Usage Log](#), Inventory Review Log, [Task Log](#), [Task State Log](#), Periodic Output Statistic Log, Periodic State Statistic Log, and Periodic Tally Statistic Log.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

State Observation Log

The State Observation Log displays the log of values over time of State Statistic elements which had *Log Observations* set to 'True'. There are twelve automatic columns that are provided with the log. Depending upon the type of state variable that is being tracked, data will appear in either the Value/Units, Rate/Units or Acceleration/Units fields. These include:

- **Time** - The simulation date and time that the state change occurred.
- **Object Type** - The type of object that includes the state statistic, which may be Model or another object.
- **Object Name** - The name of the object within the model that includes the state statistic.
- **Data Source** - The name of the state statistic being tracked or optional data source string as specified within state statistic element.
- **Category** - The category string value as specified in the state statistic element, which is UserSpecified by default.
- **Data Item** - The data item string value as specified in the state statistic element, which is StateValue by default.
- **Value** - The value of the state variable at the time specified.
- **Units** - The time units for the Value indicated.
- **Rate** - The rate for the level continuous-change state variable.
- **Units** - The time units for the Rate indicated.
- **Acceleration** - The acceleration for the level with acceleration continuous-change state variable.
- **Units** - The time units for the Acceleration indicated.

Time	Object Type	Object Name	Data Source	Category	Data Item	Value	Units	Rate	Units	Acceleration	Units
9/23/2013 12:00:00 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	0		0		0	
9/23/2013 12:00:00 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	1		0		0	
9/23/2013 12:00:00 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	2		0		0	
9/23/2013 12:00:17 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	1		0		0	
9/23/2013 12:00:23 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	2		0		0	
9/23/2013 12:00:23 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	3		0		0	
9/23/2013 12:00:31 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	2		0		0	
9/23/2013 12:00:33 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	3		0		0	
9/23/2013 12:00:33 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	4		0		0	
9/23/2013 12:00:38 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	5		0		0	
9/23/2013 12:00:38 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	6		0		0	
9/23/2013 12:00:44 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	5		0		0	
9/23/2013 12:00:52 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	6		0		0	
9/23/2013 12:00:52 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	7		0		0	
9/23/2013 12:00:56 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	6		0		0	
9/23/2013 12:01:05 AM	Model	Model	NumberInSystem_State	UserSpecified	StateValue	7		0		0	

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Tally Observation Log

The Tally Observation Log displays the log of values over time of Tally Statistic elements which had *Log Observations* set to 'True'. There are eight automatic columns that are provided with the log including:

- **Time** - The simulation date and time that a tally statistic observation was added.
- **Object Type** - The type of object that includes the tally statistic, which may be Model or another object.
- **Object Name** - The name of the object within the model that includes the tally statistic.
- **Data Source** - The name of the tally statistic being tracked or optional data source string as specified within tally statistic element.
- **Category** - The category string value as specified in the tally statistic element, which is UserSpecified by default.
- **Data Item** - The data item string value as specified in the tally statistic element, which is TallyValue by default.
- **Value** - The value of the tally statistic observation at the time specified.
- **Units** - The time units for the Value indicated.

With RPS edition, custom expression columns can be added to the log. These expressions can reference the Token's information that triggered the Tally step. This allows users to access the Token's Associate Object or Context Object references. Also Tables references could be used where the Token or the Tally Element itself has row references.

Time	Object Type	Object Name	Data Source	Category	Data Item	Value	Units
9/23/2013 12:00:17 AM	Model	Model	FlowTime	UserSpecified	TallyValue	0.293581424629643	Minutes
9/23/2013 12:00:31 AM	Model	Model	FlowTime	UserSpecified	TallyValue	0.527342431269617	Minutes
9/23/2013 12:00:44 AM	Model	Model	FlowTime	UserSpecified	TallyValue	0.344947750602105	Minutes
9/23/2013 12:00:56 AM	Model	Model	FlowTime	UserSpecified	TallyValue	0.552117402639884	Minutes
9/23/2013 12:01:12 AM	Model	Model	FlowTime	UserSpecified	TallyValue	0.65412981339366	Minutes
9/23/2013 12:01:21 AM	Model	Model	FlowTime	UserSpecified	TallyValue	0.791668676102233	Minutes
9/23/2013 12:01:35 AM	Model	Model	FlowTime	UserSpecified	TallyValue	0.94792741865533	Minutes
9/23/2013 12:01:48 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.16845109881255	Minutes
9/23/2013 12:01:56 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.07556009563727	Minutes
9/23/2013 12:02:04 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.20334927015304	Minutes
9/23/2013 12:02:13 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.13118947674869	Minutes
9/23/2013 12:02:22 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.28695651348907	Minutes
9/23/2013 12:02:37 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.25196437559051	Minutes
9/23/2013 12:02:51 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.48127662485154	Minutes
9/23/2013 12:02:59 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.3957815722294	Minutes
9/23/2013 12:03:06 AM	Model	Model	FlowTime	UserSpecified	TallyValue	1.50909924367223	Minutes

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Interactive And Experiment Results

Interactive Results Verses Experiment Results

Interactive results should not be used for making decisions or drawing conclusions. Simio allows the user to make changes to the model during the run in order to see how it will affect the system. This flexibility allows the user to see how the system might change with a modeling change while also acting as a helpful debugging tool. However, with this flexibility brings possible inaccuracy with the interactive model results. In order to draw conclusions from the model, a multi-replication experiment should be run and the results from this experiment can be considered valid.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Exporting

Exporting

Results data can be exported to a .csv file by clicking the Export Results icon in the ribbon menu when viewing the [Pivot Grid](#). If an experiment was run with more than one replication, the user has a choice of exporting the details or the summary. The detailed export will include data from all the replications. The summary export will only include the summary across all the replications.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Standard Object Library

Standard Object Library

Simio comes with a Standard Object Library that includes 15 pre-built object definitions that can be used to model a wide range of systems.

Name	Class	Description
Source	Fixed	Creates entities that arrive to the system.
Sink	Fixed	Destroys entities and records statistics.
Server	Fixed	Models a multi-channel service process with input/output queues.
Resource	Fixed	Models a resource that can be used by other objects.
Combiner	Fixed	Combines entities in batches.
Separator	Fixed	Separates entities from batches.
Vehicle	Transporter	Carries entities between Fixed objects.
Worker	Transporter	Carries entities between Fixed objects and processes entities at a fixed location.
Basic Node	Node	A simple intersection of Links.
Transfer Node	Node	An intersection where entities set destination and wait on Transporters.
Connector	Link	A zero-time connection between two Nodes.
Path	Link	A pathway between two Nodes where entities travel based on speed.
Time Path	Link	A pathway with a specified travel time.
Conveyor	Link	An accumulating/non-accumulating Conveyor device.
Workstation (Deprecated)	Fixed	Models a 3-phase workstation with setup, processing, and teardown.

Most library objects such as Entities, Sources, Sinks, and Servers have a Size property. This provides the relationship between animated size and the physical size in the model (when needed). Nodes are an exception and do not have any physical space in the model. There is no travel time required to pass through a node (although time could be required to execute any logic specified in a node).

In the Standard Library, the processing time delays for the Server, Combiner, and Separator objects are now marked as interruptible. The process in these objects performing the processing delay activity is named 'OnEnteredProcessing'. Thus, to interrupt processing activity at a Server, Combiner, or Separator, you may now use the [Interrupt step](#) to interrupt the 'OnEnteredProcessing' process of the target object.

Right clicking on an object in the Standard Library allows the user to subclass it into their own [Project Library](#). Right clicking on an existing object within the Facility window also provides an option to 'Create Object From This' which will subclass the object and include default data. See the [Creating New Objects](#) page for more information.

To view all the Functions available for these Intelligent Objects, see the [Functions](#) help page.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

BasicNode

A **Basic node** is a simple node to support connection between links. Most objects that support incoming connections have an associated (embedded) basic node. Many times, Basic nodes are used as starting and/or ending points of links. They may also be used as intersection points for vehicle networks.

Because transporters commonly travel through nodes, the state assignments functionality provided by the BasicNode and TransferNode objects make it easy for a user to distinguish between entity or transporter state assignments by providing an *Assign If* condition option. Note that these available assign condition type choices (Custom Condition, Entity Entering, No Condition, Transporter Entering) are also available within the tally condition choices if defining a tally statistic using the node's Tally Statistics -> On Entering repeat group.

The [Add-On Process Triggers](#) allow additional logic to be specified when using the Basic Node.

Listed below are the properties of the **Basic Node**:

Property	Valid Entry	Description
Initial Traveler Capacity	Expression	The initial maximum number of travelers that may simultaneously occupy this node.
Entry Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank entry into this node among competing travelers.
Entry Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' entry ranking rule.
Outbound Travel Mode	Continue, Free Space Only, Network Only, Network If Possible	Indicates whether to assign a new travel mode to entities attempting to transfer from this node to either free space or an outbound link. If the value is 'Free Space Only', then the <i>Outbound Link Preference</i> and <i>Outbound Link Rule</i> properties of the node will not be displayed.
Outbound Link Preference	Any, Available, Specific	The preference used by a traveler to select an outbound link from this node to its next destination.
Outbound Link Rule	Shortest Path, By Link Weight	The rule used by an entity to select the outbound link from this node to its destination node (if a destination node has been set).
Outbound Link Name	Link Name	The name of the specific outbound link that will be selected by the traveler.
On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the BasicNode object.
Assign If (State Assignments)	Custom Condition, Entity Entering, No Condition, Transporter	Condition required to perform the assignment.

Entering		
Condition (State Assignments)	Expression	Condition required to perform the assignment when <i>Assign If</i> is 'Custom Condition'.
State Variable Name (State Assignments)	State Name	The name of the state that will be assigned a new value.
New Value (State Assignments)	Expression	The new value to assign.
On Entering (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity enters the node object.
On Exited (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity has exited the node object.
Tally If (Tally Statistics)	Custom Condition, Entity Entering, No Condition, Transporter Entering	The condition required to record the tally statistic.
Tally Statistic Name (Tally Statistics)	TallyStatistic Element Name	The tally statistic for which the value is to be recorded.
Value Type (Tally Statistics)	Expression, TimeBetween	The type of value to record. The value type 'Expression' records the value of the specified expression. The value type 'TimeBetween' records the time between arrivals to Tally steps referencing the tally statistic.
Value (Tally Statistics)	Expression	The expression value to be recorded.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs when a traveling entity's leading edge has entered this node's crossing point.
Exited Add-On Process	Process Instance Name	Occurs when a traveling entity has departed this node's crossing point.
Sequence Expected Operation	Expression	The expression used to get a deterministic estimation of the operation time if this node is in the destination sequence of an entity's assigned sequence table.

Time

Branching Random Number Stream	Expression	The random number stream to be used if the entity is transferring from the node to an outbound link and is probabilistically selecting the outbound link using the node's 'By Link Weight' outbound link rule.
Bound External Output Node	Node Instance Name	The name of an external output node that this node has been bound to in order to transfer entities out of the parent object. The transfer attempt will be performed by an entity immediately upon entering the node and, if successful, all other entry, crossing and routing logic for the node will be ignored.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

BasicNode - Discussion and Examples

-->

Discussion

Which Path will the Entity Take?

- *Outbound Link Preference*

If the preference is 'Any', then the specified *Outbound Link Rule* will be strictly adhered to and blocked outbound links may cause the traveler to wait.

If the preference is 'Available', then the specified *Outbound Link Rule* will be automatically adjusted to minimize waiting time by first applying the rule only to outbound links that are immediately available for entry. If no outbound links are immediately available, then the preference will revert to 'Any' selection.

Note that a special case of the 'Available' preference is applied when used with a 'By Link Weight' rule and '0' link selection weights. Under those conditions, an outbound link with a '0' selection will be selected if it is the only one available. This technique can be useful for modeling primary and secondary flow (such as a backup path) where the secondary path is only used when the primary path is unable to accept new entries.

If the preference is 'Specific', then a traveler will select the specified *Outbound Link Name*. Typically, if using this feature, the user may specify the *Outbound Link Name* as a link reference state variable. Examples may include using a global 'Switch' state variable that is being dynamically assigned to 'switch' outbound flow direction from the node according to some designed control system logic. Alternatively, each entity may be executing decision logic (using steps such as Find, Search, or Decide) to do dynamically determine which outbound link it wants to select, storing that selected link in a state variable, and then routing out from the node using that selected link. Yet another example is having a property or state variable on an entity that is specifying which outbound link to use. This property might be based on entity type, or perhaps a variable that has been dynamically assigned based on where the entity is in its processing sequence or if it is 'rework' etc.

- *If the Outbound Link Rule is 'Shortest Path'*

If the entity has a destination set then this rule will choose the shortest path to the entity's set destination. If the entity does not have a destination set, then the 'Shortest Path' rule is not applicable and instead the 'By Link Weight' rule will be assumed.

- *If the Outbound Link Rule is 'By Link Weight'*

The *Selection Weight* of each outbound link will be evaluated and the probability of an outbound link being selected is then based on its proportional selection weight. For example, if there are two links, and the first has a *Selection Weight* of '.75' and the second a *Selection Weight* of '1.5', then the first link has a 33.33% chance of being selected, and the second link a 66.66% chance.

Entity Destination

- *Automatic destination is to continue*

From the basic node, the entity will simply continue down the appropriate path leaving this node, as determined by the rules mentioned above. If a [SetNode](#) step was used in the Processes window, this entity is instead going to a specific node.

- *** If the entity's destination is not reachable*

If the entity's destination is not reachable because there is no physical link between the basic node and the destination node, the entity will automatically be destroyed. For example, if there are no paths from the node leading out or no path to the specific destination specified (SetNode), the entity will be automatically destroyed. If the [Warning Level](#) is set to either Alert User or Write to Trace Only, then the user will receive a warning that this has occurred. In some cases, users may intend for this to occur, in which case, they may disable the warnings.

TransferNode

A **Transfer Node** is a more comprehensive node that supports connection to paths as well as the ability to select destination, path, and transfer device. Most objects that support outgoing connections have an associated (embedded) transfer node.

Because transporters commonly travel through nodes, the state assignments functionality provided by the BasicNode and TransferNode objects make it easy for a user to distinguish between entity or transporter state assignments by providing an *Assign If* condition option. Note that these available assign condition type choices (Custom Condition, Entity Entering, No Condition, Transporter Entering) are also available within the tally condition choices if defining a tally statistic using the node's Tally Statistics -> On Entering repeat group.

When *Entity Destination Type* is 'Select From List' or 'Custom Routing Group', the *Route Constraint Logic* repeat group is provided to support imposing additional constraints on an entity's route request by referencing [Constraint Logic](#) elements. This replaces the *Required Materials* repeat group which is deprecated as of Sprint 203. Existing models can access the *Required Materials* properties by using File, Settings and under the GUI area, setting the *Display Deprecated Properties In Properties Window* to 'True'.

The [Add-On Process Triggers](#) allow additional logic to be specified when using the Transfer Node.

For more information, see [Transfer Node - Discussion and Examples](#).

Listed below are the properties of the **Transfer Node**:

Property	Valid Entry	Description
Initial Traveler Capacity	Expression	The initial maximum number of travelers that may simultaneously occupy this node.
Entry Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank entry into this node among competing travelers.
Entry Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' entry ranking rule.
Outbound Travel Mode	Continue, Free Space Only, Network Only, Network If Possible	Indicates whether to assign a new travel mode to entities attempting to transfer from this node to either free space or an outbound link. If the value is 'Free Space Only', then the <i>Outbound Link Preference</i> and <i>Outbound Link Rule</i> properties of the node will not be displayed.
Outbound Link Preference	Any, Available, Specific	The preference used by a traveler to select an outbound link from this node to its next destination.
Outbound Link Rule	Shortest Path, By Link Weight	The rule used by a traveler to select an outbound link from this node to its next destination. Note that if this rule is set to 'Shortest Path', and the traveler does not have an assigned destination, then the 'By Link Weight' rule is assumed.
Outbound Link Name	Link Name	The name of the specific outbound link that will be selected by the traveler.

Entity Destination Type	Continue, Specific, By Sequence, Select From List, Use Custom Routing Group, None (Destroy Entity)	The method used to set a destination node for entities departing this node. 'By Sequence' requires that the entity object has been assigned a sequence table. The destination node value will be set to the next destination in the sequence.
Node Name	Node Instance name	The name of the specific node object to set the destination value to.
Node List Name	Node List Name or Table Name	The name of the list of node objects from which a destination will be selected. This may be a list name or table reference.
Routing Group Name	Routing Group Name	The name of the Routing Group element that will be used to route to a destination selected from a list of candidate nodes. Go to Definitions > Elements to add a new Routing Group to the model.
Selection Goal	Smallest Distance, Largest Distance, Preferred Order, Cyclic, Random, Smallest Value, Largest Value	The goal used to select a destination node from the list of candidates.
Value Expression	Expression	The expression criteria, evaluated for each candidate destination, used with a Smallest Value or Largest Value selection goal. Use the keyword Candidate at the beginning of the expression.
Selection Condition	Expression	An optional condition evaluated for each candidate destination that must be true for the destination node to be selected. In the condition, use the keyword 'Candidate' to reference a node object in the collection of candidates (e.g. Candidate.Node.AssociatedStationOverload)
Blocked Destination Rule	Select Any, Select Available Only, Prefer Available	The rule used to manage destination selection if there are destinations in the list of candidate nodes that are considered to be blocked. If the rule is 'Select Any', then whether or not destinations are considered blocked will be ignored. If the rule is 'Select Available Only', then only a destination that is not currently blocked may be assigned, and if all destinations are blocked, then the token will be held at the Route step until a destination becomes available. If the rule is 'Prefer Available', then a destination that is not currently blocked will be preferred, but if all destinations are blocked, then the best destination per the selection goal will still be immediately assigned.
Route Constraint Logic	Repeat Group, Constraint Logic elements	Constraint logic used to enforce additional constraints on an entity's route request.
Route	Constraint	The name of the Constraint Logic element used to enforce additional

Constraint Logic.Constraint Logic Name	Logic elements	constraints on an entity's route request.
Route Request Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static rule used to rank competing entities waiting to be assigned an available destination. Available when <i>Entity Destination Type</i> is 'Select From List' only.
Route Request Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' ranking rule.
Route Request Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	The rule used to dynamically select from competing requests waiting to be assigned an available destination.
Route Constraint Logic	Constraint Logic used to enforce additional constraints on an entity's route request.	
Ride On Transporter	Always, Conditional, Never	Indicates whether an entity must ride a transporter to exit the transfer node.
Transporter Type	Specific, From List, Any	The transporters that may be selected to ride on. 'On Same Network' indicates all transporters that are currently on the same network as the entity.
Transporter Name	Transporter Instance Name	The name of the specific transporter type that the entity may ride on.
Transporter List Name	Transporter List Name	The name of the list of transporter types that the entity may ride on.
Reservation Method	Reserve Closest, Reserve Best, First Available At Location	The method used to select and reserve a transporter object to ride on. 'ReserveClosest' will find the transporter that is closest on a Network or physically closest (if no network) to the TransferNode. If there are two that are of equal distance, the node's <i>Selection Goal</i> property will be used to determine which transporter to select. 'ReserveBest' simply looks to the <i>Selection Goal</i> and <i>Selection Criteria</i> properties and finds the Transporter that best fits the goal, regardless of the vehicle's current location. The 'FirstAvailableAtLocation' will cause the entities to wait for a Transporter to arrive to the node. This method should only be used when the vehicle has a <i>Routing Type</i> of 'Fixed Route' or when the user specifically moves the vehicle to an entity location. Otherwise, this method doesn't request the vehicle to move to the entity location.
Selection Goal	Preferred Order, Smallest	The goal used to rank transporter preference when multiple candidates are available to ride on.

	Value, Largest Value	
Value Expression	Expression	The expression criteria, evaluated for each candidate transporter object, used with a Smallest Value or Largest Value selection goal.
Selection Condition	Expression	An optional condition evaluated for each candidate transporter object, and which must be true for the transporter to be eligible to ride on.
Selection Condition	Expression	An optional condition evaluated for each candidate transporter object, and which must be true for the transporter to be eligible to ride on.
Keep Reserved If	Expression	<p>Optional condition indicating whether, on drop-off, to keep the transporter resource reserved for possible later reuse by the same entity. Other entities will be unable to use the transporter for processing tasks or new ride pickup requests unless the reservation is cancelled.</p> <p>This condition is evaluated when the entity is picked up at the TransferNode. In the expression, use the syntax <code>Entity.CurrentTransporter.[Attribute]</code> to reference an attribute of the selected transporter (e.g., <code>Entity.CurrentTransporter.Is.Vehicle1</code>).</p> <p>Note that when an entity is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p>
On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the TransferNode object.
Assign If (State Assignments)	Custom Condition, Entity Entering, No Condition, Transporter Entering	Condition required to perform the assignment.
Condition (State Assignments)	Expression	Condition required to perform the assignment when <i>Assign If</i> is 'Custom Condition'.
State Variable Name (State Assignments)	State Name	The name of the state that will be assigned a new value.
New Value (State Assignments)	Expression	The new value to assign.
On Entering (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity enters the node object.
On Exited (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity has exited the node object.
Tally If (Tally Statistics)	Custom Condition, Entity Entering, No	The condition required to record the tally statistic.

	Condition, Transporter Entering	
Tally Statistic Name (Tally Statistics)	TallyStatistic Element Name	The tally statistic for which the value is to be recorded.
Value Type (Tally Statistics)	Expression, TimeBetween	The type of value to record. The value type 'Expression' records the value of the specified expression. The value type 'TimeBetween' records the time between arrivals to Tally steps referencing the tally statistic.
Value (Tally Statistics)	Expression	The expression value to be recorded.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs when a travelling Entity or Transporter's leading edge has entered this node's crossing point.
Exited Add-On Process	Process Instance Name	Occurs when a travelling Entity or Transporter has exited this node's crossing point.
Sequence Expected Operation Time	Expression	The expression used to get a deterministic estimation of the operation time if this node is in the destination sequence of an entity's assigned sequence table.
Branching Random Number Stream	Expression	The random number stream to be used if the entity is transferring from the node to an outbound link and is probabilistically selecting the outbound link using the node's 'By Link Weight' outbound link rule.
Bound External Output Node	Node Instance Name	The name of an external output node that this node has been bound to in order to transfer entities out of the parent object. The transfer attempt will be performed by an entity immediately upon entering the node and, if successful, all other entry, crossing and routing logic for the node will be ignored.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.
Required Materials	Repeat Group	Additional material availability constraints enforced on an entity's route request. This repeat group has been deprecated. Use the Routing Logic, Other Routing Options, Route Constraint Logic property instead. To edit the deprecated repeat group, go to Files, Settings and within the GUI area, set the <i>Display Deprecated Properties in Properties Window</i> to 'True'.

TransferNode - Discussion and Examples

Discussion

Which Path will the Entity Take?

- *Outbound Link Preference*

If the preference is 'Any', then the specified *Outbound Link Rule* will be strictly adhered to. A link is immediately selected and selection of a blocked outbound link may cause the traveler to wait in the *EntryQueue* of the selected path. If an alternate link subsequently becomes available, the entity will remain in the queue for the originally selected link, unless the user supplies custom logic to remove it and reselect the newly available link.

If the preference is 'Available', then the specified *Outbound Link Rule* will be automatically adjusted to minimize waiting time by first applying the rule only to outbound links that are immediately available for entry. If no outbound links are immediately available, then the preference will revert to 'Any' selection.

Note that a special case of the 'Available' preference is applied when used with a 'By Link Weight' rule and '0' link selection weights. Under those conditions, an outbound link with a '0' selection weight will be selected if it is the only one available. This technique can be useful for modeling primary and secondary flow (such as a backup path) where the secondary path is only used when the primary path is unable to accept new entries.

If the preference is 'Specific', then a traveler will select the specified *Outbound Link Name*. Typically, if using this feature, the user may specify the *Outbound Link Name* as a link reference state variable. Examples may include using a global 'Switch' state variable that is being dynamically assigned to 'switch' outbound flow direction from the node according to some designed control system logic. Alternatively, each entity may be executing decision logic (using steps such as Find, Search, or Decide) to do dynamically determine which outbound link it wants to select, storing that selected link in a state variable, and then routing out from the node using that selected link. Yet another example is having a property or state variable on an entity that is specifying which outbound link to use. This property might be based on entity type, or perhaps a variable that has been dynamically assigned based on where the entity is in its processing sequence or if it is 'rework' etc.

- *If the OutboundLinkRule is 'Shortest Path'*

If the entity has a destination set then this rule will choose the shortest path to the entity's set destination. If the entity does not have a destination set, then the 'Shortest Path' rule is not applicable and instead the 'By Link Weight' rule will be assumed.

Note: when using 'Shortest Path', if Links between the Transfer Nodes are Connectors, one path at random will be selected at the beginning of the run as Connectors all have the same logical length of zero. The same is true if there are multiple Paths of the same length.

If there is only one outbound Link, the entity will choose to follow this Link without running the shortest path calculation. When the entity must choose between multiple outbound links, the shortest path calculation will be performed. To successfully calculate a shortest path, a viable route of Links and Nodes must exist between the entity's current Node and destination Node. Objects connected to or in-between Links and Nodes, such as a Server for example, are not included in a Network and therefore not recognized as being a part of a viable route. If the shortest path routine cannot find a viable path, a warning will be issued.

- *If the OutboundLinkRule is 'By Link Weight'*

The *Selection Weight* of each outbound link will be evaluated and the probability of an outbound link being selected is then based on its proportional selection weight. For example, if there are two links, and the first has a *Selection Weight* of '.75' and the second a *Selection Weight* of '1.5', then the first link has a 33.33% chance of being selected, and the second link a 66.66% chance.

Entity Destination Type

- *If the Entity Destination Type is 'Continue'*

The entity will simply continue down the appropriate path leaving this node, as determined by the rules mentioned above. If a [SetNode](#) step was used in the Processes Window, this entity is indeed going to a specific node but the entity destination here can be set to Continue.

- *If the Entity Destination Type is 'By Sequence'*

This entity's destination is determined by a Sequence Table. A [SetRow](#) step is needed to set which [Sequence table](#)

should be used and when the entity departs this transfer node, the next destination in the sequence is used to determine where this entity will travel to.

- *If the Entity Destination Type is 'Specific'*

This entity will travel to a specific destination that is set in the *Node Name* property. This property will appear once *Specific* is selected. The user can select one node from the drop down of *Node Name*.

- *If the Entity Destination Type is 'None (Destroy Entity)'*

The entity will simply be destroyed and no warning will be displayed.

- *If the Entity Destination Type is 'Select From List'*

The destination is selected from a Node List. A [Node List](#) can be created in the [List panel](#), which can be found by clicking on the [Definitions](#) tab. Node lists can alternatively be defined within a table (see the [Using a Table Column as a List](#) section within the [Data Tables](#) page). If 'Select from List' is selected, the *Selection Goal*, *Selection Condition* and *Blocked Destination Rule* properties appear. The *Selection Goal* determines how Simio chooses the destination node off the specified list. If *Smallest Value* or *Largest Value* is selected for the *Selection Goal*, the *Selection Expression* property appears so the user can specify what expression should be evaluated to determine the Smallest Value or the Largest Value.

The *Selection Condition* property is an optional condition evaluated for each candidate destination that must be true for the destination node to be selected. In the expression, use the syntax 'Candidate.[NodeClass].[Attribute]' or 'Candidate.[NodeAssociatedObjectClass].[Attribute]' to evaluate attributes of either the candidate destination nodes themselves or objects associated with those candidate nodes. For example, use syntax such as 'Candidate.Node.AssociatedStation.Capacity' or 'Candidate.Server.Capacity' (which would be acceptable syntax if all the candidate nodes were associated with Server objects). Data tables may also be used within the *Selection Condition* using the form 'Candidate.TableName.ColumnName'. Simio will evaluate the data table row references associated with the candidate nodes in the list. If the candidate nodes don't have a data table row reference, Simio will evaluate the associated object (Server or other associated fixed object) data table row reference.

The *Selection Expression* property is a [candidate reference expression](#) and therefore the keyword 'Candidate' must be used. The default value for this expression is `Candidate.Node.AssociatedStationOverload`, which is defined as the 'load' on the input location minus the capacity of the input location. In this expression, use the syntax 'Candidate.[NodeClass].[Attribute]' or 'Candidate.[NodeAssociatedObjectClass].[Attribute]' to evaluate attributes of either the candidate destination nodes themselves or objects associated with those candidate nodes.

The *Blocked Destination Rule* property is used to manage destination selection if there are destinations in the list of candidate nodes that are considered to be blocked. 'Select Any' will send the entity on its determined route regardless of the whether or not any destinations are considered blocked. 'Select Available Only' will only assign a destination that is not currently blocked and if all destinations are blocked, then the entity will be held at the Transfer Node until a destination becomes available. If the rule is 'Prefer Available', then a destination that is not currently blocked will be preferred, but if all destinations are blocked, then the best destination per the selection goal will be immediately assigned.

*For any of the above blocked destination rules, the *Selection Condition* must always evaluate to 'True' for a destination (whether considered blocked or unblocked) to be selected (i.e., that condition always filters the list first before applying *Selection Goal* or *Blocked Destination Rule* logic). Additionally, a destination is considered blocked in the node list specified in the TransferNode by evaluation of the following expression:

`Candidate.Node.AssociatedStation != Nothing & & Candidate.Node.AssociatedStationHasSpaceFor(Entity) == False`. If it evaluates to 'True', a candidate destination in the node list is considered 'blocked'.

- *If the Entity Destination Type is 'Use Custom Routing Group'*

The destination is determined based on custom [RoutingGroup](#) element within the model. The [RoutingGroup](#) element then specifies the [Node List](#) to be used. If 'Use Custom Routing Group' is selected, the *Routing Group Name* will be specified. This advanced option allows for multiple transfer nodes within a model to reference the same [RoutingGroup](#) element.

As with the 'Select From List' option discussed above, the *Selection Goal*, *Selection Condition* and *Blocked Destination Rule* properties appear. Refer to the notes above on specifics of these properties.

The 'Use Custom Routing Group' option is an advanced option that may be used for the following scenarios. For example, if a user is modeling blocked routing behavior and has entities waiting at multiple TransferNode objects that are trying to route to a destination selected from a common node list, and a common set of decision rules should be applied to all of those entities (e.g., entities at one location perhaps prioritized over entities at another location when a destination becomes available, etc.), then this option is available to use.

Alternatively, if a user is modeling blocked routing and wants more advanced control over how a waiting entity is selected when a destination becomes available (e.g., using a static ranking rule for the RouteRequestQueue that is not FIFO or using a dynamic selection rule for more advanced 'pulling' of entities), then this option is available to use.

Another example for this option may be if a user wants to customize the conditional expression that indicates whether a destination node in the list of candidate nodes is considered 'blocked, rather than using Simio's default condition, then this option is available to use.

- *What if a Selected Link Does Not Directly Connect to the Entity Destination?*

It is possible to select a link that does not immediately connect to the specified destination. For example, you might specify an entity destination of Server5 but directly or indirectly select a link that connects to Server1.

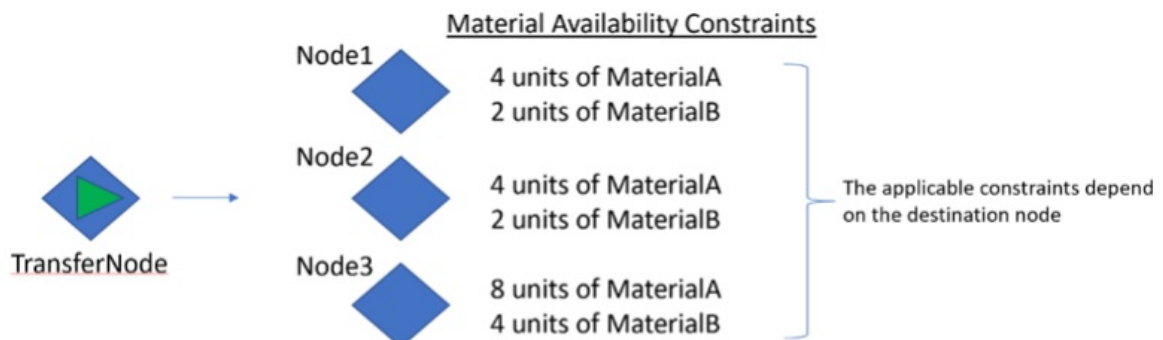
- *Arriving a node:* If an entity arrives at an object that is not its current destination, and there is no outbound link connecting to the current destination, it will enter the object -- the assumption is that an outbound link from that object will directly or indirectly connect to the entity's destination.
- *Departing a node:* When there are multiple outbound links and Simio must choose a link based on the destination (e.g. using the Shortest Path rule) then a direct path via links connected by nodes must be available leading to the specified entity destination. Otherwise a warning will be reported and the entity will be destroyed. Note: If the [Warning Level](#) is set to either Alert User or Write to Trace Only, then the user will receive a warning that this has occurred. In some cases, users may intend for this to occur, in which case, they may disable the warnings.

If links are selected by other means (e.g. By Link Weight or when there is only 1 outbound link) then no warning will be reported and the selected link will be followed.

Other Routing Out Options

- When the *Entity Destination Type* is 'Select From List' or 'Use Custom Routing Group', the *Route Constraint Logic* properties will be available. This repeating editor is an optional list of Constraint Logic element requirements for material, resource and other constraints before allowing it to be routed.

Below is simple example of imposing material availability constraints on an entity route request. An entity waiting at a transfer node is requesting to be routed to one of three possible destinations (node1, node2, or node3). Besides destination availability, there are also material availability constraints that must be satisfied before the entity may be routed, with the applicable constraints depending on the destination node assignment.



Route Constraint Logic -> Constraints

Constraint Type	Material Name	Quantity	Requested Item Condition
Material Availability	MaterialA	4	Candidate.Node.Is.Node1 Candidate.Node.Is.Node2
Material Availability	MaterialB	2	Candidate.Node.Is.Node1 Candidate.Node.Is.Node2
Material Availability	MaterialA	8	Candidate.Node.Is.Node3
Material Availability	MaterialB	4	Candidate.Node.Is.Node3

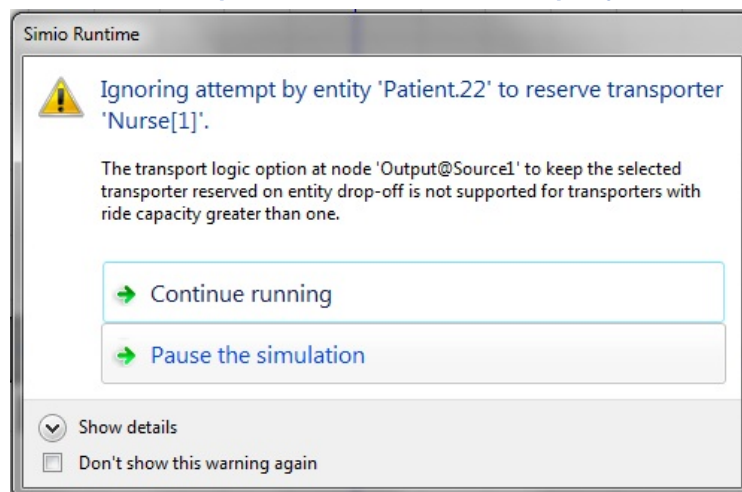
- When the *Entity Destination Type* is 'Select From List', this section of properties also includes static and dynamic routing options. This includes the *Route Request Ranking Rule* which is the static rule used to rank competing entities waiting to be assigned an available destination, as well as the *Route Request Dynamic Selection Rule* which is the rule used to dynamically select from competing requests waiting to be assigned an available destination. See the [Dynamic Selection Rules](#) for more information.

Entities that will travel on Transporters

- Transfer Nodes are used to specify whether or not the entities that enter this node will need to ride on transporters for travel out of this node. The first property that needs to be set to indicate that entities require a transporter is the *Ride On Transporter* property. It should be set to 'Always'. Setting the *Ride On Transporter* property to 'Conditional' will allow you to specify a condition that when true will require a Transporter to ride on. This will cause five other properties to be displayed in the Properties window. The *Transporter Type* can either be *Specific* or *From a List*. If *From a List* is selected, a Transporter List must be defined in the [List panel](#), which is found by clicking on the [Definitions](#) tab. The optional property *Selection Condition* can be used in conjunction with selecting from a list. The Transporter will only be selected from the list if it meets the criteria of the *Selection Condition*. If *Specific* is selected, a transporter must be part of the model. For example, the standard library object [Vehicle](#) must be in the [Facility Window](#) before it will appear in the Transporter Name property.
- The *Reservation Method* property is used to determine which transporter will be selected, if there is more than one available. *Reserve Closest* will find the transporter that is physically closest to the transfer node. If there are two that are of equal distance, the *Selection Goal* property will be used to determine which transporter to select. *Reserve Best* simply looks to the *Selection Goal* and *Selection Criteria* and finds the transporter that best fits the goal. The *First Available at Location* will cause the entities to wait for a transporter to move itself to the node.
- The *Selection Goal* property is used to rank transporter preference when there are multiple candidates available to ride on. *Preferred Order* will select in the member order of the object list or population. So if the user is selecting from a list, the transporters will be selected in the order that they are listed in the transporter list. If the user is selecting a specific transporter, preferred order will be used to select from the population of transporters, if there are more than one of that specific type in the system. When *Smallest Value* or *Largest Value* is selected, the property *Selection Expression* appears so the user can specify what should be evaluated for the Smallest value or the Largest value. Since [candidate](#) objects (such as transporters) might be used in this expression, use the keyword 'Candidate' to reference a transporter object in the collection of candidates (e.g. Candidate.Transporter.RemainingRideCapacity).
- The *Selection Condition* is an optional property that is evaluated for each candidate transporter and it must be true for that transporter to be eligible to ride on. The keyword Candidate must also be used in this expression if a transporter is referenced.
- The *Keep Reserved If* is an optional property that will determine if the transporter, upon dropping off the entity at its destination location, will be 'reserved' for possible later use. This can be used in cases where the worker/vehicle may also then be used for processing at the next step after dropoff (at a Server, for example). It may also be useful if the same transporter is used to move an entity from one location to another. It can have an expression for the *Keep Reserved If* property that results in 'True' (1) and then will reserve that same transporter for the next travel occurrence. Refer to the [Reservations](#) page for information on reserving a resource.

It is important to note that if a transporter has a ride capacity greater than 1 (default), the transporter will not be reserved by the entity(s) riding on the transporter and a runtime warning will be displayed (see below).

Transfer Node - 'Keep Reserved If' True with Ride Capacity Greater than 1



Source

A **Source** is an object that allows the creation of entities at a specified rate, by a specified arrival pattern, or the firing of an event. The Source has an Output Buffer where entities can wait before leaving the Source via a Node. From the output node the entities leave on the outgoing link(s) to wherever the designed logic is going to send them. The [Balking and Reneging Options](#) page provides additional information and examples on capacities, balking and reneging for the output buffer.

[State assignments](#) can be made before exiting, on balking or on reneging the Source. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Source.

For more information, see [Source - Discussion and Examples](#).

Listed below are the properties of the **Source**:

Property	Valid Entry	Description
Entity Type	Entity Instance Name	The type of entities created by this Source object.
Arrival Mode	Interarrival Time, Time Varying Arrival Rate, On Event, Arrival Table	The mode used by this Source object to automatically generate a stream of entity arrivals.
Time Offset	Expression	The time offset until the first arrival.
Interarrival Time	Expression	The time interval between two successive arrivals. This property may be specified using a random sample from a distribution (typically the exponential distribution).
Rate Table	Rate Table Instance	The rate table that defines how the arrival rate changes over time (arrival process is a non-stationary poisson).
Rate Scale Factor	Expression	The factor used to scale the arrival rate values specified in the rate table.
Initial Number Entities	Expression	An initial number of entities to be created by this Source object at the beginning of the simulation run, in addition to entities created by the 'On Event' arrival mode. Note that this initial entity creation is not included in (counted against) the arrival mode's Maximum Arrivals.
Event Name	Event Instance Name	The name of the event which causes entity arrivals at this Source object.
Event Count	Expression truncated to integer	The number of occurrences of Event Name required to trigger the next entity arrival at this Source object.
Arrival Time Property	Numeric Table Property (column)	The name of the numeric table property (i.e., table column) that defines the list of scheduled times when arrival events are expected to occur. The number of arrival event occurrences at each scheduled time is determined by the Arrival Events Per Time Slot property. The number of entities created by each arrival event (i.e., the arrival 'batch size') is then determined by the Entities Per Arrival property. When an arrival event occurs, a reference to the table row holding the arrival time will be

		automatically assigned to any individual entities created by the event.
Arrival Events Per Time Slot	Expression	The number of arrival events expected to occur at each arrival time specified in the Arrival Time property. Upon each arrival event occurrence, the number of entities created (i.e., the arrival 'batch size') will be determined by the Entities Per Arrival property. Note that the actual timing of an arrival event may deviate from the expected schedule if the Arrival Time Deviation and Arrival No-Show Probability options are used.
Arrival Time Deviation	Expression	Expression used to model differences (typically random) between the scheduled times in the arrival table and the times during the simulation run that the arrivals actually occur. This expression may return a negative or positive value. If a negative deviation is returned, then the actual arrival time will occur earlier than the scheduled time by that duration. If a positive deviation value is returned, then the actual arrival time will occur later than the scheduled time by that duration. Note that this feature is automatically disabled if random sampling in the simulation is disabled.
No-Show Probability	Expression	The probability that a scheduled arrival in the arrival table will be a 'no-show' and thus not actually occur. Enter the chance of a no-show as a value between 0 and 1 (including values of 0 or 1). Note that this feature is automatically disabled if random sampling in the simulation is disabled.
Entities Per Arrival	Expression Truncated to Integer	The number of entities that will be created by this Source object for each arrival event.
Repeating Arrival Pattern	True or False	Indicates if the pattern of arrival times specified in the Arrival Time property should be repeated when the end of the pattern is reached.
Maximum Arrivals	Expression truncated to integer	The maximum number of arrivals to be automatically generated by this Source object.
Maximum Time	Expression	The time duration from the beginning of the simulation run after which this Source object will stop generating arrivals.
Stop Event Name	Event	The name of an event which will cause this Source object to stop generating arrivals.
Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Source object's output buffer.
Balk Decision Type (Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Reneged Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.

Trigger Type.Renege Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Renege Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Renege Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Renege Decision Type .Renege Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Renege Condition Or Probability.Renege Triggers	Expression Specified if the <i>Renege Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renegeing as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Renege Node Name.Renege Triggers	Node object reference	Facility node location to send a renegeing entity.
Trigger Start Boundary.Renege Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
Before Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is renegeing from an input or output buffer of the object.
Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Action Type (Table Row Referencing - Before Creating	Reference Existing Row, Add New Row	The type of table row reference action to perform. The 'Reference Existing Row' action type may be used to set a reference to an existing table row in a data table or sequence

Before Creating
Entities)

table. The row index into the table is specified by the *Row Number* property.
The 'Add New Row' action type may be used to add a new row to an output table. A reference will be automatically set to the newly added row.

Table Name (Table
Row Referencing -
Before Creating
Entities)

Table Name

The name of the table.

Row Number (Table
Row Referencing -
Before Creating
Entities)

Expression

The one-based row index into the table.

Action Type (Table
Row Referencing - On
Created Entity)

Reference Existing
Row, Add New Row

The type of table row reference action to perform.
The 'Reference Existing Row' action type may be used to set a reference to an existing table row in a data table or sequence table. The row index into the table is specified by the *Row Number* property.
The 'Add New Row' action type may be used to add a new row to an output table. A reference will be automatically set to the newly added row.

Table Name (Table
Row Referencing - On
Created Entity)

Table Name

The name of the table.

Row Number (Table
Row Referencing - On
Created Entity)

Expression

The one-based row index into the table.

Parent Cost Center

Cost Center Name

The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.

Capital Cost

Expression

The initial one-time setup cost to add this object to the system.

Cost Per Use (Output
Buffer Costs)

Expression

The cost to hold an entity in this buffer irrespective of the waiting time.

Holding Cost Rate
(Output Buffer Costs)

Expression

The cost per unit time to hold an entity in this buffer.

Run Initialized Add-
On Process

Process Instance
Name

Occurs when the simulation run is initialized.

Run Ending Add-On
Process

Process Instance
Name

Occurs when the simulation run is ending.

Creating Entity Add-
On Process

Process Instance
Name

Occurs when this Source object is about to create an arrival of one or more entities.

Created Entities Add-
On Process

Process Instance
Name

Occurs when an entity has been created by this Source object.

Exited Add-On
Process

Process Instance
Name

Occurs when an entity has exited this Source object.

Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out Condition	Expression	The condition required to allow an entity to transfer out of this fixed object via an external output node.
Stock Requirements	Material stock putaway or removal requirements for each created entity.	
Stock Requirements.ID Number	Option ID number for the stock requirement.	
Stock Requirements.Material Name	The name of the material to putaway or remove.	
Stock Requirements.Quantity	The quantity to putaway or remove.	
Stock Requirements.Skip Stock Requirement Condition	Optional condition indicating whether to skip the stock requirement.	
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.
Report Gantt	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Source - Discussion and Examples

Discussion

How Often Will the Entities Arrive?

- *If the Arrival Mode is 'Interarrival Time'*

Using the Interarrival Time option requires two additional properties including Time Offset and Interarrival Time. Specifying a Time Offset greater than 0.0 will cause the Source to wait until that time elapses before generating any entities. The Interarrival Time specifies the time between arrivals of entities into the system and can be based on a constant value, expression or distribution.

- *If the Arrival Mode is 'Time Varying Arrival Rate'*

The Time Varying Arrival Rate option should be used if the arrival rate changes over time. For example, if entities (i.e., customers) arrive to the system at a less frequent rate for several hours, then at a more frequent rate for several hours, the Time Varying Arrival Rate method of arrivals is a good option. The arrival information is then specified in a [Rate Table](#), which can be defined from the Rate Tables panel, which is found by clicking on the [Data tab](#) of the model. Properties of each rate table include the *Number of Intervals* in which the rate will change and the *Interval Size*. For example, if customer arrival rates change every hour during an 8 hour day, the *Interval Size* would be 1 hour and the *Number of Intervals* would be 8. A different rate may be specified for each of the 1 hour intervals within the 8 hour day. The rate pattern will automatically repeat during the simulation run. A non-stationary Exponential distribution is used to calculate the arrivals, based on the rate specified for each time interval. The Rate Scale Factor property can be used to increase or decrease the values in the Rate Table by a given amount. For example, to increase the Rate Table values by 50%, the Rate Scale Factor property should be 1.5.

- *If the Arrival Mode is 'On Event'*

The Source object may also generate entities only when a given event occurs. This method is useful for modeling pull-type systems, inventory replenishment, and many other situations. Entities are triggered to arrive when a specified event or number of events occur. For example, a simple illustration would be when an entity exits the system through Sink1, this event generates another entity to be created. The event would be specified as Input@Sink1.Entered and once an entity enters the sink, another entity is generated through the Source. The *Event Count* property can be used to signify that a given number of occurrences of the specified event are required before triggering the next entity arrival. The *Initial Number Entities* property can be used to start a given number of entities into the system with this 'On Event' Source. The entities will be generated at the start of the simulation (time 0) and will not be included in the Event Count.

- *If the Arrival Mode is 'Arrival Table'*

The Source object generates entities based on the value of a numeric property in a table. The referenced *Data Table* property should be either a DateTime property, a Real property or an Integer property. If a DateTime property is used, the simulation returns the time span from the starting date of the simulation to itself. So if the starting date of the simulation was 1/1/2008 12:00:00 AM, and the DateTime in the table was 1/1/2008 2:30:00AM, the arrival would be scheduled at time 2.5 hours. If a Real or Integer property is used, the arrival will occur at the simulation start time plus the numeric value of the property.

Note: If the DateTime is before the Simulation Start, or the value is negative, the arrival event does not occur.

The *Arrival Events Per Time Slot* can be used to generate 'groups' of events that will occur at the specified time. The number of entities in each 'group' is based on the *Entities Per Arrival*.

The *Arrival Time Deviation* property will allow you to offset the arrival table value by a certain deviation time. For example, if the arrival table entry value is '3' hours and the *Arrival Time Deviation* is 'Random.Uniform(-.5, .5)', then the arrival may occur anywhere between 2.5 and 3.5 hours. The deviation is applied independently to each of the *Arrival Events Per Time Slot*.

If the scheduled arrival time is at the simulation start time or later, but the deviation expression is moving the actual arrival time earlier than the simulation start time, Simio will interpret that as an arrival at time 0.0.

The *Repeat Arrival Pattern* property allows the user to specify if the arrival pattern specified in the referenced table should repeat when the end of the pattern is reached. If true, the pattern will begin again from the beginning. For example, if a data table had two entries, 8 and 16, and was repeating, you would have an arrival at 8, 16, 24, 32, and so on. If you have arrivals between 9 am and 5 pm, and want that pattern to repeat, you may have a table that

includes the number of arrivals. Then you would specify pairs (Arrival Time / Arrival Quantity), such as 9/1, 10/1, 11/1, 12/1, 13/1, 14/1, 15/1, 16/1, 17/1, 24/0. This way, one arrival enters every hour between 9 and 5, but then no arrivals are specified until time 24 (end of day 1), when the pattern will start to repeat. This way, the gap between 5 pm and the next day 9 am includes no arrivals.

The *No-Show Probability* is applied independently to each of the *Arrival Events Per Time Slot* specified. However, the no-show probability is applied to the entire arrival batch size that is created at any given time. For example, if the *Arrival Events Per Time Slot* is '10' and the *No-Show Probability* is '.5', there is a 50% chance that the any of the 10 arrival groups of *Entities Per Arrival* entities will not be created at a given point in time.

How to Stop Arrivals

- *Define a Maximum Number of Arrivals*

The standard Source object has a property called *Maximum Arrivals*. When this property is set to something other than 'Infinity', the Source will stop producing entities when the number of arrivals equals the value set in the *Maximum Arrivals* property.

- *Define a Maximum Amount of Time for the Source to Generate Arrivals*

The standard Source object has a property called *Maximum Time*. When this property is set to something other than 'Infinity', the Source will stop producing entities when the simulation run has reached the specified amount of time in the *Maximum Time* property. For example, if *Maximum Time* is set to 2, the Source will stop generating arrivals after 2 hours from the beginning of the simulation run.

- *Define a Stopping Event*

The standard Source object has a property called *Stopping Event*. When an Event is specified in this property, the Source object will stop generating arrivals when this event is fired.

- *The Timer Element inside of the Source can be disabled*

The standard Source object uses a Timer element, named *EntityArrivals*, to trigger arrivals. The user can disable this Timer in order to stop arrivals from occurring. To disable the Timer, set *Source1.EntityArrivals.Enabled* to '0'.

Balking from Full Capacity Buffer

- The standard Source object has several balking properties listed under the Buffer Capacities section, including *Output Buffer* and *Balk Node Name*. If the output buffer capacity is full, the an incoming entity will leave the Source depending upon the *Balk Node Name* property.

- *If Balk Node Name is 'None (Destroy Entity)'*

A 'NumberBalked' total will be reported by the *OutputBuffer* station under *Throughput* (along with 'NumberEntered' and 'NumberExited').

Since the balked entities were automatically destroyed at the Source, those destroyed entities will not be recorded as observations in the population 'TimeInSystem' and 'CostPerItem' tallies for the entity type.

- *If Balk Node Name is a Specific Node Name*

A 'NumberBalked' total will be reported by the *OutputBuffer* station under *Throughput* (along with 'NumberEntered' and 'NumberExited').

Because the balked entities were sent to another node, where they may continue in processing, the balked entities will be recorded as observations in the population 'TimeInSystem' and 'CostPerItem' tallies for the entity type.

How to Get Data from a Table before Creating Entities

- *You can reference a Data Table before the Source creates entities*

The standard Source object has a property called *Table Name* which can be used to reference a Data Table before the Source creates an arrival. This can be used to get information from a Data Table that will be used to create an entity. For example, the Data Table might contain information on what type of entity to create or information that will be assigned to a state on the entity. The *Row Number* property is used to indicate which row number to reference in the table. The *RandomRow* function can be used to randomly select a row from the table (i.e. *OrderTable.OrderNumber.RandomRow*).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Sink

The **Sink** destroys entities. It also has the capability to collect automatic statistics.

The 'TimeInSystem' TallyStatistic generated by this object has its Confidence Level property specified as 'Default'. The user may change the Confidence Level used by the Sink's tally by simply changing the Confidence Level in the Experiment.

[State assignments](#) can be made upon entering the Sink. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Sink.

Listed below are the properties of the **Sink**:

Indicates whether the time in system observations on entities destroyed by this Sink are to be automatically logged. Go to Results > Logs > Tally Observation Log to view the logged data.

Property	Valid Entry	Description
TransferIn Time	Expression	The time required to transfer an entity into this Sink object. This property may be specified using a random sample from a distribution.
On Entering (state assignment)	Repeat Group, Assignments	Optional state assignments when an entity is entering the Sink object.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.
Capital Cost	Expression	The initial one-time setup cost to add this object to the system.
Cost Per Use (Input Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Input Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs when an entity has entered this Sink object.
Destroying Entity Add-On Process	Process Instance Name	Occurs when an entity is about to be destroyed by this Sink object.
Transfer-In	Disable,	Indicates the type of constraints used to determine whether entities can transfer

Constraints	Default, Custom Condition	into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.
Transfer-In Condition	Expression	The condition required to allow an entity to transfer into this fixed object via an external input node.
Log Time In System Observations	True, False	
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.
Report Gantt	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

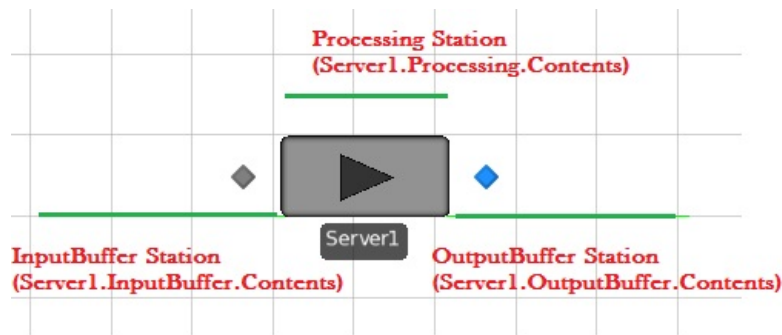
Send comments on this topic to [Support](#)

Server

In Simio, a **Server** represents a capacitated resource with optional constrained input and output buffers.

A Server is an object that contains three capacitated stations with associated queues. The three stations that make up a standard Server are the InputBuffer, Processing and OutputBuffer stations. When a Server is used in a model, it appears as though an entity enters the Server, is processed and then leaves the Server. A more detailed explanation of what happens to the entity is that when it first enters the Server, it enters the InputBuffer station, if the *InputBuffer* property is greater than 0. It waits here until there is available capacity in the Processing station. The capacity of this station is controlled by the *Initial Capacity* property of the Server. Once it has been processed, it moves into the OutputBuffer station, if the *OutputBuffer* is set greater than 0 and it waits until it can leave the Server. The three green queue lines that are placed in the Facility window with the Server, animate the entity while it is in each of these three stations. The three station approach allows for the capability to have both infinite and finite capacity queues. The [Balking and Reneging Options](#) page provides additional information and examples on buffer capacities, balking and reneging.

Standard Server placed in Facility Window



The logic that is being performed once the entity enters the Server object spawns what is called a **Token**. A Token is a completely different object than an entity. It is used to trigger logical events in the [Processes Window](#) of the model.

Within the Processing station of the Server, the *Process Type* may be a 'Specific Time' or based on a 'Task Sequence'. The default 'Specific Time' will delay the entity at the processing station of the Server for the simple *Processing Time* specified. The 'Task Sequence' option will require a number of *Processing Tasks* and associated information to be specified. This is explained in greater detail within the [Task Sequence - Processing Tasks](#) section.

A Server may have failures, which are specified within the [Reliability](#) section of the Properties window. [State assignments](#) can be made on entering, before processing, after processing, before exiting, on balking and on reneging the Server. Secondary Resources may be seized and/or released by the entity for processing or upon entering the Server, and before and after processing occurs. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Server.

An entity processing at a Server may be interrupted with the [Interrupt](#) step. The Processing Time delay is the only interruptible delay within the Server.

The *Seize Constraint Logic* repeat group supports imposing additional constraints on an entity's request to seize the object by referencing [Constraint Logic](#) elements (only if the InputBuffer is > 0).

The *Immediately Try Seize* and *Immediately Try Allocate When Released* options within the Other Processing Options (and Secondary Resources area) section of properties are typically advanced users. They deal with processing events on the Simio event calendar that occur at the same time, including arrival of multiple entities to an object, or release of multiple objects at a time for re-allocation. It's important to note that these properties do not affect processing for entities that arrive on path with *Allow Passing* set to 'False', entities arriving via Conveyor, or any other construct that causes entities to arrive one at a time at different simulation times. This feature will not provide any benefit because the second (or following) entity will not arrive before the late event happens at the first entity's simulation time.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Server.

For more information, see [Server - Discussion and Examples](#).

Listed below are the properties of the **Server**:

Property	Valid Entry	Description
Capacity Type	Fixed, Work Schedule	The method used to specify the capacity of this Server object.
Initial Capacity	Expression	The initial fixed capacity of this Server object.
Initial Work Schedule	WorkSchedule name	The name of the work schedule initially assigned to the resource. Note: Refere to the SetWorkSchedule step to dynamically assign a new work schedule to a resource during a simulation run.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order entities waiting to be allocated capacity of this Server object.
Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this object, when its capacity becomes available, dynamically selects the next allocation request from its statically ranked allocation queue using a dynamic selection rule.
Repeat Group	True, False	Indicates whether a repeat group data structure is used to define the primary dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax <code>Candidate.[EntityClass].[Attribute]</code> to reference an attribute of the candidate entities (e.g., <code>Candidate.Entity.Priority</code>).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.

TransferIn Time	Expression	The time required to transfer an entity into this Server object. This property may be specified using a random sample from a distribution.
Process Type	Specific Time, Task Sequence	The method used to model the processing of an entity at this Server object.
Processing Time	Expression	The time required for this Server object to process each entity. This property may be specified using a random sample from a distribution.
Processing Tasks	Repeat Group, Tasks	The set of tasks required for this Server object to process an entity.
Loopback Branches	Repeat Group, Processing Tasks	Conditional or probabilistic loopback branches in the task workflow.
Task Precedence Method	SequenceNumberMethod, ImmediatePredecessorsMethod, ImmediateSuccessorsMethod	The method used to define the task precedence dependencies. Can be either by specifying task sequence numbers, specifying the immediate predecessors for each task, or specifying the immediate successors for each task.
Task Resource Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting task resource requirements.
Task Materials Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting task material requirements.
Task State Assignments Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting state assignments.
Invalid Task Dependency Notification Type	Indicates the level of alert at runtime if the task sequence has missing, duplicate, or otherwise invalid data defining the task dependency relationships. Error - Throw runtime error. Warning - Throw runtime warning and skip the invalid data. Trace - Write to trace and skip the invalid data.	
Consumption Or Production Type	Material, Bill Of Materials, Bill Of Materials Group	The type of material stock consumption or production. Material - A single material. BillOfMaterials - A list of component materials. BillOfMaterialsGroup - Alternative lists of component materials. Material consumption will be required to start the processing of the task. Material production will occur after finishing the processing of the task.
Material Name	Material Reference	The name of the material.
BOM Name	Bill Of Materials Reference	The name of the bill of materials.
BOM Group Name	Bill Of Materials Group Reference	The name of the bill of materials group.

Quantity	Real	The quantity of the material or parent assembly.
Lot ID	String	Optional string value indicating the lot identifier of the material or parent assembly.
Off Shift Rule	Suspend Processing, Finish Work Already Started	<p>The processing rule used at the Server at the end of a shift.</p> <p>If the rule is 'Suspend Processing', then the Server will immediately suspend all processing and set its resource state to 'OffShift'. Processing will resume at the start of the next shift.</p> <p>If the rule is 'Finish Work Already Started', then the Server will not accept any new entities but will continue processing if necessary to finish work already started. The Server's resource state will be set to 'OffShiftProcessing' if processing entities during an off-shift period. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Resource Efficiency Rule	None, Average, Count, Maximum, Minimum, Sum	<p>If the rule is 'None', then seized resource efficiency is the processing rule used to alter the rate at which work is performed if there are seized resources with defined efficiency values. The actual work duration is the planned work duration divided by the efficiency.</p> <p>If the rule is 'Average', then the average seized resource efficiency value is used.</p> <p>If the rule is 'Count', then the number of seized resources with a defined efficiency value is used.</p> <p>If the rule is 'Maximum', then the largest efficiency value is used.</p> <p>If the rule is 'Minimum', then the smallest efficiency value is used.</p> <p>If the rule is 'Sum', then the sum of the seized resource efficiency values is used.</p>
Seize Constraint Logic	Repeat Group, Constraint Logic elements	<p>Constraint logic used to enforce additional constraints on an entity's request to seize the Server.</p> <p>NOTE: This repeat group applies only if the Server has an input buffer. Otherwise, if no input buffer, then this constraint logic will be ignored.</p>
Seize Constraint Logic.Constraint Logic Name	Constraint Logic elements	The name of the Constraint Logic element used to enforce additional constraints on an entity's request to seize the Server.
Immediately Try Seize	True, False	<p>For an arriving entity, indicates whether to immediately try seizing the Server before the execution of any other simulation logic in the system and, if successful, skipping the Server's allocation queue.</p> <p>Setting this property to False will just insert the seize request into the Server's allocation queue. An evaluation of that queue will then be scheduled on the simulation's current event calendar as a late priority event.</p> <p>NOTE: This property setting applies only if the Server has an input buffer. Otherwise, if no input buffer, then an arriving entity will always immediately try to enter the Server's processing station and seize it.</p>
Immediately Try	True, False	Once an entity has exited processing and released the

Allocate When Released		<p>Server, indicates whether to immediately try allocating the released Server capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the Server's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>
Input Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Server object's input buffer.
Balk Decision Type (Input Buffer)	None, Blocked, Conditional, Probabilistic	<p>The method used by an entity to decide whether to balk at entering the buffer.</p> <p>If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.</p>
Balk Condition Or Probability (Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Server object's output buffer.
Balk Decision Type (Output Buffer)	None, Blocked, Conditional, Probabilistic	<p>The method used by an entity to decide whether to balk at entering the buffer.</p> <p>If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.</p>
Balk Condition Or Probability (Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Output Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Reneged Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.
Trigger Type.Reneged Triggers	Time Based, Event Based	<p>The type of trigger.</p> <p>A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.</p>
Wait Duration.Reneged Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Reneged Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.

Renege Decision Type .Renege Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Renege Condition Or Probability.Renege Triggers	Expression Specified if the <i>Renege Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renegeing as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Renege Node Name.Renege Triggers	Node object reference	Facility node location to send a renegeing entity.
Trigger Start Boundary.Renege Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
Failure Type	No Failures, Calendar Time Based, Processing Count Based, Event Count Based, Processing Time Based	Specifies whether this Server object has failures that affect the object's availability, and the method used to trigger the failure occurrences. See the Reliability page for additional information on Failures.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample form a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Action Type (Table Row Referencing - Before Processing)	Reference Existing Row, Add New Row	The type of table row reference action to perform. The 'Reference Existing Row' action type may be used to set a reference to an existing table row in a data table or sequence table. The row index into the table is specified by the <i>Row Number</i> property. The 'Add New Row' action type may be used to add a new row to an output table. A reference will be automatically set to the newly added row.
Table Name (Table Row Referencing - Before Processing)	Table Name	The name of the table.
Row Number (Table Row Referencing - Before Processing)	Expression	The one-based row index into the table.
On Entering (State	Repeat Group, Assignments	Optional state assignments when an entity is entering the

Assignments)		Server object.
Before Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has been allocated capacity to be processed at the object, but before entering (or ending transfer into) the object's processing station.
After Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has completed its processing and is about to attempt its exit from the object's processing station.
Before Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the Server object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is reneging from an input or output buffer of the object.
Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Repeat Group (Secondary Resource for Processing)	True, False	Indicates whether a repeat group data structure is used to define the secondary resources for processing. If a repeat group structure is used, many of the below properties are specified within the repeat group. <i>OffShift Rule, Must Simultaneously Seize, Immediately Try Seize</i> and <i>Immediately Try Allocate When Released</i> are not within the repeat group, thus not resource specific.
Resource Type (Secondary Resource for Processing)	Specific, From List, ParentObject	The method for specifying the resource object(s) to seize.
Resource Name (Secondary Resource for Processing)	Object Instance Name	The name of the resource object to seize.
Resource List Name (Secondary Resource for Processing)	Object List Instance Name	The name of the object list from which to select the resource object(s) to seize.
Selection Goal (Secondary	Smallest Distance, Largest Distance, Preferred Order,	The goal for selecting objects to seize.

Resource for Processing)	Smallest Value, Largest Value, Random	
Value Expression (Secondary Resource for Processing)	Expression	The expression evaluated for each object that is a candidate to seize. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., Candidate.Object.Capacity.Remaining).
Request Move (Secondary Resource for Processing)	None, To Node	Indicates whether a move to a specified location will be requested from the seized resource. Processing will not be able to start until the resource has arrived to the requested location.
Destination Node (Secondary Resource for Processing)	Node Instance Name	The name of the specific node location that the seized resource will be requested to move to.
Off Shift Rule (Secondary Resource for Processing)	Finish Work Already Started, Suspend Processing, Switch Resources If Possible	<p>The processing rule used if the secondary resource is at the end of a shift because of a specified work schedule. If the rule is 'Finish Work Already Started', then the processing of the entity that is using the secondary resource will be allowed to continue processing until finished. The secondary resource will not be allowed to accept any new work.</p> <p>If the rule is 'Suspend Processing', then the processing of the entity that is using the secondary resource will be immediately suspended. Processing will resume at the start of the secondary resource's next shift.</p> <p>If the rule is 'Switch Resources If Possible', then the processing of the entity that is using the secondary resource will be immediately suspended. The entity will then try to resume processing as soon as possible by releasing the resource and seizing another available one that satisfies the same resource requirements. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Number of Resources (Secondary Resource for Processing)	Expression	The number of individual resource objects to seize capacity units of.
Units Per Resource (Secondary Resource for Processing)	Expression	The number of capacity units to seize per resource object.
Selection Condition (Secondary Resource for Processing)	Expression	Optional condition evaluated for each candidate resource that must be true for the resource to be selected. In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource objects (e.g., Candidate.Object.Capacity).
Resource	Expression	Optional value that can alter the rate at which work is

Efficiency (Secondary Resource for Processing)		<p>performed using the seized resource(s), expressed as a fraction. The actual work duration is the planned work duration divided by the efficiency. Hence, an efficiency value greater than 1 shortens the time and a value less than 1 lengthens the time.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).</p> <p>Set this property to blank to declare the resource efficiency as undefined.</p>
Must Simultaneously Seize (Secondary Resource for Processing)	True, False	If multiple resources are required, indicates whether all of the resources must be available before any can be seized.
Immediately Try Seize (Secondary Resource for Processing)	True, False	<p>Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues.</p> <p>Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.</p>
Keep Reserved If (Secondary Resource for Processing)	Expression	<p>On processing completion, an optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects without a reservation will be unable to seize the reserved capacity unless the reservation is cancelled.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>).</p> <p>Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p> <p>Leaving this property blank (no condition) is equivalent to entering False.</p>
Reservation Timeout (Secondary Resource for Processing)	Expression	<p>If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>).</p>
Immediately Try Allocate When Released (Secondary Resource for Processing)	True, False	<p>Once an entity has finished processing and released the resource, indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>

Skip Requirement If (Secondary Resource for Processing)	Expression	Optional condition indicating whether to skip the resource requirement.
On Entering (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizes when an entity is entering the Server object.
Must Simultaneously Seize (Other Resource Seizes -- On Entering)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- On Entering)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizes once an entity has been allocated Server capacity, to be completed before starting the processing time.
Must Simultaneously Seize (Other Resource Seizes -- Before Processing)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- Before Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizes after an entity has finished processing, to be completed before releasing the Server capacity.
Immediately Try Seize (Other Resource Seizes -- After Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
Must Simultaneously Seize	If multiple resources are required, indicates whether all must be available before any can be seized.	
On Entering	Repeat Group, Resource	Optional secondary resource releases to be performed

(Other Resource Releases)	Releases	when an entity is entering the Server object.
Immediately Try Allocate When Released (Other Resource Releases -- On Entering)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases once an entity has been allocated Server capacity, before starting the processing time.
Immediately Try Allocate When Released (Other Resource Releases -- Before Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases after an entity has finished processing, before releasing the Server capacity.
Immediately Try Allocate When Released (Other Resource Releases -- After Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.
Capital Cost	Expression	The initial one-time setup cost to add this object to the system.
Cost Per Use (Input Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Input Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Cost Per Use (Output Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Output Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.

Idle Cost Rate (Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of the server for each unutilized scheduled capacity unit.
Cost Per Use (Resource Costs)	Expression	The one-time cost that is accrued each time the server is used, regardless of the usage duration. This cost will be charged to the cost of the entity using the server.
Usage Cost Rate (Resource Costs)	Expression	The cost per unit time to use the server if in a utilized state. This cost will be charged to the cost of the entity using the server.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add- On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs when an entity has entered this server object.
Before Processing Add-On Process	Process Instance Name	Occurs when an entity has been allocated server capacity, but before entering (or ending transfer into) the object's processing station.
Processing Add- On Process	Process Instance Name	Occurs when an entity has been allocated server capacity and is about to start processing.
After Processing Add-On Process	Process Instance Name	Occurs when an entity has completed the processing time and is about to attempt its exit from the server's processing station in order to release the server capacity.
Exited Add-On Process	Process Instance Name	Occurs when an entity has exited this server object.
Failed Add-On Process	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process	Process Instance Name	Occurs when a failure is repaired at this object.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Server object is evaluating whether to accept or reject a request to seize capacity of the server. In the executed decision process, assigning a value of less than or equal to '0' to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
On Shift Add-On Process	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process	Process Instance Name	Occurs when the object goes 'off shift' because of its specified work schedule.
Transfer-In Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.

Transfer-In Condition	Expression	The condition required to allow an entity to transfer into this fixed object via an external input node.
Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out Condition	Expression	The condition required to allow an entity to transfer out of this fixed object via an external output node.
Expected Setup Time Expression	Expression	The expression used to get a deterministic estimation of the setup time for an entity at this fixed object.
Expected Operation Time Expression	Expression	The expression used to get a deterministic estimation of the operation time for an entity at this fixed object.
Processing Task Sequence Random Number Stream	Expression	The random number stream to be used if there is probabilistic branching in the task sequence.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Edition). If not specified, then this property defaults to a color derived from the object's unique name.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Server - Discussion and Examples

Discussion

Fixed Capacity or Capacity Varying by Work Schedule

- *Fixed*

If the *Capacity Type* property is set to 'Fixed', the value of the initial capacity will be used to determine the capacity for the Server at the start of the simulation run. The capacity can be specified with an expression. If a random distribution is specified in the capacity expression, it will only be evaluated once at the beginning of the run and the capacity will remain unchanged. The Assign step can be used to change the Server's capacity during the simulation run by using the Server state, `ServerName.CurrentCapacity`. This will change the value of the capacity function, `ServerName.Capacity`.

- *WorkSchedule*

If the *Capacity Type* property is set to 'Work Schedule', a work schedule must be specified in the *Work Schedule* property. A work schedule is defined from the Schedules panel, which is found by clicking on the [Data tab](#).

Server has ResourceState State Variable that Tracks Its States

- The ResourceState state variable is a [List State](#) automatically tracked for the Server. The numeric values for the state are Starved=0, Processing=1, Blocked=2, Failed=3, OffShift=4, FailedProcessing=5, OffShiftProcessing=6, Setup=7 and OffShiftSetup=8. If the Server has a capacity greater than 1, it is considered Processing when one or more units of the Server are busy.

The *Off Shift Rule* property of the Server will determine what happens when a Server object is scheduled to go 'OffShift'. If the rule is 'Suspend Processing', then any processing will immediately be halted and the Server will go into an 'OffShift' state. If the rule is 'Finish Work Already Started' and the Server is in the process of performing a task, then it will first complete any remaining work while in the 'OffShiftProcessing' state before going into the 'OffShift' state. If the rule is 'Finish Work Already Started' and the Server is in the process of a setup time for a changeover, then it will first complete any remaining setup work while in the 'OffShiftSetup' state before going into the 'OffShift' state. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.

You can use the ResourceState in process logic to check whether a server is 'Processing', 'Starved' or other state. This would be done by utilizing the function `Server.ResourceState` in the logic, given that the token referencing this function is associated with the server. You will see Resource State results for the Server objects in the Results window.

To animate the server ResourceState, you simply specify the Server name and the desired function. For example, if you have a Server named `Server1`, you may wish to place a Status Pie, specify the *Data Type* as 'ListState' and select `Server1.ResourceState` from the *List State* property list.

The ResourceState variable is used to display utilization statistics for the Server in the Results window. Additionally, this list state can be used to display utilization statistics during the simulation run. For example, you may wish to show the percentage of time that a Server is processing during the simulation run. This would be done by placing a status label with the expression `ServerName.ResourceState.PercentTime(1)`. For more information on list state values and utilization statistics, see [List State](#) page.

Server that has no Buffers (i.e. InputBuffer and OutputBuffer properties set to 0)

- When a user sets the *InputBuffer* property to 0, it sets this Server object to have no space for buffering inside the object. In other words, if an entity arrives to the Server and there is no available capacity of the main Processing station, the entity must wait outside of the Server until the capacity of the Server becomes available. If the entity is entering the Server via an incoming link, the entity will wait on the link.

Similarly, if the *OutputBuffer* property is set to 0, the entity must have an available physical location to move into before it can leave the Server. For example, if the entity must ride on a Transporter after it leaves the Server and the Transporter is not available at the time, the entity will wait in the Processing station of the Server until the Transporter arrives at the output node to pick up the entity.

Behavior of Transfer-In Constraints and Transfer-In Condition Property

- When the *Transfer-In Constraints* property is set to 'Default', then the *Can Transfer In and Out of Objects* property of the entity will determine whether an entity can perform a transfer into the object. For example, a ModelEntity object's *Can Transfer In and Out of Objects* property is set to 'True', by default, because entities normally go into objects for processing. Yet a Vehicle object has this property set to 'False' because normally Vehicles do not enter objects, they just carry entities between objects.

When the *Transfer-In Constraints* property is set to 'Custom Condition', then the user is prompted to provide *Transfer-In Condition*. This condition will determine which entities can enter this Server object. If the entity arrives and fails to meet the condition, it will be routed on an alternative route (if there is a path leaving the input node of the Server) or it will be destroyed if there is no path for it to take.

When the *Transfer-In Constraints* property is set to 'Disable', then all entities will be able to enter this object, regardless of what is set in their *Can Transfer In and Out of Objects* property. For example, Vehicle objects will be able to enter this object, because their default value of 'False' for *Can Transfer In and Out of Objects* property is overridden in this scenario.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Resource

A **Resource** is a static capacitated Fixed Object that can be seized. It does not move within the Facility window. If you would like a moveable Resource, consider seizing capacity of a [Vehicle](#) or [Worker](#) object.

A resource's capacity can be fixed or controlled by a work schedule. A resource object may have a capacity of 0 to N and is considered 'OffShift' if capacity = 0 or 'OnShift' if capacity > 0.

Resources can be grouped together in an object list so they can be seized as a group or selected from a list of possible resources based on a selection criteria. A seize operation requires a negotiation between the objects involved. An object could refuse the seize request because he/she has other activities to perform. Hence in Simio, objects allow or disallow themselves to be seized and released by other objects.

A Resource may have failures, which are specified within the [Reliability](#) section of the Properties window. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Resource.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Resource.

For more information, see [Resources - Discussion and Examples](#).

Listed below are the properties of the **Resource**:

Property	Valid Entry	Description
Capacity Type	Fixed, WorkSchedule	The method used to specify the capacity of this Resource object.
Initial Capacity	Expression	The initial capacity of this Resource object.
Initial Work Schedule	WorkSchedule name	The name of the work schedule initially assigned to the resource. Note: Refere to the SetWorkSchedule step to dynamically assign a new work schedule to a resource during a simulation run.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order entities waiting to be allocated capacity of this Resource object.
Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this Resource object, when allocation of its capacity is performed, dynamically selects the next allocation request using an expression-based rule.

Failure Type	No Failures, Calendar Time Based, Usage Count Based, Event Count Based, Usage Time Based	Specifies whether this Resource object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.
Capital Cost	Expression	The initial one-time setup cost to add this object to the system.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of the resource for each unutilized scheduled capacity unit.
Cost Per Use(Resource Costs)	Expression	The one-time cost that is accrued each time the resource is used, regardless of the usage duration. This cost will be charged to the cost of the owner object using the resource.
Usage Cost Rate(Resource Costs)	Expression	The cost per capacity unit time to use the resource if in a utilized state. This cost will be charged to the cost of the owner object using the resource.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Allocated Add-On Process Triggers	Process Instance Name	Occurs when this Resource object has had capacity allocated to another object.
Released Add-On Process Triggers	Process Instance Name	Occurs when this Resource object has had capacity released by another object.

Failed Add-On Process Triggers	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process Triggers	Process Instance Name	Occurs when a failure is repaired at this object.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Resource is evaluating whether to accept or reject a capacity allocation attempt. In the executed decision process, assigning a value of less than or equal to '0' to the executing token's ReturnValue state (Token.ReturnValue) indicates that the allocation attempt is rejected. The Token that executes this process is associated with the Candidate object.
On Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'off shift' because of its specified work schedule.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Edition). If not specified, then this property defaults to a color derived from the object's unique name.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

For additional information on Resources, see the help topic [Resources - Discussion and Examples](#).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Resources - Discussion and Examples

Discussion

What makes an Object a Resource?

- Of the standard library objects, the Server, Combiner, Separator, Resource, Basic Node and Transfer Node are considered resources, by definition. In other words, the property *Resource Object* in the object definition is set to 'True' for each of these standard library objects. This means that their capacity can be seized and released. A user might decide to use the Resource standard library object because as a convenience, there are two Add-On Processes in this object that are useful with capacity allocation. These are the Allocated and Evaluating Seize Request Add-On Processes. If a user is defining a new object from scratch and would like to make this object a resource, the *Resource Object* property must be set to True in the object definition.

Fixed Capacity or Capacity Varying by Work Schedule

- Fixed*

If the *Capacity Type* property is set to 'Fixed', the value of the initial capacity will be used to determine the capacity for the resource at the start of the simulation run. The capacity can be specified with an expression. If a random distribution is specified in the capacity expression, it will only be evaluated once at the beginning of the run and the capacity will remain unchanged. The Assign step can be used to change the resource's capacity during the simulation run by using the resource state, *ResourceName.CurrentCapacity*. This will change the value of the capacity function, *ResourceName.Capacity*.

- WorkSchedule*

If the *Capacity Type* property is set to 'Work Schedule', a work schedule must be specified in the *Work Schedule* property. A work schedule is defined from the Schedules panel, which is found by clicking on the [Data tab](#). The resource object may have a capacity of 0 to N and is considered 'OffShift' if capacity = 0 or 'OnShift' if capacity > 0.

Evaluating Requests for this Object's Capacity

- Ranking Rule*

A resource object has a single (internal) allocation queue with *Ranking Rule* and *Ranking Expression* properties that define the static ranking of that queue. Regardless of where the Seize step takes place, every seize request for this object goes into that single allocation queue. The *Ranking Rule* property is used to order the requests that are waiting to be allocated capacity of this object. The *Ranking Expression* property is the value that is used to evaluate the requests that are waiting allocation of this object. It is only used when *Ranking Rule* is set to 'Smallest Value' or 'Largest Value'.

- Dynamic Selection Rule*

The Resource object can dynamically select which request is addressed next. If the *DynamicSelectionRule* property is set to either 'Smallest Value First' or 'Largest Value First', it uses the expression in the *DynamicSelectionExpression* property to evaluate which request should be serviced next. This expression is a [candidate reference expression](#) since it is evaluating other objects that have made a request for this object's capacity, so the keyword 'Candidate' must be used in this expression. If both the static *Ranking Rule* and the *DynamicSelectionRule* are set, the Ranking Rule is used as a backup.

- Evaluating Seize Request Add On Process*

In the Resource object from the standard object library, there is an Add On Process called Evaluating Seize Request. This Add On Process is executed when the object is evaluating whether to accept or reject the capacity allocation request. In the executed decision process, the user should assign a value of less than or equal to '0' to the executing token's Return Value state (*Token.ReturnValue*) to indicate that the allocation attempt is rejected. The value '1' indicates that the allocation attempt was successful.

Manually Trigger an Allocation Attempt

- Simio has an [Allocate Step](#), which can be used to manually trigger an allocation attempt of the parent object's available capacity to other objects that are waiting for units of the capacity. For example, some of the standard

library objects use the Allocate Step to trigger an allocation attempt after a failure has occurred.

Resource has ResourceState State Variable that Tracks Its States

- The ResourceState state variable is a [List State](#) automatically tracked for the Resource. The numeric values for the state are Idle=0, Busy=1, Failed=3, OffShift=4, FailedBusy=5, OffShiftBusy=6. A Resource is considered busy when it is Seized by a token using a Seize step. If the resource has a capacity greater than 1, it is considered busy when one or more units of the resource are busy. Similarly, if the capacity is greater than 1 and it is busy, if the resource goes offshift, it is considered OffShiftBusy until all units are no longer allocated for processing.

You can use the ResourceState in process logic to check whether a resource is 'Busy' or 'Idle'. This would be done by utilizing the function Resource.ResourceState in the logic, given that the token referencing this function is associated with the resource. You will see Resource State results for the resource objects in the Results window.

To animate the resource ResourceState, you simply specify the resource name and the desired function. For example, if you have a Resource named Tool1, you may wish to place a Status Pie, specify the *Data Type* as 'ListState' and select Tool1.ResourceState from the *List State* property list.

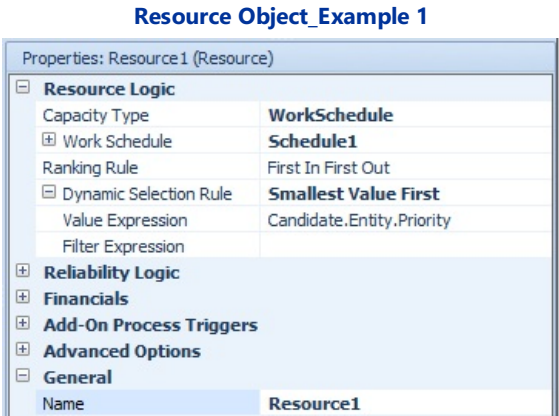
The ResourceState variable is used to display utilization statistics for the Resource in the Results window. Additionally, this list state can be used to display utilization statistics during the simulation run. For example, you may wish to show the percentage of time that a resource is busy during the simulation run. This would be done by placing a status label with the expression ResourceName.ResourceState.PercentTime(1). For more information on list state values and utilization statistics, see [List State](#) page.

Functions that can be used with Resources

- There are two functions that might be useful to use with Resource objects. The user can write logic based on the current capacity of the resource, using the function Capacity. And if the capacity of the resource changes with a schedule, the function Capacity.Previous can be used to find the capacity before the last change.

Examples

Example 1



This Resource Object from the standard object library will have a changing capacity that is based on a work schedule named Schedule1. The details of this work schedule could be found in the Schedules Panel within the Data Window of this model. This Resource Object will also dynamically select from its internal resource allocation queue using the expression ModelEntity.Priority. It will select the entity with the smallest value for ModelEntity.Priority to be allocated capacity next.

Combiner

A **Combiner** object may be used to model a process that groups multiple entities together and then attaches the batch members to a parent entity.

The Combiner object contains four capacitated stations with associated queues; one station for the parent input buffer, one for the member input buffer, the processing station and the output buffer station. The four station approach allows for the capability to have both infinite and finite capacity queues. The [Balking and Reneging Options](#) page provides additional information and examples on buffer capacities, balking and reneging.

Batching Details

An entity arrives at the parent input node of the Combiner. This station has a queue and a capacity. Entities also arrive to the member input node which has its own queue and capacity. The entities wait in these queues until they batch together according to the logic specified in the properties of the Combiner. The *Batch Quantity* properties can be specified by an expression (i.e. a variable or a random distribution) as well as a string value. The *Matching Rule* properties specifies if the entities should meet a certain criteria before they can be batched together. If there is a criteria, the *Member Match Expression* and the *Parent Match Expression* is where the criteria is specified. For the *Matching Rule* of 'Match Members' and 'Match Members and Parent', the match will occur on the first match encountered by any waiting entities, not just when the first waiting entity has a match. Therefore, entities are not necessarily matched based on their order of arrival. Refer to the [BatchLogic](#) element for additional details on batching specifics.

Releasing Batch Early Triggers

The *Release Batch Early Triggers* defined for a parent entity can be either time based (trigger fires when specified wait duration expires) or event based (trigger fires when a specified event occurs). By default, the release decision when a trigger has fired is 'Always', but may alternatively be 'Conditional' or 'Probabilistic'. A parent entity that is being released early attempts to collect as many batch members as possible up to its target batch quantity(ies) - possibly no batch members are collected. The parent entity's associated queue item is then removed from the BatchLogic element's parent queue and its associated process token exits the Batch step. * Note that the data for a BatchLogic element or Combiner's *Release Batch Early Triggers* repeat group may be defined in a data table, with the trigger definitions thus potentially differing between parent entities.

Reneging a Parent Entity From Parent Input Buffer vs. Using a Releasing Batch Early Trigger

If a renege trigger is used to renege a parent entity from a combiner's parent input buffer, then the parent entity immediately cancels its outstanding batch request. There is no attempt to collect any further batch members. The parent entity is then physically removed from the buffer and either destroyed or sent to some specified node in the model.

A release batch early trigger, on the other hand, causes the parent entity to attempt to collect as many batch members as it can up to its target batch quantity(ies). The parent entity then continues normal processing at the combiner (i.e., exiting the parent input buffer and entering the combiner's processing station).

Transferring In and Processing at the Combiner

The transfer-in times for the parent and member entities run concurrently. For example, if a *Parent Transfer-In Time* is '1' minute and a *Member Transfer-In Time* is '2' minutes, they will be ready to match (given matching requirements) after 2 minutes if they arrive at the Combiner at the same time.

The Combiner does not only batch entities together, it also has process logic and capacity logic similar to a Server Object. If the entity is processed by the Combiner, it then appears at the process station's queue. After the logic of the process has completed, if there is space the entity will be moved to the output member queue.

Within the Processing station of the Combiner, the *Process Type* may be a 'Specific Time' or based on a 'Task Sequence'. The default 'Specific Time' will delay the entity at the processing station of the Combiner for the simple *Processing Time* specified. The 'Task Sequence' option will require a number of *Processing Tasks* and associated information to be specified. This is explained in greater detail within the [Task Sequence - Processing Tasks](#) section.

Additional Information

A Combiner may have failures, which are specified within the [Reliability](#) section of the Properties window. [State assignments](#) can be made upon either parent or member entering, before processing, after processing, before exiting, on balking and on reneging the Combiner. Secondary Resources may be seized and/or released upon the parent entity entering the Combiner, and before and after processing occurs. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Combiner.

An entity processing at a Combiner may be interrupted with the [Interrupt](#) step. The Processing Time delay is the only interruptible delay within the Combiner.

The *Seize Constraint Logic* repeat group supports imposing additional constraints on an entity's request to seize the object by referencing [Constraint Logic](#) elements.

The *Immediately Try Seize* and *Immediately Try Allocate When Released* options within the Other Processing Options (and Secondary Resources area) section of properties are typically advanced users. They deal with processing events on the Simio event calendar that occur at the same time, including arrival of multiple entities to an object, or release of multiple objects at a time for re-allocation. It's important to note that these properties do not affect processing for entities that arrive on path with *Allow Passing* set to 'False', entities arriving via Conveyor, or any other construct that causes entities to arrive one at a time at different simulation times. This feature will not provide any benefit because the second (or following) entity will not arrive before the late event happens at the first entity's simulation time.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Combiner.

For additional information, see the [Batch](#) step and [BatchLogic](#) element, which are included within the Combiner object logic.

For SimBits using the Combiner object, please review [CombineThenSeparate](#), [CustomUnbatching](#), [CombineMatchingMembers](#), [CombinerReleasingBatchEarly](#), and [CombineMultipleEntityTypesOntoPallets](#).

Listed below are the properties of the **Combiner**:

Property	Valid Entry	Description
Batch Quantity	Expression truncated to an integer	The target number of member entities to collect and attach to a parent entity.
Matching Rule	AnyEntity, MatchMembers, MatchMembersAndParent	The matching rule used to group member and parent entities together. If the matching rule is 'AnyEntity', then any entity can be grouped with any other entity. If the matching rule is 'MatchMembers', then member entities must be grouped together using a specified match expression. If the matching rule is 'MatchMembersAndParent', then the parent entity must also match the member entities to attach a group to the parent.
Member Match Expression	Expression	The expression whose value must be the same between member entities to group them together.
Parent Match Expression	Expression	The expression whose value for a parent entity must be the same as the match expression for the member entities to attach those members to the parent.
Batch Quantities (More)	Repeat Group, Batch Quantities	Additional member entities to collect and attach to a parent entity.
Batch Quantity (Batch Quantities)	Expression truncated to an integer	The target number of member entities to collect and attach to a parent entity.
Matching Rule (Batch Quantities)	AnyEntity, MatchMembers, MatchMembersAndParent	The matching rule used to group member and parent entities together. If the matching rule is 'AnyEntity', then any entity can be grouped with any other entity. If the matching rule is 'MatchMembers', then member entities must be grouped together using a specified match expression. If the matching rule is 'MatchMembersAndParent', then the parent entity must also match the member entities to attach a group to the parent.
Member Match	Expression	The expression whose value must be the same between

Expression (Batch Quantities)		member entities to group them together.
Parent Match Expression (Batch Quantities)	Expression	The expression whose value for a parent entity must be the same as the match expression for the member entities to attach those members to the parent.
Parent Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank the order of entities in the parent input buffer of the Combiner.
Parent Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Member Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank the order of entities in the member input buffer of the Combiner.
Member Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Must Simultaneously Batch	True, False	Indicates whether the full target number(s) of member of entities must be available before any can be collected and attached to a parent entity.
Release Batch Early Triggers	Optional time or event-driven triggers that can cause a batch to be released early before collecting the full target number(s) of member entities.	
Trigger Type (Release Batch Early Triggers)	The type of trigger. A time based trigger can release a batch early once a wait duration expires. An event based trigger can release a batch early whenever a specified event occurs.	
Wait Duration (Release Batch Early Triggers)	The wait duration until deciding whether to release a batch early.	
Triggering Event Name (Release Batch Early Triggers)	The name of the event whose occurrence will trigger a decision whether to release a batch early.	
Release Decision Type (Release Batch Early Triggers)	Optional time or event-driven triggers that can cause a batch to be released early before collecting the full target number(s) of member entities.	
Release Condition or Probability (Release Batch Early Triggers)	The release condition or probability specified as an expression. If a probability, then enter the chance of releasing the batch early as a value between 0.0 (0%) and 1.0	

	(100%).	
Suspend Batching When Down	True, False	Indicates whether to suspend collecting batches if the Combiner has failed or is at the end of a shift because of its specified work schedule.
Capacity Type	Fixed, Work Schedule	The method used to specify the number of batching operations that this Combiner object can simultaneously process.
Initial Capacity	Expression	The initial capacity of this Combiner object.
Initial Work Schedule	WorkSchedule name	The name of the work schedule initially assigned to the resource. Note: Refer to the SetWorkSchedule step to dynamically assign a new work schedule to a resource during a simulation run.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Parent TransferIn Time	Expression	The time required to transfer a parent entity into this Combiner object. This property may be specified using a random sample from a distribution.
Member TransferIn Time	Expression	The time required to transfer a member entity into this Combiner object. This property may be specified using a random sample from a distribution.
Process Type	Specific Time, Task Sequence	The method used to model the processing of an entity at this Combiner object.
Processing Time	Expression	The time required for this Combiner object to process each entity. This property may be specified using a random sample from a distribution.
Processing Tasks	Repeat Group, Tasks	The set of tasks required for this Combiner object to process an entity.
Loopback Branches	Repeat Group, Processing Tasks	Conditional or probabilistic loopback branches in the task workflow.
Task Precedence Method	SequenceNumberMethod, ImmediatePredecessorsMethod, ImmediateSuccessorsMethod	The method used to define the task precedence dependencies. Can be either by specifying task sequence numbers, specifying the immediate predecessors for each task, or specifying the immediate successors for each task.
Task Resource Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting task resource requirements.
Task Materials Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting task material requirements.
Task State Assignments Referenced Table	Table Name	Optional name of data table to be used as the referenced data source for getting state assignments.

Name		
Invalid Task Dependency Notification Type	Indicates the level of alert at runtime if the task sequence has missing, duplicate, or otherwise invalid data defining the task dependency relationships. Error - Throw runtime error. Warning - Throw runtime warning and skip the invalid data. Trace - Write to trace and skip the invalid data.	
Consumption Or Production Type	Material, Bill Of Materials, Bill Of Materials Group	The type of material stock consumption or production. Material - A single material. BillOfMaterials - A list of component materials. BillOfMaterialsGroup - Alternative lists of component materials. Material consumption will be required to start the processing of the task. Material production will occur after finishing the processing of the task.
Material Name	Material Reference	The name of the material.
BOM Name	Bill Of Materials Reference	The name of the bill of materials.
BOM Group Name	Bill Of Materials Group Reference	The name of the bill of materials group.
Quantity	Real	The quantity of the material or parent assembly.
Lot ID	String	Optional string value indicating the lot identifier of the material or parent assembly.
Off Shift Rule	Suspend Processing, Finish Work Already Started	<p>The processing rule used at the Combiner at the end of a shift.</p> <p>If the rule is 'Suspend Processing', then the Combiner will immediately suspend all processing and set its resource state to 'OffShift'. Processing will resume at the start of the next shift.</p> <p>If the rule is 'Finish Work Already Started', then the Combiner will not accept any new entities but will continue processing if necessary to finish work already started. The Combiner's resource state will be set to 'OffShiftProcessing' if processing entities during an off-shift period. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Resource Efficiency Rule	None, Average, Count, Maximum, Minimum, Sum	<p>If the rule is 'None', then seized resource efficiency is the processing rule used to alter the rate at which work is performed if there are seized resources with defined efficiency values. The actual work duration is the planned work duration divided by the efficiency.</p> <p>If the rule is 'Average', then the average seized resource efficiency value is used.</p> <p>If the rule is 'Count', then the number of seized resources with a defined efficiency value is used.</p> <p>If the rule is 'Maximum', then the largest efficiency value is used.</p>

		<p>If the rule is 'Minimum', then the smallest efficiency value is used.</p> <p>If the rule is 'Sum', then the sum of the seized resource efficiency values is used.</p>
Seize Constraint Logic	Repeat Group, Constraint Logic elements	<p>Constraint logic used to enforce additional constraints on an entity's request to seize the Combiner.</p> <p>NOTE: This repeat group applies only if the Combiner has an input buffer. Otherwise, if no input buffer, then this constraint logic will be ignored.</p>
Seize Constraint Logic.Constraint Logic Name	Constraint Logic elements	The name of the Constraint Logic element used to enforce additional constraints on an entity's request to seize the Combiner.
Immediately Try Seize	True, False	<p>For a parent entity that has collected its batch, indicates whether to immediately try seizing the Combiner before the execution of any other simulation logic in the system and, if successful, skipping the Combiner's allocation queue.</p> <p>Setting this property to False will just insert the seize request into the Combiner's allocation queue. An evaluation of that queue will then be scheduled on the simulation's current event calendar as a late priority event.</p>
Immediately Try Allocate When Released	True, False	<p>Once a parent entity has exited processing and released the Combiner, indicates whether to immediately try allocating the released Combiner capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the Combiner's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>
Parent Input Buffer Capacity	Integer ≥ 0	The number of discrete entities that can be held in this Combiner object's parent input buffer.
Balk Decision Type (Parent Input Buffer)	None, Blocked, Conditional, Probabilistic	<p>The method used by an entity to decide whether to balk at entering the buffer.</p> <p>If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.</p>
Balk Condition Or Probability (Parent Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Parent Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Member Input Buffer Capacity	Integer ≥ 0	The number of discrete entities that can be held in this Combiner object's member input buffer.
Balk Decision Type (Member Input Buffer)	None, Blocked, Conditional, Probabilistic	<p>The method used by an entity to decide whether to balk at entering the buffer.</p> <p>If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the</p>

		object is blocked.
Balk Condition Or Probability (Member Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Member Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Combiner object's output buffer.
Balk Decision Type (Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Output Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Reneged Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.
Trigger Type.Reneged Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Reneged Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Reneged Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Reneged Decision Type .Reneged Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Reneged Condition Or Probability.Reneged Triggers	Expression Specified if the <i>Reneged Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renege as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Reneged Node Name.Reneged Triggers	Node object reference	Facility node location to send a reneging entity.

Trigger Start Boundary.Renege Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
Action Type (Table Row Referencing - Before Processing)	Reference Existing Row, Add New Row	The type of table row reference action to perform. The 'Reference Existing Row' action type may be used to set a reference to an existing table row in a data table or sequence table. The row index into the table is specified by the <i>Row Number</i> property. The 'Add New Row' action type may be used to add a new row to an output table. A reference will be automatically set to the newly added row.
Table Name (Table Row Referencing - Before Processing)	Table Name	The name of the table.
Row Number (Table Row Referencing - Before Processing)	Expression	The one-based row index into the table.
Failure Type	No Failures, Calendar Time Based, Processing Count Based, Event Count Based, Processing Time Based	Specifies whether this Combiner object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample form a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
On Parent Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when a parent entity is entering the Combiner object.
On Member Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when a member entity is entering the Combiner object.
Before Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when a batch has been formed and the parent entity has been allocated capacity to be processed at the object, but before entering (or ending transfer into) the object's processing station.
After Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has completed its processing and is about to attempt its exit from the object's processing station.

Before Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the Combiner object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is reneging from an input or output buffer of the object.
Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Repeat Group (Secondary Resource for Processing)	True, False	Indicates whether a repeat group data structure is used to define the secondary resources for processing. If a repeat group structure is used, many of the below properties are specified within the repeat group. <i>OffShift Rule, Must Simultaneously Seize, Immediately Try Seize</i> and <i>Immediately Try Allocate When Released</i> are not within the repeat group, thus not resource specific.
Resource Type (Secondary Resource for Processing)	Specific, From List, ParentObject	The method for specifying the resource object(s) to seize.
Resource Name (Secondary Resource for Processing)	Object Instance Name	The name of the resource object to seize.
Resource List Name (Secondary Resource for Processing)	Object List Instance Name	The name of the object list from which to select the resource object(s) to seize.
Selection Goal (Secondary Resource for Processing)	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value, Random	The goal for selecting objects to seize.
Value Expression (Secondary Resource for Processing)	Expression	The expression evaluated for each object that is a candidate to seize. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., Candidate.Object.Capacity.Remaining).
Request Move (Secondary)	None, To Node	Indicates whether a move to a specified location will be requested from the seized resource. Processing will not be

Resource for Processing)		able to start until the resource has arrived to the requested location.
Destination Node (Secondary Resource for Processing)	Node Instance Name	The name of the specific node location that the seized resource will be requested to move to.
Off Shift Rule (Secondary Resource for Processing)	Finish Work Already Started, Suspend Processing, Switch Resources If Possible	<p>The processing rule used if the secondary resource is at the end of a shift because of a specified work schedule. If the rule is 'Finish Work Already Started', then the processing of the entity that is using the secondary resource will be allowed to continue processing until finished. The secondary resource will not be allowed to accept any new work.</p> <p>If the rule is 'Suspend Processing', then the processing of the entity that is using the secondary resource will be immediately suspended. Processing will resume at the start of the secondary resource's next shift.</p> <p>If the rule is 'Switch Resources If Possible', then the processing of the entity that is using the secondary resource will be immediately suspended. The entity will then try to resume processing as soon as possible by releasing the resource and seizing another available one that satisfies the same resource requirements. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Number of Resources (Secondary Resource for Processing)	Expression	The number of individual resource objects to seize capacity units of.
Units Per Resource (Secondary Resource for Processing)	Expression	The number of capacity units to seize per resource object.
Selection Condition (Secondary Resource for Processing)	Expression	<p>Optional condition evaluated for each candidate resource that must be true for the resource to be selected.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).</p>
Resource Efficiency (Secondary Resource for Processing)	Expression	<p>Optional value that can alter the rate at which work is performed using the seized resource(s), expressed as a fraction. The actual work duration is the planned work duration divided by the efficiency. Hence, an efficiency value greater than 1 shortens the time and a value less than 1 lengthens the time.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).</p> <p>Set this property to blank to declare the resource efficiency as undefined.</p>

Must Simultaneously Seize (Secondary Resource for Processing)	True, False	If multiple resources are required, indicates whether all of the resources must be available before any can be seized.
Immediately Try Seize (Secondary Resource for Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
Keep Reserved If (Secondary Resource for Processing)	Expression	On processing completion, an optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects without a reservation will be unable to seize the reserved capacity unless the reservation is cancelled. In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>). Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal. Leaving this property blank (no condition) is equivalent to entering False.
Reservation Timeout (Secondary Resource for Processing)	Expression	If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation. In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>).
Immediately Try Allocate When Released (Secondary Resource for Processing)	True, False	Once an entity has finished processing and released the resource, indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of an other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
On Parent Entering (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizures when an entity is entering the parent input buffer of the Combiner object.
Must Simultaneously Seize (Other Resource Seizes -- On Parent Entering)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try	True, False	Indicates whether to immediately try seizing the resource

Seize (Other Resource Seizes -- On Parent Entering)		before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizes once a batch's parent entity has collected its batch members and been allocated Combiner capacity, to be completed before starting the processing time.
Must Simultaneously Seize (Other Resource Seizes -- Before Processing)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- Before Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizes after a batch's parent entity has finished processing, to be completed before releasing the Combiner capacity.
Must Simultaneously Seize	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- After Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
On Parent Entering (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases to be performed when an entity is entering the parent input buffer of the Combiner object.
Immediately Try Allocate When Released (Other Resource Releases -- On Parent Entering)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases once a batch's parent entity has collected its batch members and been allocated Combiner capacity, before starting the

		processing time.
Immediately Try Allocate When Released (Other Resource Releases -- Before Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases after a batch's parent entity has finished processing, before releasing the Combiner capacity.
Immediately Try Allocate When Released (Other Resource Releases -- After Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.
Capital Cost	Expression	The initial one-time setup cost to add this object to the system.
Cost Per Use (Parent Input Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Parent Input Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Cost Per Use (Member Input Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Member Input Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Cost Per Use (Output Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Output Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of the combiner for each unutilized scheduled capacity unit.
Cost Per	Expression	The one-time cost that is accrued each time the combiner

Use(Resource Costs)		is used, regardless of the usage duration. This cost will be charged to the cost of the entity using the combiner.
Usage Cost Rate(Resource Costs)	Expression	The cost per unit time to use the combiner if in a utilized state. This cost will be charged to the cost of the entity using the combiner.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
ParentEntered Add-On Process	Process Instance Name	Occurs when an entity has entered the parent input buffer of this combiner object.
Member Entered Add-On Process	Process Instance Name	Occurs when an entity has entered the member input buffer of this combiner object.
Before Processing Add-On Process	Process Instance Name	Occurs when batch's parent entity has been allocated combiner capacity, but before entering (or ending transfer into) the object's processing station.
Processing Add-On Process	Process Instance Name	Occurs when a batch's parent entity has been allocated combiner capacity and is about to be removed from the parent input buffer and start processing.
After Processing Add-On Process	Process Instance Name	Occurs when a batch's parent entity has completed the processing time and is about to attempt its exit from the combiner's processing station in order to release the combiner capacity.
Exited Add-On Process	Process Instance Name	Occurs when a batch's parent entity has exited this combiner object.
Failed Add-On Process	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process	Process Instance Name	Occurs when a failure is repaired at this object.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Combiner object is evaluating whether to accept or reject a request to seize capacity of the combiner. In the executed decision process, assigning a value of less than or equal to '0' to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
On Shift Add-On Process	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process	Process Instance Name	Occurs when the object goes 'off shift' because of its specified work schedule.
Transfer-In Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.

Transfer-In Condition	Expression	The condition required to allow an entity to transfer into this fixed object via an external input node.
Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out Condition	Expression	The condition required to allow an entity to transfer out of this fixed object via an external output node.
Expected Setup Time Expression	Expression	The expression used to get a deterministic estimation of the setup time for an entity at this fixed object.
Expected Operation Time Expression	Expression	The expression used to get a deterministic estimation of the operation time for an entity at this fixed object.
Processing Task Sequence Random Number Stream	Expression	The random number stream to be used if there is probabilistic branching in the task sequence.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Edition). If not specified, then this property defaults to a color derived from the object's unique name.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Combiner - Discussion and Examples

Discussion

Batch Quantity

- If the *Batch Quantity* is set to an integer value X, the Combiner will wait until X members arrive to the member entry node before the batch is processed. This *Batch Quantity* does not include the parent entity arriving to the parent entry node.
- If the *Batch Quantity* is set to an expression based on the entity, for example 'ModelEntity.BatchSize', the parent entity's value for 'ModelEntity.BatchSize' is evaluated and not the member entity values. By using an entity property or state, the batch size that will be used is based on the entity, and therefore can vary.

Matching Rule

- If the *Matching Rule* is set to 'Any Entity', the Combiner will use the *Batch Quantity* property to determine how many entities to batch together and it will batch the first X entities that arrive to the Combiner, where X is the number specified in *Batch Quantity*.
- If the *Matching Rule* is set to 'Match Members', the Combiner will only batch member entities together if the value of the *Member Match Expression* is the same amongst all the member entities. If the *Member Match Expression* of an entity arriving to the member input buffer does not match the other *Member Match Expressions* of the other entities watching to be batched, the entity will wait until enough other entities arrive that provide a match for this value and the total number of matching entities equal the *Batch Quantity*.
- If the *Matching Rule* is set to 'Match Members and Parent', the Combiner will only batch member entities together if the value of the *Member Match Expression* is the same amongst all the member entities and the parent entity. If the *Member Match Expression* of an entity arriving to the member input buffer does not match the other *Member Match Expressions* of the other entities watching to be batched and the parent entity waiting to be batched, the entity will wait until enough other entities arrive that provide a match for this value and the total number of matching entities equal the *Batch Quantity*.

Hierarchical Batching

- Since a parent represents the container that holds batch members, if a parent arrives with an existing batch (e.g. the container is partly full) then the new batch will be added to the existing batch and the outgoing batch size of the parent will be the sum of the incoming batch size plus the additional members added. If you desire to build hierarchical batches (e.g., items batch into boxes, then boxes batch into pallets) it is important to remember that the batch items (boxes, in the example) must go into a combiner as members, with the container (a pallet in the example) as the new parent.

Fixed Capacity or Capacity Varying by Work Schedule

- *Fixed*

If the *Capacity Type* property is set to 'Fixed', the value of the initial capacity will be used to determine the capacity for the Combiner at the start of the simulation run. The capacity can be specified with an expression. If a random distribution is specified in the capacity expression, it will only be evaluated once at the beginning of the run and the capacity will remain unchanged. The Assign step can be used to change the Combiner's capacity during the simulation run by using the Combiner state, CombinerName.CurrentCapacity. This will change the value of the capacity function, CombinerName.Capacity.

- *WorkSchedule*

If the *Capacity Type* property is set to 'Work Schedule', a work schedule must be specified in the *Work Schedule* property. A work schedule is defined from the Schedules panel, which is found by clicking on the [Data tab](#).

Combiner has ResourceState State Variable that Tracks Its States

- The ResourceState state variable is a [List State](#) automatically tracked for the Combiner. The numeric values for the state are Starved=0, Processing=1, Blocked=2, Failed=3, OffShift=4, FailedProcessing=5, OffShiftProcessing=6, Setup=7 and OffShiftSetup=8. If the Combiner has a capacity greater than 1, it is considered Processing when one or more units of the Combiner are busy.

The *Off Shift Rule* property of the Combiner will determine what happens when a Combiner object is scheduled to go 'OffShift'. If the rule is 'Suspend Processing', then any processing will immediately be halted and the Combiner will go into an 'OffShift' state. If the rule is 'Finish Work Already Started' and the Combiner is in the process of performing a task, then it will first complete any remaining work while in the 'OffShiftProcessing' state before going into the 'OffShift' state. If the rule is 'Finish Work Already Started' and the Combiner is in the process of a setup time for a changeover, then it will first complete any remaining setup work while in the 'OffShiftSetup' state before going into the 'OffShift' state. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.

You can use the ResourceState in process logic to check whether a server is 'Processing', 'Starved' or other state. This would be done by utilizing the function Combiner.ResourceState in the logic, given that the token referencing this function is associated with the server. You will see Resource State results for the Combiner objects in the Results window.

To animate the combiner ResourceState, you simply specify the Combiner name and the desired function. For example, if you have a Combiner named Palletizer1, you may wish to place a Status Pie, specify the *Data Type* as 'ListState' and select Palletizer1.ResourceState from the *List State* property list.

Send comments on this topic to [Support](#)

Separator

A **Separator** represents a capacitated resource with one input buffers and two output buffers.

The Separator object contains four capacitated stations with associated queues; one station for the input buffer, one for the processing station, one for the parent output buffer and another station for the member output buffer. The four station approach allows for the capability to have both infinite and finite capacity queues. The [Balking and Reneging Options](#) page provides additional information and examples on buffer capacities, balking and reneging.

An entity arrives at the input node of the Separator. This station has a queue and a capacity. If the *Separation Mode* is set to 'Make Copies', the Separator will create the number of entities that is specified in *Copy Quantity*. The copies may be specified as a different entity type by using the *Copy Entity Type* property. The original entity will depart the Separator from the parent output node and the copies will depart from the member output node. Similarly, if the *Separation Mode* is set to 'Create New (No Copying)', the Separator will create a number of new entities that are specified in *New Entity Quantity*. The new entities may be specified as a different entity type by using the *New Entity Type* property. The original entity will depart the Separator from the parent output node and the newly created entities will depart from the member output node. If the *Separation Mode* is set to 'Split Batch', the Separator will unbatch the specified number of entities from the parent entity. The number is specified in *Desired Split Quantity*. The original entity, which is the parent entity, will depart from the parent node and the member entities depart from the member output node. The *Match Condition* property may be used to unbatch only certain entities from the batch based on a condition specified. For example, a *Match Condition* may be specified as 'Candidate.ModelEntity.Is.RedPart' where RedPart is the name of the entity. When unbatched from the parent, only those entities of type RedPart will be unbatched and any other members of the batch (BluePart, GreenPart, etc.) will remain with the batch. The *Removal Order* can be used to determine the order of searching for the *Desired Split Quantity* number of members to unbatch.

The Separator does not only unbatch entities, make copies of entities or make new entities, it also has process logic and capacity logic similar to a Server Object. If the entity is be processed by the Separator, it appears at the process station's queue while its being processed and then performs the logic of separating, copying or making new entities. After the logic of the process has completed, if there is space the entity will be moved to the output member queue.

Within the Processing station of the Separator, the *Process Type* may be a 'Specific Time' or based on a 'Task Sequence'. The default 'Specific Time' will delay the entity at the processing station of the Separator for the simple *Processing Time* specified. The 'Task Sequence' option will require a number of *Processing Tasks* and associated information to be specified. This is explained in greater detail within the [Task Sequence - Processing Tasks](#) section.

A Separator may have failures, which are specified within the [Reliability](#) section of the Properties window. [State assignments](#) can be made upon entering, before processing, after processing, before either parent or member exiting, on balking and on reneging the Separator. Secondary Resources may be seized and/or released by the entity upon entering the Separator, and before and after processing occurs. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Separator.

An entity processing at a Separator may be interrupted with the [Interrupt](#) step. The Processing Time delay is the only interruptible delay within the Separator.

The *Seize Constraint Logic* repeat group supports imposing additional constraints on an entity's request to seize the object by referencing [Constraint Logic](#) elements.

The *Immediately Try Seize* and *Immediately Try Allocate When Released* options within the Other Processing Options (and Secondary Resources area) section of properties are typically advanced users. They deal with processing events on the Simio event calendar that occur at the same time, including arrival of multiple entities to an object, or release of multiple objects at a time for re-allocation. It's important to note that these properties do not affect processing for entities that arrive on path with *Allow Passing* set to 'False', entities arriving via Conveyor, or any other construct that causes entities to arrive one at a time at different simulation times. This feature will not provide any benefit because the second (or following) entity will not arrive before the late event happens at the first entity's simulation time.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Separator.

For more information, see [Separator - Discussion and Examples](#).

Listed below are the properties of the **Separator**:

Property	Valid Entry	Description
Separation Mode	Create New (No Copying), Split	The method for separating the incoming entity. If 'Split

	Batch, Make Copies	Batch', then the Separator object will attempt to split off (unbatch) the desired number of batch members from the parent entity. If 'Make Copies', then the Separator object will make the specified number of copies of the original incoming entity. If 'Create New (No Copying)', then the Separator object will create the specified number of new entities without copying over any attributes of the original incoming entity
Desired Split Quantity	Expression	The desired number of batch members to split off (unbatch) from the parent entity.
Removal Order	LastBatchedFirst, FirstBatchedFirst	The order in which to search for and split off batch members from the parent entity.
Match Condition	Expression	Optional match condition used to filter the batch member entities. Only members that match the criteria given by the expression will be split off from the batch. In the condition, use the keyword 'Candidate' to explicitly reference a candidate entity in the batch members being searched (e.g., Candidate.Entity.Priority > 0).
Copy Quantity	Expression	The number of copies to make of the original incoming entity.
Copy Entity Type	Entity Name	The entity type of the created copies. If this property is not specified, then the same type as the original incoming entity will be assumed.
New Entity Quantity	Expression	The number of new entities to create.
New Entity Type	Entity Name	The entity type of the new entities to create.
Capacity Type	Fixed, Work Schedule	The method used to specify the number of batching operations that this Separator object can simultaneously process.
Initial Work Schedule	WorkSchedule name	The name of the work schedule initially assigned to the resource. Note: Refere to the SetWorkSchedule step to dynamically assign a new work schedule to a resource during a simulation run.
Work Schedule	WorkSchedule name	The capacity work schedule that this Separator object follows.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order entities waiting to be allocated capacity of this Separator object.
Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Dynamic Selection	None, or one of several	Indicates whether this object, when its capacity becomes

Rule	Dynamic Selection Rules	available, dynamically selects the next allocation request from its statically ranked allocation queue using a dynamic selection rule.
Repeat Group	True, False	Indicates whether a repeat group data structure is used to define the primary dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax <code>Candidate.[EntityClass].[Attribute]</code> to reference an attribute of the candidate entities (e.g., <code>Candidate.Entity.Priority</code>).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.
TransferIn Time	Expression	The time required to transfer an entity into this Separator object. This property may be specified using a random sample from a distribution.
Process Type	Speific Time, Task Sequence	The method used to model the processing of an entity at this Separator object.
Processing Time	Expression	The time required for this Separator object to process each entity. This property may be specified using a random sample from a distribution.
Processing Tasks	Repeat Group, Tasks	The set of tasks required for this Separator object to process an entity.
Loopback Branches	Repeat Group, Processing Tasks	Conditional or probabilistic loopback branches in the task workflow.
Task Precedence Method	SequenceNumberMethod, ImmediatePredecessorsMethod, ImmediateSuccessorsMethod	The method used to define the task precedence dependencies. Can be either by specifying task sequence numbers, specifying the immediate predecessors for each task, or specifying the immediate successors for each task.
Task Resource Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting task resource requirements.

Task Materials Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting task material requirements.
Task State Assignments Referenced Table Name	Table Name	Optional name of data table to be used as the referenced data source for getting state assignments.
Invalid Task Dependency Notification Type	Indicates the level of alert at runtime if the task sequence has missing, duplicate, or otherwise invalid data defining the task dependency relationships. Error - Throw runtime error. Warning - Throw runtime warning and skip the invalid data. Trace - Write to trace and skip the invalid data.	
Consumption Or Production Type	Material, Bill Of Materials, Bill Of Materials Group	The type of material stock consumption or production. Material - A single material. BillOfMaterials - A list of component materials. BillOfMaterialsGroup - Alternative lists of component materials. Material consumption will be required to start the processing of the task. Material production will occur after finishing the processing of the task.
Material Name	Material Reference	The name of the material.
BOM Name	Bill Of Materials Reference	The name of the bill of materials.
BOM Group Name	Bill Of Materials Group Reference	The name of the bill of materials group.
Quantity	Real	The quantity of the material or parent assembly.
Lot ID	String	Optional string value indicating the lot identifier of the material or parent assembly.
Off Shift Rule	Suspend Processing, Finish Work Already Started	<p>The processing rule used at the Separator at the end of a shift.</p> <p>If the rule is 'Suspend Processing', then the Separator will immediately suspend all processing and set its resource state to 'OffShift'. Processing will resume at the start of the next shift.</p> <p>If the rule is 'Finish Work Already Started', then the Separator will not accept any new entities but will continue processing if necessary to finish work already started. The Separator's resource state will be set to 'OffShiftProcessing' if processing entities during an off-shift period. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Resource Efficiency Rule	None, Average, Count, Maximum, Minimum, Sum	If the rule is 'None', then seized resource efficiency is the processing rule used to alter the rate at which work is performed if there are seized resources with defined efficiency values. The actual work duration is the planned

		<p>work duration divided by the efficiency.</p> <p>If the rule is 'Average', then the average seized resource efficiency value is used.</p> <p>If the rule is 'Count', then the number of seized resources with a defined efficiency value is used.</p> <p>If the rule is 'Maximum', then the largest efficiency value is used.</p> <p>If the rule is 'Minimum', then the smallest efficiency value is used.</p> <p>If the rule is 'Sum', then the sum of the seized resource efficiency values is used.</p>
Seize Constraint Logic	Repeat Group, Constraint Logic elements	<p>Constraint logic used to enforce additional constraints on an entity's request to seize the Separator.</p> <p>NOTE: This repeat group applies only if the Separator has an input buffer. Otherwise, if no input buffer, then this constraint logic will be ignored.</p>
Seize Constraint Logic.Constraint Logic Name	Constraint Logic elements	The name of the Constraint Logic element used to enforce additional constraints on an entity's request to seize the Separator.
Immediately Try Seize	True, False	<p>For an arriving entity, indicates whether to immediately try seizing the Separator before the execution of any other simulation logic in the system and, if successful, skipping the Separator's allocation queue.</p> <p>Setting this property to False will just insert the seize request into the Separator's allocation queue. An evaluation of that queue will then be scheduled on the simulation's current event calendar as a late priority event.</p> <p>NOTE: This property setting applies only if the Separator has an input buffer. Otherwise, if no input buffer, then an arriving entity will always immediately try to enter the Separator's processing station and seize it.</p>
Immediately Try Allocate When Released	True, False	<p>Once an entity has exited processing and released the Separator, indicates whether to immediately try allocating the released Separator capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the Separator's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>
Input Buffer Capacity	Integer ≥ 0	The number of discrete entities that can be held in this Separator object's input buffer.
Balk Decision Type (Input Buffer)	None, Blocked, Conditional, Probabilistic	<p>The method used by an entity to decide whether to balk at entering the buffer.</p> <p>If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.</p>
Balk Condition Or Probability (Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.

Parent Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Separator object's parent output buffer.
Balk Decision Type (Parent Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Parent Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Parent Output Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Member Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Separator object's member output buffer.
Balk Decision Type (Member Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Member Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Member Output Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Renegé Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.
Trigger Type.Renegé Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Renegé Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Renegé Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Renegé Decision Type .Renegé Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Renegé Condition Or	Expression Specified if the <i>Renegé Decision Type</i> is	The renege condition or probability specified as an expression. If a probability then enter the chance of

Probability.Renege Triggers	'Conditional' or 'Probabilistic'.	reneging as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Renege Node Name.Renege Triggers	Node object reference	Facility node location to send a reneging entity.
Trigger Start Boundary.Renege Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
Failure Type	No Failures, Calendar Time Based, Event Count Based, Event Count Based, Processing Time Based	Specifies whether this Separator object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Action Type (Table Row Referencing - Before Processing)	Reference Existing Row, Add New Row	The type of table row reference action to perform. The 'Reference Existing Row' action type may be used to set a reference to an existing table row in a data table or sequence table. The row index into the table is specified by the <i>Row Number</i> property. The 'Add New Row' action type may be used to add a new row to an output table. A reference will be automatically set to the newly added row.
Table Name (Table Row Referencing - Before Processing)	Table Name	The name of the table.
Row Number (Table Row Referencing - Before Processing)	Expression	The one-based row index into the table.
On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the Separator object.
Before Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has been allocated capacity to be processed at the object, but before entering (or ending transfer into) the object's processing station.

After Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has completed its processing and is about to do the separation logic, but before attempt its exit from the object's processing station.
Before Parent Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when a parent entity is ready to exit the Separator object.
Before Member Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when a separated member entity is ready to exit the Separator object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is reneging from an input or output buffer of the object.
Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Repeat Group (Secondary Resource for Processing)	True, False	Indicates whether a repeat group data structure is used to define the secondary resources for processing. If a repeat group structure is used, many of the below properties are specified within the repeat group. <i>OffShift Rule</i> , <i>Must Simultaneously Seize</i> , <i>Immediately Try Seize</i> and <i>Immediately Try Allocate When Released</i> are not within the repeat group, thus not resource specific.
Resource Type (Secondary Resource for Processing)	Specific, From List, ParentObject	The method for specifying the resource object(s) to seize.
Resource Name (Secondary Resource for Processing)	Object Instance Name	The name of the resource object to seize.
Resource List Name (Secondary Resource for Processing)	Object List Instance Name	The name of the object list from which to select the resource object(s) to seize.
Selection Goal (Secondary Resource for Processing)	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value, Random	The goal for selecting objects to seize.

Value Expression (Secondary Resource for Processing)	Expression	The expression evaluated for each object that is a candidate to seize. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., Candidate.Object.Capacity.Remaining).
Request Move (Secondary Resource for Processing)	None, To Node	Indicates whether a move to a specified location will be requested from the seized resource. Processing will not be able to start until the resource has arrived to the requested location.
Destination Node (Secondary Resource for Processing)	Node Instance Name	The name of the specific node location that the seized resource will be requested to move to.
Off Shift Rule (Secondary Resource for Processing)	Finish Work Already Started, Suspend Processing, Switch Resources If Possible	<p>The processing rule used if the secondary resource is at the end of a shift because of a specified work schedule. If the rule is 'Finish Work Already Started', then the processing of the entity that is using the secondary resource will be allowed to continue processing until finished. The secondary resource will not be allowed to accept any new work.</p> <p>If the rule is 'Suspend Processing', then the processing of the entity that is using the secondary resource will be immediately suspended. Processing will resume at the start of the secondary resource's next shift.</p> <p>If the rule is 'Switch Resources If Possible', then the processing of the entity that is using the secondary resource will be immediately suspended. The entity will then try to resume processing as soon as possible by releasing the resource and seizing another available one that satisfies the same resource requirements. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Number of Resources (Secondary Resource for Processing)	Expression	The number of individual resource objects to seize capacity units of.
Units Per Resource (Secondary Resource for Processing)	Expression	The number of capacity units to seize per resource object.
Selection Condition (Secondary Resource for Processing)	Expression	<p>Optional condition evaluated for each candidate resource that must be true for the resource to be selected.</p> <p>In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource objects (e.g., Candidate.Object.Capacity).</p>
Resource Efficiency (Secondary Resource for Processing)	Expression	Optional value that can alter the rate at which work is performed using the seized resource(s), expressed as a fraction. The actual work duration is the planned work

Resource for Processing)		<p>duration divided by the efficiency. Hence, an efficiency value greater than 1 shortens the time and a value less than 1 lengthens the time.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).</p> <p>Set this property to blank to declare the resource efficiency as undefined.</p>
Must Simultaneously Seize (Secondary Resource for Processing)	True, False	If multiple resources are required, indicates whether all of the resources must be available before any can be seized.
Immediately Try Seize (Secondary Resource for Processing)	True, False	<p>Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues.</p> <p>Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.</p>
Keep Reserved If (Secondary Resource for Processing)	Expression	<p>On processing completion, an optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects without a reservation will be unable to seize the reserved capacity unless the reservation is cancelled.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>).</p> <p>Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p> <p>Leaving this property blank (no condition) is equivalent to entering False.</p>
Reservation Timeout (Secondary Resource for Processing)	Expression	<p>If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>).</p>
Immediately Try Allocate When Released (Secondary Resource for Processing)	True, False	<p>Once an entity has finished processing and released the resource, indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>
On Parent Entering (Other Resource)	Repeat Group, Resource Seizes	Optional secondary resource seizes when an entity is entering the Separator object.

Seizes)		
Must Simultaneously Seize (Other Resource Seizes -- On Entering)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- On Entering)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizes once an entity has been allocated Separator capacity, to be completed before starting the processing time.
Must Simultaneously Seize (Other Resource Seizes -- Before Processing)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- Before Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Optional secondary resource seizes after an entity has finished processing, to be completed before releasing the Separator capacity and then executing the separation logic.
Must Simultaneously Seize	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- After Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
On Parent Entering (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases to be performed when an entity is entering the Separator object.
Immediately Try Allocate When Released (Other	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the

Resource Releases -- On Entering)		system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases once an entity has been allocated Separator capacity, before starting the processing time.
Immediately Try Allocate When Released (Other Resource Releases -- Before Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Releases)	Repeat Group, Resource Releases	Optional secondary resource releases after an entity has finished processing, before releasing the Separator capacity and then executing the separation logic.
Immediately Try Allocate When Released (Other Resource Releases -- After Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.
Capital Cost	Expression	The initial one-time setup cost to add this object to the system.
Cost Per Use (Parent Output Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Parent Output Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Cost Per Use (Member Output Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Member Output Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of the separator for each unutilized scheduled capacity unit.
Cost Per Use(Resource	Expression	The one-time cost that is accrued each time the separator is used, regardless of the usage duration. This cost will be

Costs)		charged to the cost of the entity using the separator.
Usage Cost Rate(Resource Costs)	Expression	The cost per unit time to use the separator if in a utilized state. This cost will be charged to the cost of the entity using the separator.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs when an entity has entered the input buffer of this separator object.
Before Processing Add-On Process	Process Instance Name	Occurs when an entity has been allocated separator capacity, but before entering (or ending transfer into) the object's processing station.
Processing Add-On Process	Process Instance Name	Occurs when an entity has been allocated separator capacity and is about to start processing.
After Processing Add-On Process	Process Instance Name	Occurs when an entity has completed the processing time and is about to do the separation logic, before attempting its exit from the separator's processing station in order to release the separator capacity.
Parent Exited Add-On Process	Process Instance Name	Occurs when a parent entity has exited this separator object.
Member Exited Add-On Process	Process Instance Name	Occurs when a separated member entity has exited this separator object.
Failed Add-On Process	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process	Process Instance Name	Occurs when a failure is repaired at this object.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Separator object is evaluating whether to accept or reject a request to seize capacity of the separator. In the executed decision process, assigning a value of less than or equal to '0' to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
OnShift Add-On Process	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
OffShift Add-On Process	Process Instance Name	Occurs when the object goes 'off shift' because of its specified work schedule.
Transfer-In Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.
Transfer-In	Expression	The condition required to allow an entity to transfer into

Condition		this fixed object via an external input node.
Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out Condition	Expression	The condition required to allow an entity to transfer out of this fixed object via an external output node.
Expected Setup Time Expression	Expression	The expression used to get a deterministic estimation of the setup time for an entity at this fixed object.
Expected Operation Time Expression	Expression	The expression used to get a deterministic estimation of the operation time for an entity at this fixed object.
Processing Task Sequence Random Number Stream	Expression	The random number stream to be used if there is probabilistic branching in the task sequence.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Edition). If not specified, then this property defaults to a color derived from the object's unique name.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Separator - Discussion and Examples

Discussion

Separation Mode

- If the *Separation Mode* is set to 'Split Batch', the Separator will use the *Split Quantity* property to determine how many entities to separate. By default, *Entity.BatchMembers* is specified in the *Split Quantity* property because this holds the number of member entities in the batch if the entity is a parent entity. However, any value can be specified in the *Split Quantity* property.
- If the *Separation Mode* is set to 'Make Copies', the Separator will make X number of copies of the parent entity, where X is the quantity specified in *Copy Quantity*. The original entity will depart through the parent output node and the new copies will depart through the member output node. The copies may be specified as a different entity type than the original, by specifying a *Copy Entity Type* property. If the entities are of a different type, the much information from the original entity is copied to the newly created entities. This includes all table references assigned to the original, all the original object's state values, and the original's assigned network and destination. If the original is following a sequence table, the current sequence table index is copied as well. Note that if the entity type of the copy is different than the original's type, then only the state values of common inherited object states can be copied. Also, the size state values will not be copied and thus a copy of a different type will be sized according to the default size for that different type.

Fixed Capacity or Capacity Varying by Work Schedule

- *Fixed*

If the *Capacity Type* property is set to 'Fixed', the value of the initial capacity will be used to determine the capacity for the Separator at the start of the simulation run. The capacity can be specified with an expression. If a random distribution is specified in the capacity expression, it will only be evaluated once at the beginning of the run and the capacity will remain unchanged. The Assign step can be used to change the Separator's capacity during the simulation run by using the Separator state, *SeparatorName.CurrentCapacity*. This will change the value of the capacity function, *SeparatorName.Capacity*.

- *WorkSchedule*

If the *Capacity Type* property is set to 'Work Schedule', a work schedule must be specified in the *Work Schedule* property. A work schedule is defined from the Schedules panel, which is found by clicking on the [Data tab](#).

Separator has ResourceState State Variable that Tracks Its States

- The ResourceState state variable is a [List State](#) automatically tracked for the Separator. The numeric values for the state are Starved=0, Processing=1, Blocked=2, Failed=3, OffShift=4, FailedProcessing=5, OffShiftProcessing=6, Setup=7 and OffShiftSetup=8. If the Separator has a capacity greater than 1, it is considered Processing when one or more units of the Separator are busy.

The *Off Shift Rule* property of the Separator will determine what happens when a Separator object is scheduled to go 'OffShift'. If the rule is 'Suspend Processing', then any processing will immediately be halted and the Separator will go into an 'OffShift' state. If the rule is 'Finish Work Already Started' and the Separator is in the process of performing a task, then it will first complete any remaining work while in the 'OffShiftProcessing' state before going into the 'OffShift' state. If the rule is 'Finish Work Already Started' and the Separator is in the process of a setup time for a changeover, then it will first complete any remaining setup work while in the 'OffShiftSetup' state before going into the 'OffShift' state. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.

You can use the ResourceState in process logic to check whether a separator is 'Processing', 'Starved' or other state. This would be done by utilizing the function *Separator.ResourceState* in the logic, given that the token referencing this function is associated with the separator. You will see Resource State results for the Separator objects in the Results window.

To animate the separator ResourceState, you simply specify the Separator name and the desired function. For example, if you have a Separator named *Depalletizer1*, you may wish to place a Status Pie, specify the *Data Type* as

'ListState' and select Depalletizer.ResourceState from the *List State* property list.

The ResourceState variable is used to display utilization statistics for the Separator in the Results window. Additionally, this list state can be used to display utilization statistics during the simulation run. For example, you may wish to show the percentage of time that a separator is processing during the simulation run. This would be done by placing a status label with the expression SeparatorName.ResourceState.PercentTime(1). For more information on list state values and utilization statistics, see [List State](#) page.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Vehicle

Vehicles

A Vehicle object may be used to define a dynamic population of moveable unit resources in the modeled system. Each individual vehicle member in the population is tracked separately, in terms of both animation and statistics.

Vehicles are modeling objects that can pickup entity objects at a location, carry those objects through a network of links or free space, then drop the entities off at a destination location. This object also has the ability to move off of a network while maintaining association with a node on the network (i.e., park at a node in a network). You can create other transporter models that behave differently than the standard Vehicle model; e.g. you could create an object definition for a subway car, AGV, or bus.

An 'On Demand' Vehicle object may be used as a moveable resource that is seized and released by model logic for non-transport tasks. A 'Fixed Route' vehicle will never be available to be seized, as the behavior of that type of vehicle is designed to maintain its specified fixed routing sequence and only do pickups and drop-offs at the locations specified along the route. If an 'On Demand' Vehicle object is being used for both transport and non-transport tasks, then by default, the non-transport tasks will always be prioritized before transport tasks.

If a Vehicle's CurrentTravelMode is 'Network If Possible', then the vehicle will accept any node in the same Facility view as a valid destination for pickups, drop-offs, processing tasks, etc. (because the Vehicle's travel movements will be able to be performed using either its currently assigned network or in free space).

You can define the *Initial Number In System* for the Vehicle population. You may also use the Create step or Source object to dynamically create and add new members to the population during the run. However, note that you cannot dynamically Destroy a Vehicle because resource objects in Simio are not destroyable.

A Vehicle may have failures, which are specified within the [Reliability](#) section of the Properties window. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Vehicle.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Vehicle.

Note: When auto-creating Vehicles from a data table, you must specify the Transporter Reference Column as a Key Column.

For more information, see [Vehicle - Discussion and Examples](#).

Listed below are the properties of the **Vehicle**:

Property	Valid Entry	Description
Initial Ride Capacity	Integer	The initial carrying capacity of this Vehicle object.
Task Selection Strategy	Smallest Distance, Largest Distance, Smallest Priority, Largest Priority, First In Queue	The strategy used by the Vehicle object to select its next transport pickup or dropoff task.
Load Time	Expression	The time required for this Vehicle object to load an entity (i.e., the load time per entity).
Unload Time	Expression	The time required for this Vehicle object to unload an entity (i.e., the unload time per entity).
Park To Load/Unload	True or False	Indicates whether this Vehicle object should park to load and unload entities at a node. Parking a vehicle can be useful to avoid blocking other travelers on the network.
Minimum	Dwell Until	Specifies the minimum dwell time requirement for this Vehicle object when

Dwell Time Type	Event, Dwell Until Full, No Requirement, Specific Time	loading and unloading entities at a node. A 'minimum dwell time' is a minimum amount of time the vehicle is required to wait, or 'dwell', at a node to load and unload entities.
Event Name	Event Name	The name of the event that will indicate the expiration of the minimum wait time requirement for this Vehicle object when loading and unloading entities at a node.
Minimum Dwell Time	Expression	The specific minimum amount of time that this Vehicle object is required to wait, or 'dwell', at a node to load and unload entities.
Dwell Only If	Expression	If specified, this condition will be used to indicate whether the vehicle must respect the minimum dwell time at a node if that requirement has not yet expired, thereby forcing the vehicle to wait, or 'dwell', at the node for possible additional loads. Example uses of this property might include allowing the vehicle to exit a node before the minimum dwell time has expired if the vehicle has reached full ride capacity, or if there is little to no chance of additional loads. Another example condition might be to only respect the minimum dwell time requirement if the vehicle is at a node meeting some specific criteria.
Initial Desired Speed	Expression	The initial desired speed value for this Vehicle object when traveling between locations.
Initial Travel Mode	Free Space Only, Network Only, Network If Possible	<p>The initial travel mode for entities of this type.</p> <p>'Free Space Only' indicates that the entities are required to perform travel movements in free space.</p> <p>'Network Only' indicates that the entities are required to perform travel movements using the currently assigned network.</p> <p>'Network If Possible' indicates a preference for the entities to perform travel movement using the currently assigned network, but if no followable network path exists to the next destination then travel in free space is allowed.</p> <p>NOTE: Refer to the user-assignable 'CurrentTravelMode' state variable of an entity to dynamically change its travel mode during a simulation run. An entity's travel mode may also be changed using the 'Outbound Travel Mode' property provided by a node.</p>
Initial Network	Network Instance Name	The initial network of links used by this Vehicle object to travel between node locations.
Network Turnaround Method	Exit & Re-enter, Rotate in Place, Reverse	When traveling on a network, the default method used by objects of this type to turn around at a node in order to move in the opposite direction on a bidirectional link. The 'Turnaround Method' property of a Network element may also be used to override this default on a network-by-network basis.
Free Space Steering Behavior	Direct To Destination, Follow Network Path If Possible	<p>The behavior used to steer an entity of this type when traveling in free space to a destination.</p> <p>'Direct To Destination' will steer an entity in a straight line to its destination.</p> <p>'Follow Network Path If Possible' will prefer to steer an entity along a path following its currently assigned network, staying within the boundaries of the drawn path width. The Entity will travel based on the drawn length of the path even if a different logical length is specified. However, if no followable network path exists to the entity's destination then the entity will be steered in a straight line. Note: In order to use this steering behavior, make sure the entity's travel mode is set to 'Free Space Only'.</p>
Update Interval	Expression	The time interval between updates checking an entity's adherence to the network path.
Avoid Collisions	True or False	Indicates whether the steering behavior will attempt to avoid collisions with other entities that are following the same network path.

Initial Priority	Expression	The initial priority value of this Vehicle object at the beginning of the simulation run.
Initial Node (Home)	Node Instance Name	The initial node location of this Vehicle object at the beginning of the simulation run. Also indicates the vehicle's 'Home' location if the object's 'Idle Action' is specified as 'Park At Home' or 'Go To Home'.
Routing Type	On Demand, Fixed Route	Specifies the routing behavior of this Vehicle. If 'On Demand', then the vehicle will route to locations on the network according to visit requests. An 'On Demand' vehicle may also be used as a moveable resource that is seized and released for non-transport related tasks by model process logic. If 'Fixed Route', then the vehicle will follow a fixed route sequence to transport riders and will only load and unload entities at locations on the specified route.
Route Sequence	Sequence Instance Name	The sequence table that defines the route sequence that this vehicle will loop.
Idle Action	Park At Node, Go to Home, Remain In Place, Park At Home, Roam	The action of this Vehicle object when it does not have any transportation tasks to perform.
Off Shift Action	Park At Node, Go to Home, Remain In Place, Park At Home	The action of this Vehicle object when it goes 'off shift'. Note that, if the vehicle is busy or transporting when an off shift period begins, then this action will be taken when the vehicle is released from (i.e., finishes) the remaining tasks.
Capacity Type	Fixed, Work Schedule	The availability of this vehicle type to perform tasks. 'Fixed' indicates that this type of vehicle is always available. 'WorkSchedule' indicates that each vehicle of this type follows a work schedule defining its 'On-shift', 'Off-shift' availability over time.
Initial Work Schedule	WorkSchedule name	The name of the work schedule initially assigned to the resource. Note: Refere to the SetWorkSchedule step to dynamically assign a new work schedule to a resource during a simulation run.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order requests waiting to seize this Vehicle object for a process task. Note: This feature only applies to an 'On Demand' routing type vehicle.
Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' static ranking rule for ordering requests waiting to be allocated capacity of this object.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this vehicle, when it becomes available, dynamically selects the next seize request from its statically ranked allocation queue using a dynamic selection rule. Note: This feature only applies to an 'On Demand' routing type vehicle.

Repeat Group	True, False	Indicates whether a repeat group data structure is used to define the primary dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (e.g., Candidate.Entity.Priority).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.
Park While Busy	True or False	Indicates whether this Vehicle object should park while busy at a node due to a seize request. Parking a vehicle can be useful to avoid blocking other travelers on the network.
Failure Type	No Failures, Calendar Time Based, Event Count Based	Specifies whether this Vehicle object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The event count between failure occurrences.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.

Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to vehicles of this type are rolled up into. If a parent cost center is not explicitly specified, then costs allocated to a vehicle will be automatically rolled up into the parent object that contains the vehicle's current location.
Capital Cost Per Vehicle	Expression	The initial one-time setup cost to add a vehicle of this type to the system.
Cost Per Rider (Transport Costs)	Expression	The cost to load and transport an entity using a transporter of this type, irrespective of the transportation time. This cost is allocated to an entity when it is loaded onto the transporter.
Transport Cost Rate (Transport Costs)	Expression	The cost per unit time to transport an entity using a transporter of this type.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of a vehicle of this type when it is idle.
Cost Per Use(Resource Costs)	Expression	The one-time cost that is accrued each time a vehicle of this type is used for a non-transport task, regardless of the usage duration. This cost will be charged to the cost of the owner object(i.e., task) using the vehicle.
Usage Cost Rate(Resource Costs)	Expression	The cost per unit time to use a vehicle of this type for a non-transport task. This cost will be charged to the cost of the owner object (i.e., task) using the vehicle.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized. Note: This trigger will only occur for Vehicle objects created by the Initial Number in System setting.
Run Ending Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is ending.
Allocated Add-On Process	Process Instance Name	Occurs when this Vehicle resource has been allocated to perform some transport tasks, or when another object has seized this Vehicle resource for some non-transport related task.

Triggers		
Released Add-On Process Triggers	Process Instance Name	Occurs when this Vehicle resource has been released after completing a set of transport tasks, or when another object has released this Vehicle resource from some non-transport related task.
Failed Add-On Process Triggers	Process Instance Name	Occurs when the vehicle has failed.
Repaired Add-On Process Triggers	Process Instance Name	Occurs when a failure is repaired at the vehicle.
Entered Node Add-On Process Triggers	Process Instance Name	Occurs when this vehicle has entered a node.
Unloaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is unloaded from this vehicle.
Loaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is loaded onto the vehicle.
Exiting Node Add-On Process Triggers	Process Instance Name	Occurs when this vehicle object is about to begin exiting a node.
Evaluating Transport Request Add-On Process Triggers	Process Instance Name	Occurs when this vehicle is evaluating whether or not to accept the pickup of a rider. In the executed decision process, assigning the value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the potential rider is rejected.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when an 'On Demand' routing type Vehicle is evaluating whether to accept or reject a seize allocation attempt by some process in the system. In the executed decision process, assigning a value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the allocation attempt is rejected.
On Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'off shift' because of a specified work schedule.
Initial Number in System	Integer	The initial number of Vehicle objects of this type in the system. Vehicles are initially located in free-space at the object instance location.
Maximum	Integer	The maximum number of objects of this instance that can be simultaneously

Number in System		in the system. If this limit is exceeded then a runtime error is generated.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Edition). If not specified, then this property defaults to a color derived from the object's unique name.
Can Transfer In & Out of Objects	True or False	Indicates whether objects of this type are by default allowed to transfer into and out of fixed objects via those object's external input / output nodes.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Vehicle - Discussion and Examples

Discussion

First Steps in Using a Standard Vehicle Object

- Drag and drop a Vehicle object from the Standard Library window into the Facility Window. If you want the vehicle to ride on a specific Network of Paths, you must change the associated Network property of the Vehicle.
- There is one key property of the Vehicle object that must be set. Although all nodes and links are part of the Global network by default, until an initial position is specified for a Transporter, it is not accessible on the network. Specify a starting or "home" node in the *Initial Node (Home)* property.
- Vehicles can serve as Transporters which move entities to and from locations. Somehow, the model has to be told to have the entities "ride" on the vehicle. In order to have the vehicles move the entities from one Node to another, the Transfer Node has to be configured to have the entities leave on the vehicle. Select the node by left-clicking on the Transfer Node of an object, such as a Source or Server. Then, change the *Ride On Transporter* property in the Properties window to 'Always' or 'Conditional'. Setting the *Ride On Transporter* property to 'Conditional' will allow you to specify a condition that when true will require a Transporter to ride on. Next, expand the *Transporter Type* by clicking on the + and change the value in the *Transporter Name* property to the determined vehicle's name.

Fixed Capacity or Capacity Varying by Work Schedule

- Regardless of whether a Vehicle object has a Fixed Capacity or Capacity Varying by Work Schedule, the number of vehicles available to the system is based on the *Initial Number In System* property, shown under the Population section. The *Initial Ride Capacity* specifies the number of objects that the particular vehicle may move at any time. If the *Vehicle.CurrentCapacity* is manually assigned to a value other than 0 or 1, a warning will be displayed (if warnings are not disabled) and the capacity will automatically adjusted to the value of '1'. In the SimBit, [DynamicallyCreatingVehicles](#), the Create step is used to create a new vehicle into the system, automatically created with a capacity of 1.

- *Fixed*

If the *Capacity Type* property is set to 'Fixed', then each of the *Initial Number In System* vehicles will have a fixed capacity during the simulation run; that is, no are breaks or off-shift time.

- *WorkSchedule*

If the *Capacity Type* property is set to 'Work Schedule', a work schedule must be specified in the *Work Schedule* property. A work schedule is defined from the Schedules panel, which is found by clicking on the [Data tab](#). This work schedule will be utilized for each of the *Initial Number in System* vehicles to allow them to go off-shift for breaks and/or off-shift time. The schedule is intended to be used for On-Shift and Off-Shift times for the Vehicle. The *Value* property within the schedule should always be '1', as the capacity of the Vehicle is set through the *Initial Number in System*.

Loading and Unloading Times

- Load and unload times are incurred per entity. For example, if the capacity of a Vehicle is 2, up to two waiting entities may be picked up to be moved. If two entities are waiting for pickup, the first one will be picked up, and incur its loading time. Before moving, the second entity is picked up and incurs its loading time. Thus, the first entity incurs its own load time, plus the load time of the second entity before the vehicle starts the transfer.
- If the *Park to Load/Unload* property is specified as 'True', the vehicle(s) will be placed in the parking area of the node at which it is loading or unloading. There are two ways to animate the parking area. First, click on a specific node and then select the Parking Queue button from the Appearance ribbon. Alternatively, click on a specific node and select Draw Queue / ParkingStation.Contents and draw the animated queue.

How the Task Selection Strategy Property Works

- When Vehicle object searches a RidePickupQueue for the next pickup at a node, or searches the global VisitRequestQueue for a ride reservation, or searches its RideStation to determine its next dropoff destination, it will use the option specified in the Task Selection Strategy property to determine how to choose from the appropriate queue.

Vehicle has ResourceState State Variable that Tracks Its States

- The ResourceState state variable is a [List State](#) automatically tracked for the Vehicle, identical to the ResourceState state provided by all of the processor objects in the standard library (Server, Resource, Combiner, etc.). The numeric values for the state are Idle=0, Busy=1, Failed=3, FailedBusy=5, Transporting=7. A Vehicle is considered busy when it is Seized by a token using a Seize step. The vehicle is considered transporting when it is utilized for a movement type task (either moving to the entity for transport or moving with the entity to its destination location).

You can use the ResourceState in process logic to check whether a vehicle is 'Busy' or 'Idle'. This would be done by utilizing the function Vehicle.ResourceState in the logic, given that the token referencing this function is associated with the vehicle. You might also easily change the symbol animation on the vehicle to reflect its current state (e.g., show 'Busy' versus 'Idle' symbols) by using the Vehicle.ResourceState function in the *Current Symbol Index* property of the vehicle. You will see Resource State results for the vehicle objects in the Results window.

To animate the vehicle ResourceState, you must explicitly reference the index of the vehicle within the dynamic population. For example, if you have a vehicle with the *Name* 'Vehicle1' and the *Initial Number In System* property for the vehicle set to '2', two Status Pies may be placed and the List States can be referenced as Vehicle1[1].ResourceState and Vehicle1[2].ResourceState.

The ResourceState variable is used to display utilization statistics for the Vehicle in the Results window. Additionally, this list state can be used to display utilization statistics during the simulation run. For example, you may wish to show the percentage of time that a vehicle is doing transport type tasks during the simulation run. This would be done by placing a status label with the expression VehicleName[unit#].ResourceState.PercentTime(7). For more information on list state values and utilization statistics, see [List State](#) page.

Seizing an 'On-Demand' Vehicle for Non-Transport Tasks

- You may use a Seize step to seize an 'On Demand' vehicle and then have that vehicle move to a requested location. It might delay for some processing time, then release the vehicle using the Release step. This allows the same 'On Demand' Vehicle object to be utilized for both transport as well as non-transport tasks. Note that a 'Fixed Route' vehicle will never be available to be seized, as the behavior of that type of vehicle is designed to maintain its specified fixed routing sequence and only do pickups and drop-offs at the locations specified along the route.

If an 'On Demand' Vehicle object is being used for both transport and non-transport tasks, then note that by default non-transport tasks will always be prioritized before transport tasks.

To prioritize specifically among transport tasks, specify the *Task Selection Strategy* in the Transport Logic category of the Vehicle object's properties. To prioritize specifically among non-transport tasks, you may use the *Ranking Rule* or *Dynamic Selection Rule* properties available in the Process Logic category of the Vehicle object's properties. If you want to implement even more customized control of how an 'On Demand' Vehicle object selects its next task then you may take advantage of the *Evaluating Transport Request* and *Evaluating Seize Request* Add-On process triggers. For example, you can use those Add-On processes to implement logic on a particular vehicle such as "Don't accept any non-transport tasks until all my transport tasks are finished first".

If the *Park While Busy* property is specified as 'True', the vehicle(s) will be placed in the parking area of the node at which it is processing. There are two ways to animate the parking area. First, click on a specific node and then select the Parking Queue button from the Appearance ribbon. Alternatively, click on a specific node and select Draw Queue / ParkingStation.Contents and draw the animated queue.

Referring to a Specific Vehicle within the Dynamic Population

- You may use a bracket and number to explicitly refer to a specific unit of the dynamic population of vehicles in the system, for example, Vehicle1[1].FunctionName. Therefore, if you have a vehicle with the *Name* 'Vehicle1' and the *Initial Number In System* property for the vehicle set to '2', they can be referenced as Vehicle1[1] and Vehicle1[2] with regards to functions. For example, within an expression, you may wish to evaluate a function of a specific unit, 'Vehicle1[1].RideStation.Capacity.Remaining > 3' within a Decide step. You may use any agent instance name, followed by brackets, with an expression in those brackets giving the index into the agent population, for example, Vehicle1[1].Function, Vehicle[1+0].Function or Vehicle1[Vehicle1[x].SiblingVehicleIndex].Function.

Network Turnaround Method and Bidirectional Paths

- The *Network Turnaround Method* property for an entity, worker or vehicle is used in conjunction with bidirectional paths. By default, the 'Exit & Re-enter' method is used. With this method, the entity that wants to turn around will always immediately first exit the link (moving essentially into free-space at the node) before attempting to re-enter. And, if the entity cannot immediately re-enter, then it will wait in the link's EntryQueue. The other two turnaround methods are 'Rotate in Place' and 'Reverse'. With both of these methods, the entity that wants to turn around will not leave the link, but will simply turn around and move in the opposite direction. 'Rotate in Place' will always keep the

'head' of the entity at the front, whereas 'Reverse' will appear that the entity is moving backwards. For more information regarding the Turnaround Method, see the [Path - Discussion and Examples](#) page.

Initial Node (Home)

- The *Initial Node (Home)* property determines where the vehicle(s) will initially be located at the start of the simulation run. This node name is stored in the HomeNode state of the vehicle and can be assigned during the simulation run to change the vehicle's home location. This would be done by assigning a *State Variable Name* of 'Vehicle.HomeNode' to a *New Value* of 'BasicNode1', for example, if the associated object is the vehicle. Similarly, the user may reference a specific vehicle by specifying *State Variable Name* as 'ForkTruckA[1]' and *New Value* as 'BasicNode1'.

Vehicles and Deadlocking

- Vehicles traveling along bidirectional paths have the potential to deadlock, causing the vehicles to appear to hang at a given node location. Please see the discussion on the [Path - Discussion and Examples](#) page, as well as the [Warning Level](#) page.

Minimum Dwell Time

- Both the Vehicle and Worker objects have the ability to 'dwell', or wait, at a node for a specified period of time. There are several options for dwelling and a condition that may also be specified.

'No Requirement' for the *Minimum Dwell Time Type* indicates that the vehicle will not wait at a node before moving on. This is the default behavior of both Vehicles and Workers.

'Dwell Until Event' indicates that the specified event must be fired before the vehicle moves from the node to its next destination. This may be useful in cases where the vehicle moves only when a certain condition is met and an event is fired from an upstream location. Note that the event used can be different from node to node. For example, the event name might be coming from sequence table data that the vehicle is following.

The 'Dwell Until Full' option cause the vehicle to wait at the node until its load is equal to the ride capacity of the vehicle.

'Specific Time' specifies a particular time that the vehicle must wait until it moves on to its next destination. This time may also vary from node to node. This option may be useful for vehicles such as buses, that wait a particular amount of time at a given stop before moving to the next stop.

With all of the dwelling options, a *Dwell Only If* property is available to specify condition(s) associated with dwelling. By default, the following condition is specified. '!Vehicle.CurrentNode.IsInputNode && (Vehicle.RideStation.Capacity > Vehicle.RideStationLoad) && (Vehicle.ResourceState == 7)'. The first part of this expression (!Vehicle.CurrentNode.IsInputNode) checks to see if the vehicle is not at an input node (such as a BasicNode on an object). This is done as usually dwelling may occur when picking up entities at an output node instead of when dropping them off at an input node. Remember that BasicNode's have no ability to call a transporter for pickup within the standard properties. The second part of the *Dwell Only If* expression (Vehicle.RideStation.Capacity > Vehicle.RideStationLoad) indicates dwelling only when the ride capacity isn't yet full. For example, if the vehicle is already full, yet Specific Time is indicated, the vehicle will not dwell. Finally, the last part of the expression (Vehicle.ResourceState == 7) checks to see if the vehicle is in a Transporting state, meaning it has entities and is not going offshift. In that's not the case, the vehicle will not wait additional time for dwelling. It is **important to note** that this *Dwell Only If* condition is checked when the vehicle arrives to the node for pickup and after each entity pickup at the node (it is not constantly evaluated for a change in condition). Of course, this default condition may be deleted or modified to suit your modeling needs.

Vehicles and Free Space Travel

- Vehicles and Workers may travel on links in a network or in free space. When a vehicle or worker moves in free space, there are several things to note. A Vehicle or Worker, if entering free space from a node and there is no destination yet assigned, will do one of the following things. If the *Idle Action* is 'Roam', the vehicle / worker will travel to the nearest Output node (to possibly do 'first available' transport pickups) or travel to any nearest destination as second choice. If the vehicle/worker is transporting, it will travel to the nearest Input node (to do transport drop-offs) or travel to any nearest node as second choice.

If a vehicle / worker is travelling in free space and is scheduled to go off-shift / on-shift due to a work schedule, the vehicle / worker will immediately adjust its direction of travel and go to the home node if the *OffShift Action* or *Idle Action* are set to 'Park at Home' or 'Go To Home'.

Examples

Example 1

Vehicle_Example 1

Properties: Output@Server1 (TransferNode)

Crossing Logic	
Initial Traveler Capa...	Infinity
Entry Ranking Rule	First In First Out
Routing Logic	
Outbound Link Prefe...	Any
Outbound Link Rule	Shortest Path
Entity Destination...	Specific
Node Name	Input@Sink1
Transport Logic	
Ride On Transporter	True
Transporter Type	Specific
Transporter Name	Vehicle1
Reservation Method	Reserve Closest
Selection Goal	Preferred Order
Selection Condition	

Properties of the Output Node at Server1

Properties: Vehicle1 (Vehicle)

Transport Logic	
Initial Ride Capacity	Infinity
Task Selection Strat...	First In Queue
Load Time	0.0
Unload Time	0.0
Park to Load/Unload	False
Minimum Dwell Time ...	No Requirement
Travel Logic	
Initial Desired Sp...	2.0
Initial Network	Global
Network Turnaroun...	Exit & Re-enter
Routing Logic	
Initial Priority	1.0
Initial Node (Home)	Input@Sink1
Routing Type	On Demand
Idle Action	Park At Node
Off Shift Action	Park At Node

Properties of the Vehicle

This shows the Transfer Node properties window for the Output Node of a Server and the Vehicle properties window for Vehicle1. The entities that enter this node will wait to be picked up by the Transporter named Vehicle1. If there are more than one type of Vehicle1 objects in the model at the time, the Closet will be reserved. The Vehicle1 object will take the entities from the Output Node of the Server to the Input Node of Sink1, as specified in the *Node Name* property of the Node. The *Initial Ride Capacity* property is set to 'Infinity' so the Transporter will pick up all the entities that are waiting in the Output@Server1.RidePickupQueue at the time it arrives at the Output node.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Worker

A Worker object may be used to define a dynamic population of moveable unit resources in the modeled system. Each individual worker member in the population is tracked separately, in terms of both animation and statistics. Each individual worker member in the population may have a resource capacity of 0 (OffShift) or 1 (OnShift). A Worker can follow an OnShift/OffShift work schedule or have a fixed capacity.

In Simio, a Worker may be used as a moveable resource that is seized and released to process tasks by model process logic. Additionally, a worker may be used to transport entity objects between node locations. By default, new seize requests are always prioritized over new transport requests.

Workers have characteristics of both the Standard Library objects of Vehicles and Resources. Workers may have a fixed capacity or may utilize a defined work schedule, similar to a resource. Workers also physically move between objects and can carry one or more entities from one location to another, similar to vehicles. Workers have a physical location in the Facility window, defined by the *Initial Node (Home)*, *Idle Action* and *Off Shift Action* properties specified with the Worker. The worker may have a home location to which it moves when idle / off-shift, may remain at its last utilized location or may roam the network of paths.

When a Worker object is scheduled to go 'Off-Shift', if the worker is still in the process of performing a task then it will first complete any remaining work before going into the 'OffShift' resource state. No new seize or transport requests will be accepted during the off-shift period. While finishing those tasks, the worker object will be in either the 'OffShiftBusy' state or 'OffShiftTransporting' state. If in the 'OffShiftTransporting' state, the Worker will complete any pickups at a reserved node location (if it has already accepted a pickup transport request). The Worker will also finish any remaining rider drop-offs. Once all remaining drop-offs are completed and the Worker object's RideStation queue is empty, it will go into the 'OffShift' state and perform the *Off Shift Action* (Park at Node, etc.). If in the 'OffShiftBusy' state, the Worker will complete any traveling to the requested task destination. It will then remain at that location until released. Once released, it will go into the 'OffShift' state and perform the *Off Shift Action* specified. Note: If the user would like to interrupt a process task delay when a Worker goes off-shift (to immediately preempt usage of and release the worker), there is the option of using the Interrupt step to do so.

If a Worker's CurrentTravelMode is 'Network If Possible', then the worker will accept any node in the same Facility view as a valid destination for pickups, drop-offs, processing tasks, etc. (because the Worker's travel movements will be able to be performed using either its currently assigned network or in free space).

The *Initial Number In System* may be defined for the Worker population. Additionally, the Create step or Source object may be used to dynamically create and add new members to the population during the run. However, note that Worker objects cannot be dynamically destroyed (Destroy step) because resource objects in Simio are not destroyable.

The [Add-On Process Triggers](#) allow additional logic to be specified when using the Worker.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Worker.

Note: When auto-creating Workers from a data table, you must specify the Transporter Reference Column as a Key Column.

For more information, see [Worker - Discussion and Examples](#).

Listed below are the properties of the **Worker**:

Property	Valid Entry	Description
Capacity Type	Fixed, Work Schedule	The availability of this worker type to perform tasks. 'Fixed' indicates that this type of worker is always available. 'WorkSchedule' indicates that each worker of this type follows a work schedule defining its 'On-shift', 'Off-shift' availability over time.
Initial Work Schedule	WorkSchedule name	The name of the work schedule initially assigned to the resource. Note: Refere to the SetWorkSchedule step to dynamically assign a new work schedule to a resource during a simulation run.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.

Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order requests waiting to seize this Worker object for a process task.
Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this Worker object, when it becomes available, dynamically selects the next seize request from its statically ranked allocation queue using a dynamic selection rule.
Repeat Group	True, False	Indicates whether a repeat group data structure is used to define the primary dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (e.g., Candidate.Entity.Priority).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.
Park While Busy	True or False	Indicates whether this Worker object should park while busy at a node due to a seize request. Parking a worker can be useful to avoid blocking other travelers on the network.
Initial Desired Speed	Expression	The initial value for the desired travel speed of this Worker object when traveling between locations.
Initial Travel Mode	Free Space Only, Network Only, Network If Possible	<p>The initial travel mode for entities of this type.</p> <p>'Free Space Only' indicates that the entities are required to perform travel movements in free space.</p> <p>'Network Only' indicates that the entities are required to perform travel movements using the currently assigned network.</p> <p>'Network If Possible' indicates a preference for the entities to perform travel movement using the currently assigned network, but if no followable network path exists to the next destination then travel in free space is allowed.</p> <p>NOTE: Refer to the user-assignable 'CurrentTravelMode' state variable of an entity to dynamically change its travel mode during a simulation run. An</p>

entity's travel mode may also be changed using the 'Outbound Travel Mode' property provided by a node.		
Initial Network	Network Instance Name	The initial network of paths used by this Worker object to travel between locations.
Network Turnaround Method	Exit & Re-enter, Rotate in Place, Reverse	When traveling on a network, the default method used by objects of this type to turn around at a node in order to move in the opposite direction on a bidirectional link. The 'Turnaround Method' property of a Network element may also be used to override this default on a network-by-network basis.
Free Space Steering Behavior	Direct To Destination, Follow Network Path If Possible	The behavior used to steer an entity of this type when traveling in free space to a destination. 'Direct To Destination' will steer an entity in a straight line to its destination. 'Follow Network Path If Possible' will prefer to steer an entity along a path following its currently assigned network, staying within the boundaries of the drawn path width. The Entity will travel based on the drawn length of the path even if a different logical length is specified. However, if no followable network path exists to the entity's destination then the entity will be steered in a straight line. Note: In order to use this steering behavior, make sure the entity's travel mode is set to 'Free Space Only'.
Update Interval	Expression	The time interval between updates checking an entity's adherence to the network path.
Avoid Collisions	True or False	Indicates whether the steering behavior will attempt to avoid collisions with other entities that are following the same network path.
Initial Priority	Real	The initial priority value of this Worker object.
Initial Node (Home)	Node Instance Name	The initial node location of this Worker object at the beginning of the simulation run. Also indicates the worker's 'Home' location if the object is specified to 'Park At Home' or 'Go To Home' when idle or off shift.
Idle Action	Park At Node, Go to Home, Remain In Place, Park At Home, Roam	The action of this Worker object when it does not have any tasks to perform.
Off Shift Action	Park At Node, Go to Home, Remain In Place, Park At Home	The action of this Worker object when it goes 'off shift'. Note that, if the worker is busy when an off shift period begins, then this action is not taken until the worker is released from its last task.
Initial Ride Capacity	Integer	The initial carrying capacity of this Worker object.
Task Selection Strategy	Smallest Distance, Largest Distance, Smallest Priority, Largest Priority, First In Queue	The strategy used by the Worker object to select its next transport task.
Load Time	Expression	The time required for this Worker object to load an entity (i.e., load time per entity).
Unload Time	Expression	The time required for this Worker object to unload an entity (i.e., unload time per entity).
Park To	True or False	Indicates whether this Worker object should park to load and unload entities

Load/Unload		at a node. Parking a worker can be useful to avoid blocking other travelers on the network.
Minimum Dwell Time Type	Dwell Until Event, Dwell Until Full, No Requirement, Specific Time	Specifies the minimum dwell time requirement for this Worker object when loading and unloading entities at a node. A 'minimum dwell time' is a minimum amount of time the worker is required to wait, or 'dwell', at a node to load and unload entities.
Event Name	Event Name	The name of the event that will indicate the expiration of the minimum wait time requirement for this Worker object when loading and unloading entities at a node.
Minimum Dwell Time	Expression	The specific minimum amount of time that this Worker object is required to wait, or 'dwell', at a node to load and unload entities.
Dwell Only If	Expression	If specified, this condition will be used to indicate whether the worker must respect the minimum dwell time at a node if that requirement has not yet expired, thereby forcing the worker to wait, or 'dwell', at the node for possible additional loads. Example uses of this property might include allowing the worker to exit a node before the minimum dwell time has expired if the worker has reached full ride capacity, or if there is little to no chance of additional loads. Another example condition might be to only respect the minimum dwell time requirement if the worker is at a node meeting some specific criteria.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to workers of this type are rolled up into. If a parent cost center is not explicitly specified, then costs allocated to a worker will be automatically rolled up into the parent object that contains the worker's current location.
Capital Cost Per Worker	Expression	The initial one-time setup cost to add a worker of this type to the system.
Cost Per Rider (Transport Costs)	Expression	The cost to load and transport an entity using a transporter of this type, irrespective of the transportation time. This cost is allocated to an entity when it is loaded onto the transporter.
Transport Cost Rate (Transport Costs)	Expression	The cost per unit time to transport an entity using a transporter of this type.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of a worker of this type when it is idle.
Cost Per Use(Resource Costs)	Expression	The one-time cost that is accrued each time a worker of this type is used for a non-transport task, regardless of the usage duration. This cost will be charged to the cost of the owner object(i.e., task) using the worker.
Usage Cost Rate(Resource Costs)	Expression	The cost per unit time to use a worker of this type for a non-transport task. This cost will be charged to the cost of the owner object (i.e., task) using the worker.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized. Note: This trigger will only occur for Worker objects created by the Initial Number in System setting.

Run Ending Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is ending.
Allocated Add-On Process Triggers	Process Instance Name	Occurs when this Worker resource has been allocated to perform some transport tasks, or when another object has seized this Worker resource for some non-transport related task.
Released Add-On Process Triggers	Process Instance Name	Occurs when this Worker resource has been released (freed) after completing a set of transport tasks, or when another object has released this Worker resource from some non-transport related task.
Entered Node Add-On Process Triggers	Process Instance Name	Occurs when this Worker object has entered a node.
Unloaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is unloaded from this Worker.
Loaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is loaded onto the Worker.
Exiting Node Add-On Process Triggers	Process Instance Name	Occurs when this Worker object is about to begin exiting a node.
Evaluating Transport Request Add-On Process Triggers	Process Instance Name	Occurs when this Worker object is evaluating whether or not to accept or reject a transport request from a potential rider. In the executed decision process, assigning the value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the transport request is rejected.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Worker object is evaluating whether to accept or reject a seize allocation attempt by some process in the system. In the executed decision process, assigning a value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
On Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'off shift' because of its specified work schedule.
Initial Number in System	Integer	The initial number of Worker objects of this type in the system.
Maximum Number in	Integer	The maximum number of objects of this type that can be simultaneously in the system. If this limit is exceeded then a runtime error is generated.

System		
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Edition). If not specified, then this property defaults to a color derived from the object's unique name.
Can Transfer In & Out of Objects	True or False	Indicates whether objects of this type are by default allowed to transfer into and out of fixed objects via those object's external input / output nodes.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Worker - Discussion and Examples

Discussion

First Steps in Using a Worker

- Place a Worker object from the Standard Library window into the Facility window. The Worker physically moves around the Facility window and has specific locations; therefore, if you want the Worker to move on a specific Network of Paths, you must change the associated Network property of the Worker.
- Workers can serve as Transporters which move entities to and from locations. Somehow, the model has to be told to have the entities "ride" on the worker. In order to have the worker move entities from one Node to another, the Transfer Node has to be configured to have the entities leave with the Worker. Select the node by left-clicking on the Transfer Node of an object, such as a Source or Server. Then, change the *Ride On Transporter* property in the Properties window to 'True'. Next, expand the *Transporter Type* by clicking on the + and change the value in the *Transporter Name* property to the determined worker's name.
- The Worker may also be used for stationary tasks (as opposed to moving an entity from location to location). The Seize and Release steps in the Processes window should be used to request a Worker object to perform a stationary task. Use of these steps can be done through the Add-On Process Triggers in any of the Standard Library objects, such as Server, BasicNode or TransferNode. Upon seizing a Worker object, you may use the Request Move option in the Seize step to optionally require the seized worker to move to a specified node location (for example, the requesting entity's location) before the Seize is considered 'completed'. Note: This applies to Vehicle as well.

Fixed Capacity or Capacity Varying by Work Schedule

- Regardless of whether a Worker object has a Fixed Capacity or Capacity Varying by Work Schedule, the number of workers available to the system is based on the *Initial Number In System* property, shown under the Population section. If the Worker.CurrentCapacity is manually assigned to a value other than 0 or 1, a warning will be displayed (if warnings are not disabled) and the capacity will automatically adjusted to the value of '1'.

- *Fixed*

If the *Capacity Type* property is set to 'Fixed', then each of the *Initial Number In System* workers will have a fixed capacity during the simulation run; that is, no are breaks or off-shift time.

- *WorkSchedule*

If the *Capacity Type* property is set to 'Work Schedule', a work schedule must be specified in the *Work Schedule* property. A work schedule is defined from the Schedules panel, which is found by clicking on the [Data tab](#). This work schedule will be utilized for each of the *Initial Number in System* workers to allow them to go off-shift for breaks and/or off-shift time. The schedule is intended to be used for On-Shift and Off-Shift times for the Worker. The *Value* property within the schedule should always be '1', as the capacity of the Worker is set through the *Initial Number in System*.

Loading and Unloading Times

- Load and unload times are incurred per entity. For example, if the capacity of a Worker is 2, up to two waiting entities may be picked up to be moved. If two entities are waiting for pickup, the first one will be picked up, and incur its loading time. Before moving, the second entity is picked up and incurs its loading time. Thus, the first entity incurs its own load time, plus the load time of the second entity before the vehicle starts the transfer.
- If the *Park to Load/Unload* property is specified as 'True', the worker(s) will be placed in the parking area of the node at which it is loading or unloading. There are two ways to animate the parking area. First, click on a specific node and then select the Parking Queue button from the Appearance ribbon. Alternatively, click on a specific node and select Draw Queue / ParkingStation.Contents and draw the animated queue.

How the Task Selection Strategy Property Works

- When Worker object searches a RidePickupQueue for the next pickup at a node, or searches the global VisitRequestQueue for a ride reservation, or searches its RideStation to determine its next dropoff destination, it will use the option specified in the Task Selection Strategy property to determine how to choose from the appropriate queue.

Worker has ResourceState State Variable that Tracks Its States

- The ResourceState state variable is a [List State](#) automatically tracked for the Worker, identical to the ResourceState state provided by various other objects in the standard library (Server, Resource, Combiner, Vehicle, etc.). The numeric values for the state are Idle=0, Busy=1, OffShift=4, OffShiftBusy=6, Transporting=7, OffShiftTransporting=9. A Worker is considered 'Busy' when it is Seized by a token using a Seize step for a process task. The Worker is considered 'Transporting' when it is utilized for a movement type task (either moving to the entity for transport or moving with the entity to its destination location).

You can use the ResourceState in process logic to check whether a Worker is 'Busy' or 'Idle' or 'Transporting'. This would be done by utilizing the function Worker.ResourceState in the logic, given that the token referencing this function is associated with the worker. You might also easily change the symbol animation on the Worker to reflect its current state (e.g., show 'Busy' versus 'Idle' symbols) by using the Worker.ResourceState function in the *Current Symbol Index* property of the vehicle. Note that you would need to animate all possible symbols to reflect them properly. For example, the symbols for Idle (0), Busy(1), and Transporting(7) should be utilized if the worker has a fixed capacity. A different color or symbol could be used for each of those states. You will see Resource State results for the Worker objects in the Results window.

To animate the worker ResourceState, you must explicitly reference the index of the worker within the dynamic population. For example, if you have a worker with the *Name* 'Worker1' and the *Initial Number In System* property for the worker set to '2', two Status Pies may be placed and the List States can be referenced as Worker1[1].ResourceState and Worker1[2].ResourceState.

The ResourceState variable is used to display utilization statistics for the Worker in the Results window. Additionally, this list state can be used to display utilization statistics during the simulation run. For example, you may wish to show the percentage of time that a worker is doing transport type tasks during the simulation run. This would be done by placing a status label with the expression WorkerName[unit#].ResourceState.PercentTime(7). For more information on list state values and utilization statistics, see [List State](#) page.

Seizing a Worker for Non-Transport Tasks

- You may use a Seize step to seize a Worker. It might delay for some processing time, then release the worker using the Release step. This allows the Worker object to be utilized for both transport as well as non-transport tasks. If a Worker object is being used for both transport and non-transport tasks, then note that by default non-transport tasks will always be prioritized before transport tasks.

To prioritize specifically among transport tasks, specify the *Task Selection Strategy* in the Transport Logic category of the Vehicle object's properties. To prioritize specifically among non-transport tasks, you may use the *Ranking Rule* or *Dynamic Selection Rule* properties available in the Process Logic category of the Worker object's properties. If you want to implement even more customized control of how a Worker object selects its next task then you may take advantage of the *Evaluating Transport Request* and *Evaluating Seize Request* Add-On process triggers. For example, you can use those Add-On processes to implement logic on a particular worker such as "Don't accept any non-transport tasks until all my transport tasks are finished first".

If the *Park While Busy* property is specified as 'True', the worker(s) will be placed in the parking area of the node at which it is processing. There are two ways to animate the parking area. First, click on a specific node and then select the Parking Queue button from the Appearance ribbon. Alternatively, click on a specific node and select Draw Queue / ParkingStation.Contents and draw the animated queue.

Referring to a Specific Worker within the Dynamic Population

- You may use a bracket and number to explicitly refer to a specific unit of the dynamic population of workers in the system, for example, Worker1[1].FunctionName. Therefore, if you have a worker with the *Name* 'Worker1' and the *Initial Number In System* property for the vehicle set to '2', they can be referenced as Worker1[1] and Worker1[2] with regards to functions. For example, within an expression, you may wish to evaluate a function of a specific unit, 'Worker1[1].RideStation.Capacity.Remaining > 3' within a Decide step. You may use any agent instance name, followed by brackets, with an expression in those brackets giving the index into the agent population, for example, Worker1[1].Function, Worker1[1+0].Function or Worker1[Worker1[x].SiblingWorkerIndex].Function.

Network Turnaround Method and Bidirectional Paths

- The *Network Turnaround Method* property for an entity, worker or vehicle is used in conjunction with bidirectional paths. By default, the 'Exit & Re-enter' method is used. With this method, the entity that wants to turn around will always immediately first exit the link (moving essentially into free-space at the node) before attempting to re-enter. And, if the entity cannot immediately re-enter, then it will wait in the link's EntryQueue. The other two turnaround methods are 'Rotate in Place' and 'Reverse'. With both of these methods, the entity that wants to turn around will not leave the link, but will simply turn around and move in the opposite direction. 'Rotate in Place' will always keep the

'head' of the entity at the front, whereas 'Reverse' will appear that the entity is moving backwards. For more information regarding the Turnaround Method, see the [Path - Discussion and Examples](#) page.

Initial Node (Home)

- The *Initial Node (Home)* property determines where the worker(s) will initially be located at the start of the simulation run. This node name is stored in the HomeNode state of the worker and can be assigned during the simulation run to change the worker's home location. This would be done by assigning a *State Variable Name* of 'Worker.HomeNode' to a *New Value* of 'BasicNode1', for example, if the associated object is the worker. Similarly, the user may reference a specific worker by specifying *State Variable Name* as 'OperatorA[1]' and *New Value* as 'BasicNode1'.

Workers and Deadlocking

- Workers traveling along bidirectional paths have the potential to deadlock, causing the workers to appear to hang at a given node location. Please see the discussion on the [Path - Discussion and Examples](#) page, as well as the [Warning Level](#) page.

Minimum Dwell Time

- Both the Worker and Vehicle objects have the ability to 'dwell', or wait, at a node for a specified period of time. There are several options for dwelling and a condition that may also be specified.

'No Requirement' for the *Minimum Dwell Time Type* indicates that the worker will not wait at a node before moving on. This is the default behavior of both Workers and Vehicles.

'Dwell Until Event' indicates that the specified event must be fired before the vehicle moves from the node to its next destination. This may be useful in cases where the worker moves only when a certain condition is met and an event is fired from an upstream location. Note that the event used can be different from node to node. For example, the event name might be coming from sequence table data that the worker is following.

The 'Dwell Until Full' option cause the worker to wait at the node until its load is equal to the ride capacity of the worker.

'Specific Time' specifies a particular time that the worker must wait until it moves on to its next destination. This time may also vary from node to node.

With all of the dwelling options, a *Dwell Only If* property is available to specify condition(s) associated with dwelling. By default, the following condition is specified. '!Worker.CurrentNode.IsInputNode && (Worker.RideStation.Capacity > Worker.RideStationLoad) && (Worker.ResourceState == 7)'. The first part of this expression (!Worker.CurrentNode.IsInputNode) checks to see if the worker is not at an input node (such as a BasicNode on an object). This is done as usually dwelling may occur when picking up entities at an output node instead of when dropping them off at an input node. Remember that BasicNode's have no ability to call a transporter for pickup within the standard properties. The second part of the *Dwell Only If* expression (Worker.RideStation.Capacity > Worker.RideStationLoad) indicates dwelling only when the ride capacity isn't yet full. For example, if the worker is already full, yet Specific Time is indicated, the worker will not dwell. Finally, the last part of the expression (Worker.ResourceState == 7) checks to see if the worker is in a Transporting state, meaning it has entities and is not going offshift. In that's not the case, the worker not wait additional time for dwelling. It is **important to note** that this *Dwell Only If* condition is checked when the worker arrives to the node for pickup and after each entity pickup at the node (it is not constantly evaluated for a change in condition). Of course, this default condition may be deleted or modified to suit your modeling needs.

Workers and Free Space Travel

- Vehicles and Workers may travel on links in a network or in free space. When a vehicle or worker moves in free space, there are several things to note. A Vehicle or Worker, if entering free space from a node and there is no destination yet assigned, will do one of the following things. If the *Idle Action* is 'Roam', the vehicle / worker will travel to the nearest Output node (to possibly do 'first available' transport pickups) or travel to any nearest destination as second choice. If the vehicle/worker is transporting, it will travel to the nearest Input node (to do transport drop-offs) or travel to any nearest node as second choice.

If a vehicle / worker is travelling in free space and is scheduled to go off-shift / on-shift due to a work schedule, the vehicle / worker will immediately adjust its direction of travel and go to the home node if the *OffShift Action* or *Idle Action* are set to 'Park at Home' or 'Go To Home'.

Connector

A **Connector** object is a link that may be used to define a non passing pathway between two node locations where the travel time is instantaneous. It is a Link object that is hard-coded with its *DrawnToScale* property set to 'False' and its *LogicalLength* property set to '0.0'. Entities will never actually enter, move across or exit a Connector, but will always instead immediately do a node-to-node direct transfer across the link. Note that entities travel across a Connector one at a time, which is different behavior than the standard Path link type. Entities waiting to enter a Connector are inserted into the Connector's EntryQueue.

Connectors can be animated with Path Decorators, see the [Animating Links](#) page for more details.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Path

A **Path** object is a link that may be used to define a pathway between two node locations where the travel time is determined by the path length and a traveler's speed. Passing may or may not be allowed. Paths can be animated with Path Decorators, see the [Animating Links](#) page for more details.

The [Add-On Process Triggers](#) allow additional logic to be specified when using the Path.

There are two categories of statistics that are generated for paths, including traveler statistics and flow statistics. The Data Source 'Travelers' provides information related to the throughput, flowtime and content on a particular path. The Data Source section 'Flow' provides throughout information on the flow in and out of a particular path. Flow values are reported only if at least one of them is greater than 0.

For more information, see [Path - Discussion and Examples](#).

Listed below are the properties of the **Path**:

Property	Valid Entry	Description
Type	Unidirectional, Bidirectional	The type of traffic movement through this path.
Traffic Direction Rule	First In Entry Queue, Match Desired Direction, Prefer Desired Direction	The rule used to direct traffic entry onto this bidirectional path.
Initial Desired Direction	None, Either, Forward, Reverse	The initial desired direction of traffic on this bidirectional path.
Initial Traveler Capacity	Expression	The initial maximum number of travelers that may simultaneously occupy this link.
Entry Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank entry onto this path among competing entry requests.
Entry Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Drawn To Scale	True or False	Specifies whether the path's drawn length in the Facility View is the length to be used for the simulation logic.
Logical Length	Double	The path length to be used for the simulation logic.
Allow Passing	True or False	Indicates whether passing is allowed on this path. Passing means that entities will not be blocked along the path, but instead will pass each other if speeds vary.
Speed Limit	Double	The maximum speed at which an entity can travel along this path.
Selection Weight	Expression	The weight expression for this link when using the 'By Link Weight' rule to select an outbound link from a node.
On Entering (State Assignment)	Repeat Group, Assignments	Optional state assignments when an entity is entering the Path object.

Before Exiting (State Assignment)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the Path object.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation model is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process Triggers	Process Instance Name	Occurs when a traveler's leading edge has entered this path.
Trailing Edge Entered Add-On Process Triggers	Process Instance Name	Occurs when a traveler's trailing edge has entered this path.
Reached End Add-On Process Triggers	Process Instance Name	Occurs when a traveler's leading edge has reached the end of this path.
Exited Add-On Process Triggers	Process Instance Name	Occurs when a traveler has exited this path.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Path - Discussion and Examples

Discussion

Using Bidirectional Paths

- The *Type* property allows the user to specify whether or not the Path is 'Unidirectional' or 'Bidirectional'. It is important to note that traffic cannot travel in two different directions on one Link at the same time. So even if the capacity of the Link is set to Infinity, if there is traffic traveling one direction and an entity or a transporter would like to enter the link and travel in the opposite direction, the entity or transporter will wait until the traveling entity is off the link before it enters. In order to allow traffic to travel in opposite directions at the same time, consider having two unidirectional links side by side each going in the opposite direction.
- When using bidirectional paths, you may need to use occasional unidirectional 'bypass' paths to minimize deadlocking. See the SimBit [BidirectionalPaths](#) for an example. Additionally, SimBit [PathSelectionRule](#) demonstrates the use of the *Traffic Selection Rule* property of a path.

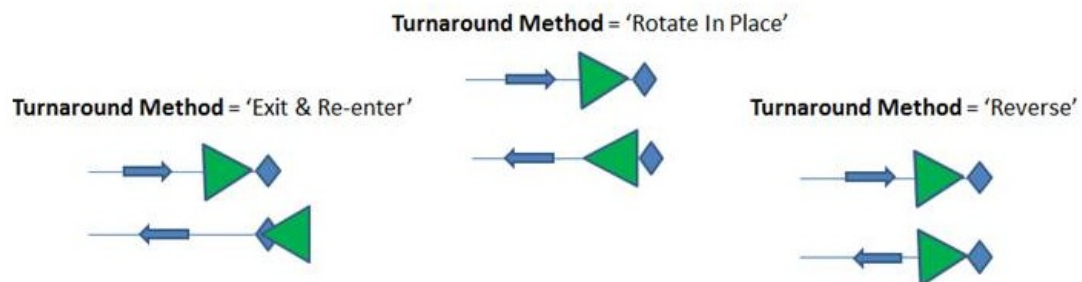
When referencing the Link's Type in a table, the link will show unidirectional arrows during design time, but during runtime the arrows for bidirectional links will change to bidirectional arrows.

Bidirectional Paths: Deadlocking

- Whenever bidirectional paths are used, there is a potential for deadlocking to occur. What this means is that entity, worker, or vehicle movement may stop because one object is at a node trying to move onto a bidirectional link, while another object is trying to move through the same node onto another link. If this occurs, the simulation model will appear to 'hang' at that particular node. Simio checks for potential deadlocks at a node if there is an entity object (entity, worker, vehicle) on a bi-directional link attempting transfer through the node onto another bi-directional link. If a potential deadlock is found, a [warning](#) will be displayed.
- To help prevent deadlocking, there are some modeling suggestions. When there is a node where an entity can get onto a bidirectional link, there should be an adequate area for the entity to wait without blocking the exit for the link (by-pass area). The size of this by-pass area depends upon the rate and size of the incoming entities. Additionally, if you have a network of paths, any spurs on that network should have the link capacity of 1. This will allow entities to enter and exit the spur without another entity blocking the entry. Please refer to the [BidirectionalPaths](#) SimBit, as well as the *MiningExample.spfx* under Example Projects to see examples of by-pass links.

Bidirectional Paths: Network Turnaround Method

- The *Network Turnaround Method* property for an entity, worker or vehicle is used in conjunction with bidirectional paths. By default, the 'Exit & Re-enter' method is used. With this method, the entity that wants to turn around will always immediately first exit the link (moving essentially into free-space at the node) before attempting to re-enter. And, if the entity cannot immediately re-enter, then it will wait in the link's EntryQueue. The other two turnaround methods are 'Rotate in Place' and 'Reverse'. With both of these methods, the entity that wants to turn around will not leave the link, but will simply turn around and move in the opposite direction. 'Rotate in Place' will always keep the 'head' of the entity at the front, whereas 'Reverse' will appear that the entity is moving backwards.



Bidirectional Paths: Traffic Direction Rule and Desired Direction

- When the *Type* of path is 'Bidirectional', two additional properties become available within the Path object. The *Traffic Direction Rule* is used to direct traffic entry onto the bidirectional link. The *Initial Desired Direction* property is

used to specify the direction of traffic on the link. A bidirectional link has a direction which is based on the way that it is drawn (as an original unidirectional path). That direction is considered 'Forward'. When defining a path as 'bidirectional', you specify the *Traffic Direction Rule* which is either 'First in Entry Queue', 'Match Desired Direction' or 'Prefer Desired Direction'. 'First in Entry Queue' indicates that if multiple entities are attempting to enter a bidirectional link (from either side), the first entity arriving to the node to enter will be the one to enter the link. 'Match Desired Direction' means that the original placement of the link will determine what the 'desired direction' of the link is, which also depends upon the *Initial Desired Direction* property specified. So, for example if the link was placed from a left most node to a right most node, the 'Forward' direction will be from left to right. If the *Initial Desired Direction* is specified as 'Forward', then only those entities coming from the left most node can enter the link (when it is bidirectional). The state `Path.DesiredDirection` may be assigned during the simulation run to change direction of the path to Reverse (or Either / None). 'Either' indicates that traffic can travel in both directions, forward or reverse. 'None' means entities cannot travel on this link. When 'Prefer Desired Direction' is specified, when multiple entities are waiting to enter the path, those moving in the 'desired direction' specified will have priority over those in a different direction.

Selection Weight Property

- The *Selection Weight* property can be used to specify which entities should travel on this path. It can be set to a numerical value, such as .2 so that if there is a choice between this path and another path, 20% of the entities will travel on this path. It can also be set to an expression, such as `ModelEntity.Priority == 2` so that all entities with their Priority state equal to 2 will travel down this path. In this example, if there is no other choice of path, in other words, this is the only path for entities to take, they will all travel on this path regardless of whether or not their Priority state equals 2.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

TimePath

A **TimePath** object is a link that may be used to define a pathway between two node locations where the travel time is user specified. TimePaths can be animated with Path Decorators, see the [Animating Links](#) page for more details.

The [Add-On Process Triggers](#) allow additional logic to be specified when using the TimePath.

There are two categories of statistics that are generated for time paths, including traveler statistics and flow statistics. The Data Source 'Travelers' provides information related to the throughput, flowtime and content on a particular time path. The Data Source section 'Flow' provides throughout information on the flow in and out of a particular time path. Flow values are reported only if at least one of them is greater than 0.

Listed below are the properties of the **TimePath**:

Property	Valid Entry	Description
Type	Unidirectional, Bidirectional	The type of traffic movement through this path.
Traffic Direction Rule	First In Entry Queue, Match Desired Direction, Prefer Desired Direction	The rule used to direct traffic entry onto this bidirectional path.
Initial Desired Direction	None, Either, Forward, Reverse	The initial desired direction of traffic on this bidirectional path.
Initial Traveler Capacity	Expression	The initial maximum number of travelers that may simultaneously occupy this link.
Entry Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank entry onto this path among competing entry requests.
Entry Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Travel Time	Expression	The time required to travel the length of this path.
Selection Weight	Expression	The weight expression for this link when using the 'By Link Weight' rule to select an outbound link from a node.
On Entering (State Assignment)	Repeat Group, Assignments	Optional state assignments when an entity is entering the TimePath object.
Before Exiting (State Assignment)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the TimePath object.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process Triggers	Process Instance Name	Occurs when a traveler's leading edge has entered this path.
Trailing Edge	Process Instance Name	Occurs when a traveler's trailing edge has entered

Entered Add-On Process Triggers		this path.
Reached End Add-On Process Triggers	Process Instance Name	Occurs when a traveler's leading edge has reached the end of this path.
Exited Add-On Process Triggers	Process Instance Name	Occurs when a traveler has exited this path.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Conveyor

Conveyor

A Conveyor is a link that can represent both accumulating and non-accumulating devices. The specific conveyor defined in the Standard Object Library supports both fixed and variable spacing and accurately models the merging of entities of various sizes on the conveyor. Conveyors can be animated with Path Decorators, see the [Animating Links](#) page for more details.

A Conveyor may have failures, which are specified within the [Reliability](#) section of the Properties window. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Conveyor.

There are two categories of statistics that are generated for conveyors, including traveler statistics and flow statistics. The Data Source 'Travelers' provides information related to the throughput, flowtime and content on a particular conveyor. The Data Source section 'Flow' provides throughout information on the flow in and out of a particular conveyor. Flow values are reported only if at least one of them is greater than 0.

For more information, see [Conveyor - Discussion and Examples](#).

Listed below are the properties of the **Conveyor**:

Property	Valid Entry	Description
Initial Traveler Capacity	Expression	The initial maximum number of travelers that may simultaneously occupy this link.
Entry Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank entry onto this conveyor among competing entry requests.
Entry Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Initial Desired Speed	Expression	The initial desired speed of this Conveyor object.
Entity Alignment	Any Location, Cell Location	The method for aligning engaged entities on this Conveyor object. If 'Any Location', then an entity can engage at any location on the conveyor. If 'Cell Location', then the conveyor length has a discrete number of cell locations, and the leading edge of an engaged entity must align with a cell boundary.
Cell Spacing Type	Fixed Cell Size, Fixed Number Cells	The method used to specify the fixed interval spacing of cells on the link. If 'Fixed Cell Size', then the cell size can be specified. If 'Fixed Number Cells', the number of cells is specified.
Cell Size	Real	The distance between cell boundaries on the link.
Auto Align Cells	With First Entity, No	Indicates whether the position of the cell boundaries on the link will be automatically adjusted to align with the first entity that engages the link. Using this option allows the cell boundaries of each link on which an entity travels to become synchronized with the first entity that passes through that link.
Number Of	Integer	The number of discrete cell locations on this Conveyor object.

Cells

Drawn To Scale	True or False	Specifies whether the conveyor's drawn length in the Facility View is the length to be used for the simulation logic.
Logical Length	Double	The conveyor length to be used for the simulation logic.
Accumulating	True or False	Specifies whether this Conveyor object is accumulating or non-accumulating. If non-accumulating, then the entire conveyor stops if an entity reaches the end and cannot exit. If accumulating, then entities are allowed to continue moving and accumulate if progress is blocked.
Selection Weight	Expression	The weight expression for this link when using the 'By Link Weight' rule to select an outbound link from a node.
Failure Type	No Failures, Calendar Time Based, Event Count Based	Specifies whether this conveyor object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
On Entering (State Assignment)	Repeat Group, Assignments	Optional state assignments when an entity is entering the Conveyor object.
Before Exiting (State Assignment)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the Conveyor object.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.
Capital Cost	Expression	The initial one-time setup cost to add this object to the system.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-	Process Instance	Occurs when a traveler's leading edge has entered this conveyor.

On Process Triggers	Name	
Trailing Edge Entered Add-On Process Triggers	Process Instance Name	Occurs when a traveler's trailing edge has entered this conveyor.
Reached End Add-On Process Triggers	Process Instance Name	Occurs when a traveler's leading edge has reached the end of this conveyor.
Exited Add-On Process Triggers	Process Instance Name	Occurs when a traveler has exited this conveyor.
Failed Add-On Process Triggers	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process Triggers	Process Instance Name	Occurs when a failure is repaired at this object.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Conveyor - Discussion and Examples

Discussion

Accumulation

The following functions are available to track Accumulated entities on a conveyor (or other type of links):

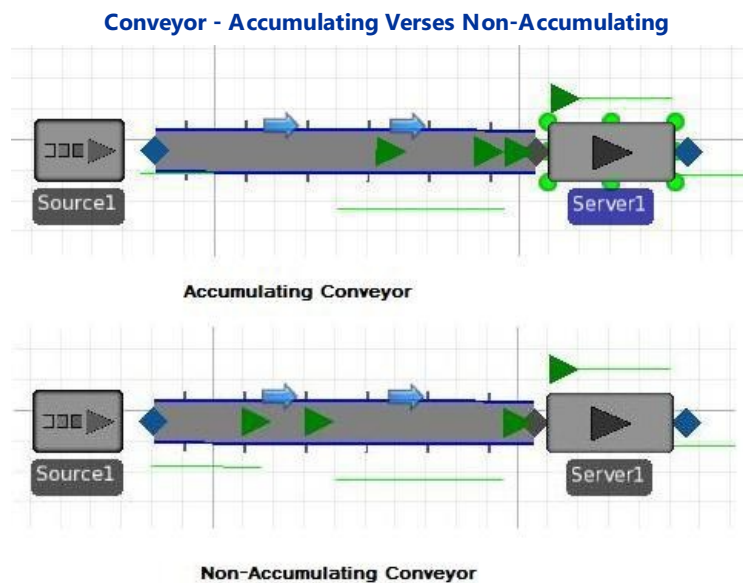
- NumberTravelers.Accumulated
- NumberTravelers.Accumulated.Minimum
- NumberTravelers.Accumulated.Maximum
- NumberTravelers.Accumulated.Average

The Conveyor's *Accumulating* property indicates that if the entity at the end of the conveyor is stopped, then the conveyor itself (i.e., everybody else) should stop. If the *Accumulating* property of the standard library Conveyor object is set to 'False', the NumberTravelers.Accumulated function may still be positive. For instance, if an entity is stopped at the end of the conveyor, it will be considered Accumulated. See the definition of Accumulation in the Link section of the [Object Types](#) page.

Examples

Example 1

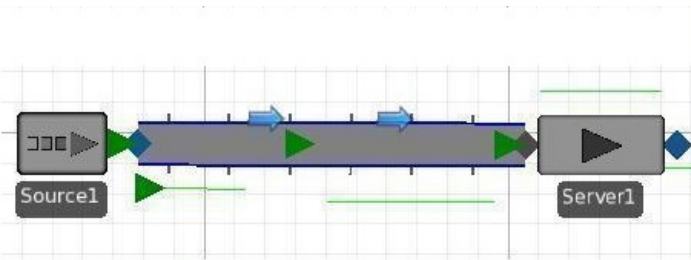
Conveyor object with *Accumulating* property set to True and then False. When Server1 is processing an entity, all entities on the conveyor will continue moving to the server with an accumulating conveyor. If the conveyor is non-accumulating, the conveyor stops and all entities keep their spacing when the server is busy. Both scenarios assume "0" input buffer capacity for Server1.



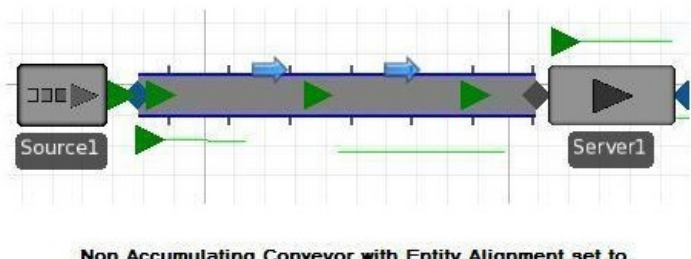
Example 2

Conveyor object with *Entity Alignment* property set to 'Cell Location'. The example below shows one conveyor with *Number of Cells* set to '2' and the other with *Number of Cells* set to '3'.

Conveyor with Specific Cell Location



Non Accumulating Conveyor with Entity Alignment set to Cell Location and Number of Cells set to 2



Non Accumulating Conveyor with Entity Alignment set to Cell Location and Number of Cells set to 3

Properties: Conveyor1 (Conveyor)

- Travel Logic**
 - Initial Traveler Capacity: Infinity
 - Entry Ranking Rule: First In First Out
 - Initial Desired Speed: 2.0
 - Entity Alignment: **Cell Location**
 - Cell Spacing Type: **Fixed Number Cells**
 - Number Of Cells: **2**
 - Auto Align Cells: With First Entity
 - Drawn To Scale: True
 - Accumulating: **False**
- Routing Logic**
 - Selection Weight: 1.0
- Reliability Logic**
- State Assignments**
- Financials**
- Add-On Process Triggers**
- Advanced Options**
- General**
 - Name: Conveyor1

Properties: Conveyor1 (Conveyor)

- Travel Logic**
 - Initial Traveler Capacity: Infinity
 - Entry Ranking Rule: First In First Out
 - Initial Desired Speed: 2.0
 - Entity Alignment: **Cell Location**
 - Cell Spacing Type: **Fixed Number Cells**
 - Number Of Cells: **3**
 - Auto Align Cells: With First Entity
 - Drawn To Scale: True
 - Accumulating: **False**
- Routing Logic**
 - Selection Weight: 1.0
- Reliability Logic**
- State Assignments**
- Financials**
- Add-On Process Triggers**
- Advanced Options**
- General**
 - Name: Conveyor1

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

WorkStation - DEPRECATED

In Simio, a **Workstation** represents a capacitated resource with optional constrained input and output buffers.

A Workstation is modeled as a sequence of three Activities: Setup, Processing, and Teardown. The sequence of Activities is referred to as an Operation. The Setup Time can be Specific, Change-Dependent, or Sequence-Dependent. A Specific time can be a constant or be product/order dependent by referencing a table. Change/Sequence dependent times can be based on any characteristic of the product (e.g. size or color). The entity that is processed through the workstation represents a production lot. The data for a production lot is typically held in data tables, and then referenced by the entity. The next production lot to be processed can be selected based on both a static ranking and a dynamic selection process. The workstation may have a fixed capacity, or follow a capacity schedule. However, take note that the standard Workstation object only processes one job at a time. In other words, it has a fixed capacity of 1.

The *Operation Quantity* specifies the number of "items" in the production lot that is represented by each arriving entity. This value can be a constant or be product/order dependent by referencing a table. The lot can be processed as a single production unit, or the lot can be broken into a sequence of batches that are processed one at a time until the entire lot has been produced. For example a lot of 1000 units may be processed in batches of 100 at a time. Note that a single setup and teardown activity is performed, but 10 separate batches of 100 are sequentially processed through the processing activity.

Multiple materials or a bill of material may be required to start the processing of each batch. Additionally, multiple materials or a bill of material may be produced at the end of each batch processing. A batch of produced material may also have a transfer time before it becomes available by other workstations. These materials can be specified through the *Materials* repeating property editor. This makes it easy to model situations involving intermediate transfer batching. Typically, a bill of materials is used for consumption and a material is produced, however Simio also allows for a bill of materials to be produced, to model cases such as disassembly.

The Workstation object is considered to be a primary resource that is required during the entire operation: i.e. setup, the processing of all batches, and teardown. In addition, any number of secondary resources may be required and held by each batch across one or more of the three activities. Secondary resources can be specific resources or selected based on rules from a list of resources.

A production lot may have a maximum make span that limits when it can be started based on the availability of the primary resource. The maximum make span may be a constant or order/product dependent by referencing a table. The make span check verifies that the primary resource is available to complete the operation within the maximum make span. For example a lot that has a maximum make span of 3 hours will not start 2 hours before an off-shift period of 8 hours begins, because its make span would exceed its maximum value. However, the make span check does not guarantee that material and/or secondary resources will be available at the needed times. Hence a production lot that passes the make span check may still exceed its make span if delays in material/secondary resources occur.

The [Balking and Reneging Options](#) page provides additional information and examples on buffer capacities, balking and reneging.

A Workstation may have failures, which are specified within the [Reliability](#) section of the Properties window. [State assignments](#) can be made on entering, before exiting, on balking and on reneging the Workstation. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Workstation.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the WorkStation.

For more information, see [Workstation - Discussion and Examples](#).

Listed below are the properties of the **Workstation**:

Property	Valid Entry	Description
Capacity Type	Fixed, Work Schedule	The method used to specify the capacity of this Workstation object. The capacity is fixed at 1 and cannot be changed.
Work Schedule	Work Schedule Name	The capacity work schedule that this Workstation object follows. *Note* Currently the GUI allows a user to add a Work Schedule with capacity greater than 1 to a Workstation, however because the standard Workstation object can only process one job at a time,

		if the schedule has a capacity greater than 1, it is simply interpreted as 1 by the Workstation.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order entities waiting to be allocated capacity of this Workstation object.
Ranking Expression	Expression	The expression used with a Smallest Value First or Largest Value First ranking rule.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this Workstation object dynamically selects the next entity to work on when the workstation is ready to process another job, instead of only using the ranking rule to determine processing order.
TransferIn Time	Expression	The time required to transfer an entity into this Workstation object. This property may be specified using a random sample from a distribution.
Operation Quantity	Expression	The item quantity assumed for each operation performed at this Workstation. This property may be used with <i>Processing Batch Size</i> property to perform an operation's processing activity in a single batch or a set of sequential batches.
Setup Time Type	Specific, Change Dependent, Sequence Dependent	The method used for specifying the setup time of an operation.
Setup Time	Expression	The setup time required at this Workstation object before processing an entity. This property may be specified using a random sample from a distribution.
Operation Attribute	Expression	The expression evaluated for each operation that is used to determine the change-dependent or sequence dependent setup time. Note that, for a sequence dependent time, the operation attribute expression is evaluated as a zero-based index into the Changeover Matrix string list for which 'from-to' pairs are defined. For example, an operation attribute of '0' means that the entity's comparison value is the first value in the changeover matrix string list. Referencing a list property on the processing entity as the operation attribute is particularly useful for modeling sequence-dependent setup times.
Setup Time If Same	Expression	The setup time if there has not been a change in the <i>Operation Attribute</i> from the previous operation to the current operation. This property may be specified using a random sample from a distribution.
Setup Time If Different	Expression	The setup time if there has been a change in the

		<i>Operation Attribute</i> from the previous operation to the current operation. This property may be specified using a random sample from a distribution.
ChangeOver Matrix	Changeover Matrix Instance	The changeover matrix that defines the operation's setup time dependent on the attribute change from the previous operation to the current operation.
Processing Batch Size	Expression	Optional batch size used instead of the <i>Operation Quantity</i> to perform the processing activity as a set of sequential batches. Must be less than or equal to the operation quantity. If not specified, then the batch size is assumed to be the operation quantity and the processing activity is performed in a single batch.
Processing Time	Expression	The time required to process an individual batch. This property may be specified using a random sample from a distribution.
Teardown Time	Expression	The teardown time required for an operation after completing the processing of all batches. This property may be specified using a random sample from a distribution.
Input Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Workstation object's input buffer.
Balk Decision Type (Input Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Workstation object's output buffer.
Balk Decision Type (Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Output Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Renege Triggers	Repeat Group	Optional time or event-driven triggers that can cause

entities to abandon waiting in the buffer.		
Trigger Type.Renege Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Renege Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Renege Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Renege Decision Type .Renege Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Renege Condition Or Probability.Renege Triggers	Expression Specified if the <i>Renege Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renegeing as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Renege Node Name.Renege Triggers	Node object reference	Facility node location to send a renegeing entity.
Trigger Start Boundary.Renege Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
Failure Type	No Failures, Calendar Time Based, Processing Count Based, Event Count Based, Processing Time Based	Specifies whether this Workstation object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample form a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.

On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the Workstation object.
Before Exiting (State assignments)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the Workstation object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is reneging from an input or output buffer of the object.
Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Secondary Resources	Repeating Property	The secondary resources that are required for activities at this Workstation object.
SecondaryResources.Object Type	Specific, From List	The method for specifying the object(s) required as a secondary resource.
SecondaryResources.Object Name	Object Instance Name	The name of the object type that is required.
SecondaryResources.Object List Name	Object List Instance Name	The name of the object list from which one or more objects will be selected.
SecondaryResources.Activity Range	All, Specific, All But Specific	The range of activities that the secondary resource is required for.
SecondaryResources.Activity	Setup, Processing, Teardown	The specific activity at this Workstation object that the secondary resource either is or is not required for.
SecondaryResources.Request Move	None, To Node	Indicates whether a move to a specified location will be requested from the seized object(s). Workstation activity will not continue until all resources have arrived to the requested location.
SecondaryResources.Destination Node	Node Instance Name	The name of the specific node location that the seized resource(s) will be requested to move to.
SecondaryResources.Move Priority	Expression	The priority of the move request if a PlanVisit step or SelectVisit step is used by the seized resource(s) to choose a visit destination from multiple candidates. If not specified, and the owner object performing the seize is an entity object, then this property defaults to the processing entity's Priority state value.
SecondaryResources.Number Of Objects	Expression truncated to integer	The number of objects that are required.

SecondaryResources.Units Per Object	Expression truncated to integer	The number of capacity units that are required per object.
(SecondaryResources)Selection Goal	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value	The goal used to rank object preference when multiple candidates are available.
(SecondaryResources)Selection Condition	Expression	An optional condition evaluated for each candidate object that must be true for the object to be eligible for seizing. In the condition, use the keyword Candidate to reference an object in the collection of candidates.
SecondaryResources.Selection Goal	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value, Random	The goal for selecting objects to seize.
SecondaryResources.Selection Expression	Expression	The expression evaluated for each object that is a candidate to seize. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., Candidate.Object.Capacity.Remaining).
SecondaryResources.Selection Condition	Expression	An optional condition evaluated for each candidate object that must be true for the object to be eligible for seizing. In the condition, use the keyword Candidate to reference an object in the collection of candidates (e.g., Candidate.Object.Capacity.Remaining)
Materials	Repeating Property	Materials consumed or produced by processing activities at this workstation
Materials.Action Type	Consume, Produce	The type of material-related action required.
Materials.Consumption Type	Material, Bill Of Materials	Indicates whether to consume a single material or bill of materials. The material consumption will be required to start the processing time of a batch (i.e., is a material consumption requirement per batch).
Materials.Production Type	Material, Bill Of Materials	Indicates whether to produce a single material or bill of materials. The material production will occur after finishing the processing time of a batch (i.e., is a material production result per batch).
Materials.Material Name	Material Instance Name	The name of the material which is to be either specifically consumed or produced, or whose Bill of Materials is to be consumed or produced.
Materials.Quantity	Expression	The quantity to be consumed or produced.
Materials.Lot ID	String	Optional string value indicating the lot identifier of the consumed or produced material.
Materials.Produced Delay Time	Expression	An optional delay time after finishing the processing time of a batch until the produced material is actually added to the system.

Maximum Makespan	Expression	The maximum length of the time allowed for this operation, from the start of the setup activity until the completion of the teardown activity at this Workstation object. An operation cannot be started unless its estimated makespan is less than this Maximum value minus the Makespan Buffer Time. Note that, if the Workstation object is following a Work Schedule, then 'off-shift' periods in the schedule will be factored into and extend the estimated makespan time.
Makespan Buffer Time	Expression	An operation cannot be started unless its estimated makespan is less than the Maximum Makespan minus this Buffer Time.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to this object are rolled up into. If a parent cost center is not explicitly specified, then costs will be automatically rolled up into the parent object containing this object.
Capital Cost	Expression	The initial one-time setup cost to add this object to the system.
Cost Per Use (Input Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Input Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Cost Per Use (Output Buffer Costs)	Expression	The cost to hold an entity in this buffer irrespective of the waiting time.
Holding Cost Rate (Output Buffer Costs)	Expression	The cost per unit time to hold an entity in this buffer.
Idle Cost Rate (Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of the workstation during periods that the workstation is scheduled but unutilized.
Cost Per Use (Resource Costs)	Expression	The one-time cost that is accrued each time the workstation is used, regardless of the usage duration. This cost will be charged to the cost of the entity using the workstation.
Setup Cost Rate (Resource Costs)	Expression	The cost per unit time to use the workstation for setup activity. This cost will be charged to the cost of the entity using the workstation.
Processing Cost Rate (Resource Costs)	Expression	The cost per unit time to use the workstation for processing activity. This cost will be charged to the cost of the entity using the workstation.
Teardown Cost Rate (Resource Costs)	Expression	The cost per unit time to use the workstation for teardown activity. This cost will be charged to the cost of the entity using the workstation.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance	Occurs when the simulation run is ending.

	Name	
Entered Add-On Process	Process Instance Name	Occurs immediately after an entity as entered this object and is about to start TransferIn Time.
Setting Up Add-On Process	Process Instance Name	Occurs when an operation's setup time is about to begin.
Finished SetUp Add-On Process	Process Instance Name	Occurs when an operation's setup time is finished and the release of secondary resources due to finish setup is about to occur.
Processing Batch Add-On Process	Process Instance Name	Occurs when the processing time for a batch is about to begin.
Processed Batch Add-On Process	Process Instance Name	Occurs when the processing time for a batch has completed and the material production or release of secondary resources due to finish the batch is about to occur.
Tearing Down Add-On Process	Process Instance Name	Occurs when an operation's teardown time is about to begin.
Finished Teardown Add-On Process	Process Instance Name	Occurs when an operation's teardown time is finished and the release of secondary resources due to finish teardown is about to occur.
Finished Good Operation Add-On Process	Process Instance Name	Occurs when an operation has finished and the makespan of the operation has satisfied the Maximum Makespan condition.
Finished Bad Operation Add-On Process	Process Instance Name	Occurs when an operation has finished and the makespan of the operation has exceeded the Maximum Makespan condition.
Exited Add-On Process	Process Instance Name	Occurs when an entity as exited this object.
Failed Add-On Process	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process	Process Instance Name	Occurs when a failure is repaired at this object.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Workstation object is evaluating whether to accept or reject a request to seize capacity of the workstation. In the executed decision process, assigning a value of less than or equal to '0' to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
On Shift Add-On Process	Process Instance Name	Occurs when the object goes On Shift because of its specified Work Schedule.
Off Shift Add-On Process	Process Instance Name	Occurs when the object goes Off Shift because of its specified Work Schedule.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data.

Transfer-In Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.
Transfer-In Condition	Expression	The condition required to allow an entity to transfer into this fixed object via an external input node.
Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out Condition	Expression	The condition required to allow an entity to transfer out of this fixed object via an external output node.
Expected Setup Time Expression	Expression	The expression used to estimate an expected setup time for an entity at this fixed object.
Expected Operation Time Expression	Expression	The expression used to estimate an expected operation time for an entity at this fixed object.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Workstation (Deprecated) - Discussion and Examples

Discussion

Operation Quantity and Batch Size

- The operation quantity specifies the number of "items" in the production lot that is represented by each arriving entity. This value can be a constant or be product/order dependent by referencing a table. The lot can be processed as a single production unit, or the lot can be broken into a sequence of batches that are processed one at a time until the entire lot has been produced. For example a lot of 1000 units may be processed in batches of 100 at a time. Note that a single setup and teardown activity is performed, but 10 separate batches of 100 are sequentially processed through the processing activity.

Material and Bill of Material

- Material or a Bill of Material may be required to start the processing of each batch. Material or a Bill of Material may also be produced at the end of each batch processing. A batch of produced material may also have a transfer time before it becomes available for consumption by other workstations. This makes it easy to model situations involving intermediate transfer batching.

Add a new [Material element](#) from the Elements panel of the Definitions Window. The Material icon can be found in the Workflow group of the Ribbon. *Bill of Material* is a repeating property that may contain component materials used to build the parent material. If 'Bill of Materials' is selected for the *Material Consumption* or the *Material Production* properties of the Workstation, all of the components of that Material are consumed or produced. If 'Material' is selected for *Material Consumption* or the *Material Production* properties of the Workstation, both units of the Material and units of the components of the Material (if there are components in the Bill of Materials) are consumed or produced.

If the workstation consumes material (either through the Bill of Materials or simply Material), the quantity of those materials must be available for the operation to continue. These material quantities are typically initialized using the Initial Quantity property of the Material element. The Produce step can be used to increase the available quantity of a given material or bill of materials during the simulation run.

In the example below, the Workstation object consumes one unit of Hamburger, FrenchFries, Toy and Milk each time a batch is processed. This is because a HappyMeal contains these 4 different Materials.

Consuming Material

Properties: Workstation1 (Workstation)

Process Logic	
Capacity Type	Fixed
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Operation Quantity	1.0
Setup Time Type	Specific
Setup Time	0.0
Processing Batch Size	
Processing Time	Random.Triangular(.1,.2,.3)
Teardown Time	0.0
Buffer Capacity	
Reliability Logic	
State Assignments	
Secondary Resources	
Materials & Other Constraints	
Material Consumption	Bill Of Materials
Consumed Material ...	HappyMeal
Consumed Material ...	1.0

Workstation Object Properties

Properties: HappyMeal (Material Element Instance)

Basic Logic	
Initial Quantity	100
Bill Of Materials	4 Rows
Financials	
Advanced Options	
General	
Name	HappyMeal
Description	
Public	True
Report Statistics	True

Material Properties (Definitions Window/Elements Panel)

Bill Of Materials - Repeating Property Editor

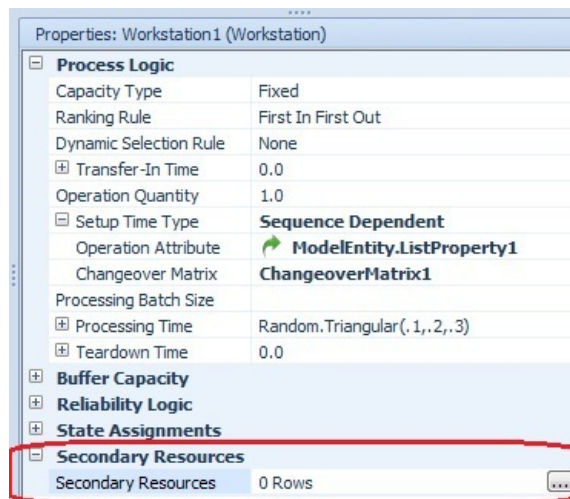
Items:	
FrenchFries, 1.0	
Hamburger, 1.0	
Milk, 1.0	
Toy, 1.0	
Add Delete	
Properties:	
Basic Logic	
Material Name	FrenchFries
Quantity	1.0
Basic Logic	
The bill of materials list for the material.	
Close	

Bill Of Materials (Happy Meal Material) - 4 Materials are required to make 1 Happy Meal

Primary and Secondary Resources

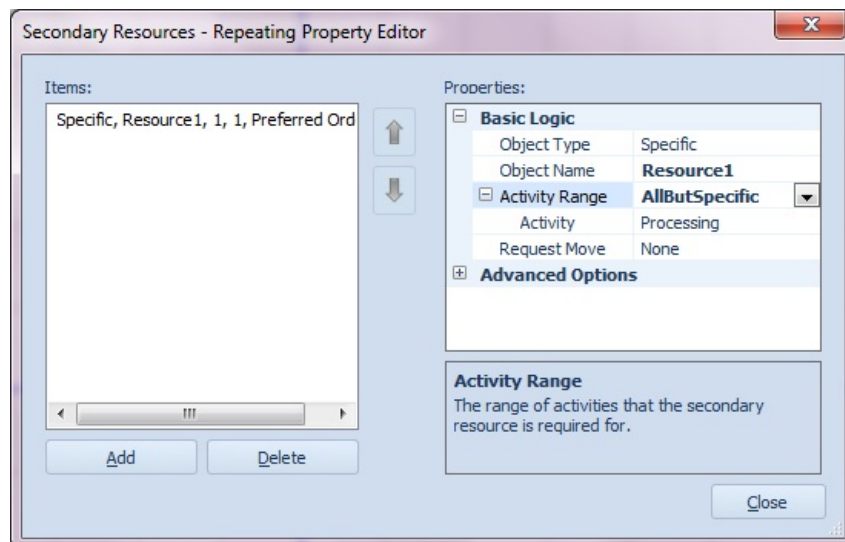
- The Workstation object is considered to be a primary resource that is required during the entire operation: i.e. setup, the processing of all batches, and teardown. In addition, any number of secondary resources may be required and held by each batch across one or more of the three activities. Secondary resources can be specific resources or selected based on rules from a list of resources. A secondary resource is added through the *Secondary Resource* property on the Workstation.

Adding a Secondary Resource to Workstation



In this example below, a Workstation is the primary resource and the secondary resource is an object named Resource1. Notice the *Activity Range* and *Activity* properties. Resource1 is only seized for the Setup and Teardown activities, not the Processing activity.

Details of the Secondary Resource on a Workstation



Change Dependent Setup Time

- The Setup Time for each entity arriving to the Workstation can be set to one duration if the value for Operation Attribute of the current operation is the same as it was for the previous operation and set to a different duration if the value for Operation Attribute of the current operation is different than the last operation. The *Setup Time Type* property should be set to 'Change Dependent'. The *Operation Attribute* is set to an expression that is evaluated to compare this operation to the previous operation and therefore leads to the use of either *Setup Time if Same* or *Setup Time if Different*. The following example of a Change Dependent set up time uses the State ModelEntity.Color to compare sequential operations and determine which setup time should be used. If two entities with the same value for ModelEntity.Color are processed directly after one another, the setup time for this Workstation is 1 minute. If two entities with different values for ModelEntity.Color are processed directly after one another, the setup time for this Workstation is 5 minutes.

Change Dependent Setup Time

Properties: Workstation1 (Workstation)

Process Logic	
Capacity Type	Fixed
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Operation Quantity	1.0
Setup Time Type	Change Dependent
Operation Attribute	ModelEntity.Color
Setup Time If Same	1
Units	Minutes
Setup Time If Dif...	5
Units	Minutes

Sequence Dependent Setup Time

- The Setup Time for each entity arriving to the Workstation can be set to a value that is determined by the combination of an attribute on the previous entity and the same attribute on the current entity (i.e. Setup times for From-To pairs are defined in a ChangeOverMatrix). The *Setup Time Type* property should be set to 'Sequence Dependent'. The *Operation Attribute* is evaluated as a zero based index into the Changeover Matrix string list. An *Operation Attribute* value of '0' means that the entity's comparison value is the first in the String List. The following example of a Sequence Dependent Setup time uses a List Property to compare sequential operations and determine which setup time should be used. The ModelEntity object has a String List with three values; Small, Medium and Large. Each ModelEntity has a different value for the ListProperty. Since the *Operation Attribute* is set to ModelEntity.ListProperty1, this is what is used to determine the Setup Time from the ChangeOver Matrix.

Sequence Dependent Setup Time

Properties: Workstation1 (Workstation)

Process Logic	
Capacity Type	Fixed
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Operation Quantity	1.0
Setup Time Type	Sequence Dependent
Operation Attribute	ModelEntity.ListProperty1
Changeover Matrix	ChangeoverMatrix1
Processing Batch Size	
Processing Time	Random.Triangular(.1,.2,.3)
Teardown Time	0.0

Name				
▼ Changeover Matrix				
ChangeoverMatrix1				
↓ From \ To →	Small	Medium	Large	
Small	0	0	0	
Medium	0	0	0	
Large	0	0	0	

Makespan and Capacity Changes

- The Workstation object has a fixed capacity of 1 unless you specify a work schedule. Please note that the Workstation object does currently have a Maximum Makespan feature that prohibits allowing an operation to start unless the job can finish within the specified makespan. The makespan estimate includes spanning any specified work schedule.

If the user manually assigns the capacity of a workstation using the state, WorkstationName.CurrentCapacity, instead of using the Work Schedule, care should be taken not to use the Maximum Makespan property. If the WorkstationName.CurrentCapacity is manually assigned to a value other than 0 or 1, a warning will be displayed (if warnings are not disabled) and the capacity will automatically adjusted to the value of '1'.

Workstation has ResourceState State Variable that Tracks Its States

-
- The ResourceState state variable is a [List State](#) automatically tracked for the Workstation. The numeric values for the state are Starved=0, Processing=1, Blocked=2, Failed=3, OffShift=4, Setup=5, Teardown=6.

You can use the ResourceState in process logic to check whether a workstation is 'Processing', 'Setup', 'Teardown' or other state. This would be done by utilizing the function Workstation.ResourceState in the logic, given that the token referencing this function is associated with the workstation. You will see Resource State results for the Workstation objects in the Results window.

To animate the workstation ResourceState, you simply specify the Workstation name and the desired function. For example, if you have a workstation named Workstation1, you may wish to place a Status Pie, specify the *Data Type* as 'ListState' and select Workstation1.ResourceState from the *List State* property list.

The ResourceState variable is used to display utilization statistics for the Workstation in the Results window. Additionally, this list state can be used to display utilization statistics during the simulation run. For example, you may wish to show the percentage of time that a workstation is processing during the simulation run. This would be done by placing a status label with the expression WorkstationName.ResourceState.PercentTime(1). For more information on list state values and utilization statistics, see [List State](#) page.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Flow Library

Flow Object Library

Simio has developed a Flow Library that currently includes 10 pre-built object definitions that can be used to model flow processing systems.

Name	Class	Description
FlowSource	Fixed	Generates a flow of fluid or other mass of a specified entity type.
FlowSink	Fixed	Destroys flow entities representing quantities of fluids or other mass that have finished processing in the model.
Tank	Fixed	Models a volume or weight capacity constrained location for holding entities representing quantities of fluids or other mass.
ContainerEntity	Entity	Models a type of simple moveable container (e.g., barrels or totes) for carrying flow entities representing quantities of fluids or other mass.
Filler	Fixed	Object used to model a process that fills container entities with flow entities representing quantities of fluids or other mass. The filling flow rate of the Filler is regulated by its 'FlowInput' flow node. Container entities enter the Filler through its 'ContainerInput' basic node and exit the Filler from its 'Output' transfer node.
Emptier	Fixed	Object used to model a process that empties the flow contents of container entities. The emptying flow rate of the Emptier is regulated by its 'FlowOutput' flow node. Container entities enter the Emptier through its 'Input' basic node and exit the emptier from its 'ContainerOutput' transfer node.
ItemToFlowConverter	Fixed	Object used to model a process that converts entities representing discrete items into flow entities representing quantities of fluids or other mass. The flow rate out of the ItemToFlowConverter is regulated by its 'Output' flow node. Discrete item entities enter the object through its 'Input' basic node and are then processed into flow.
FlowToItemConverter	Fixed	Object used to model a process that converts flow entities representing quantities of fluids or other mass into entities representing discrete items. The flow rate into the FlowToItemConverter is regulated by its 'Input' flow node. Discrete item entities exit the object from its 'Output' transfer node.
FlowNode	Node	Node designed to regulate the flow of entities representing quantities of fluid or other mass.
FlowConnector	Link	A zero-time connection between two FlowNodes.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FlowSource

The FlowSource object may be used to generate a flow of fluid or other mass of a specified entity type. The source of supply is by default considered to be infinite, but alternatively may be configured to run out if a stopping condition is met.

The flow rate out of the FlowSource is regulated by its 'Output' flow node.

Please refer to the SimBit [FlowConcepts](#) for models using the FlowSource.

Listed below are the properties of the **FlowSource**:

Property	Valid Entry	Description
Entity Type	Entity Instance Name	The entity type generated by the flow source.
Maximum Volume Out	Expression	The maximum volume to be generated by and flow out of this FlowSource object. If this limit is reached, then the FlowSource object will stop generating flow.
Maximum Weight Flow Out	Expression	The maximum weight to be generated by and flow out of this FlowSource object. If this limit is reached, then the FlowSource object will stop generating flow.
Maximum Time	Expression	The maximum time from the beginning of the simulation run after which this FlowSource object will stop generating flow.
Stop Event Name	Event Name	The name of the event which will cause this FlowSource object to stop generating flow.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FlowSink

The FlowSink object may be used to destroy the flow of entities representing quantities of fluids or other mass that have finished processing in the model.

The flow rate into the FlowSink is regulated by its 'Input' flow node.

Please refer to the SimBit [FlowConcepts](#) for models using the FlowSink.

Listed below are the properties of the **FlowSink**:

Property	Valid Entry	Description
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Tank

The Tank object may be used to model a volume or weight capacity constrained location for holding entities representing quantities of fluids or other mass.

The flow rate into the Tank is regulated by its 'Input' flow node. The flow rate out of the tank is regulated by its 'Output' flow node.

The tank may include *Initial Contents* of various entity types. Additionally, the *Auto Refill Mode* allows the specification of when, if at all, the tank will be automatically refilled. The tank may also be automatically emptied or cleaned upon particular event triggers and conditions. These are specified in the *Purge Contents Triggers* or *Clean In-Place Triggers* repeating properties.

[State assignments](#) can be made when the Tank reaches various levels when rising and falling. Additionally, the [Add-On Process Triggers](#) allow additional logic to be specified when using the Tank.

Please refer to the [FillingEmptyingTank](#) and [TransferringFromOneTankToAnother](#) models within the SimBit [FlowConcepts](#), as well as the [InfectionPropagationUsingContinuousAndFlow](#) SimBit for Tank examples.

Listed below are the properties of the **Tank**:

Property	Valid Entry	Description
Capacity Unit Type	Volume, Weight	The unit type to be used for specifying the tank's physical capacity.
Initial Volume Capacity	Expression	The initial maximum physical volume that can be stored in this tank.
Initial Weight Capacity	Expression	The initial maximum physical weight that can be stored in this tank.
Initial Contents	Repeat Group, Initial Contents	The contents present in the tank when the system is initialized.
Entity Type (Initial Contents)	Entity Instance Name	The initial content entity type.
Unit Type (Initial Contents)	Volume, Weight	The unit type to be used for specifying the initial content quantity.
Quantity (Initial Contents)	Expression	The initial content quantity.
Auto Refill Mode	No Automatic Refills, Refill When Empty, Refill When Low, Refill When Specific Event	Indicates whether the tank has automatic refills, and if so, whether a refill is triggered when the tank becomes empty, when its contents level has fallen below the 'Low' mark, or when some other specified triggering event occurs. The tank will be refilled to the maximum level determined by its physical volume or weight capacity.
Refill Triggering Event Name	Valid Event Name	The name of the event that will indicate when the tank is to be refilled.

Delay Time Until Refill	Expression	The delay time after a refill triggering event before the refill action is actually performed.
Refill Entity Type	Entity Instance Name	The entity type that the tank will be refilled with.
Purge Contents Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately remove and dispose of all contents in the tank, putting the tank into an empty state.
Triggering Event Name (Purge Contents Triggers)	Valid Event Name	The name of the event whose occurrence will indicate that all contents from the tank are to be immediately removed and disposed of, putting the tank into an empty state.
Condition (Purge Contents Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to purge the tank's contents.
Clean In-Place Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately remove and dispose of any contents held in the tank and then suspend new inflow until a specified cleaning time has elapsed.
Triggering Event Name (Clean In-Place Triggers)	Valid Event Name	The name of the event whose occurrence will indicate that the tank is to be cleaned.
Condition (Clean In-Place Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to trigger a cleaning operation.
Cleaning Time Type (Clean In-Place Triggers)	Specific, Sequence Dependent	The method used to specify the time required to clean the tank.
Cleaning Time (Clean In-Place Triggers)	Expression	The time required to clean the tank.
Operation Attribute (Clean In-Place Triggers)	Expression	The attribute expression to be evaluated for each cleaning operation and whose 'from/to' value changes will determine the sequence-dependent times required to clean the tank.
Initial Cleaning Time (Clean In-Place Triggers)	Expression	The time required to clean the tank if the first cleaning operation or if the changeover state stored for the previous operation has been cleared.
Changeover	Changeover	The name of the changeover matrix that defines the sequence-dependent

Matrix (Clean In-Place Triggers)	Matrix Name	cleaning times.
Low-Low Mark	Expression	The contents level value to be used as the tank's 'Low-Low' mark.
Low Mark	Expression	The contents level value to be used as the tank's 'Low' mark.
Mid Mark	Expression	The contents level value to be used as the tank's 'Mid' mark.
High Mark	Expression	The contents level value to be used as the tank's 'High' mark.
High-High Mark	Expression	The contents level value to be used as the tank's 'High-High' mark.
On New Inflow Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when a new inflow is starting to enter this Tank object, which may be either the first inflow into an empty tank or a change in inflow entity type into a partially full tank.
Above Low-Low Mark (On Tank Level Rising State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'Low-Low' mark.
Above Low Mark (On Tank Level Rising State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'Low' mark.
Above Mid Mark (On Tank Level Rising State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'Mid' mark.
Above High Mark (On Tank Level Rising State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'High' mark.
Above High-High Mark (On Tank Level Rising State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'High-High' mark.
Tank Full (On Tank Level Rising State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank has reached the tank's maximum volume or weight capacity.
Below High-	Repeat Group,	Optional state assignments when the contents level of the tank is rising

High Mark (On Tank Level Falling State Assignments)	Assignments	above the tank's 'High-High' mark.
Below High Mark (On Tank Level Falling State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'High' mark.
Above Mid Mark (On Tank Level Falling State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'Mid' mark.
Below Low Mark (On Tank Level Falling State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'Low' mark.
Below Low- Low Mark (On Tank Level Falling State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank is rising above the tank's 'Low-Low' mark.
Tank Empty (On Tank Level Falling State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of the tank has fallen to zero and the tank has thus become empty.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
New Inflow Entering Add-On Process	Process Instance Name	Occurs when a new inflow is starting to enter this Tank object, which may be either the first inflow into an empty tank or a change in the inflow entity type into a partially full tank. Note than a token executing this add-on process will be associated with the flow entity that is entering the tank. Also, if the logic of the specified process includes any delays, then the inflow rate into the tank will be halted until the process has completed.
Tank Level Rising - Above Low- Low Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is rising above the tank's 'Low-Low' mark.

Tank Level Rising - Above Low Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is rising above the tank's 'Low' mark.
Tank Level Rising - Above Mid Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is rising above the tank's 'Mid' mark.
Tank Level Rising - Above High Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is rising above the tank's 'High' mark.
Tank Level Rising - Above High-High Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is rising above the tank's 'High-High' mark.
Tank Level Rising - Tank Full Add-On Process	Process Instance Name	Occurs when the contents level of the tank has reached the tank's maximum volume or weight capacity.
Tank Level Falling - Below High-High Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is falling below the tank's 'High-High' mark.
Tank Level Falling - Below High Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is falling below the tank's 'High' mark.
Tank Level Falling - Below Mid Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is falling below the tank's 'Mid' mark.
Tank Level Falling - Below Low Mark Add-On Process	Process Instance Name	Occurs when the contents level of the tank is falling below the tank's 'Low' mark.
Tank Level Falling - Below Low-Low Mark Add-On	Process Instance Name	Occurs when the contents level of the tank is falling below the tank's 'Low-Low' mark.

Process		
Tank Level Falling - Tank Empty Add-On Process	Process Instance Name	Occurs when the contents level of the tank has fallen to zero and the tank has thus become empty.
Contents Ranking Rule	First In First Out, Last In First Out, Smallest value First, Largest Value First	The static rule used to rank flow entities located in this tank's 'StorageTank.Contents' queue.
Contents Ranking Expression	Expression	The expression used with a 'SmallestValueFirst' or 'LargestValueFirst' ranking rule.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ContainerEntity

The ContainerEntity object may be used to model a type of moveable container (e.g., barrels or totes) for carrying flow entities representing quantities of fluids or other mass.

The ContainerEntity provides an easy way to define a discrete entity type that represents a type of container for carrying flow. In terms of object design, this object is essentially a simplified combination of the ModelEntity and Tank objects.

[State assignments](#) can be made when the ContainerEntity becomes full or empty. Additionally, the [Add-On Process Triggers](#) allow additional logic to be specified when using the ContainerEntity.

The ContainerEntity has a Container Element defined inside it called "FlowContainer". The Container Element can be referenced to get information about its contents and current flow in and out. The expression to reference those items might look something like this "ContainerEntity.FlowContainer.Contents.Volume" or "ContainerEntity.FlowContainer.CurrentVolumeFlowOut". See the [Container](#) page for more information on the Container Element. Check out the SimBit [TrackingTankContentsByProductTypeUsingTable](#) for an example of the use of these expressions.

Listed below are the properties of the **ContainerEntity**:

Property	Valid Entry	Description
Capacity Unit Type	Volume, Weight	The unit type to be used for specifying the physical capacity of a container entity of this type.
Initial Volume Capacity	Expression	The initial maximum physical volume that can be stored in a container entity of this type.
Initial Weight Capacity	Expression	The initial maximum physical weight that can be stored in a container entity of this type.
Initial Contents	Repeat Group	The initial contents present in a container entity of this type when the entity is created.
Entity Type (Initial Contents)	Entity Instance Name	The initial content entity type.
Unit Type (Initial Contents)	Volume, Weight	The unit type to be used for specifying the initial content quantity.
Quantity (Initial Contents)	Expression	The initial content quantity.
Initial Desired Speed	Expression	The initial desired speed value for entities of this type.
Initial Network	Network Element Property	The initial network of links used by entities of this type to travel between node locations. This includes the default 'Global' network and 'No Network (FreeSpace)'.
Network Turnaround	Exit & Re- enter, Rotate	When traveling on a network, the default method used by objects of this type to turn around at a node in order to move in the opposite direction on a bidirectional

Method	in Place, Reverse	link. The 'Turnaround Method' property of a Network element may also be used to override this default on a network-by-network basis.
Free Space Steering Behavior	Direct To Destination, Follow Network Path If Possible	<p>The behavior used to steer an entity of this type when traveling in free space to a destination.</p> <p>'Direct To Destination' will steer an entity in a straight line to its destination.</p> <p>'Follow Network Path If Possible' will prefer to steer an entity along a path following its currently assigned network, staying within the boundaries of the drawn path width. The Entity will travel based on the drawn length of the path even if a different logical length is specified. However, if no followable network path exists to the entity's destination then the entity will be steered in a straight line. Note: In order to use this steering behavior, make sure the entity's travel mode is set to 'Free Space Only'.</p>
Update Interval	Expression	The time interval between updates checking an entity's adherence to the network path.
Avoid Collisions	True or False	Indicates whether the steering behavior will attempt to avoid collisions with other entities that are following the same network path.
Initial Priority	Expression	The initial priority value for entities of this type.
Initial Sequence	Sequence Property	An initial destination sequence that entities of this type use for routing purposes.
On Container Full (State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of a container entity of this type has reached the container's maximum volume or weight capacity.
On Container Empty (State Assignments)	Repeat Group, Assignments	Optional state assignments when the contents level of a container entity of this type has fallen to zero and the container has thus become empty.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Container Full Add-On Process	Process Instance Name	Occurs when the contents level of a container entity of this type has reached the container's maximum volume or weight capacity.
Container Empty Add-On Process	Process Instance Name	Occurs when the contents of a container entity of this type has fallen to zero and the container has thus become empty.
Initial Number in System	Numeric	The initial number of objects of this type in the system at the beginning of the simulation run. Entities are initially located in free-space at the object instance location.
Maximum Number in System	Numeric	The maximum number of objects of this type that can simultaneously be in the system. If this limit is exceeded then a runtime error is generated.
Destroyable	True or False	Specifies whether objects of this type can be destroyed using the Destroy Step.

Can Transfer In & Out of Objects	True or False	Indicates whether entities of this type are by default allowed to transfer into and out of fixed objects via those object's external input / output nodes.
Due Date Expression	Expression	The expression used to return a 'due date' value for an entity of this type.
Gantt Visibility	True or False	An expression evaluated at run time to determine if the entity should appear in the Entity Gantt window.
Contents Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static rule to rank flow entities located in the 'FlowContainer.Contents' queue of a container entity of this type.
Contents Ranking Expression	Expression	The expression used with a 'SmallestValueFirst' or 'LargestValueFirst' ranking rule.
Current Size Index	State Variable	The optional name of a state variable that indexes into the object's associated symbol list, and whose value will automatically resize the object to match the size of the indexed symbol.
Size.Length	Numeric	The initial graphical length of the entity.
Size.Width	Numeric	The initial graphical width of the entity.
Size.Height	Numeric	The initial graphical height of the entity.
Volume	Numeric	The initial logical volume of the object in cubic meters. If blank, the initial logical volume will be the implicit volume defined by the Size. At runtime, the ratio between the logical volume and the Size implicit volume will be kept constant.
Weight Density	Numeric	The weight density of an entity of this type in kilograms per cubic meter. The default value for density is 1 kilogram per cubic meter. At runtime, a change to Weight or Volume will make a change in the other to make sure Density remains constant.
Current Symbol Index	Expression	An expression that returns a value for the index in the list for the current symbol to display.
Random Symbol	True or False	If true, each new object will be randomly assigned a symbol from the associated list of symbols.
Current Animation Index	Expression	An expression that returns the numeric index or string name of the current animation of the active symbol. If the expression is not defined, returns an index that is out of the range of the list of animations, or a name of an animation that does not exist, Simio will fallback to using the action specified in the Default Animation Action.
Default Animation Action	Expression	Indicates if and how Simio will animate objects that don't have an explicit animation set via the Current Animation Index expression. None indicates no action will be taken. Moving indicates actively moving entities will be animated with any available forward movement animations from the active symbol. MovingandIdle indicates actively moving entity and idle objects will be animated with any available forward movement or idle animations from the active symbol.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Filler

The Filler object may be used to fill any type of entity that has one or more container locations, including an entity type defined using the ContainerEntity object. The properties, structure, and process logic design of the Filler is very similar to a Server from the Standard Library. The Filler has InputBuffer, Processing, and OutputBuffer station locations, all of which are animated within the Facility window. Container entities transfer through those stations in sequence, entering the Filler using its 'ContainerInput' BasicNode and exiting the Filler using its 'Output' TransferNode. The primary difference between a Filler and Server is that instead of a 'Processing Time' delay incurred by an entity that uses a Server, the processing delay incurred by an entity at a Filler is instead determined by the time it takes to fill the entity's container location(s). The flow stream that fills containers at the Filler is pulled from the Filler's 'FlowInput' FlowNode. Note that the capacity of the Filler is automatically set to '1' (Fixed) and should be set to a *Value* of '1' when using a work schedule.

The Filler object may fill the ContainerEntity object provided in the Flow Library, or a user specified type of entity (or transporter) object that has containers. If the entity that enters the Filler has a Container element, then that container can be filled. If an entity enters the Filler object and it has no container locations, then the filling time will be 0.0 and a warning will be displayed to alert the user that there may be a modeling logic issue. Similarly, if an entity enters the Filler object and its container location(s) are already full, then the filling time will be 0.0. The Filler object provides two *Fill Target Types*. A container can either be filled until full, or a *Desired Quantity* can be added to each container using an expression. Note that if the *Desired Quantity* expression does not return a positive quantity, then the filling time will be 0.0. Thus the user might use a desired quantity expression that returns a value of 0.0 if the current container is not desired to be filled (e.g., don't fill a container unless it already contains flow contents of a certain type). The current flow container location being filled by a Filler may be referenced in an expression by looking at its CurrentFlowContainer state variable. If the *Fill Target Type* is 'Add Specific Quantity', then the filling operation will be considered completed when the desired quantity is added or the container becomes full, whichever event occurs first. Note that if using work schedules and the Filler goes off-shift, then the current filling operation is immediately suspended and will then be resumed when the Filler goes back on-shift. This immediate suspension of all current processing activity is consistent with Simio's other 'Processor' type objects.

The Filler may stop filling earlier than the specified fill target by using the *Stop Early Triggers* repeating properties.

The [Balking and Reneging Options](#) page provides additional information and examples on buffer capacities, balking and reneging for the input and output buffers.

A Filler may have failures, which are specified within the [Reliability](#) section of the Properties window. [State assignments](#) can be made upon entering, before processing, after processing, before exiting, on balking and on reneging the Filler. [Secondary Resources](#) may be seized and/or released by the entity upon entering the Filler, and before and after filling occurs. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Filler. The Filler Add-On Process Triggers are unique in that there are processes available for Before Processing and After Processing (as with Server), but there are also processes for Filling Container and Filled Container, as these may be triggered multiple times for an entity object entering the Filler with multiple containers.

An entity processing at a Filler may NOT be interrupted with the Interrupt step.

The *Seize Constraint Logic* repeat group supports imposing additional constraints on an entity's request to seize the object by referencing [Constraint Logic](#) elements.

The *Immediately Try Seize* and *Immediately Try Allocate When Released* options within the Other Processing Options (and Secondary Resources area) section of properties are typically advanced users. They deal with processing events on the Simio event calendar that occur at the same time, including arrival of multiple entities to an object, or release of multiple objects at a time for re-allocation. It's important to note that these properties do not affect processing for entities that arrive on path with *Allow Passing* set to 'False', entities arriving via Conveyor, or any other construct that causes entities to arrive one at a time at different simulation times. This feature will not provide any benefit because the second (or following) entity will not arrive before the late event happens at the first entity's simulation time.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Filler.

Listed below are the properties of the **Filler**:

Property	Valid Entry	Description
Capacity Type	Fixed, Work Schedule	The availability of this Filler for processing. 'Fixed' indicates that this Filler is always available. 'WorkSchedule' indicates that this Filler follows a work schedule defining its 'On-Shift, Off-Shift' availability over time.

Work Schedule	Work Schedule Name	The name of the schedule that specifies the capacity for this resource over time.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order entities waiting to be allocated capacity of this Filler object.
Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' ranking rule.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this resource, when its capacity becomes available, dynamically selects the next seize request from its statically ranked allocation queue using a dynamic selection rule.
Repeat Group	True, False	Indicates whether a repeat group data structure is used to define the primary dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (e.g., Candidate.Entity.Priority).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.
Transfer-In Time	Expression	The time required to transfer an entity into this object.
Fill Target Type	Fill Until Full, Add Specific Quantity	Indicates whether the Filler object will fill each container until full or add a specific desired quantity.
Quantity Unit Type	Volume, Weight	The unit type to be used for specifying the desired quantity to add to a container.
Desired Quantity	Expression	The desired quantity to add to a container. Filling activity will end once this volume or weight quantity has been added or the container has become full, whichever event occurs first.
Stop Early Triggers	Repeat Group,	Optional event-driven triggers that will immediately cause the Filler

	Triggers	object to stop filling a container early, before reaching the specified fill target.
Triggering Event Name (Stop Early Triggers)	Valid Event Name	The name of the event whose occurrence will trigger the Filler object to stop filling a container early, before reaching the specified fill target.
Condition (Stop Early Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to trigger the Filler object to stop early.
Off Shift Rule	Suspend Processing, Finish Work Already Started	<p>The processing rule used at the Filler at the end of a shift.</p> <p>If the rule is 'Suspend Processing', then the Filler will immediately suspend all processing and set its resource state to 'OffShift'. Processing will resume at the start of the next shift.</p> <p>If the rule is 'Finish Work Already Started', then the Filler will not accept any new entities but will continue processing if necessary to finish work already started. The Filler's resource state will be set to 'OffShiftProcessing' if processing entities during an off-shift period. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Seize Constraint Logic	Repeat Group, Constraint Logic elements	<p>Constraint logic used to enforce additional constraints on an entity's request to seize the Filler.</p> <p>NOTE: This repeat group applies only if the Filler has an input buffer. Otherwise, if no input buffer, then this constraint logic will be ignored.</p>
Seize Constraint Logic.Constraint Logic Name	Constraint Logic elements	The name of the Constraint Logic element used to enforce additional constraints on an entity's request to seize the Filler.
Immediately Try Seize	True, False	<p>For an arriving container entity, indicates whether to immediately try seizing the Filler before the execution of any other simulation logic in the system and, if successful, skipping the Filler's allocation queue. Setting this property to False will just insert the seize request into the Filler's allocation queue. An evaluation of that queue will then be scheduled on the simulation's current event calendar as a late priority event.</p> <p>NOTE: This property setting applies only if the Filler has an input buffer. Otherwise, if no input buffer, then an arriving entity will always immediately try to enter the Filler's processing station and seize it.</p>
Immediately Try Allocate When Released	True, False	<p>Once a container entity has exited processing and released the Filler, indicates whether to immediately try allocating the released Filler capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the Filler's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>
Input Buffer Capacity	Integer ≥ 0	The number of discrete entities that can be held in this Filler object's input buffer.
Balk Decision Type (Input Buffer)	None, Blocked, Conditional, Probabilistic	<p>The method used by an entity to decide whether to balk at entering the buffer.</p> <p>If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.</p>

Balk Condition Or Probability (Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Filler object's output buffer.
Balk Decision Type (Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Output Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Reneged Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.
Trigger Type.Reneged Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Reneged Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Reneged Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Reneged Decision Type .Reneged Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Reneged Condition Or Probability.Reneged Triggers	Expression Specified if the <i>Reneged Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renegeing as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Reneged Node Name.Reneged Triggers	Node object reference	Facility node location to send a renegeing entity.
Trigger Start Boundary.Reneged Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).

Failure Type	No Failures, Calendar Time Based, Processing Count Based, Event Count Based, Processing Time Based	Specifies whether this Filler object has failures that affect the object's availability, and the method used to trigger the failure occurrences. See the Reliability page for additional information on Failures.
Failure Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Action Type (Table Row Referencing - Before Processing)	Reference Existing Row, Add New Row	<p>The type of table row reference action to perform.</p> <p>The 'Reference Existing Row' action type may be used to set a reference to an existing table row in a data table or sequence table. The row index into the table is specified by the <i>Row Number</i> property.</p> <p>The 'Add New Row' action type may be used to add a new row to an output table. A reference will be automatically set to the newly added row.</p>
Table Name (Table Row Referencing - Before Processing)	Table Name	The name of the table.
Row Number (Table Row Referencing - Before Processing)	Expression	The one-based row index into the table.
On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the Filler object.
Before Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has been allocated capacity to be processed at the object, but before entering (or ending transfer into) the object's processing station.
After Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has completed its processing and is about to attempt its exit from the object's processing station.
Before Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the Filler object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is reneging from an input or output buffer of the object.

Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Repeat Group (Secondary Resource for Processing)	True, False	Indicates whether a repeat group data structure is used to define the secondary resources for processing. If a repeat group structure is used, many of the below properties are specified within the repeat group. <i>OffShift Rule</i> , <i>Must Simultaneously Seize</i> , <i>Immediately Try Seize</i> and <i>Immediately Try Allocate When Released</i> are not within the repeat group, thus not resource specific.
Resource Type (Secondary Resource for Processing)	Specific, From List, ParentObject	The method for specifying the resource object(s) to seize.
Resource Name (Secondary Resource for Processing)	Object Instance Name	The name of the resource object to seize.
Resource List Name (Secondary Resource for Processing)	Object List Instance Name	The name of the object list from which to select the resource object(s) to seize.
Selection Goal (Secondary Resource for Processing)	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value, Random	The goal for selecting objects to seize.
Value Expression (Secondary Resource for Processing)	Expression	The expression evaluated for each object that is a candidate to seize. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., Candidate.Object.Capacity.Remaining).
Request Move (Secondary Resource for Processing)	None, To Node	Indicates whether a move to a specified location will be requested from the seized resource. Processing will not be able to start until the resource has arrived to the requested location.
Destination Node (Secondary Resource for Processing)	Node Instance Name	The name of the specific node location that the seized resource will be requested to move to.
Off Shift Rule (Secondary Resource for Processing)	Finish Work Already Started, Suspend Processing, Switch	The processing rule used if the secondary resource is at the end of a shift because of a specified work schedule. If the rule is 'Finish Work Already Started', then the processing of the

Processing)	Resources If Possible	<p>entity that is using the secondary resource will be allowed to continue processing until finished. The secondary resource will not be allowed to accept any new work.</p> <p>If the rule is 'Suspend Processing', then the processing of the entity that is using the secondary resource will be immediately suspended. Processing will resume at the start of the secondary resource's next shift.</p> <p>If the rule is 'Switch Resources If Possible', then the processing of the entity that is using the secondary resource will be immediately suspended. The entity will then try to resume processing as soon as possible by releasing the resource and seizing another available one that satisfies the same resource requirements. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Number of Resources (Secondary Resource for Processing)	Expression	The number of individual resource objects to seize capacity units of.
Units Per Resource (Secondary Resource for Processing)	Expression	The number of capacity units to seize per resource object.
Selection Condition (Secondary Resource for Processing)	Expression	<p>Optional condition evaluated for each candidate resource that must be true for the resource to be selected.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).</p>
Must Simultaneously Seize (Secondary Resource for Processing)	True, False	If multiple resources are required, indicates whether all of the resources must be available before any can be seized.
Immediately Try Seize (Secondary Resource for Processing)	True, False	<p>Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues.</p> <p>Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.</p>
Keep Reserved If (Secondary Resource for Processing)	Expression	<p>On processing completion, an optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects without a reservation will be unable to seize the reserved capacity unless the reservation is cancelled.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>).</p> <p>Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p>

		Leaving this property blank (no condition) is equivalent to entering False.
Reservation Timeout (Secondary Resource for Processing)	Expression	If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation. In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released (e.g., Candidate.Object.Capacity).
Immediately Try Allocate When Released (Secondary Resource for Processing)	True, False	Once an entity has finished processing and released the resource, indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of an other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
On Entering (Other Resource Seizes)	Repeat Group, Resource Seizes	Secondary resource seizes required when an entity is entering the Filler object.
Must Simultaneously Seize (Other Resource Seizes -- On Entering)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- On Entering)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Secondary resource seizes required once an entity has been allocated Filler capacity, but before entering (or ending transfer into) its processing station.
Must Simultaneously Seize (Other Resource Seizes -- Before Processing)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- Before Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Secondary resource seizes required after all possible container elements of an entity have been filled and the entity is now about to attempt its exit from the Filler's processing station.
Immediately Try Seize (Other Resource Seizes -- After Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues.

After Processing)		Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
Must Simultaneously Seize	If multiple resources are required, indicates whether all must be available before any can be seized.	
On Entering (Other Resource Releases)	Repeat Group, Resource Releases	Secondary resource releases to be performed when an entity is entering the Filler object.
Immediately Try Allocate When Released (Other Resource Releases -- On Entering)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Releases)	Repeat Group, Resource Releases	Secondary resource releases to be performed once an entity has been allocated Filler capacity, but before entering (or ending transfer into) its processing station.
Immediately Try Allocate When Released (Other Resource Releases -- Before Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Releases)	Repeat Group, Resource Releases	Secondary resource releases to be performed after all possible container elements of an entity have been filled and the entity is now about to attempt its exit from the Filler's processing station.
Immediately Try Allocate When Released (Other Resource Releases -- After Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs immediately after an entity has entered this object and is about to start the 'Transfer-In Time'.
Before Processing Add-On Process	Process Instance Name	Occurs when an entity has been allocated Filler capacity, but before entering (or ending transfer into) the Filler's processing station.
Filling Container Add-On Process	Process Instance Name	Occurs when the Filler is about to start filling a container element of the current entity in its processing station.
Filled Container	Process Instance	Occurs when the Filler has finished filling a container element of the

Add-On Process	Name	current entity in its processing station.
After Processing Add-On Process	Process Instance Name	Occurs when all possible container elements of an entity have been filled, and the entity is now about to attempt its exit from the Filler's processing station.
Exited Add-On Process	Process Instance Name	Occurs when an entity has exited this object.
Failed Add-On Process	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process	Process Instance Name	Occurs when a failure is repaired at this object.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Filler object is evaluating whether to accept or reject a request to seize capacity of the filler. In the executed decision process, assigning a value of less than or equal to '0' to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
On Shift Add-On Process	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process	Process Instance Name	Occurs when the object goes 'off shift' because of its specified work schedule.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Transfer-In Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.
Transfer-In Condition	Expression	The condition required to allow an entity to transfer into this fixed object via an external input node.
Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out Condition	Expression	The condition required to allow an entity to transfer out of this fixed object via an external output node.
Expected Setup Time Expression	Expression	The expression used to get a deterministic estimation of the setup time for an entity at this fixed object.
Expected Operation Time	Expression	The expression used to get a deterministic estimation of the operation time for an entity at this fixed object.

Expression		
Stop Early Event Name	Valid Event Name	Optional name of an event whose occurrence will trigger the Filler object to stop filling a container early, before reaching the specified fill target. NOTE: This property has been deprecated. Use the Process Logic Stop Early Triggers repeat group instead.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Emptier

The Emptier object may be used to empty any type of entity that has one or more container locations, including an entity type defined using the ContainerEntity object. The properties, structure, and process logic design of the Emptier is very similar to a Server from the Standard Library. The Emptier has InputBuffer, Processing, and OutputBuffer station locations, all of which are animated within the Facility window. Container entities transfer through those stations in sequence, entering the Emptier using its 'Input' BasicNode. Discrete entities then exit the Emptier using its 'ContainerOutput' TransferNode while flow stream exits the Emptier using the 'FlowOutput' FlowNode. The primary difference between an Emptier and Server is that instead of a 'Processing Time' delay incurred by an entity that uses a Server, the processing delay incurred by an entity at an Emptier is instead determined by the time it takes to empty the entity's container location(s). Note that the capacity of the Emptier is automatically set to '1' (Fixed) and should be set to a *Value* of '1' when using a work schedule.

The Emptier object may empty the ContainerEntity object provided in the Flow Library, or a user specified type of entity (or transporter) object that has containers. If the entity that enters the Emptier has a Container element, then that container can be emptied. If an entity enters the Emptier object and it has no container locations, then the emptying time will be 0.0 and a warning will be displayed to alert the user that there may be a modeling logic issue. Similarly, if an entity enters the Emptier object and its container location(s) are already empty, then the emptying time will be 0.0. The Emptier object provides two *Empty Target Types*. A container can either be emptied until empty, or a *Desired Quantity* can be removed from each container using an expression. Note that if the *Desired Quantity* expression does not return a positive quantity, then the emptying time will be 0.0. Thus the user might use a desired quantity expression that returns a value of 0.0 if the current container is not desired to be emptied (e.g., don't empty a container unless it already contains flow contents of a certain type). The current flow container location being emptied by an Emptier may be referenced in an expression by looking at its CurrentFlowContainer state variable. If the *Empty Target Type* is 'Remove Specific Quantity', then the emptying operation will be considered completed when the desired quantity is removed or the container becomes empty, whichever event occurs first. Note that if using work schedules and the Emptier goes off-shift, then the current emptying operation is immediately suspended and will then be resumed when the Emptier goes back on-shift. This immediate suspension of all current processing activity is consistent with Simio's other 'Processor' type objects.

The Emptier may stop emptying earlier than the specified empty target by using the *Stop Early Triggers* repeating properties.

The [Balking and Reneging Options](#) page provides additional information and examples on buffer capacities, balking and reneging for the input and output buffers.

An Emptier may have failures, which are specified within the [Reliability](#) section of the Properties window. [State assignments](#) can be made upon entering, before processing, after processing, before exiting, on balking and on reneging the Emptier. [Secondary Resources](#) may be seized and/or released by the entity upon entering the Emptier, and before and after emptying occurs. The [Add-On Process Triggers](#) allow additional logic to be specified when using the Emptier. The Emptier Add-On Process Triggers are unique in that there are processes available for Before Processing and After Processing (as with Server), but there are also processes for Emptying Container and Emptied Container, as these may be triggered multiple times for an entity object entering the Emptier with multiple containers.

An entity processing at a Emptier may NOT be interrupted with the Interrupt step.

The *Seize Constraint Logic* repeat group supports imposing additional constraints on an entity's request to seize the object by referencing [Constraint Logic](#) elements.

The *Immediately Try Seize* and *Immediately Try Allocate When Released* options within the Other Processing Options (and Secondary Resources area) section of properties are typically advanced users. They deal with processing events on the Simio event calendar that occur at the same time, including arrival of multiple entities to an object, or release of multiple objects at a time for re-allocation. It's important to note that these properties do not affect processing for entities that arrive on path with *Allow Passing* set to 'False', entities arriving via Conveyor, or any other construct that causes entities to arrive one at a time at different simulation times. This feature will not provide any benefit because the second (or following) entity will not arrive before the late event happens at the first entity's simulation time.

The [Work Day and Work Period Exceptions](#) pages provides additional information on work schedule exceptions for the Emptier.

Listed below are the properties of the **Emptier**:

Property	Valid Entry	Description
Capacity Type	Fixed, Work Schedule	The availability of this Emptier. 'Fixed' indicates that this Emptier is always available. 'WorkSchedule' indicates that this Emptier follows a work schedule defining its 'On-Shift, Off-Shift' availability over time.

Work Schedule	Work Schedule Name	The name of the schedule that specifies the capacity for this resource over time.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order entities waiting to be allocated capacity of this Emptier object.
Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' ranking rule.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this resource, when its capacity becomes available, dynamically selects the next seize request from its statically ranked allocation queue using a dynamic selection rule.
Repeat Group	True, False	Indicates whether a repeat group data structure is used to define the primary dispatching rule and any tie breaker rules.
Dispatching Rule	See Dynamic Selection Rules	The primary criteria used to select the next entity from the queue. Note that using a particular dispatching rule may require some specific model data about the candidate entities, such as due dates, job routings, expected setup or operation times, etc.
Tie Breaker Rule	None, or one of several Dynamic Selection Rules	The secondary criteria used to break ties.
Dispatching Rules	Repeat Group, Dispatching Rules	The primary dispatching rule and any tie breaker rules, applied in the order listed.
Filter Expression	Expression	Optional condition evaluated for each candidate entity that must be true for the entity to be selected. In the expression, use the syntax Candidate.[EntityClass].[Attribute] to reference an attribute of the candidate entities (e.g., Candidate.Entity.Priority).
Look Ahead Window (Days)	Expression	Only candidate entities whose due dates fall within this specified time window will be considered for selection. Note that if there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.
Transfer-In Time	Expression	The time required to transfer an entity into this object.
Empty Target Type	Empty All Contents, Remove Specific Quantity	Indicates whether the Emptier object will empty all contents from each container or remove a specific desired quantity.
Quantity Unit Type	Volume, Weight	The unit type to be used for specifying the desired quantity to remove from a container.
Desired Quantity	Expression	The desired quantity to remove from a container. Emptying activity will end once this volume or weight quantity has been removed or the container has become empty, whichever event occurs first.

Stop Early Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately cause the Emptier object to stop emptying a container early, before reaching the specified empty target.
Triggering Event Name (Stop Early Triggers)	Valid Event Name	The name of the event whose occurrence will trigger the Emptier object to stop emptying a container early, before reaching the specified empty target.
Condition (Stop Early Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to trigger the Emptier object to stop early.
Off Shift Rule	Suspend Processing, Finish Work Already Started	<p>The processing rule used at the Emptier at the end of a shift. If the rule is 'Suspend Processing', then the Emptier will immediately suspend all processing and set its resource state to 'OffShift'. Processing will resume at the start of the next shift.</p> <p>If the rule is 'Finish Work Already Started', then the Emptier will not accept any new entities but will continue processing if necessary to finish work already started. The Emptier's resource state will be set to 'OffShiftProcessing' if processing entities during an off-shift period. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Seize Constraint Logic	Repeat Group, Constraint Logic elements	<p>Constraint logic used to enforce additional constraints on an entity's request to seize the Emptier.</p> <p>NOTE: This repeat group applies only if the Emptier has an input buffer. Otherwise, if no input buffer, then this constraint logic will be ignored.</p>
Seize Constraint Logic.Constraint Logic Name	Constraint Logic elements	The name of the Constraint Logic element used to enforce additional constraints on an entity's request to seize the Emptier.
Immediately Try Seize	True, False	<p>For an arriving container entity, indicates whether to immediately try seizing the Emptier before the execution of any other simulation logic in the system and, if successful, skipping the Emptier's allocation queue.</p> <p>Setting this property to False will just insert the seize request into the Emptier's allocation queue. An evaluation of that queue will then be scheduled on the simulation's current event calendar as a late priority event.</p> <p>NOTE: This property setting applies only if the Emptier has an input buffer. Otherwise, if no input buffer, then an arriving entity will always immediately try to enter the Emptier's processing station and seize it.</p>
Immediately Try Allocate When Released	True, False	<p>Once a container entity has exited processing and released the Emptier, indicates whether to immediately try allocating the released Emptier capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the Emptier's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>
Input Buffer Capacity	Integer ≥ 0	The number of discrete entities that can be held in this Emptier object's input buffer.
Balk Decision Type (Input Buffer)	None, Blocked, Conditional,	The method used by an entity to decide whether to balk at entering the buffer.

	Probabilistic	If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Output Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this Emptier object's output buffer.
Balk Decision Type (Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Output Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Reneged Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.
Trigger Type.Reneged Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Reneged Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Reneged Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Reneged Decision Type .Reneged Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Reneged Condition Or Probability.Reneged Triggers	Expression Specified if the <i>Reneged Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renegeing as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queueing' namespace provides several functions for that purpose (e.g., Entity.Queueing.IsWaitingRidePickupQueue).
Reneged Node Name.Reneged Triggers	Node object reference	Facility node location to send a renegeing entity.

Trigger Start Boundary.Renege Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
Failure Type	No Failures, Calendar Time Based, Processing Count Based, Event Count Based, Processing Time Based	Specifies whether this Emptier object has failures that affect the object's availability, and the method used to trigger the failure occurrences. See the Reliability page for additional information on Failures.
Failure Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample form a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Action Type (Table Row Referencing - Before Processing)	Reference Existing Row, Add New Row	<p>The type of table row reference action to perform.</p> <p>The 'Reference Existing Row' action type may be used to set a reference to an existing table row in a data table or sequence table. The row index into the table is specified by the <i>Row Number</i> property.</p> <p>The 'Add New Row' action type may be used to add a new row to an output table. A reference will be automatically set to the newly added row.</p>
Table Name (Table Row Referencing - Before Processing)	Table Name	The name of the table.
Row Number (Table Row Referencing - Before Processing)	Expression	The one-based row index into the table.
On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the Emptier object.
Before Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has been allocated capacity to be processed at the object, but before entering (or ending transfer into) the object's processing station.
After Processing (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity has completed its processing and is about to attempt its exit from the object's processing station.
Before Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is ready to exit the Emptier object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.

On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is reneging from an input or output buffer of the object.
Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Repeat Group (Secondary Resource for Processing)	True, False	Indicates whether a repeat group data structure is used to define the secondary resources for processing. If a repeat group structure is used, many of the below properties are specified within the repeat group. <i>OffShift Rule</i> , <i>Must Simultaneously Seize</i> , <i>Immediately Try Seize</i> and <i>Immediately Try Allocate When Released</i> are not within the repeat group, thus not resource specific.
Resource Type (Secondary Resource for Processing)	Specific, From List, ParentObject	The method for specifying the resource object(s) to seize.
Resource Name (Secondary Resource for Processing)	Object Instance Name	The name of the resource object to seize.
Resource List Name (Secondary Resource for Processing)	Object List Instance Name	The name of the object list from which to select the resource object(s) to seize.
Selection Goal (Secondary Resource for Processing)	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value, Random	The goal for selecting objects to seize.
Value Expression (Secondary Resource for Processing)	Expression	The expression evaluated for each object that is a candidate to seize. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., <code>Candidate.Object.Capacity.Remaining</code>).
Request Move (Secondary Resource for Processing)	None, To Node	Indicates whether a move to a specified location will be requested from the seized resource. Processing will not be able to start until the resource has arrived to the requested location.
Destination Node (Secondary Resource for Processing)	Node Instance Name	The name of the specific node location that the seized resource will be requested to move to.

Processing)		
Off Shift Rule (Secondary Resource for Processing)	Finish Work Already Started, Suspend Processing, Switch Resources If Possible	<p>The processing rule used if the secondary resource is at the end of a shift because of a specified work schedule.</p> <p>If the rule is 'Finish Work Already Started', then the processing of the entity that is using the secondary resource will be allowed to continue processing until finished. The secondary resource will not be allowed to accept any new work.</p> <p>If the rule is 'Suspend Processing', then the processing of the entity that is using the secondary resource will be immediately suspended. Processing will resume at the start of the secondary resource's next shift.</p> <p>If the rule is 'Switch Resources If Possible', then the processing of the entity that is using the secondary resource will be immediately suspended. The entity will then try to resume processing as soon as possible by releasing the resource and seizing another available one that satisfies the same resource requirements. The processing rule used if all of the units of capacity of a resource are at the end of a shift because of the specified work schedule while processing entities. If the capacity is reduced but not to zero because of the specified work schedule, the entities will finish processing and then the units of capacity will go off shift while the remaining capacity will continue to process entities as normal.</p>
Number of Resources (Secondary Resource for Processing)	Expression	The number of individual resource objects to seize capacity units of.
Units Per Resource (Secondary Resource for Processing)	Expression	The number of capacity units to seize per resource object.
Selection Condition (Secondary Resource for Processing)	Expression	<p>Optional condition evaluated for each candidate resource that must be true for the resource to be selected.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource objects (e.g., <code>Candidate.Object.Capacity</code>).</p>
Must Simultaneously Seize (Secondary Resource for Processing)	True, False	If multiple resources are required, indicates whether all of the resources must be available before any can be seized.
Immediately Try Seize (Secondary Resource for Processing)	True, False	<p>Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues.</p> <p>Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.</p>
Keep Reserved If (Secondary Resource for Processing)	Expression	<p>On processing completion, an optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects without a reservation will be unable to seize the reserved capacity unless the reservation is cancelled.</p> <p>In the expression, use the syntax <code>Candidate.[ObjectClass].[Attribute]</code> to reference an attribute of the candidate resource object(s) being released (e.g., <code>Candidate.Object.Capacity</code>).</p>

		<p>Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p> <p>Leaving this property blank (no condition) is equivalent to entering False.</p>
Reservation Timeout (Secondary Resource for Processing)	Expression	<p>If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation.</p> <p>In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released (e.g., Candidate.Object.Capacity).</p>
Immediately Try Allocate When Released (Secondary Resource for Processing)	True, False	<p>Once an entity has finished processing and released the resource, indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.</p>
On Entering (Other Resource Seizes)	Repeat Group, Resource Seizes	Secondary resource seizes required when an entity is entering the Emptier object.
Must Simultaneously Seize (Other Resource Seizes -- On Entering)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- On Entering)	True, False	<p>Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues.</p> <p>Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.</p>
Before Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Secondary resource seizes required once an entity has been allocated Emptier capacity, but before entering (or ending transfer into) its processing station.
Must Simultaneously Seize (Other Resource Seizes -- Before Processing)	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- Before Processing)	True, False	<p>Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues.</p> <p>Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.</p>
After Processing (Other Resource Seizes)	Repeat Group, Resource Seizes	Secondary resource seizes required after all possible container elements of an entity have been emptied and the entity is about to

Seizes)		attempt its exit from the Emptier's processing station.
Must Simultaneously Seize	If multiple resources are required, indicates whether all must be available before any can be seized.	
Immediately Try Seize (Other Resource Seizes -- After Processing)	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event.
On Entering (Other Resource Releases)	Repeat Group, Resource Releases	Secondary resource releases to be performed when an entity is entering the Emptier object.
Immediately Try Allocate When Released (Other Resource Releases -- On Entering)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Before Processing (Other Resource Releases)	Repeat Group, Resource Releases	Secondary resource releases to be performed once an entity has been allocated Emptier capacity, but before entering (or ending transfer into) its processing station.
Immediately Try Allocate When Released (Other Resource Releases -- Before Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
After Processing (Other Resource Releases)	Repeat Group, Resource Releases	Secondary resource releases to be performed after all possible container elements of an entity have been emptied and entity is now about to attempt its exit from the Emptier's processing station.
Immediately Try Allocate When Released (Other Resource Releases -- After Processing)	True, False	Indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of any other simulation logic in the system. Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource's allocation queue will be scheduled on the simulation's current event calendar as a late priority event.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs immediately after an entity has entered this object and is about to start the 'TransferIn Time'.
Before Processing Add-On Process	Process Instance Name	Occurs when an entity has been allocated Emptier capacity, but before entering (or ending transfer into) the Emptier's processing

		station.
Emptying Container Add-On Process	Process Instance Name	Occurs when the Emptier is about to start filling a container element of the current entity in its processing station.
Emptied Container Add-On Process	Process Instance Name	Occurs when the Emptier has finished emptying a container element of the current entity in its processing station.
After Processing Add-On Process	Process Instance Name	Occurs when all possible container elements of an entity have been emptied, and the entity is now about to attempt its exit from the Emptier's processing station.
Exited Add-On Process	Process Instance Name	Occurs when an entity has exited this object.
Failed Add-On Process	Process Instance Name	Occurs when this object has failed.
Repaired Add-On Process	Process Instance Name	Occurs when a failure is repaired at this object.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when this Emptier object is evaluating whether to accept or reject a request to seize capacity of the emptier. In the executed decision process, assigning a value of less than or equal to '0' to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
On Shift Add-On Process	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process	Process Instance Name	Occurs when the object goes 'off shift' because of its specified work schedule.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Transfer-In Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.
Transfer-In Condition	Expression	The condition required to allow an entity to transfer into this fixed object via an external input node.
Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out	Expression	The condition required to allow an entity to transfer out of this fixed

Condition		object via an external output node.
Expected Setup Time Expression	Expression	The expression used to get a deterministic estimation of the setup time for an entity at this fixed object.
Expected Operation Time Expression	Expression	The expression used to get a deterministic estimation of the operation time for an entity at this fixed object.
Stop Early Event Name	Valid Event Name	Optional name of an event whose occurrence will trigger the Emptier object to stop emptying a container early, before reaching the specified empty target. NOTE: This property has been deprecated. Use the Process Logic Stop Early Triggers repeat group instead.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ItemToFlowConverter

The ItemToFlowConveyor object is used to model a process that converts entities representing discrete items into flow entities representing quantities of fluids or other mass.

The flow rate out of the ItemToFlowConverter is regulated by its 'Output' flow node. Discrete item entities enter the object through its 'Input' basic node and are then processed into flow.

When creating flow, the user has the option to either copy attributes from the input discrete entity, or to create new flow with no copying. If copying, then all possible states, table references, and destination settings are copied over. See the *Flow Entity Creation Method* property for more information.

The conversion logic properties are evaluated in the context of the input. Thus, for the ItemToFlowConverter, the conversion logic properties can be based on attributes of the discrete item entity.

The discrete item input buffer capacity for the ItemToFlowConverter object can be infinite, limited, or specified as '0' to not have any buffer. If a limited buffer or no buffer situation, then blocking can occur. The [Balking and Reneging Options](#) page provides additional information and examples on capacities, balking and reneging for the input buffer.

The converter may stop producing outflow early before reaching flow quantity specified by using the *Stop Early Triggers* repeating properties. The converter may also require the removal and disposal of generated flow upon particular event triggers and conditions. These are specified in the *Purge Contents Triggers* and *Clean In-Place Triggers* repeating properties.

Listed below are the properties of the **ItemToFlowConverter**:

Property	Valid Entry	Description
Flow Quantity Unit Type	Volume, Weight	The unit type to be used for specifying the flow quantity per discrete item.
Flow Quantity Per Item	Expression	The amount of outflow created from a discrete item entity.
Stop Early Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately cause the ItemToFlowConverter object to stop producing outflow for the current discrete item entity early, before reaching the specified flow quantity per discrete item. The converter's flow container will be put into an empty state and ready to process the next discrete item.
Triggering Event Name (Stop Early Triggers)	Valid Event Name	The name of the event whose occurrence will trigger the ItemToFlowConverter object to stop producing outflow for the current discrete item early, before reaching the specified flow quantity per discrete item. The converter's flow container will be put into an empty state and ready to process the next discrete item.
Condition (Stop Early Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to trigger the ItemToFlowConverter object to stop early.
Flow Output Entity Type	Same As Item, Valid Entity Type	The entity type of the created outflow.
Item Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order discrete item entities to be processed into flow.
Item Ranking	Expression	The expression used with a 'Smallest Value First' or 'Largest Value

Expression		First' ranking rule.
Flow Entity Creation Method	Copy Item Attributes,Create New (No Copying)	The method used to create a flow entity. If copying item attributes, then all possible state values, destination settings, and table references will be copied from the processed discrete item entity to the created flow entity.
Purge Contents Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately remove and dispose of any generated flow waiting to exit the ItemToFlowConverter object, putting the converter's flow container into an empty state and cancelling any further outflow for the discrete item entity whose conversion was in-process.
Triggering Event Name (Purge Contents Triggers)	Valid Event Name	The name of the event whose occurrence will indicate that any generated flow waiting to exit the converter object is to be immediately removed and disposed of, putting the converter's flow container into an empty state and cancelling any further outflow for the discrete item entity whose conversion was in-process.
Condition (Purge Contents Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to purge the converter's flow container.
Clean In-Place Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately remove and dispose of any generated flow waiting to exit the converter object and then suspend further processing until a specified cleaning time has elapsed.
Triggering Event Name (Clean In-Place Triggers)	Valid Event Name	The name of the event whose occurrence will indicate that the converter's flow container is to be cleaned.
Condition (Clean In-Place Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to trigger a cleaning operation.
Cleaning Time Type (Clean In-Place Triggers)	Specific, Sequence Dependent	The method used to specify the time required to clean the converter's flow container.
Cleaning Time (Clean In-Place Triggers)	Expression	The time required to clean the converter's flow container.
Operation Attribute (Clean In-Place Triggers)	Expression	The attribute expression to be evaluated for each cleaning operation and whose 'from/to' value changes will determine the sequence-dependent times required to clean the converter's flow container.
Initial Cleaning Time (Clean In-Place Triggers)	Expression	The time required to clean the converter's flow container if the first cleaning operation or if the changeover state stored for the previous operation has been cleared.
Changeover Matrix (Clean In-Place Triggers)	Changeover Matrix Name	The name of the changeover matrix that defines the sequence-dependent cleaning times.
Input Buffer Capacity	Integer >= 0	The number of discrete entities that can be held in this ItemToFlowConverter object's input buffer.
Balk Decision Type (Input Buffer)	None, Blocked, Conditional,	The method used by an entity to decide whether to balk at entering the buffer.

	Probabilistic	If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Input Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name (Input Buffer)	Node object reference	Facility node location to send an entity that has balked at entering the buffer.
Reneged Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.
Trigger Type.Reneged Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Reneged Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Reneged Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Reneged Decision Type .Reneged Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Reneged Condition Or Probability.Reneged Triggers	Expression Specified if the <i>Reneged Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renege as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Reneged Node Name.Reneged Triggers	Node object reference	Facility node location to send a renege entity.
Trigger Start Boundary.Reneged Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
On Item Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when a discrete item entity is entering this ItemToFlowConverter object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is renege from an input or output buffer of the object.

Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State Assignments	Expression	Custom condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Item Entered Add-On Process	Process Instance Name	Occurs immediately after a discrete item entity has entered this ItemToFlowConverter object.
Processing Item Add-On Process	Process Instance Name	Occurs when a discrete item entity is about to be processed into outflow.
Created Flow Add-On Process	Process Instance Name	Occurs when flow content has been created from a discrete item entity, just before the flow entity attempts to start exiting the ItemToFlowConverter object.
Flow Exited Add-On Process	Process Instance Name	Occurs when the outflow created from a discrete item entity has fully exited this ItemToFlowConverter object.
Cleaning Converter Add-On Process	Process Instance Name	Occurs when a clean-in-place operation is beginning at the converter.
Cleaned Converter Add-On Process	Process Instance Name	Occurs when a clean-in-place operation is finishing at the converter.
Transfer-In Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer into this fixed object via external input nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer into the object.
Transfer-In Condition	Expression	The condition required to allow an entity to transfer into this fixed object via an external input node.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FlowToItemConverter

The FlowToItemConverter object used to model a process that converts flow entities representing quantities of fluids or other mass into entities representing discrete items.

The flow rate into the FlowToItemConverter is regulated by its 'Input' flow node. Discrete item entities exit the object from its 'Output' transfer node.

When creating discrete entities, the user has the option to either copy attributes from the flow, or to create new discrete entities with no copying. If copying, then all possible states, table references, and destination settings are copied over. See the Item Entity Creation Method property for more information.

The conversion logic properties are evaluated in the context of the input. Thus, for the FlowToItemConverter object, the conversion logic properties can be based on the attributes of the new inflow entity.

The discrete item output buffer capacity for the FlowToItemConverter object can be infinite, limited, or specified as '0' to not have any buffer. If a limited buffer or no buffer situation, then blocking can occur. The [Balking and Reneging Options](#) page provides additional information and examples on capacities, balking and reneging for the output buffer.

The converter may stop waiting for inflow and create a discrete item earlier than the flow quantity specified by using the *Stop Early Triggers* repeating properties. The converter may also may require the removal and disposal of inflow collected upon particular event triggers and conditions. These are specified in the *Purge Contents Triggers* and *Clean In-Place Triggers* repeating properties.

Listed below are the properties of the **FlowToItemConverter**:

Property	Valid Entry	Description
Flow Quantity Unit Type	Volume, Weight	The unit type to be used for specifying the flow quantity per discrete item.
Flow Quantity Per Item	Expression	The amount of inflow required to create a discrete item entity.
Stop Early Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately cause the FlowToItemConverter object to stop waiting for additional inflow and create the next discrete item entity early, using whatever inflow that has accumulated in the converter's flow container but before reaching the specified flow quantity per discrete item.
Triggering Event Name (Stop Early Triggers)	Valid Event Name	The name of the event whose occurrence will trigger the FlowToItemConverter object to stop waiting for additional inflow and create the next discrete item entity early, using whatever inflow that has accumulated in the converter's flow container but before reaching the specified flow quantity per discrete item.
Condition (Stop Early Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to trigger the FlowToItemConverter object to stop early.
Item Output Entity Type	Same As Inflow, Valid Entity Type	The entity type of the created discrete items.
Item Entity Creation Method	Copy Flow Attributes, Create New (No Copying)	The method used to create a discrete item entity. If copying flow attributes, then all possible state values, destination settings, and table references will be copied from the accumulated flow entity to the created discrete item entity.
Purge Contents	Repeat Group,	Optional event-driven triggers that will immediately remove and

Triggers	Triggers	dispose of any inflow collected by the converter object for creating the next discrete item entity, putting the converter's flow container back into an empty state.
Triggering Event Name (Purge Contents Triggers)	Valid Event Name	The name of the event whose occurrence will indicate that any inflow collected by the converter object is to be immediately removed and disposed of, putting the converter's flow container back into an empty state.
Condition (Purge Contents Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to purge the converter's flow container.
Clean In-Place Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately remove and dispose of any inflow collected by the converter object for creating the next discrete item entity and then suspend new inflow until a specified cleaning time has elapsed.
Triggering Event Name (Clean In-Place Triggers)	Valid Event Name	The name of the event whose occurrence will indicate that the converter's flow container is to be cleaned.
Condition (Clean In-Place Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to trigger a cleaning operation.
Cleaning Time Type (Clean In-Place Triggers)	Specific, Sequence Dependent	The method used to specify the time required to clean the converter's flow container.
Cleaning Time (Clean In-Place Triggers)	Expression	The time required to clean the converter's flow container.
Operation Attribute (Clean In-Place Triggers)	Expression	The attribute expression to be evaluated for each cleaning operation and whose 'from/to' value changes will determine the sequence-dependent times required to clean the converter's flow container.
Initial Cleaning Time (Clean In-Place Triggers)	Expression	The time required to clean the converter's flow container if the first cleaning operation or if the changeover state stored for the previous operation has been cleared.
Changeover Matrix (Clean In-Place Triggers)	Changeover Matrix Name	The name of the changeover matrix that defines the sequence-dependent cleaning times.
Output Buffer Capacity	Integer ≥ 0	The number of discrete entities that can be held in this FlowToItemConverter object's output buffer.
Balk Decision Type (Output Buffer)	None, Blocked, Conditional, Probabilistic	The method used by an entity to decide whether to balk at entering the buffer. If the decision type is 'Blocked', then an entity attempting to enter the buffer will balk if the buffer is full rather than be blocked. Or, if a zero capacity buffer, the entity will balk if its attempt to transfer directly into or out of the object is blocked.
Balk Condition Or Probability (Output Buffer)	Expression	The balk condition or probability specified as an expression. If a probability then enter the chance of balking as a value between 0.0 (0%) and 1.0 (100%).
Balk Node Name	Node object	Facility node location to send an entity that has barked at entering

(Output Buffer)	reference	the buffer.
Renega Triggers	Repeat Group	Optional time or event-driven triggers that can cause entities to abandon waiting in the buffer.
Trigger Type.Renega Triggers	Time Based, Event Based	The type of trigger. A time based trigger can cause an entity waiting in the buffer to renege after a tolerable wait duration expires. An event based trigger can cause an entity waiting in the buffer to renege if a specific event occurs.
Wait Duration.Renega Triggers	Expression,vspecified if the <i>Trigger Type</i> is 'Time Based'.	The tolerable wait duration until a renege decision by an entity waiting in the buffer.
Triggering Event Name.Renega Triggers	Event Reference, specified if the <i>Trigger Type</i> is 'Event Based'.	The name of the event whose occurrence will trigger a renege decision by an entity waiting in the buffer.
Renega Decision Type .Renega Triggers	Always, Conditional, Probabilistic	The method used by an entity when the trigger occurs to decide whether to abandon waiting in the buffer.
Renega Condition Or Probability.Renega Triggers	Expression Specified if the <i>Renega Decision Type</i> is 'Conditional' or 'Probabilistic'.	The renege condition or probability specified as an expression. If a probability then enter the chance of renegeing as a value between 0.0 (0%) and 1.0 (100%). NOTE: If it is necessary to check in a conditional expression whether an entity is currently waiting for a specific type of constraint, the entity 'Queuing' namespace provides several functions for that purpose (e.g., Entity.Queuing.IsWaitingRidePickupQueue).
Renega Node Name.Renega Triggers	Node object reference	Facility node location to send a renegeing entity.
Trigger Start Boundary.Renega Triggers	Entered, Ended Transfer In	Indicates when the trigger becomes active for an entity occupying the buffer. Can be either immediately when the entity has entered the buffer or, alternatively, not until its transfer into the buffer has been ended (e.g., after a transfer-in time or any other entry related logic).
On New Inflow Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when a new inflow is entering this FlowToItemConverter object, which may be either the first inflow to begin creation of a new discrete item or a change in the inflow entity type.
Before Item Exiting (State Assignments)	Repeat Group, Assignments	Optional state assignments when a discrete item entity is ready to exit this FlowToItemConverter object.
On Balking (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is balking at entering an input or output buffer of the object.
On Reneging (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is reneging from an input or output buffer of the object.
Assign If.State Assignments	Expression	Condition required to perform the assignment. Used only with On Balking and On Reneging assignments.
Condition.State	Expression	Custom condition required to perform the assignment. Used only

Assignments		with On Balking and On Reneging assignments.
State Variable Name.State Assignments	State Variable Name or Reference	Name of the state variable that will be assigned a new value.
New Value.State Assignments	Expression	The new value to assign.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
New Inflow Entering Add-On Process	Process Instance Name	Occurs when a new inflow is starting to enter this FlowToItemConverter object, which may be either the first inflow to begin creation of a new discrete item or a change in the inflow entity type. Note that a token executing this add-on process will be associated with the flow entity that is entering the converter. Also, if the logic of the specified process includes any delays, then the inflow into the converter will be paused until the process has completed.
Create Item Add-On Process	Process Instance Name	Occurs when a discrete item entity has been created from accumulated inflow, just before the entity attempts to enter the output buffer or exit the FlowToItemConverter object.
Item Exited Add-On Process	Process Instance Name	Occurs when a discrete item entity has exited this FlowToItemConverter object.
Cleaning Converter Add-On Process	Process Instance Name	Occurs when a clean-in-place operation is beginning at the converter.
Cleaned Converter Add-On Process	Process Instance Name	Occurs when a clean-in-place operation is finishing at the converter.
Transfer-Out Constraints	Disable, Default, Custom Condition	Indicates the type of constraints used to determine whether entities can transfer out of this fixed object via external output nodes. If specified as 'Default', then the <i>Can Transfer In & Out of Objects</i> property setting for the entity type will determine whether an entity can perform a transfer out of the object.
Transfer-Out Condition	Expression	The condition required to allow an entity to transfer out of this fixed object via an external output node.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FlowNode

A FlowNode object is a node specifically designed to regulate the flow of entities representing quantities of fluid or other mass.

Common applications of a FlowNode object include use as an 'Input' or 'Output' node for controlling flow into or out of another object (such as an object representing a tank). A FlowNode may be used to model a flow control point in a network of links.

The output entity type of the outflow from the node can be dynamically changed by assigning the flow node's FlowRegulator.CurrentDesiredOutputEntityType' state variable.

The FlowNode allows optional dynamic updating of either maximum flow rate and/or the output yield factor of the node's flow regulator. Values are updated based on the Update Interval time specified. Note that the first update is done to the initial flow rate and/or initial yield value at simulation time 0, and then updated every Update Interval time specified.

Please refer to the SimBit [FlowConcepts](#) for models using the FlowNode. Note that FlowNodes are also incorporated into the FlowSource, FlowSink and Tank objects.

Listed below are the properties of the **FlowNode**:

Property	Valid Entry	Description
Flow Rate Unit Type	Volume Flow Rate, Weight Flow Rate	The unit type to be used for specifying the maximum flow rate of the node's flow regulator.
Initial Maximum Flow Rate	Expression	The initial maximum flow rate allowed by the node's flow regulator.
Initial Output Yield Factor	Expression	The initial output yield factor for the regulator. This factor is entered as a ratio of output to inflow, and may be used to scale the flow into the regulator such that there is a physical loss or gain represented in the regulator's output flow.
Initial Output Entity Type	Entity Name	Optional property specifying the initial desired entity type of the outflow produced by the node's flow regulator. If unspecified, then the output flow entity type will be automatically determined by the inflow entity types.
Regulator Initially Enabled	True, False	Specifies whether the node's flow regulator is enabled when the system is initialized.
Flow Control Mode (Input Flow Control)	SingleFlow (No Merging), Merge Flow	The mode used by the node's flow regulator to control the inflow from inbound links.
Inbound Link Rule	By Link Weight, By Flow Priority	The rule used by the node's flow regulator to prioritize inflows from inbound links. If the rule specified is 'By Link Weight', then the flow entity on the inbound link with the largest selection weight will be selected first. If the rule is specified as 'By Flow Priority', then the flow entity with the largest priority value (regardless of the link used) will be selected first.

Switch Control Variable (Input Flow Control)	State Variable Name	Optional discrete variable to use as a switch control mechanism for dynamically changing the current inbound link selection. If this property is specified, then the selection weight expressions of the node's inbound links will be reevaluated whenever the switch control variable's value changes in order to change priority to the link with the now largest weight. The inbound link selection weight expressions, as a typical modeling approach, are recommended to be based on the switch control variable's current value. For example, enter link selection weight expressions such as 'MySwitchControlVar == 1', 'MySwitchControlVar == 2', etc.
Flow Priority Expression	Expression	The expression used to return the priority value of an inflow request at the node.
Merge Matching Rule	Any Entity Type, Same Entity Type	Specifies the match conditions required to merge inflow requests at this node into a single output flow. If the matching rule is specified as 'Any Entity Type', then all inflows may be merged together regardless of entity type. If the matching rule is specified as 'Same Entity Type', then only the inflows of the same entity type may be merged.
Merge Allocation Rule	Proportional Based on Inflow Rates, Preferred Order By Link Weight, Proportional Based on Link Weights	The allocation rule used by the node's flow regulator to merge inflow from inbound links.
Flow Control Mode (Output Flow Control)	Single Flow (No Splitting), Split Flow	The mode used by the node's flow regulator to control outflow to outbound links.
Outbound Link Rule	By Link Weight, Shortest Path	The rule used by the node's flow regulator to send outflows to outbound links. If the rule is specified as 'By Link Weight', then the outbound link with the largest selection weight will be selected by a flow entity. If the rule is specified as 'Shortest Path' and the flow entity has an assigned destination, then the next link in the shortest path to that destination will be selected. For any flow entity that does not have an assigned destination, then the 'By Link Weight' is always assumed.
Switch Control Variable (Output Flow Control)	State Variable Name	Optional discrete variable to use as a switch control mechanism for dynamically changing the current outbound link selection. If this property is specified, then the selection weight expressions of the node's outbound links will be reevaluated whenever the switch control variable's value changes in order to change priority to the link with the now largest weight. The outbound link selection weight expressions, as a typical modeling approach, are recommended to be based on the switch control variable's current value. For example, enter link selection weight expressions such as 'MySwitchControlVar == 1', 'MySwitchControlVar == 2', etc.
Split Allocation Rule	Evenly If Possible, Preferred Order By Link Weight, Proportional Based On Link Weights, Proportional Based On Link	<p>The allocation rule used by the node's flow regulator to split outflow to outbound links.</p> <p>If the rule is specified as 'Evenly If Possible', then the flow regulator will distribute flow as evenly as possible to the node's outbound links while still maximizing total outflow.</p> <p>If the rule is specified as 'Preferred Order By Link Weight', then the flow regulator will distribute flow in priority order to the node's outbound links while still maximizing total outflow. The outbound link with the largest selection weight will be selected first, and if that link cannot accept the full output then overflow will be sent to the outbound link with the next largest selection weight and so forth.</p>

	Weights If Possible	<p>If the rule is specified as 'Proportional Based On Link Weights', then the flow regulator will proportionally distribute flow to the node's outbound links using the ratios of the link selection weights. This rule will not maximize total outflow, but rather, will reduce the output to all links as necessary to maintain the specified ratios.</p> <p>If the rule is specified as 'Proportional Based On Link Weights If Possible', then the flow regulator will distribute flow as proportionally as possible to the node's outbound links using the ratios of the link selection weights, while still adjusting when necessary to maximize total outflow. For example, if there are three outbound links designated to receive 20%, 30%, and 50% of the flow and the outbound link designated for 50% becomes blocked, then this rule will automatically adjust so that the other two outbound links will receive 40% and 60% of the flow until the blockage is cleared.</p>
Update Interval	Expression	The time interval between updates.
Maximum Flow Rate Equation	Expression	The expression used to regularly update the maximum flow rate of the node's flow regulator.
Output Yield Factor Equation	Expression	The expression used to regularly update the output yield factor of the node's flow regulator.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Bound External Output Node	Node Name	The name of an external output node that this node has been bound to in order to transfer flow out of the containing (parent) object. The transfer attempt will be performed by a flow entity immediately upon entering the node and, if successful, any other flow regulator logic for the node will be ignored.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

FlowConnector

The FlowConnector object may be used to define a direct, zero travel distance connection from one flow node location to another.

Please refer to the SimBit [FlowConcepts](#) for models using the FlowConnector.

Listed below are the properties of the **FlowConnector**:

Property	Valid Entry	Description
Selection Weight	Expression	The weight expression for this link if using a 'By Link Weight' rule to select an outbound link from a node.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

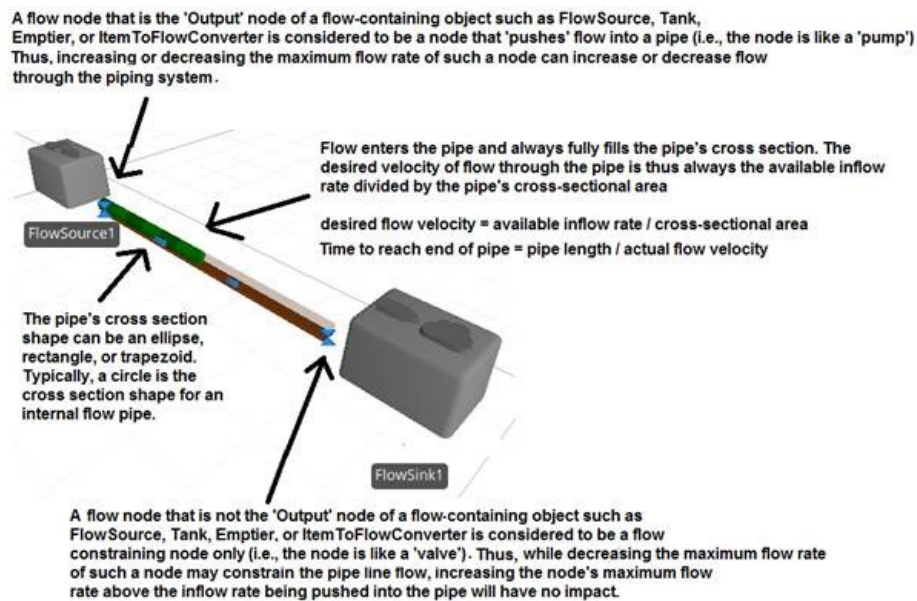
Send comments on this topic to [Support](#)

Pipe

The Pipe object may be used to define a flow connection from one flow node location to another where flow travel time and pipeline volume is of significant interest. NOTE: There are known issues with the *New Inflow Entering* options when multiple flow entities of the same entity type arrive into a single pipe; currently, the state assignments and add-on processes are triggered for each unique flow entity.

The pipe may be automatically emptied upon particular event triggers and conditions. These are specified in the *Purge Contents Triggers* repeating properties.

[State assignments](#) can be made when the Pipe has a new inflow entity type or is full or empty. Additionally, the [Add-On Process Triggers](#) allow additional logic to be specified when using the Pipe object.



Listed below are the properties of the **Pipe**:

Property	Valid Entry	Description
Drawn To Scale	True, False	Specifies whether this link's drawn length in the Facility window is to be used for the simulation logic.
Auto Stop If No Inflow	True, False	Indicates whether to automatically stop flow through this Pipe object if there is no available inflow. If this property is set to False and the pipe's inflow rate goes to zero, then any flow already present in the pipe will continue trying to move at the last desired flow velocity until the pipe is emptied. Otherwise, if this property is set to True, then all flow through the pipe will be stopped until the inflow rate becomes positive again.
Purge Content Triggers	Repeat Group, Triggers	Optional event-driven triggers that will immediately remove and dispose any flow present in the pipe, putting the pipe into an empty state.
Triggering Event Name (Purge Contents Triggers)	Valid Event Name	The name of the event whose occurrence will indicate that all flow present in the pipe is to be immediately removed and disposed of, putting the pipe into an empty state.

Condition (Purge Contents Triggers)	Expression	Optional condition to be evaluated whenever the triggering event occurs, and which must also be true to purge the pipe's contents.
Selection Weight	Expression	The weight expression for this link if using the 'By Link Weight' rule to select an outbound link from a node.
On New Inflow Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when a new inflow is starting to enter this Pipe object, which may be either the first inflow to an empty pipe or a change in the inflow entity type.
On Pipe Full (State Assignments)	Repeat Group, Assignments	Optional state assignments when the pipe's volume has become completely filled.
On Pipe Empty (State Assignments)	Repeat Group, Assignments	Optional state assignments when the pipe's volume has become empty.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
New Inflow Entering Add-On Process	Process Instance Name	Occurs when a new inflow is starting to enter this Pipe object, which may be either the first inflow into an empty pipe or a change in the inflow entity type. Note that a token executing this add-on process will be associated with the flow entity that is entering the pipe. Also, if the logic of the specified process includes any delays, then the inflow rate into the pipe will be paused until the process has completed.
Pipe Full Add-On Process	Process Instance Name	Occurs when the pipe's volume has become completely filled.
Pipe Empty Add-On Process	Process Instance Name	Occurs when the pipe's volume has become empty.
Report Statistics	True or False	Specifies if statistics are to be automatically reported for this object.
Cross Section Shape	Ellipse, Rectangle, Trapezoid	Specifies the geometric shape of this link's cross section.

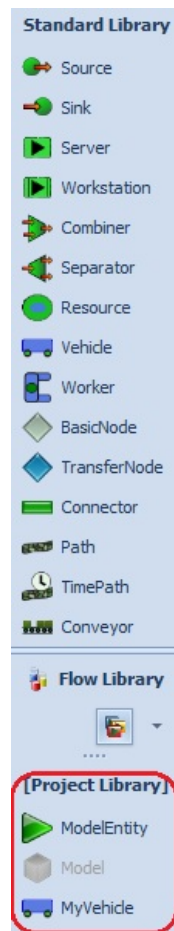
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Project Library

The Project Library is a list of objects (i.e. models) that are contained within a project. The Project Library is found in the Libraries window, from within the Facility window. It appears under the Standard Library and it can be expanded vertically in the case of a large library. When a new model is created, it appears in the Project Library. A model can be dragged into the Facility window of another model from the Project Library. If the model appears grey in the Project Library, this indicates that it cannot be placed into the active Facility window because that particular model is currently active and you cannot drag a model into itself.

The Project Library



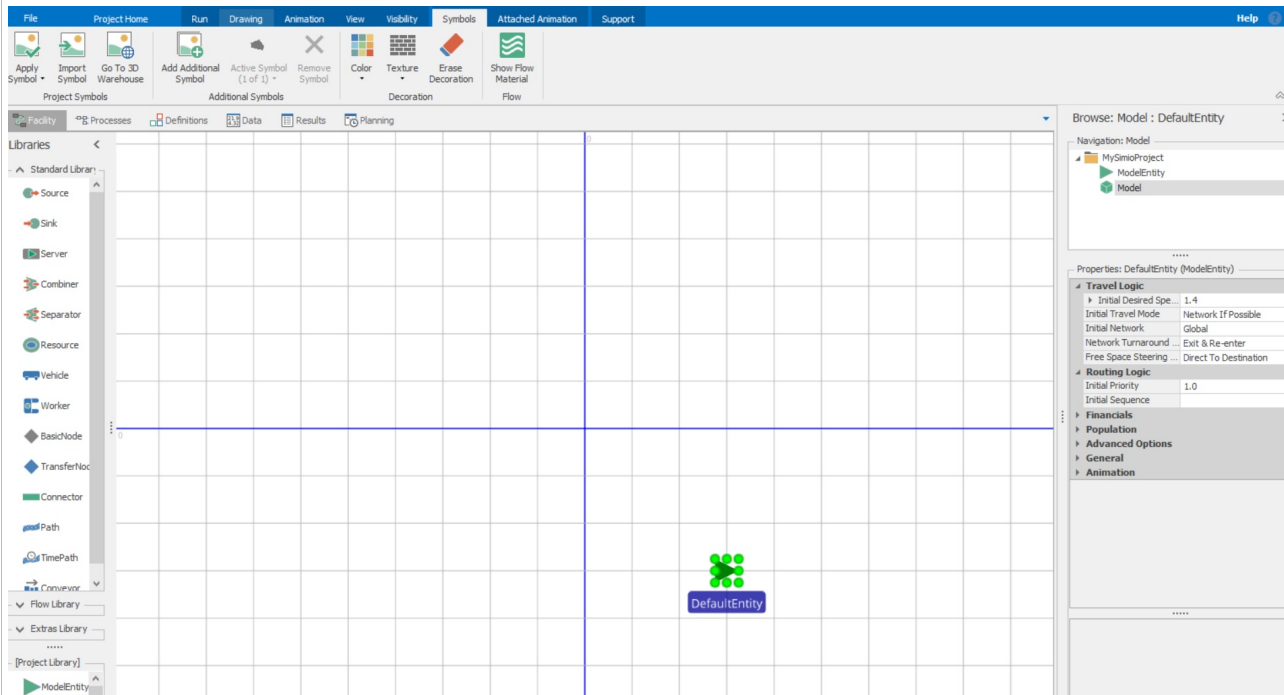
For information on how to create a new object for the Project Library, see [Creating New Objects](#).

The models within the Project Library can also be viewed in the [Project window](#), where they can be renamed, edited and deleted. To navigate to the Project window, select the name of the project within the [Navigation window](#). You can also edit a model by selecting the model from within the Navigation window.

A user can import a library of objects from another project into the current project and they will appear in the Project Library and therefore available for use in the current project. To import a library, select the Load Library icon from the Project Home ribbon (or right click in the Libraries panel and select Load Library). Select an .spfx file and the models from that selected project will then appear in a new Library called MySimioProject. Models from this imported library can be copied or subclassed into the current project from within the Project window. They can also be subclassed with a simple right click from the Project Library window. Unloading a library from the Libraries panel can also be done by using the right click option within that Library.

ModelEntity

You will find one **ModelEntity** automatically created within a new Project Library. The ModelEntity is the default entity definition in your project. A user does not need to drag it into the Facility window. A ModelEntity called DefaultEntity is already part of the Project and can therefore be created by a Source object or with the Create Step. However, if you would like to change the symbol of the Entity or change any of the properties of the instance, you need to drag a ModelEntity into the Facility window from the Project Library. Additional instances of ModelEntity can also be dragged into the model.



For information on the difference between Entities and Tokens, see the [Tokens and Entities](#) help page.

Listed below are the properties of the **ModelEntity**:

Property	Valid Entry	Description
Initial Desired Speed	Rate Table Instance	The initial desired speed value for objects of this type.
Initial Travel Mode	Free Space Only, Network Only, Network If Possible	The initial travel mode for entities of this type. 'Free Space Only' indicates that the entities are required to perform travel movements in free space. 'Network Only' indicates that the entities are required to perform travel movements using the currently assigned network. 'Network If Possible' indicates a preference for the entities to perform travel movement using the currently assigned network, but if no followable network path exists to the next destination then travel in free space is allowed. NOTE: Refer to the user-assignable 'CurrentTravelMode' state variable of an entity to dynamically change its travel mode during a simulation run. An entity's travel mode may also be changed using the 'Outbound Travel Mode' property provided by a node.
Initial Network	Network Element Property	The network of links used by objects of this type to travel between node locations. This includes the default 'Global' network and 'No Network (FreeSpace)'.
Network Turnaround Method	Exit & Re-enter, Rotate in Place, Reverse	When traveling on a network, the default method used by objects of this type to turn around at a node in order to move in the opposite direction on a bidirectional link. The 'Turnaround Method' property of a Network element may also be used to override this default on a network-by-network basis.
Free Space Steering Behavior	Direct To Destination, Follow Network Path If Possible	The behavior used to steer an entity of this type when traveling in free space to a destination. 'Direct To Destination' will steer an entity in a straight line to its destination. 'Follow Network Path If Possible' will prefer to steer an entity along a path following its currently assigned network, staying within the boundaries of the drawn path width. The Entity will travel based on the drawn length of the path even if a different logical length is specified. However, if no followable network path exists to the entity's destination then the entity will be steered in a straight line. Note: In order to use this steering behavior, make sure the entity's travel mode is set to 'Free Space Only'.
Update Interval	Expression	The time interval between updates checking an entity's adherence to the network path.
Avoid Collisions	True or False	Indicates whether the steering behavior will attempt to avoid collisions with other entities that are following the same network path.
Initial Priority	Expression	The initial priority value for objects of this type.
Initial Sequence	Sequence Property	An initial destination sequence that objects of this type use for routing purposes.
Initial Cost	Expression	The initial value of the Cost state for an object of this type, which stores the total cost that has been allocated to the object.
Initial Cost Rate	Expression	The initial value of the Cost.Rate parameter for an object of this type, which indicates a cost per unit time that is to be allocated to the object.
Initial Number in System	Numeric	The initial number of objects of this type in the system at the beginning of the simulation run. Entities are initially located in free-space at the object instance location.
Maximum Number in System	Numeric	The maximum number of objects of this type that can simultaneously be in the system. If this limit is exceeded then a runtime error is generated.
Destroyable	True or False	Specifies whether objects of this type can be destroyed using the Destroy Step.
Can Transfer In & Out of Objects	True or False	Indicates whether entities of this type are by default allowed to transfer into and out of fixed objects via those object's external input / output nodes.
Due Date Expression	Expression	The expression used to return a 'due date' value for an entity of this type.

Gantt Visibility	True or False	An expression evaluated at run time to determine if the entity should appear in the Entity Gantt window.
Current Size and Orientation Index	State Variable	The optional name of a state variable that indexes into the object's associated symbol list, and whose value will automatically resize and orient the object to match the size and orientation of the indexed symbol.
Size.Length	Numeric	The initial graphical length of the entity.
Size.Width	Numeric	The initial graphical width of the entity.
Size.Height	Numeric	The initial graphical height of the entity.
Orientation.Yaw	Numeric	The yaw (or heading) angle in degrees, measured clockwise from -Z, of the object in the X-Z plane.
Orientation.Pitch	Numeric	The pitch angle of the object in the Y-Z plane.
Orientation.Roll	Numeric	The roll angle of the object in the X-Y plane.
Volume	Numeric	The initial logical volume of the object in cubic meters. If blank, the initial logical volume will be the implicit volume defined by the Size. At runtime, the ratio between the logical volume and the Size implicit volume will be kept constant.
Weight Density	Numeric	The weight density of an entity of this type in kilograms per cubic meter. The default value for density is 1 kilogram per cubic meter. At runtime, a change to Weight or Volume will make a change in the other to make sure Density remains constant.
Current Symbol Index	Expression	An expression that returns a value for the index in the list for the current symbol to display.
Random Symbol	True or False	If true, each new object will be randomly assigned a symbol picture from the associated list of symbols which is stored internally. Note that, since physical sizes are determined by the state variable specified as the Current Size and Orientation Index (defaulted to 'ModelEntity.Picture' in new model entities), the physical size of the entity will not change until the state variable does. To make the entity have both the symbol picture and size specified by the symbol, set Random Symbol to 'False' and make sure Current Size Index and Current Symbol Index use the same state variable (which in new model entities is already defaulted to 'ModelEntity.Picture for you). Then randomly assign a value to that state variable in process logic.
Current Animation Index	Expression	An expression that returns the numeric index or string name of the current animation of the active symbol. If the expression is not defined, returns an index that is out of the range of the list of animations, or a name of an animation that does not exist, Simio will fallback to using the action specified in the Default Animation Action.
Default Animation Action	Expression	Indicates if and how Simio will animate objects that don't have an explicit animation set via the Current Animation Index expression. None indicates no action will be taken. Moving indicates actively moving entities will be animated with any available forward movement animations from the active symbol. Moving and Idle indicates actively moving entity and idle objects will be animated with any available forward movement or idle animations from the active symbol.
Link Segment Transition Type	Smooth, Immediate	The type of link movement transitions for this object. Smooth indicates the object will smoothly move between different angles of the different link segments. Immediate will immediately change the angle of the object at the next link segment.
Dynamic Label Text	Expression	An expression returning the string to show in a floating label above a dynamic entity at runtime.
Draw Type	Single, Segmented	Indicates the way the entity should be drawn. 'Single' indicates a single symbol, scaled to the entity's length, width and height. 'Segmented' indicates multiple symbols scaled to the entity's width and height, with each symbol drawn along the entity's length along its occupied links. *NOTE: To have the entity actually move segmented, this property should be 'segmented' and the entity symbol itself must be able to be segmented. For example, to animate a train, go to the Project Home ribbon, click New Symbol -> Create New Symbol, and place a number of individual cars into the symbol. The new symbol should be applied to the ModelEntity placed in the Facility window.

*NOTE: Default values for both the Link Segment Transition Type and Dynamic Label Text can be set within the Definitions tab / [External](#) view of the ModelEntity.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Entity Routing and Movement

Entity Routing

- Entities can be routed in Simio in a number of different ways, such as using Sequences, going to specific nodes or following along on paths and using their Selection Weights to determine where to go next. Entity routing is generally determined by the TransferNodes that the entities pass through. The *Entity Destination Type* property determines how the entity will travel when it leaves the node. Read this [TransferNode help page](#) for more detailed information on how to determine how the entity will travel through the system.

The Movement State on ModelEntity

- The standard ModelEntity has an inherited movement state called, "Movement". It returns the total linear distance that this object has traveled in free space or on its current network link. It has certain state parameters to it, such as
 - Initial State Value
 - Rate Value
 - Acceleration
 - AccelerationDuration
 - Heading
 - Pitch
 - Roll (not currently supported)
 - X
 - Y
 - Z

You can see the information about a ModelEntity's movement during a run by accessing Movement.Rate, Movement.X, Movement.Y, Movement.Z, Movement.Pitch and Movement.Heading. These functions will return the current value. For example, Movement.X will return the entity's current location along the X axis.

You can also use the [Assign Step](#) to assign new values to these parameters during the run, to change the movement of the entity. Note: CHANGING ACCELERATION, HEADING AND PITCH ON A MODELENTITY TRAVELING ON A LINK IS NOT YET FULLY SUPPORTED. FREE SPACE MOVEMENT IS ACTIVE.

Heading is measured in degrees from "north" where "north" points up when viewed in the 2D top down view. These equate to compass headings with a compass placed on the floor and pointing with north in the up direction. Hence the following headings produce the following movements:

- 0 – along the floor to the north (up)
- 90 – along the floor to the east (right)
- 180 – along the floor to the south (down)
- 270 – along the floor to the west (left)

The user can set Pitch relative to the floor in degrees. For example, a 45 degree Pitch will cause the entity to fly at an angle of 45 degrees to the floor (i.e. climb at 45 degrees to the surface). A -30 degree Pitch will cause the entity to descend at an angle of 30 degrees relative to the surface.

- Notes on an Entity's speed
 - An entity in Free Space or on a network Link will by default travel at its DesiredSpeed state value (ModelEntity.DesiredSpeed). You can set the initial value of the DesiredSpeed state using the Initial Desired Speed property. You can also have logic manually slow down or speed up an entity by assigning the DesiredSpeed state directly using the Assign step. Some Link objects, such as the Conveyor object or Path object, have a Speed Limit property, which causes their own process logic to ultimately control the entity's actual movement rate on the Link. But the DesiredSpeed value is considered the entity's default rate for traveling. Note: Acceleration of an entity on a traveling on a Link is not yet supported.

Using Acceleration

- An entity's acceleration can be accessed or assigned through the Movement.Acceleration variable. Acceleration on Links is currently not supported in Simio, so this variable will not be used unless the entity is traveling in free space.

-
- When an entity is moving in Free Space, it is typically directed by a Travel step. The Travel step gives you property options such as *Destination Type*, *Maximum Movement Rate*, *Acceleration*, *Deceleration*, and others. We suggest using the Travel step to assign any Acceleration or Deceleration.
 - A Travel step will overwrite any Assignment to the Movement variable. So to include acceleration information for an entity, it would need to be on the Travel step, or Assigned to the Movement Variable after the Travel step, when the entity is moving in free space. Additionally, the acceleration will not be used if the entity is already traveling at its desired speed. You may need to Assign a new desired speed, or stop the entity by Assigning a Movement.Rate of zero.
 - We might suggest editing or subclassing your entity object so you are able to edit all the entity's Travel steps and include the acceleration logic. To quickly find all Travel steps in an existing object definition, you can use the Properties Window, on the Project Home ribbon, to locate all steps. Before all the steps, you might consider adding an Assign step to change the Movement.Rate to '0'. Then, in the subsequent Travel step, add in values or Reference Properties for the acceleration logic.
-

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Entity Size and Volume

Entities have a number of characteristics, including Picture, Size, Volume, Weight and Density. Within the Project Library, the ModelEntity object is typically used to place an Entity type within the Facility window. Once an ModelEntity (named DefaultEntity once placed) is in the Facility window, the user can change its various characteristics.

Making up the entity's 'Size' characteristic are three dimensions, **Length**, **Width**, and **Height**, found under the 'General / Physical Characteristics' section of properties of the entity. By default, those are 0.5 x. 0.5 x 0.25 meters, respectively. Therefore the implicit Size volume of an entity is 0.0625 cubic meters. These characteristics can be referenced (or changed within an Assign step) by using the ModelEntity.Size.Length, ModelEntity.Size.Width and ModelEntity.Size.Height states of the entity. The implicit Size volume calculated is referenced by the state ModelEntity.Size.

The entity also includes a state named **Volume**, which can be specified within the 'General / Physical Characteristics' section of properties for the entity. If not specified, the model will use the Size calculation of volume as the ModelEntity.Volume state. If specified, then this ModelEntity.Volume state is the logical volume of the entity. The ratio of the Size volume to the logical Volume is kept constant during the simulation run. In other words, if you change one value, you will see the other immediately change as well. The ModelEntity.Volume may be changed during the simulation run by using an Assign step.

The entity has a **Density** characteristic, which has an initial value specified through the 'General / Physical Characteristics' section of properties of the entity. The default value for the density is '1' Kilograms per Cubic Meter. Density is not a state of the entity and therefore, cannot be changed during the simulation run. If the Weight of an entity changes, the Volume of that entity will change to keep a constant Density. Likewise, if the Volume of the entity changes, the Weight will automatically change to keep a constant Density.

The entity has a **Weight** state, referred to as ModelEntity.Weight, that can be changed during the simulation run. The entity's Weight is initially calculated based on the Volume and Density specified.

Both Weight and Volume have an associated **Rate**, referred to as ModelEntity.Weight.Rate and ModelEntity.Volume.Rate, respectively. These rates are initially set to '0' (in the appropriate units based on the Units Settings) and can be changed during the simulation run with an Assign step. These rate states will automatically change the associated Weight or Volume of the entity over time.

ModelEntity.Weight and ModelEntity.Volume can both be monitored, which would be done through the ModelEntity object in the Navigation window by placing a Monitor element within the Definitions window / Elements panel.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Assigning States

You might be familiar with the term “attribute” or “variable” from other software packages. In Simio, you’ll find States and Properties.

Properties of an object are the input parameters associated with that object. A Property holds a specific data type. It is a static input parameter (e.g. processing time, batch size) for the object and does not change during the running of the simulation. Properties are used to send information between objects (inputs and outputs) and they are also used for [experimentation](#) when a user wants to experiment with using different input parameters for their model.

A **State** represents a numeric or string value that may change during the run. It is sometimes referred to as a *State Variable* because its value can *vary* during the run. A State can be added to any object in your Project Library. When a State is added to the main model object, it can be seen and accessed by all the objects within that main model. Therefore, it is similar to what is often referred to as a “global variable” in other software packages. When a State is added to another object, such as the ModelEntity object, it is part of that object’s definition. Therefore, a state on a ModelEntity is similar to what is often referred to as an “attribute” of each entity that is going through the system. For example, if you add a Discrete State to the default ModelEntity object, called “MyCustomState”, then each entity that enters the system will have MyCustomState on it and therefore can have a unique value assigned to that State. For example, a State on an entity might contain a value that specifies this entity’s unique bar code, this entity’s due date or a number that will be used to identify which shipment this entity should go into. And the value of the state can be updated or referenced during the run with the syntax ModelEntity.MyCustomState, in an expression.

A common task in simulation is to assign values to a State within the model logic. In Simio, we provide a number of different ways to make assignments. The easiest and most straightforward way to make an assignment to a State variable is to use the properties found under the State Assignments category of each object. These properties provide an easy, straightforward way to make an assignment to a State when an entity is either entering or exiting this object. As an alternative, you can use the Add-On Process triggers to execute a process, where you can use an Assign Step, along with other logic, to make an assignment to a State.

Two Different Approaches for Making a State Assignment upon Entering the Server Object

Properties: Server1 (Server)

Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Processing Time	Random.Triangular(.1,.2,.3)
Buffer Capacity	
Reliability Logic	
State Assignments	
On Entering	0 Rows
Before Exiting	0 Rows
Secondary Resources	

Can make a State Assignment right here in this repeat property

Properties: Server1 (Server)

Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Processing Time	Random.Triangular(.1,.2,.3)
Buffer Capacity	
Reliability Logic	
State Assignments	
Secondary Resources	
Financials	
Add-On Process Triggers	
Run Initialized	
Run Ending	
Entered	Server1_Entered
Before Processing	
Processing	
After Processing	

Server1 AddOn Processes

```

graph LR
    Begin((Begin)) --> Decide1[Decide1]
    Decide1 -- true --> Assign1[Assign1]
    Decide1 -- false --> End1((End))
    Assign1 --> End2((End))
  
```

Alternatively, you can use an Add On Process to make a State assignment. You would use this approach if you want additional logic, such as a Decide step in front of the assignment.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Secondary Resources

Secondary Resources

Secondary resources are available within the Server, Combiner and Separator objects in the Standard Library, as well as the Workstation (Deprecated) object. Within the Flow Library, the Filler and Emptier objects both contain secondary resources as well. A secondary resource is simply an additional resource that may be seized and/or released to assist with processing at the object. Each of the objects mentioned has a 'primary' resource that is included with the object that is used for processing. You may consider an operator, worker, tool or any other limited resource that is needed for all or part of processing to be a secondary resource.

Within Servers, Combiners and Separators of the Standard Library and Filler and Emptier of the Flow Library

There are also secondary resources that can be used with the Server, Combiner and Separator objects, as well as the Filler and Emptier within the Flow Library. These resources are specified within the Secondary Resources section of properties for the object. The secondary resources section of properties is broken up into three sections, including 'For Processing', 'Other Resource Seizes' and 'Other Resource Releases'.

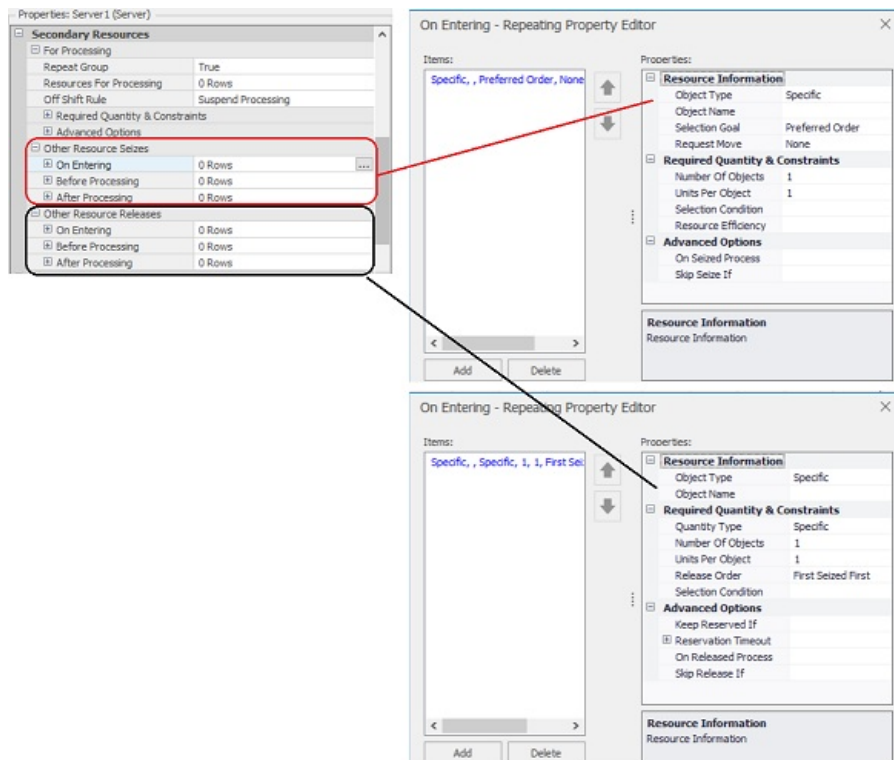
Within the 'For Processing' section, any resource(s) specified here will be seized before the entity begins within the object and will be released once processing is complete. This can be a single resource or one from a list, or can be defined through a repeat group of multiple resources. The properties are specified directly in the main property window for the object or within the repeat group, if defined. Resources that are specified as the 'For Processing' will be seized before any resources specified in the 'Other Resource Seizes' / 'Before Processing' and will be released after any resources in the 'Other Resource Releases' / 'After Processing'.

Additionally, multiple resources can be specified for seizing / releasing at various other times within the object. For the Server and Separator, this includes On Entering, Before Processing and After Processing. For the Combiner, it includes On Parent Entering, Before Processing and After Processing. Within each of these categories, the repeating property editor may be opened by pressing the small box to the right to enter multiple resources to seize or release at the specified time. Within each of the 'Other Resource Seizes' sections, the *Must Simultaneously Seize* property determines whether or not multiple resources must all be available at the same time before any of them is seized.

Secondary resources allow the user the flexibility to 'carry' resources between objects. For example, you may seize an operator upon entering Server1 and not release that operator within Server1. Then the entity continues through several other servers and finally releases the Operator after processing at Server5.

Within any of the secondary resources sections, the resource seized may be 'reserved' if the *Keep Reserved If* property expression results in a 'True' or '1' type evaluation. See the [Reservations](#) page for more details about keeping resources reserved. Optional *Skip Seize If* and *Skip Release If* expressions can be used for conditional seizing and releasing.

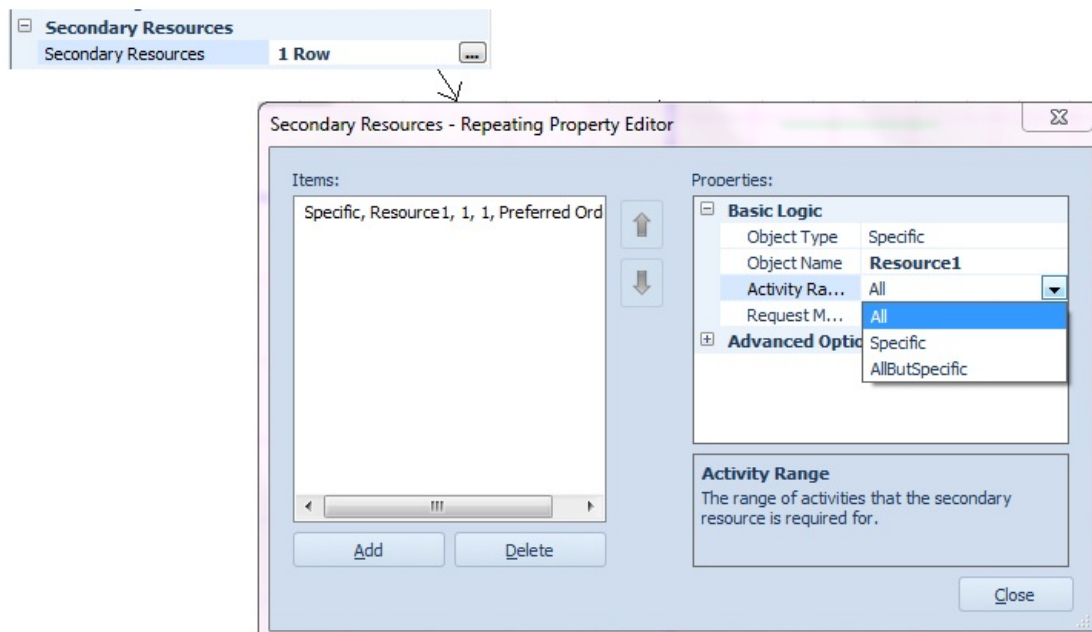
Secondary Resources within a Server



Within Workstations (Deprecated)

There are secondary resources that may be used with the Workstation (Deprecated) object. These resources are specified within the Other Requirements section of properties for the Workstation and are allocated to the entity for specific processing tasks within the workstation. Any secondary resources specified in this manner are allocated for setup, processing, teardown or all activities, based on the *Activity Range* property specified. The activities which the resources are used for are specified within the Secondary Resources repeating property editor. There are no options to keep a resource 'reserved' within this section of Secondary Resources of a Workstation object.

Secondary Resources within a Workstation (Deprecated)



Properties for the 'Resources for Processing' are found within the properties of Server, Combiner, Separator, Filler and Empty.

Listed below are the properties of the **Other Resource Seizes**:

Property	Valid Entry	Description
----------	-------------	-------------

Object Type	Specific, From List	The method for specifying the resource object(s) to be seized.
Object Name	Object Instance Name	The name of the resource object to be seized.
Object List Name	Object List Instance Name	The name of the object list from which one or more resource objects will be selected to be seized.
Selection Goal	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value, Random	The goal for selecting resource objects to seize.
Selection Expression	Expression	The expression evaluated for each resource that is a candidate to seize. In the expression, use the keyword Candidate to reference an object in the collection of candidates to be seized (e.g., Candidate.Object.Capacity.Remaining).
Request Move	None, To Node	Indicates whether a move to a specified location will be requested from the seized resource(s). Server activity will not continue until all resources have arrived to the requested location.
Destination Node	Node Instance Name	The name of the specific node location that the seized resource(s) will be requested to move to. It is sometimes useful to use a function of the fixed object to access its input or output node, such as 'Server.Input' or 'Combiner.Output' to move the resource to a node associated with the object.
Number of Objects	Expression	The number of individual resource objects of which to seize capacity units.
Units Per Object	Expression	The number of capacity units to seize per resource object.
Selection Condition	Expression	An optional condition evaluated for each candidate resource that must be true for the object to be eligible for seizing. In the condition, use the keyword 'Candidate' to reference an object in a collection of candidates (e.g., Candidate.Object.Capacity.Remaining).
Resource Efficiency	Expression	Optional value that can alter the rate at which work is performed using the seized resource(s), expressed as a fraction. The actual work duration is the planned work duration divided by the efficiency. Hence, an efficiency value greater than 1 shortens the time and a value less than 1 lengthens the time. In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource objects (e.g., Candidate.Object.Capacity). Set this property to blank to declare the resource efficiency as undefined.
On Seized Process	Process Instance Name	Optional process that is executed by tokens associated with each seized resource.
Skip Seize If	Expression	Optional condition indicating whether to skip the seize requirement.

Listed below are the properties of the **Other Resource Releases**:

Property	Value Entry	Description
Object Type	Specific, From List	The method for specifying the resource object(s) to be released.
Object Name	Object Instance Name	The name of the resource object to be released.
Object List Name	Object List Instance Name	The name of the object list from which one or more resource objects will be released.
Release Quantity Type	All, Specific	Indicates whether to release all owned resources of the specified type or to release a specific quantity.
Number of Objects	Expression	The number of individual resource objects of which to release capacity units.
Units Per Object	Expression	The number of capacity units to release per resource object.
Release Order	First Seized First, Last Seized First	The order in which to release the resources that have been seized by the entity.
Selection Condition	Expression	An optional condition evaluated for each seized resource that must evaluate to true for the resource to be eligible to be released. In the condition, use the keyword 'Candidate' to reference an object in the collection of candidates (e.g., Candidate.Object.ID).
Keep Reserved If	Expression	<p>Optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects will be unable to seize the reserved capacity unless the reservation is cancelled.</p> <p>In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released (e.g., Candidate.Object.Capacity).</p> <p>Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p> <p>Leaving this property blank (no condition) is equivalent to entering False.</p>
Reservation Timeout	Expression	<p>If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation.</p> <p>In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released (e.g., Candidate.Object.Capacity).</p>
On Released Process	Process Instance Name	Optional process that is executed by tokens associated with each released resource.
Skip Release If	Expression	Optional condition indicating whether to skip the release requirement.

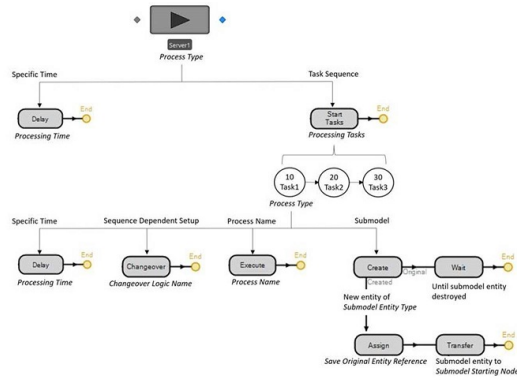
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Task Sequence - Processing Tasks

For a Task Sequence, the Server, Combiner and Separator objects allow you to model the processing of an individual task using one of the following general process types.

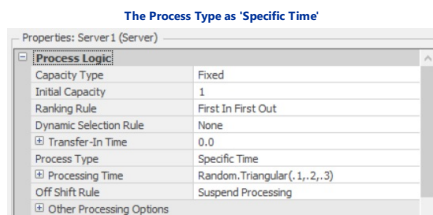
- **Specific Time** – The processing of the task is defined as a specific processing time. The task is considered finished once the time duration has elapsed.
- **Process Name** – The processing of the task is handled by executing a specified process name. The task is considered finished once the token that was created to execute the invoked process has ended its processing.
- **Submodel** – The processing of the task is handled by executing a part of the overall model that is defined elsewhere in the Facility window. The task creates a new entity of a specified entity type and then sends that new entity to a specified node (the starting point for the submodel). The task is considered finished once the entity that was created to execute the submodel has ended its processing (i.e., has been destroyed).



Specific Time

Specific Time as Process Type

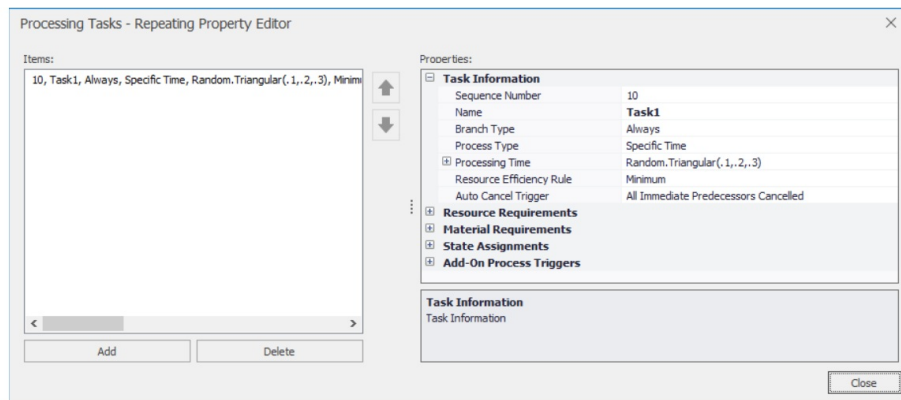
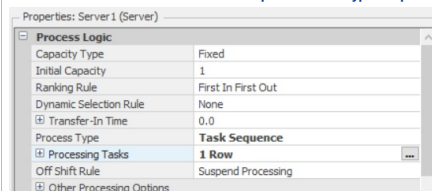
If the *Process Type* property is specified as 'Specific Time', the entity will be delayed for that specific amount of time before it moves from the processing station. This is the default behavior for the Server/Combiner/Separator objects and only behavior for the Server in Simio sprint versions prior to Simio 7.119 (and Combiner/Separator prior to Simio 7.130).



Specific Time within Task Sequence

If the *Process Type* property is specified as 'Task Sequence' and the *Process Type* within the particular Task is specified as 'Specific Time', the entity will be delayed for that specific amount of time before it moves from that task to the next.

The Task Sequence Process Type as 'Specific Time'



Process Name

If the *Process Type* property is specified as 'Task Sequence' and the *Process Type* within the particular Task is specified as 'Process Name', the task is considered finished once the token that was created to execute the invoked process has ended its processing.

The Task Sequence Process Type as 'Process Name'

Task Information	
Sequence Number	10
Name	Task1
Branch Type	Conditional
Condition Or Probability	Entity.Is.PartA
Process Type	Process Name
Process Name	Server1_Task1_ProcessA
Auto Cancel Trigger	All Immediate Predecessors Cancelled

☐ Resource Requirements
☐ Material Requirements
☐ State Assignments
☐ Add-On Process Triggers

Submodel

If the *Process Type* property is specified as 'Task Sequence' and the *Process Type* within the particular Task is specified as 'Submodel', the task creates a new entity of a specified entity type and then sends that new entity to a specified node (the starting point for the submodel). The task is considered finished once the entity that was created to execute the submodel has ended its processing (i.e., has been destroyed).

The Task Sequence Process Type as 'Submodel'

Task Information	
Sequence Number	10
Name	Task1
Branch Type	Conditional
Condition Or Probability	.8
Process Type	Submodel
Submodel Entity Creation Method	Copy Original Entity Attributes
Submodel Entity Type	LabOrder
Submodel Starting Node	Input@LabProcessing
Save Original Entity Reference	
Auto Cancel Trigger	All Immediate Predecessors Cancelled

☐ Resource Requirements
☐ Material Requirements
☐ State Assignments
☐ Add-On Process Triggers

Sequence Dependent Setup

If the *Process Type* property is specified as 'Task Sequence' and the *Process Type* within the particular Task is specified as 'Sequence Dependent Setup', the *Changeover Logic Name* is evaluated to determine if there are any setup times to be incurred. A Changeover step will be executed by the processing task token, with the token then held in that step for the required setup time (if any). Note that if there is no required setup time then any specified task resource actions (the Seize and Release steps) as well as any specified material actions (the Consume and Produce steps) will be skipped/ignored. Thus, for example, a Worker required for setup at the Server will only be seized and moved to the Server if there is an actual setup time.

The Task Sequence Process Type as 'Sequence Dependent Setup'

Task Information	
Sequence Number	10
Name	
Branch Type	Always
Process Type	Sequence Dependent Setup
Changeover Logic Name	SetupTimes
Resource Efficiency Rule	Minimum
Auto Cancel Trigger	All Immediate Predecessors Cancelled

☐ Resource Requirements
☐ Material Requirements
☐ State Assignments
☐ Add-On Process Triggers

Resource Requirements

Within each processing task, Simio also supports the definition of resource requirements, which includes specifying the resource that will be seized and released for the task and whether the resource will be 'reserved' for additional tasks.

The *Off Shift Rule* property is specified for each particular task. Therefore, if multiple resources are specified through the use of data tables (see below), the *Off Shift Rule* is by task and pertains to all resources seized.

Note that the default values for the *Keep Reserved If* and *Reservation Timeout* properties (as shown below) are set to 'True' and 'Math.Epsilon' respectively. If these property values are left unchanged by the user, then by default once the task is finished the entity will always momentarily keep reserved the resource(s) that were seized to perform the task, just in case there happens to be an immediate next task whose specified resource requirements will prefer to re-seize (continue using) the same resource(s).

If there is no immediate re-seize, then the resource reservation(s) will be automatically cancelled at the end of the same time step as the epsilon reservation timeout period expires. This default behavior is intended to make it easy for the user to model the same task resource(s) being used for multiple tasks that are performed consecutively.

The *Skip Requirement If* property allows the user to specify an optional condition that is evaluated for each task. If evaluated to 'True', any resource(s) requirements specified will not be seized and released.

The Resource Requirements for a Processing Task

Task Information	
Sequence Number	10
Name	
Branch Type	Always
Process Type	Specific Time
Processing Time	Random.Triangular(, 1, 2, 3)
Resource Efficiency Rule	Minimum
Auto Cancel Trigger	All Immediate Predecessors Cancelled

Resource Requirements	
Object Type	Specific
Object Name	
Selection Goal	Preferred Order
Request Move	None
Off Shift Rule	Suspend Processing

Required Quantity & Constraints	
Number Of Objects	1
Units Per Object	1
Selection Condition	
Resource Efficiency	1.0
Must Simultaneously Seize	False

Advanced Options	
Immediately Try Seize	False
Keep Reserved If	True
Reservation Timeout	Math.Epsilon
Immediately Try Allocate When Released	False
Skip Requirement If	

☐ Material Requirements
☐ State Assignments
☐ Add-On Process Triggers

Material Requirements

In addition to resource requirements for a given task, Simio also supports material requirements, which includes specifying

the material (or bill of materials) that will be consumed or produced for the task. The material (or bill of materials) consumed or produced may also be associated with a given inventory storage location. A *Lot ID* (string) may also be specified for a material.

Note that an *Action Type* property with choices 'Consume' or 'Produce' allows the user specify either material consumption required to start the processing of the task or material production that occurs after the processing of the task is finished. The material consumption will occur after all resources are seized, but before the *Starting Task* add-on process. The material production will occur after the *Finished Task* add-on process, but before releasing any seized resources.

For the consumption of materials, the user can specify, using the *Must Simultaneously Consume* property, whether or not all required materials must be available for material allocation. The *Skip Requirement If* property allows the user to specify an optional condition that is evaluated for each task. If evaluated to 'True', any material requirements specified will not be consumed/produced.

Note: If consuming a Material (directly or through Inventory) where the backorder is declined due to the *Allow Backorder Policy* and there is no Material available, the Entity (via the Token) balks from the Material's Allocation Queue, thereby eliminating the requirement. Furthermore, the Token that was executing the Material consumption is destroyed, ending the containing Process. When this happens for a Task Sequence, the Token executing the Consume step is destroyed and the Task is stopped. No subsequent actions will be taken for that specific Task. For example, the Processing Time does not occur, nor does *Finished Task Add-On Process* trigger. The Task simply ends but is not considered canceled. Because the Task was not canceled, successor tasks will then commence.

The Material and Inventory elements provide an *On Balked At Backorder Process* to handle the balked Entity. This Process will run after the related Token balks from Material Allocation Queue (i.e., Material unable to be consumed and backorder is declined).

You might, for example, assign a new value to an Entity state variable using *On Balked At Backorder Process* to indicate a balked backorder, and in a more complex implementation, move to a different branch of the Task Sequence to achieve desired behavior. Or, you might choose to simply Transfer that balked Entity out of the Server (or other object), sending balked orders to a designated location.

The Material Requirements for a Processing Task

Task Information	
Sequence Number	10
Name	
Branch Type	Always
Process Type	Specific Time
Processing Time	Random.Triangular(, 1, 2, 3)
Resource Efficiency Rule	Minimum
Auto Cancel Trigger	All Immediate Predecessors Cancelled
Resource Requirements	
Material Requirements	
Action Type	Consume
Consumption Type	Material
Material Name	
Inventory Site Type	None
Required Quantity & Constraints	
Quantity	1.0
Lot ID	
Must Simultaneously Consume	False
Advanced Options	
Skip Requirement If	
State Assignments	
Add-On Process Triggers	

State Assignments

There are three times during a particular task when one or more state assignments can be made. 'Task Ready' state assignments will occur once all of the task's predecessor tasks have been satisfied and the entity is about to start seizing resources and consuming required materials needed for the task. 'Starting Task' state assignments will occur once all resources and materials have been seized and consumed and the task is about to start. 'Finished Task' state assignments are made when the task has been finished.

The State Assignments for a Processing Task

Task Information	
Sequence Number	10
Name	
Branch Type	Always
Process Type	Specific Time
Processing Time	Random.Triangular(, 1, 2, 3)
Resource Efficiency Rule	Minimum
Auto Cancel Trigger	All Immediate Predecessors Cancelled
Resource Requirements	
Material Requirements	
State Assignments	
Assign When	Task Ready
State Variable Name	
New Value	0.0
Add-On Process Triggers	

Add-On Process Triggers

The *Task Ready* Add-On process begins when all of the task's predecessor dependencies have been satisfied and the entity is about to seize resources and consume/produce materials specified. The *Starting Task* Add-On process begins after the seizing of the specified resource(s) and consuming/producing specified material(s). The *Finished Task* Add-On process begins after processing, but prior to the resource(s) being released.

The Add-On Process Triggers for a Processing Task

Task Information	
Sequence Number	10
Name	
Branch Type	Always
Process Type	Specific Time
Processing Time	Random.Triangular(, 1, 2, 3)
Resource Efficiency Rule	Minimum
Auto Cancel Trigger	All Immediate Predecessors Cancelled
Resource Requirements	
Material Requirements	
State Assignments	
Add-On Process Triggers	
Task Ready	
Starting Task	
Finished Task	

Using Task Sequences with Data Tables

In all of the above examples, the data for all the properties within the Task Sequence Processing Tasks dialog is specified directly within the repeating group properties. Data tables (within the Data tab) can also be extremely useful in storing information regarding the processing tasks for one or more objects.

First, the 'schema' for the data table can be generated by going to the Data tab and clicking on the Add Data Table arrow. A list will then appear with the various repeat group options, based on the objects currently placed in the Facility window. If at least one Server has been placed in the Facility window, the option for *Server.ProcessingTasks* will be shown (and

likewise for Combiner/Separator). Clicking on this option will automatically generate a data table with its columns matching the name and type of each property within the repeat group. These columns can then be used to store information regarding the one or more servers into which it will feed information. Unused columns can be deleted.

Add Data Table
A table consisting of columns of properties, states, and targets.

Add Data Table From Schema
A table with a schema matching a repeat group of an existing object.

Add Sequence Table
A data table with a destination column for defining routing logic.

Add Output Table
A table of state columns, with rows created at runtime.

Available Schemas

- BasicNode.AssignmentsOnEntering
- BasicNode.TalliesOnEntering
- BasicNode.TalliesOnExited
- Server.AssignmentsAfterProcessing
- Server.AssignmentsBeforeExiting
- Server.AssignmentsBeforeProcessing
- Server.AssignmentsOnBalking
- Server.AssignmentsOnEntering
- Server.AssignmentsOnReneging
- Server.InputBufferRenegTriggers
- Server.OutputBufferRenegTriggers
- Server.ProcessingLoopbackBranches
- Server.ProcessingTasks**
- Server.SecondaryResourceReleasesAfterProcessing
- Server.SecondaryResourceReleasesBeforeProcessing
- Server.SecondaryResourceReleasesOnEntering
- Server.SecondaryResourceSeizesAfterProcessing
- Server.SecondaryResourceSeizesBeforeProcessing
- Server.SecondaryResourceSeizesOnEntering
- TransferNode.AssignmentsOnEntering
- TransferNode.RoutingOutRequiredMaterials
- TransferNode.TalliesOnEntering
- TransferNode.TalliesOnExited

Server - Processing Tasks Repeating Property Editor within the Facility window

Data Table with Column Properties based on Processing Tasks repeat group above

Table 1	Sequence Number	Name	Branch Type	Condition Or Probability	Process Type	Processing Time (Minutes)	Process Name

Once the data table has been generated, data can be entered into the table. Within the Facility window, as shown below, once the Server has the *Process Type* of 'Task Sequence', the *Processing Tasks* property will be visible. Right-clicking on the *Processing Tasks* property will allow the user to 'Set Referenced Property' to the table named 'Table1'. The name 'Table1' will then appear within the *Processing Tasks* property field with a green (reference) arrow. Clicking on the ... button to the right will open the repeating property editor and the properties within this repeat group will then automatically reference the table columns within Table1. Any column property that was previously deleted (because it would not be used for whatever reason) will appear with the property default value.

Right-click on Processing Tasks

Properties: Server1 (Server)

Process Logic

- Capacity Type: Fixed
- Initial Capacity: 1
- Ranking Rule: First In First Out
- Dynamic Selection Rule: None
- Transfer-In Time: 0.0
- Process Type: Task Sequence
- Processing Tasks: Table1
- Other Task Sequence Options: Reset

Set Referenced Property

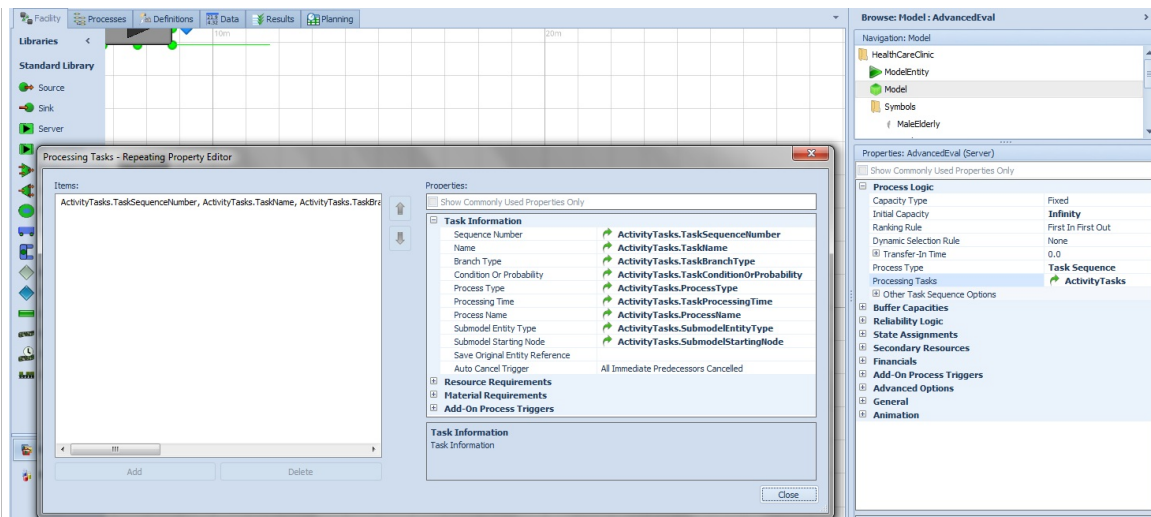
Table1

Processing Tasks

Task Information

- Sequence Number: Table1.TaskSequenceNumber
- Name: Table1.TaskName
- Branch Type: Table1.TaskBranchType
- Condition Or Probability: Table1.TaskConditionOrProbability
- Process Type: Table1.TaskProcessType
- Processing Time: Table1.TaskProcessingTime
- Process Name: Table1.TaskProcessName
- Submodel Entity Type: Table1.TaskSubmodelEntityType
- Submodel Starting Node: Table1.TaskSubmodelStartingNode
- Save Original Entity Reference: Table1.TaskSaveOriginalEntityReference
- Auto Cancel Trigger: Table1.TaskAutoCancelTrigger

Within the HealthCareClinic example (See Support ribbon / Examples), there are two servers in the system that help to process two different types of entities. Data tables are used in this example to store the various information for specific process tasks, as can be seen in the picture below. Relational tables and key columns are used within this example so that both Server objects reference the same data table for processing task information, based on the patient (entity) type and activity (key).

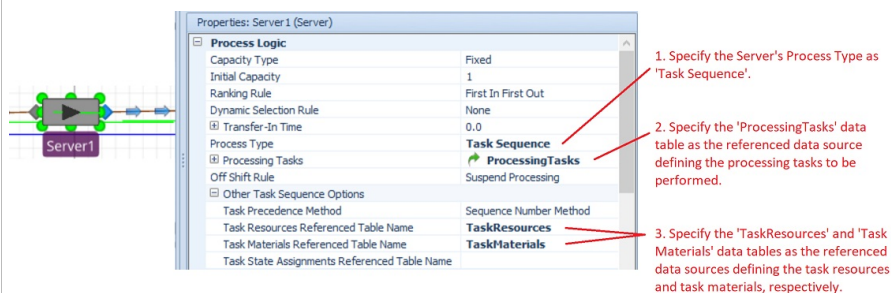


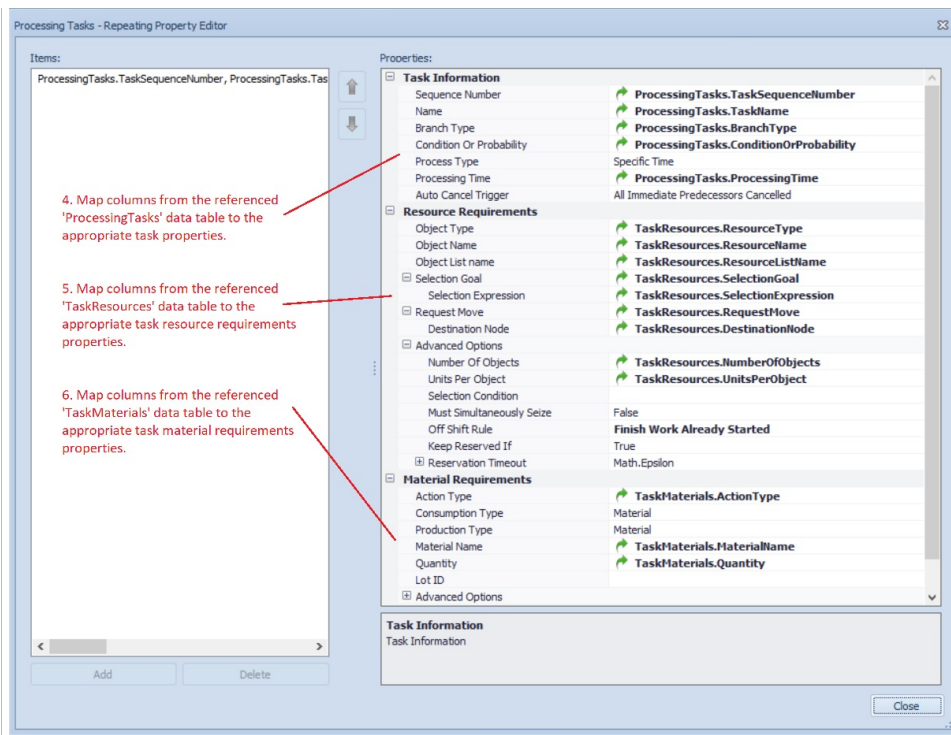
Patient Type	Activities	Patient Flow
Patient Type	Step	Destination
1	MildSick	1 Input@Registration
2	MildSick	2 Input@BasicEval
3	MildSick	3 Input@Checkout
4	Sick	1 Input@Registration
5	Sick	2 Input@AdvancedEval
6	Sick	3 Input@Checkout

Activity	Task Sequence Number	Task Name	Branch Type	Condition Or Probability	Process Type	Processing Time (Minutes)	Process Name	Submodel Entity Type	Submodel Starting Node
1	Registration	FindPatientName	Always	0.0	Specific Time	5			
2	Registration	SendInfoToNurse	Always	0.0	Specific Time	10			
3	BasicEvaluation	GetVitals	Always	0.0	Specific Time	4			
4	BasicEvaluation	EvalWithDoctor	Always	0.0	Specific Time	6			
5	BasicEvaluation	FollowupTask	Always	0.0	Specific Time	2			
6	AdvancedEvaluation	EvalWithDoctor	Always	0.0	Specific Time	15			
7	AdvancedEvaluation	SendLabWork	Probabilistic	0.7	Submodel			LabWorkOrder	Input@LabWork

Using Task Sequences with Data Tables - Multiple Resources, Materials or State Assignments for a Specific Task

Another example of using task sequences with data tables is within the SimBit Server using TaskSequenceWithDataTables.FlowLine. This small example takes the data table concept a bit further so that multiple resources (and/or materials or state assignments) can be used within a single process task. Note that, by default, each task in the Processing Tasks repeating properties has a single resource/resource list specification and single material requirement. If multiple resources/workers are required for a particular task, the resource requirements can be stored in a separate data table than that described above. This similar approach can also be taken for specifying multiple materials or state assignments within a given task. Within an object using the 'Task Sequence' option, the Other Task Sequence Options section of properties is available. The *Task Resources Referenced Table Name*, *Task Materials Referenced Table Name* and *Task State Assignments Referenced Table Name* properties have a pull down list of data tables. In the below example, the 'Task Resources' table stores information regarding the multiple task resources for a given task. Within the Processing Tasks repeating properties, the Resource Requirements properties will then reference a different table than the table that stores the individual task information. The tables must be relational with foreign key and key columns as shown in the SimBit documentation.





Note, if the Task Resources, Materials, or State Assignments Table Name has been specified under Other Task Sequence Options, then only that table should be referenced in the Processing Tasks' associated properties in the Repeating Property Group Item.

For more information on structuring Task Sequences, please refer to the [Task Sequence](#) element and [Task Sequence - Discussion and Examples](#).

For SimBit examples that include Task Sequences, please refer to [ServerUsingTaskSequence](#), [ServersUsingTaskSequenceWithDataTables_FlowLine](#), [ServersUsingTaskSequenceWithDataTables_JobShop](#) and [ServerUsingTaskSequenceWithWorkers](#). Additionally, the [HealthCareClinic](#) example includes task sequences using data tables, as noted in the above discussion.

Listed below are the properties of the **Processing Tasks Repeating Properties in Server/Combiner/Separator**.

Property	Valid Entry	Description
Sequence Number	Real	Sequence number used to define the task precedence constraints. To model a task sequence that is simply a serially ordered set of tasks, define the tasks with increasing sequence numbers (e.g., the first task is numbered '10', the second task is numbered '20', and so forth). Shown if the <i>Task Precedence Method</i> (Other Task Sequence Options) is 'SequenceNumberMethod'. For a more in-depth discussion on task sequence numbering including examples of how to model sequences with tasks processed in parallel, please refer to the Task Sequence element documentation.
ID Number	Integer	Unique integer identifier number for the task. Shown if the <i>Task Precedence Method</i> (Other Task Sequence Options) is 'ImmediatePredecessorsMethod' or 'ImmediateSuccessorsMethod'.
Name	Task Name	Name field for the task.
Branch Type	Always, Conditional, Probabilistic	Indicates whether the task is always performed, whether it is instead treated as the first task of a conditional or probabilistic branch in the process workflow.
Condition or Probability	Expression	The branch condition or probability specified as an expression. If a condition, then enter the logical condition. If a probability, enter the chance of selecting the task as a value between 0.0 (0%) and 1.0 (100%).
Process Type	Specific Time, Process Name, Submodel	The method used to model the processing of a task.
Processing Time	Expression	The time required to perform the task.
Process Name	Process Instance Name	The name of the process that will be executed in order to perform the task. The task will be considered finished once this invoked process has ended.
Submodel Entity Creation Method	Create New (No Copying), Copy Original Entity Attributes	The method used to create an entity that will be sent to some other distinct part of the overall model. The task will be considered finished once this created submodel entity has ended its processing (i.e., been destroyed). If copying the original entity's attributes, then all possible state values and table references will be copied from the original entity to the created submodel entity.
Submodel Entity Type	Valid Entity Name	The type of entity to be created that will be sent to the submodel. The task will be considered finished once this entity has ended its processing (i.e., has been destroyed).
Submodel Starting Node	Node Name	The node that is the starting point for the submodel.
Save Original Entity Reference	Entity Reference State Variable	Optional entity reference state variable to save a reference to the original entity being processed at the server. Can be a state variable defined on the submodel entity, to allow easy access to submodel logic to the original entity's attributes.
Changeover Logic Name	ChangeoverLogic Name	The name of the ChangeoverLogic element used to determine any sequence dependent setup times. Go to Definitions, Elements to add a new ChangeoverLogic element to the model.
Resource	None, Average,	The processing rule used to alter the rate at which the task is performed if

Efficiency Rule	Count, Maximum, Minimum, Sum	there are seized resources with defined efficiency values. The actual task duration is the planned task duration divided by the efficiency. If the rule is 'None', then seized resource efficiency is ignored. If the rule is 'Average', then the average seized resource efficiency value is used. If the rule is 'Count', then the number of seized resources with a defined efficiency value is used. If the rule is 'Maximum', then the largest efficiency value is used. If the rule is 'Minimum', then the smallest efficiency value is used. If the rule is 'Sum', then the sum of the seized resource efficiency values is used.
Immediate Predecessors	Valid Integer ID Numbers	Lists the ID numbers of the tasks which must be finished or cancelled before this task can start. To specify more than one predecessor, enter multiple task ID numbers separated by commas. Shown when the Task Precedence Method is 'ImmediatePredecessorsMethod'.
Immediate Successors	Valid Integer ID Numbers	Lists the ID numbers of the tasks which can't start until this task is finished or cancelled. To specify more than one successor, enter multiple task ID numbers separated by commas. Shown when the Task Precedence Method is 'ImmediateSuccessorsMethod'.
Auto Cancel Trigger	None, All Immediate Predecessors Cancelled	Event type that will trigger an automatic cancellation of the task. The default setting is 'All Immediate Predecessors Cancelled'. If left unchanged, then the task will be automatically cancelled if all its immediate predecessors in the task sequence are cancelled. This default setting is recommended if you want conditional or probabilistic branching to automatically cancel all tasks in unselected branches. For a more in-depth discussion of this option including examples, please refer to the Task Sequence element documentation.
Resource Type	Specific, Select From List	The method for specifying the resource object(s) to be seized.
Resource Name	Object Name	The name of the resource object to be seized.
Resource List Name	Object List Name	The name of the object list from which one or more resource objects will be selected to be seized.
Selection Goal	Smallest Distance, Largest Distance, Preferred Order, Smallest Value, Largest Value, Random	The goal for selecting the resource object(s) to seize.
Request Move	None, To Node	Indicates whether a move to a specified location will be requested from the seized resource(s). The task will not start until all resources have arrived to the requested location.
Destination Node	Object Name	The name of the destination node to which the seized resource(s) will be requested to move.
Off Shift Rule	Finish Work Already Started, Suspend Processing, Switch Resources If Possible	The processing rule used if a resource required for the task is at the end of a shift because of its specified work schedule. If the rule is 'Finish Work Already Started', then the processing of the task will be allowed to continue until finished. The resource will not be allowed to accept any new work. If the rule is 'Suspend Processing', then the processing of the task will be immediately suspended. Processing will resume at the start of the resource's next shift. If the rule is 'Switch Resources If Possible', then the processing of the task will be immediately suspended. The affected entity will then try to resume the task's processing as soon as possible by releasing the resource and seizing another available one that satisfies the same task resource requirements. NOTE: This property setting may vary from task to task but not from resource requirement to resource requirement for the same task. Thus, if using a table driven approach to define task sequence data, the value of this property must be either left constant or specified as a column reference in the table that defines each individual processing task. Referencing a column in the Task Resources Referenced Table Name (if using that option) is invalid.
Number of Resources	Expression	The number of individual resource objects to seize capacity units of.
Units Per Resource	Expression	The number of capacity units to seize per resource.
Selection Condition	Expression	An optional condition evaluated for each candidate resource that must be true for the resource to be eligible for seizing. In the condition, use the keyword 'Candidate' to reference an object in the collection of candidates (e.g., Candidate.Object.Capacity.Remaining)
Resource Efficiency	Expression	Optional value that can alter the rate at which the task is performed using the seized resource(s), expressed as a fraction. The actual task duration is the planned task duration divided by the efficiency. Hence, an efficiency value greater than 1 shortens the time and a value less than 1 lengthens the time. In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource objects (e.g., Candidate.Object.Capacity). Set this property to blank to declare the resource efficiency as undefined.
Must Simultaneously Seize	True, False	If multiple resources are required for the task, indicates whether all of the resources must be available before any can be seized. NOTE: This property setting may vary from task to task but not from resource requirement to resource requirement for the same task. Thus, if using a table driven approach to define the task sequence data, the value of this property must either be left as a constant or specified as a column reference in the table that defines each individual processing task. Referencing a column in the Task Resources Referenced Table Name (if using that option) is invalid.
Immediately Try Seize	True, False	Indicates whether to immediately try seizing the resource before the execution of any other simulation logic in the system and, if successful, skipping the resource allocation queues. Setting this property to False will just insert the seize request into the resource allocation queues. An evaluation of the queues will then be scheduled on the simulation's current event calendar as a late priority event. NOTE: This property setting may vary from task to task but not from resource requirement to resource requirement for the same task. Thus, if using a table driven approach to define the task sequence data, the value of this property may only be specified as a constant or by referring to a column in the table that defines each individual processing task.
Keep Reserved If	Expression	Optional condition indicating whether to keep the released resource capacity reserved for possible later reuse by the same owner object. Other objects will be unable to seize the reserved capacity unless the reservation is cancelled. In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released

		<p>(e.g., Candidate.Object.Capacity).</p> <p>Note that when an owner object is attempting to select a resource from a group of candidates (e.g., from a list or from a population of some resource type), by default a preference will be given to select a resource that has already been reserved by that entity irrespective of the specified selection goal.</p> <p>Leaving this property blank (no condition) is equivalent to entering False.</p>
Reservation Timeout	Expression	<p>If the keep reserved condition is true, then an optional wait time before automatically cancelling the reservation.</p> <p>In the expression, use the syntax Candidate.[ObjectClass].[Attribute] to reference an attribute of the candidate resource object(s) being released (e.g., Candidate.Object.Capacity).</p>
Immediately Try Allocate When Released	True, False	<p>Once the task has completed and released the resource requirements, indicates whether to immediately try allocating the released resource capacity to waiting seize requests before the execution of an other simulation logic in the system.</p> <p>Setting this property to False will skip immediate allocation. Instead, an evaluation of the resource allocation queues will be scheduled on the simulation's current event calendar as a late priority event.</p> <p>NOTE: This property setting may vary from task to task but not from resource requirement to resource requirement for the same task. Thus, if using a table driven approach to define task sequence data, the value of this property may only be specified as a constant or by referencing a column in the table that defines each individual processing task.</p>
Skip Requirement If	Expression	Optional condition indicating whether to skip the resource requirement.
Action Type	Consume, Produce	The type of material-related action required.
Consumption Type	Material, BillOfMaterials	Indicates whether to consume a single material or a bill of materials. The material consumption will be required to start the processing of the task.
Production Type	Material, BillOfMaterials	Indicates whether to produce a single material or a bill of materials. The material consumption will occur after finishing the processing of the task.
Material Name	Material Name	The name of the material which is to be either specifically consumed or produced, or whose bill of materials is to be consumed or produced.
Inventory Site Type	None, ParentObject, SpecificObject	Indicates the fixed object that is the inventory location. Applies only to material elements whose <i>Location Based Inventory</i> property is set to True. The 'ParentObject' setting may be used to indicate that the inventory site is the Server, Combiner or Separator.
Site Object Name	Fixed Object	The name of the fixed object that is the inventory location.
Quantity	Expression	The quantity to be consumed or produced.
Lot ID	String	Optional string value indicating the lot identifier of the consumed or produced material.
Must Simultaneously Consume	True, False	<p>Indicates whether all of the task's required materials must be available before any can be consumed.</p> <p>NOTE: This property setting may vary from task to task but not from material requirement to material requirement for the same task. Thus, if using a table driven approach to define task sequence data, the value of this property may only be specified as a constant or by referencing a column in the table that defines each individual processing task.</p>
Skip Requirement If	Expression	Optional condition indicating whether to skip the material requirement.
Assign When	Task Ready, Starting Task, Finished Task	<p>Indicates when to perform the state assignment.</p> <p>'Task Ready' is when all the task's predecessor dependencies have been satisfied and the entity is about to try seizing or consuming the resources and materials needed (if any) to perform the task.</p> <p>'Starting Task' is when all of the task's resource and material requirements (if any) have been satisfied and the processing of the task is about to begin.</p> <p>'Finished Task' is when the processing of the task has been finished.</p>
State Variable Name	State Variable Name	Name of the state variable that will be assigned a new value.
New Value	Expression	The new value to assign.
Task Ready Add-On Process	Process Instance Name	Occurs when all of the task's predecessor dependencies have been satisfied and the entity is about to try seizing or consuming the resources and materials needed (if any) to perform the task.
Starting Task Add-On Process	Process Instance Name	Occurs when all of the task's resource and material requirements (if any) have been seized or consumed by the entity and the processing of the task is about to begin.
Finished Task Add-On Process	Process Instance Name	Occurs when the processing of the task has been finished.
Seized Resource Add-On Process	Process Instance Name	Occurs when a resource required to perform the task has been seized.
Released Resource Add-On Process	Process Instance Name	Occurs when a resource after finishing the task has been released.
Consumed Material Add-On Process	Process Instance Name	Occurs when a material required to perform the task has been consumed.
Produced Material Add-On Process	Process Instance Name	Occurs when a material after finishing the task has been produced.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Balking And Reneging Options

Balking and Reneging Options

Balking and Reneging Options are available on all objects where input and/or output buffer capacities may be specified. These options allow balking and/or reneging behavior to be easily included in the modeled buffer logic.

Balking occurs when an entity arriving to a buffer decides not to enter the queue, instead immediately leaving the system or going someplace else. Reneging occurs when an entity does enter a buffer but then later leaves the queue before being served or processed.

A specific type of reneging behavior is switching between waiting lines, sometimes referred to as jockeying. Jockeying is when there are parallel servers with distinct waiting lines, and an entity waiting in one of the queues decides to switch to another due to a perceived lower waiting time.

Balking and reneging (including jockeying) are particularly common in service-oriented systems such as call centers, health care clinics, banks, stores, etc.

Balking

If an entity balks at entering a buffer, the entity may either be automatically destroyed or, alternatively, redirected to a specified Facility node location. Statistics on the total number of bailed entities for a buffer are automatically included in reported results.

Common approaches used to model balking behavior at a queue include:

- The queue has a finite capacity (or zero capacity) and an arriving entity always balks if blocked.
- The queue is assumed to have an infinite capacity and an arriving entity uses a probabilistic or conditional rule (typically based on the current queue length) to determine whether to balk or not.

Reneging

If an entity enters a buffer and then reneges at some point, the entity may either be automatically destroyed or, alternatively, redirected to a specified Facility node location. Statistics on the total number of reneged entities for a buffer are automatically included in reported results.

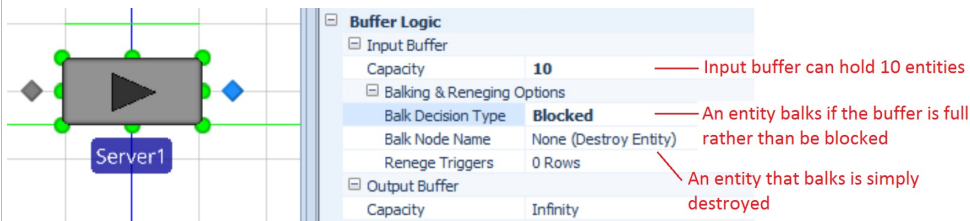
Common approaches used to model reneging behavior at a queue include:

- An entity has a waiting time tolerance, and if that tolerance is exceeded before being served then the entity reneges if it is not within a maximum tolerable position from the front of the queue.
- An event occurrence (e.g., server downtime event, queue length change, periodic evaluation etc.) causes an entity waiting in a queue to make a probabilistic or conditional renege decision.

Balking Examples

Example 1

A buffer has a capacity of 10. An arriving entity always balks if the buffer is full rather than be blocked.

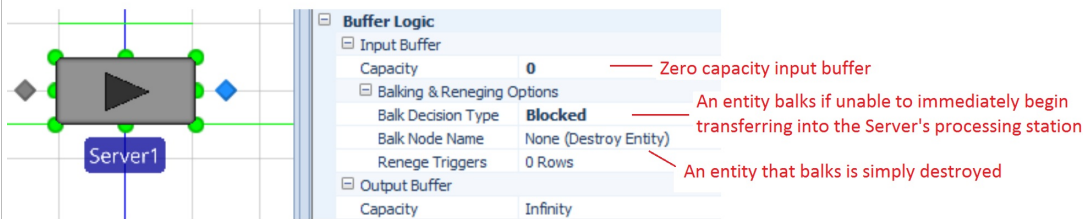


The diagram shows a server icon labeled 'Server1' with a buffer in front of it. The configuration panel for 'Buffer Logic' is shown on the right:

- Input Buffer**
 - Capacity: **10** (An entity balking if the buffer is full rather than be blocked)
- Balking & Reneging Options**
 - Balk Decision Type: **Blocked** (An entity balking if the buffer is full rather than be blocked)
 - Balk Node Name: **None (Destroy Entity)** (An entity that balks is simply destroyed)
 - Reneg Triggers: **0 Rows**
- Output Buffer**
 - Capacity: **Infinity**

Example 2

A buffer has zero capacity. An arriving entity balks if it is unable to begin transferring into the next desired location in the same time step.



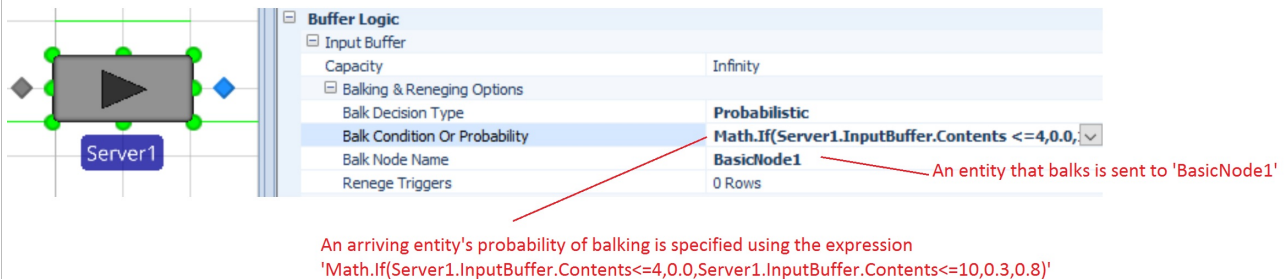
The diagram shows a server icon labeled 'Server1' with a buffer in front of it. The configuration panel for 'Buffer Logic' is shown on the right:

- Input Buffer**
 - Capacity: **0** (Zero capacity input buffer)
- Balking & Reneging Options**
 - Balk Decision Type: **Blocked** (An entity balking if unable to immediately begin transferring into the Server's processing station)
 - Balk Node Name: **None (Destroy Entity)** (An entity that balks is simply destroyed)
 - Reneg Triggers: **0 Rows**
- Output Buffer**
 - Capacity: **Infinity**

Example 3

An arriving entity balks entering a buffer using the following conditional probability distribution:

Condition	Probability of Balking
Queue Length <= 4	0.0
5 <= Queue Length <= 10	0.3
Queue Length > 10	0.8

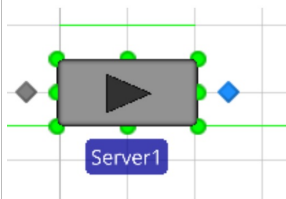


The diagram shows a server icon labeled 'Server1' with a buffer in front of it. The configuration panel for 'Buffer Logic' is shown on the right:

- Input Buffer**
 - Capacity: **Infinity**
- Balking & Reneging Options**
 - Balk Decision Type: **Probabilistic**
 - Balk Condition Or Probability: **Math.If(Server1.InputBuffer.Contents <=4,0.0,Server1.InputBuffer.Contents >=10,0.3,0.8)** (An arriving entity's probability of balking is specified using the expression 'Math.If(Server1.InputBuffer.Contents<=4,0.0,Server1.InputBuffer.Contents>=10,0.3,0.8)')
 - Balk Node Name: **BasicNode1** (An entity that balks is sent to 'BasicNode1')
 - Reneg Triggers: **0 Rows**

Example 4

An arriving entity has a maximum tolerable queue length, determined by generating a sample from Random.Triangular(3,6,15). The entity balks if the buffer's queue length is greater than its tolerance.



Buffer Logic	
Input Buffer	
Capacity	Infinity
Balking & Reneging Options	
Balk Decision Type	Conditional
Balk Condition Or Probability	Server1.InputBuffer.Contents > Random.Triangular(3,6,15)
Balk Node Name	BasicNode1
Reneg Triggers	0 Rows

An arriving entity balks if the current number waiting in the buffer is greater than the entity's maximum tolerable queue length of Random.Triangular(3,6,15)

An entity that balks is sent to 'BasicNode1'

Reneging Examples

Example 1

An arriving entity enters the queue but will always renege if waiting in the queue between 10-12 minutes.

Reneg Trigger	
Trigger Type	Time Based
Wait Duration	Random.Uniform(10,12)
Units	Minutes
Reneg Decision Type	Always
Reneg Node Name	None (Destroy Entity)
Advanced Options	

An entity that decides to enter the queue will renege from the queue after waiting between 10-12 minutes based on Uniform distribution

Reneging entities are destroyed

Example 2

An arriving entity enters the queue but will renege when the event ER_EvalTime is triggered only if the entity's priority value is less than 3 and there are 5 or more entities in the queue.

Reneg Trigger	
Trigger Type	Event Based
Triggering Event Name	ER_EvalTime
Reneg Decision Type	Conditional
Reneg Condition Or Probability	Entity.Priority < 3 && Server1.InputBuffer.Contents >= 5
Reneg Node Name	None (Destroy Entity)
Advanced Options	

Reneging is evaluated based on an event, ER_EvalTime and entities in the queue with priority less than 3 will renege if the buffer has 5 or more entities

Automatically Reported Results

For each physical buffer location in the Standard and Flow Libraries, a total number bailed and total number reneged statistics will be recorded and automatically reported in the Throughput results. Note that these statistics may also be referenced in a model expression using the syntax 'ObjectName.BufferName.NumberBailed' or 'ObjectName.BufferName.NumberReneged'.

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Server	Server 1	InputBuffer	Content	NumberInStation	Average	1.0053
					Maximum	5.0000
			HoldingTime	TimeInStation	Average (Hours)	0.0899
					Maximum (Hours)	0.4544
					Minimum (Hours)	0.0000
			Throughput	NumberBailed	Total	2.0000
				NumberEntered	Total	94.0000
				NumberExited	Total	94.0000
				NumberReneged	Total	15.0000

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Reservations

Reservations for Resources (including Resource, Worker and Vehicle within the Standard Library)

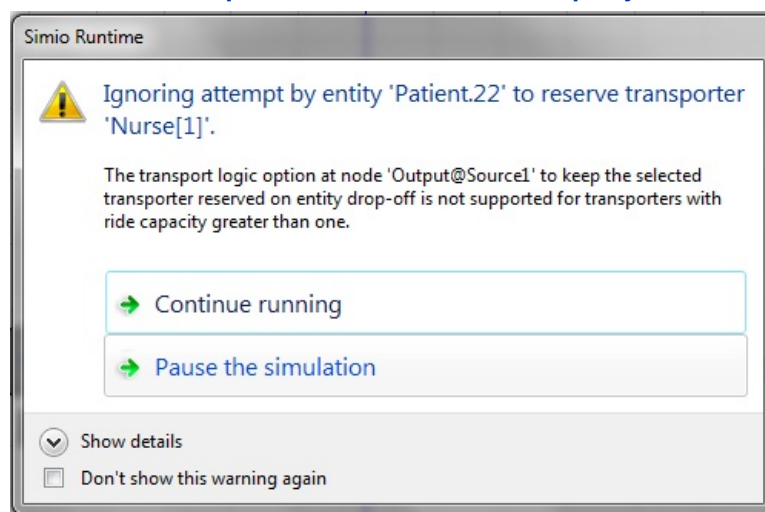
The 'Reserve' capability is automatically incorporated into various Standard Library objects with the Secondary Resources options within Server, Combiner, and Separator. See the *Keep Reserved If* and *Reservation Timeout* properties in these objects. Reservation capability is also available within the Secondary Resource options in the Filler and Emptier of the Flow Library.

If there are multiple reservations for the same resource, then standard ranking and dynamic selections rules apply in terms of the order of who (among the owners with reservations) gets to use the resource. Note that there is no limit on the number of active concurrent reservations for a resource (i.e., you can have a 100 entities with reservations for the same resource at any given time). When a resource becomes available, if there are active reservations then it will only allow an entity with an active reservation to seize it. If there happen to be multiple entities with active reservations waiting in the resource's AllocationQueue trying to seize it, then the best entity with a reservation per the allocation queue's ranking rule and/or the resource's dynamic selection rule will be the next entity to get the resource. For example, suppose Entity1 (Priority = 1), Entity2 (Priority = 2), and Entity3 (Priority = 3) all are trying to seize Worker1[1]. But only Entity1 and Entity2 have reservations. If Worker1[1] is released, it will look in its AllocationQueue for the next entity to seize it. Let's suppose its ranking rule is 'Largest Priority First'. Entity3 is the highest ranked entity in its allocation queue, but that entity has no reservation so it can't seize the worker. Both Entity1 and Entity2 have reservations, so Entity2 is the next entity to get the worker because it has the highest priority among all the waiting entities with reservations.

When attempting a Seize or Ride step, the entity will prefer to immediately select a reserved resource. But if it cannot select that resource, then it will simply select the best resource immediately available per the specified selection goal/rule at the Seize or Ride step. For example, suppose an entity is trying to select a resource from a list that has Worker1[1], Worker2[1], and Worker3[1] using Preferred Order selection goal, and it has a reservation for Worker1[1]. It will prefer to immediately seize Worker1[1] but if that worker is being used by someone else then the entity will instead seize any available worker if possible using the specified Preferred Order selection goal. So if Worker2[1] is immediately available then it will simply seize Worker2[1].

Additionally, within the Transfer node of any Standard Library object, if a Transporter is used for transfer (Worker/Vehicle), the *Keep Reserved If* property is available to reserve the transporter upon dropping off an entity after transport. This then allows resources to be allocated for transport to a Server, for example, reserved and used for processing at that Server and then reserved and used for transport to the next destination. It is important to note that if a transporter has a ride capacity greater than 1 (default), the transporter will not be reserved by the entity(s) riding on the transporter and a runtime warning will be displayed (see below).

Transfer Node - 'Keep Reserved If' True with Ride Capacity Greater than 1



While many objects include the ability for reservations, Simio also includes this capability within the Steps in the Processes window. The [Reserve](#) step in a process may be used to arbitrarily reserve a resource's capacity for possible later use. Reserved resource capacity is capacity that, although unallocated, has been set aside for a specific future owner thus preventing its use by anyone else without a reservation. The [UnReserve](#) step may be used to cancel any reservations for a resource made by a specified owner up to a desired number of reserved capacity units cancelled.

There are a number of functions available related to reservations on objects. These include ResourceName.Capacity.Reserved, ResourceName.Capacity.ReservedTo(owner), and a number of ResourceName.ReservationOwners.* functions. These are listed on the [Functions, States and Events for All Intelligent Objects](#) page under the Functions available for Intelligent Objects that are Enabled as Resource Objects.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Stock Reservation

Stock Reservation

A stock reservation represents a material stock putaway or removal reservation and can only be created by the ReserveStock and ReserveBin steps. Once a stock reservation is created, an AddStock or RemoveStock step can be used to add or remove stock using the reservation. A stock reservation always has a type, either 'Putaway' or 'Removal'. This type can be accessed using the Type function. Removal reservations can only be used with the RemoveStock step. Similarly, putaway reservations can only be used with the AddStock step.

For more information about using stock reservations, [Material Storage – Discussion and Examples](#).

Listed below are the properties of the **RemoveStock**

Function Name	Description
IDNumber	Unique ID number for the stock reservation.
Type	The stock reservation type. Possible Values: Putaway = 0, Removal = 1
Quantity	The incoming quantity or quantity reserved.
StockRequirement	The associated stock requirement.
StockQuant	The associated stock quant.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

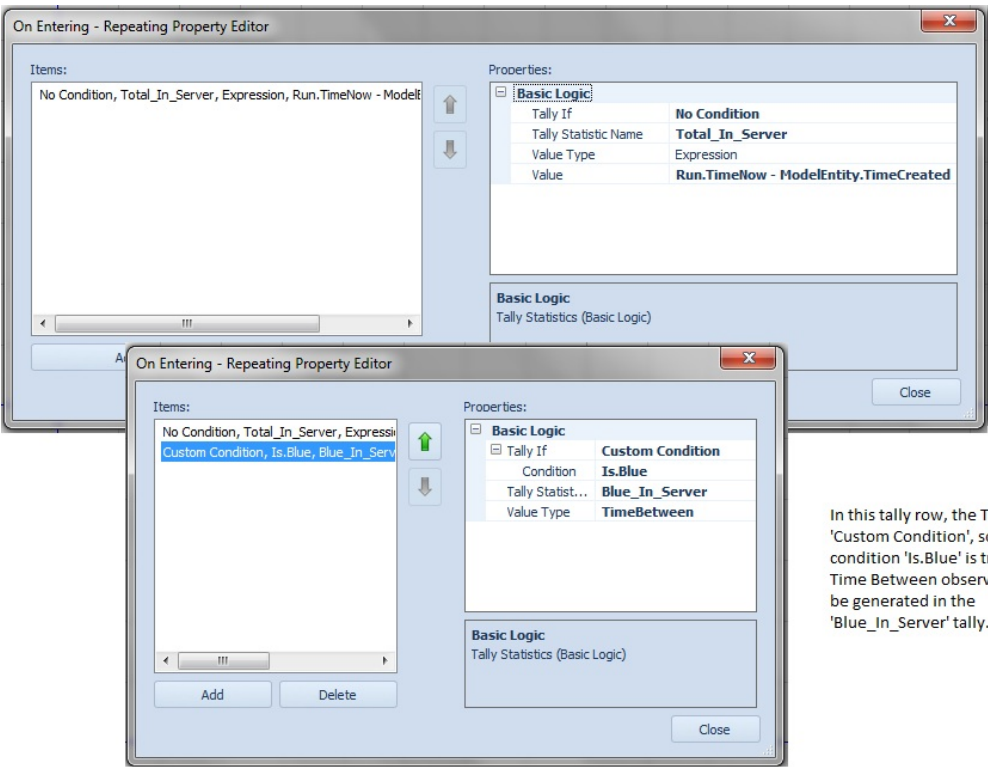
Tally Statistics

Tally Statistics

Tally statistics are available within the BasicNode and TransferNode objects, as well as the Source, Sink, Server, Combiner, Separator and Workstation (Deprecated) external nodes. Tally statistics can be specified when an object enters a node (On Entered) and when an object exits a node (On Exited).

Within the repeating property editor, the user may select the situation when the tally statistic will be calculated. The *Tally If* property determines whether tallies are generated when an entity enters/exits the node, or when a transporter (vehicle or worker) enters/exits the node. Alternatively, the tally may be generated when any object enters ('No Condition') or when a specific condition is met ('Custom Condition'). Multiple tallies can be generated by simply specifying multiple rows in the repeat group, as shown below.

On Entering Tally Statistics



All entering objects will cause the 'Total_In_Server' tally statistic to generate an observation based on the current simulation run time Run.TimeNow and when the entity was created. This is because the Tally If is specified as 'No Condition'.

In this tally row, the Tally If is 'Custom Condition', so if the condition 'Is.Blue' is true, then a Time Between observation will be generated in the 'Blue_In_Server' tally.

As with the Tally step within the Processes window, any tallies that are generated with the Tally Statistics section of an object must have an associated [TallyStatistic element](#) defined within the Elements panel of the Definitions window. For example, in the above screen shot, the Tally Statistic Names of 'Blue_In_Server' and 'Total_In_Server' must be defined.

With either basic or transfer type node, the On Entered tally statistics are generated prior to the 'On Entered' add on process and crossing capacity (if specified) is allocated. The On Exited tally statistics are generated prior to releasing the crossing capacity (if specified) and Exited add on process.

Add-on Process Triggers

Add-on Processes are a special case of [processes](#). Add-on processes are typically defined by a user to help customize object behavior or add specific behavior into a model. The objects in the Standard Library contain Add On Process triggers. When an Add On Process is defined in the Add On Process trigger property of an object, when the object fires a particular event, the Add On Process is executed. The example below shows a trigger called "Created Entity". When the Source object creates an entity, an event is fired that this trigger picks up and if there is a process defined in the "Created Entity" Add On Process Trigger, it then executes the defined process.

How to Utilize the Add On Process Triggers of a Standard Object

Select the Object and the Process Trigger

- Select the object in the Facility Window and then expand the Add-On Process Triggers list. Click on the trigger that you would like to use and select Create New from the drop down list of that trigger.

Add Steps to the Process

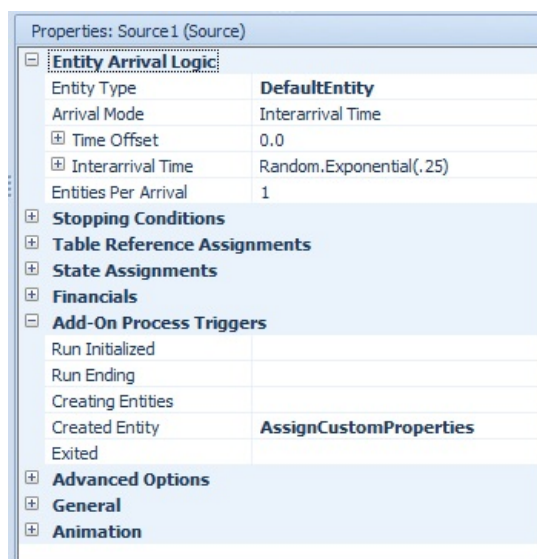
- Click on the [Processes](#) tab and the new Add-On Process will appear in the window.
- Click and drag a [Step](#) into the Process between the beginning and end step instances. The functionality of this Step can be customized by changing the properties in the Properties window in the lower right corner.

▲ Add-On Process Example

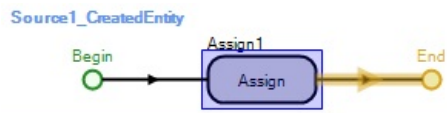
The Source object currently has five Add-On Process Triggers:

- **Run Initialized** occurs when the simulation run is initialized.
- **Run Ending** occurs when the simulation run is ending.
- **Creating Entities** occurs when a Source object is about to create an arrival of one or more entities.
- **Created Entity** occurs when an entity has been created by this Source object
- **Exited** occurs when an entity has departed the Source object.

The example below illustrates a Source object with an add-on process named AssignCustomProperties that is called just after each entity has been created.



These images are from the Process Window of the new Add-On Process, with an Assign Step that assigns a new State Variable to the Entity.



Properties: Assign1 (Assign Step Instance)	
Basic Logic	
State Variable Name	NewStateVariable
New Value	TimeCreated
Assignments (More)	0 Rows
Advanced Options	
General	
General	
The Assign step may be used to assign a new value to a state variable.	

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Reliability

Reliability

The Reliability section of Properties window is included in the following Standard Library objects: Server, Combiner, Separator, Workcenter, Resource, Vehicle and Conveyor. This section allows the user to define failure and repair time information for the object. By default, there are no failures when an object is initially placed in the Facility Window.

No Failures

- If the *Failure Type* is set to 'No Failures', then failures will not occur for the specific object. By default, there are no failures for an object when it is originally placed in the Facility Window.

Calendar Time Based Failures

- If the *Failure Type* is set to 'Calendar Time Based' failures, the user will specify both the *Uptime Between Failures*, as well as the *Time to Repair* properties. With this type of failure, the simulation clock is used to determine when the failure will begin.
- If the entity is being processed by the object at the time the failure occurs, the entity will stop processing (and its remaining processing time stored), and the failure begins for the repair time specified. Once the failure has been repaired, the entity will resume processing at the object.

Event Count Based Failures

- If the *Failure Type* is set to 'Event Count Based' failures, then an associated event, *Event Name*. *Uptime Between Failures*, must occur a number of times before the failure occurs. The *Count Between Failures* property determines how many times the given *Event Name* must be called before a failure will take place. This allows the user to associate something happening in the system, via an Event, to become the cause of the failure. The *Time to Repair* determines how long object is failed.

Processing Count Based and Usage Count Based Failures

- The *Failure Type* of 'Processing Count Based' is available for the Server, Combiner, Separator, and Workstation (Deprecated) objects, while 'Usage Count Based' may be specified for a Resource object. Both of these failure types work in a similar way.
- With a count based failure, the *Count Between Failures*, as well as the *Time to Repair* are specified. After the object has processed the number of entities determined by the count (which may be based on an integer value, expression or distribution), the failure will occur. Once the repair is completed, the processing or usage count will restart at zero.

Processing Time Based and Usage Time Based Failures

- The *Failure Type* of 'Processing Time Based' is available for the Server, Combiner, Separator, and Workstation (Deprecated) objects, while while 'Usage Time Based' may be specified for a Resource object. Both of these failure types work in a similar way.
- With a processing or usage time based failure, the *Uptime Between Failures*, as well as the *Time to Repair* are specified. After the object has processed a specified amount of time determined by the uptime (which may be based on an integer value, expression or distribution), the failure will occur. Once the repair is completed, the processing time duration will restart at zero.

Note

The standard library objects are defined so that if they go off-shift during a repair, the repair time continues during the off-shift period. If the user would like the repair time to stop during the off-shift period, the object should be subclassed and customized.

Examples

[Example 1 - Calendar Time Based Failures](#)

Reliability_Example 1

Properties: Server1 (Server)	
Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Processing Time	10
Units	Minutes
Buffer Capacity	
Reliability Logic	
Failure Type	Calendar Time Based
Uptime Between Failures	8
Units	Minutes
Time To Repair	2
Units	Minutes
State Assignments	
Secondary Resources	
Financials	
Add-On Process Triggers	
Advanced Options	
General	
Name	Server1

In this example, the Server may begin processing an entity at time 0.0 for 10 minutes. When the failure occurs at time 8 minutes, the entity processing stops and the failure repair time begins for 2 minutes. Once the failure has been repaired at 10 minutes, the remaining 2 minutes of the entity processing begins.

Example 2 - Event Count Based Failures

Reliability_Example 2

Properties: Server1 (Server)	
Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Processing Time	Random.Triangular(.1,.2,.3)
Buffer Capacity	
Reliability Logic	
Failure Type	Event Count Based
Event Name	Tool_Change
Count Between Failures	10
Time To Repair	Random.Uniform(3,5)
Units	Minutes
State Assignments	
Secondary Resources	
Financials	
Add-On Process Triggers	
Advanced Options	
General	
Name	Server1

In this example, the Server has event count based failures, based on the event called 'Tool_Change'. After this event has been called 10 times, the failure will occur and will take between 3 to 5 minutes, based on a uniform distribution, to be repaired.

Example 3 - Processing Count Based Failures

Reliability_Example 3

Properties: Server1 (Server)

Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Processing Time	10
Units	Minutes
Buffer Capacity	
Reliability Logic	
Failure Type	Processing Count Based
Count Between Failures	Random.Uniform(50,100)
Time To Repair	Random.Triangular(2,4,6)
Units	Minutes
State Assignments	
Secondary Resources	
Financials	
Add-On Process Triggers	
Advanced Options	
General	
Name	Server1

In this example, the Server has a processing count based failure based on a Uniform distribution between 50 and 100 entities. Therefore, once that number of entities has completed processing through the Server, the failure will occur for a repair time based on a triangular distribution with a mean of 2 minutes, mode of 4 minutes and maximum of 6 minutes.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

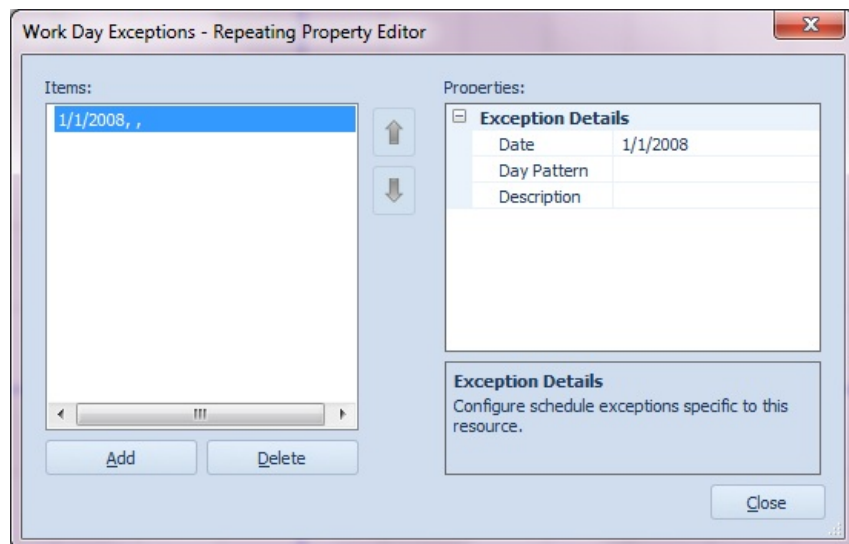
Work Day and Work Period Exceptions

Work Day Exceptions and Work Period Exceptions are available when a Work Schedule is used for the Server, Combiner, Separator, Resource, Worker, Vehicle and Workstation (Deprecated). Exceptions override the repeating work schedule for the duration of the exception.



Work Day Exceptions

A Work Day Exception may be used for a holiday or single day and is defined by a single calendar day and an alternative Day Pattern for that calendar day. Day Patterns are defined within the [Schedules](#) panel of the Data window.



Listed below are the properties of the **Work Day Exception**:

Property	Valid Entry	Description
Date	Valid Date and Time	Specifies the date for the day pattern that will be replaced by this exception's day pattern.
Day Pattern	Day Pattern Name	Specifies the Day Pattern to use for this exception.
Description	Expression	Specifies an optional description for this exception.

Work Period Exceptions

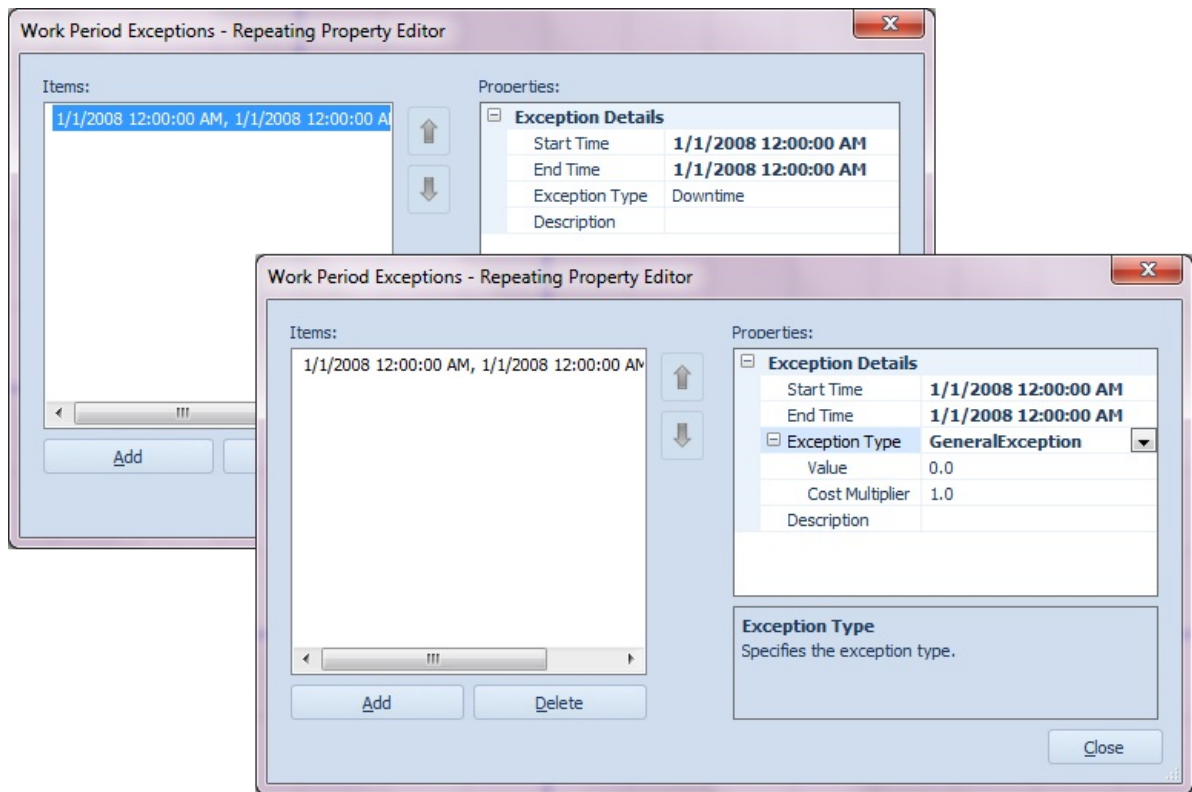
Work Period Exceptions may be used for overtime, planned maintenance, vacation periods, etc., and are defined by starting day/time and ending day/time.

Within the Work Period Exceptions, the *Exception Type* will determine whether the user can specify a capacity *Value* and/or *Cost Multiplier*.

Downtimes assume a capacity *Value* of '0' and a *Cost Multiplier* of '1' and these values are not shown to or editable by the user.

For GeneralExceptions, the capacity *Value* can be specified. It is important to note that the capacity *Value* for a Worker or Vehicle should be either '1' or '0'. See [Worker - Discussion and Examples](#) and [Vehicle - Discussion and Examples](#) for more information on schedules for these dynamic objects.

A *Cost Multiplier* property may also be specified for the GeneralException, which may be useful for incurring overtime costing.



Listed below are the properties of the **Work Period Exceptions**:

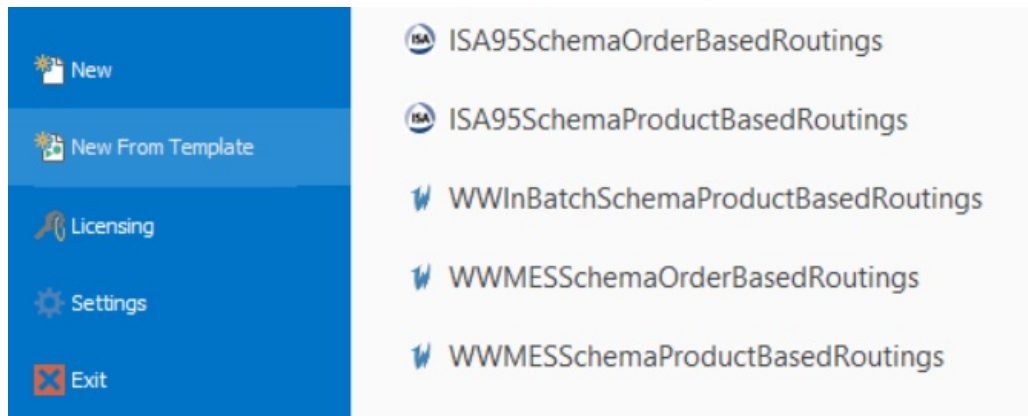
Property	Valid Entry	Description
Start Time	Valid Date and Time	Specifies the starting date and time for this exception.
End Time	Valid Date and Time	Specifies the ending date and time for this exception.
Exception Type	Downtime, GeneralException	Specifies the exception type.
Value	Real	Specifies the capacity value to be used for the duration of this exception.
Cost Multiplier	Real	Specifies the cost multiplier to be used for the duration of this exception.
Description	Expression	Specifies an optional description for this exception.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Project Templates

On the File tab, users may select New From Template, which will open an existing project that may include existing data tables and/or models/objects within the project. Clicking on the New From Template option opens a selection of templates that are pre-installed with Simio:



A user can create their own templates by just saving a project file, ideally with a *Name*, *Icon*, and *Description* set in the Project Properties, and saving it to the Project Templates location: C:\Program Files\Simio LLC\Simio\ProjectTemplates.

Scheduling and Wonderware MES Templates

The two **ISA95 Schema** templates will automatically add either seven (7) or nine(9) new data tables to the simulation model, including Resources, Routing Destinations, Materials, Material Lots, ManufacturingOrders, Routings, BillOfMaterials, WorkInProgress (Product Based only) and Manufacturing Orders Output (Product Based only). Within each of the tables, multiple columns have been added, including key columns to relate several of the tables. The tables have been added to be similar (although not identical) in structure to the SchedulingDiscretePartProduction example, available through the Support ribbon, Examples button.

In addition to automatically adding tables to the model, this template also generates three (3) string lists within the Definitions tab / Lists panel. This includes the RoutingType, MaterialColor and OrderStatus. MaterialColor list includes None, Red, Green and Blue while OrderStatus includes New and WIP. These string lists are referenced in the various tables listed above. A Changeover Matrix, named 'MaterialColorMatrix' as well as a ChangoverLogic element named 'SequenceDependentSetupMatrix' have also been originally configured within this template.

The **ISA95 Schema** templates will also add new Scheduling based objects to the Project, as seen in the Navigation window, each with a 'Sched' preceeding the original Standard Library object name. These objects are pre-filled with data for scheduling type examples.

The **Wonderware InBatch Schema** and **Wonderware MES Schema** templates adds the lists and tables into a Simio model that are used to integrate with Wonderware MES. These templates add MES based objects to the Project, as seen in the Navigation window, each with a 'MES' preceeding the original Standard Library object name. These objects are pre-filled with data for scheduling type examples. See the [Content Ribbon - RPS](#) page for more add-ins within Simio RPS Edition.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Creating New Objects

There are several ways to create new objects in Simio. Objects can be created from scratch by the user by selecting one of the Simio object classes or by using APIs. A more common and easier way to create a new object is to copy (or subclass) one of the Simio Standard Library objects and use the standard logic as a base when customizing the logic for your own needs. Models from other libraries can also be subclassed and used as a basis for creating a new object. Once a new object is created within the project, it can be added to models by dragging it from the Project Library into the Facility window of a model. The Project Library is located below the Standard Object Library on the left side of the interface.

It is worth noting here that Model and Object are used interchangeably throughout this document. This is because in Simio a model is actually an object that can be used inside of another model. And an object itself is a model because it has its own internal logic and there might be other objects inside of an object, which is why it can also be referred to as a model. Therefore, try to keep in mind that the Model and Object are one in the same.

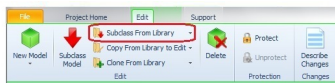
The main model within a project is usually a Fixed class object. The simplest way of adding logic to a model is by using the Standard Library objects in this new Fixed Class object. This is done by creating a new model and then placing objects from the Standard Library into the Facility window of the model. By placing a number of objects, such as Sources, Servers, Combiners, Separators and Sinks, and connecting them with various types of links, you can represent a wide variety of systems in a model.

An object definition has five primary components: properties, states, events, external view, and logic. There are three approaches to defining the model logic for an object. One approach is to subclass an existing object definition and then change/extend its behavior using processes. This approach is typically used when there is an existing object that has behavior similar to the desired object, and can be "tweaked" in terms of property names, descriptions, and behavior to meet the needs of the new object. Objects can also be subclassed with default data already specified. A second approach is to create your model hierarchically using a facility model. This approach can also be combined with the use of add-on processes to define custom behavior within the facility objects. This approach is typically used for building up higher-level facility components such as a work center comprised of two machines, a worker, and tooling. The third and most flexible approach is to create the object definition behavior from scratch using a process model. This is the approach that was used to create the Standard Library. All three approaches are described below.

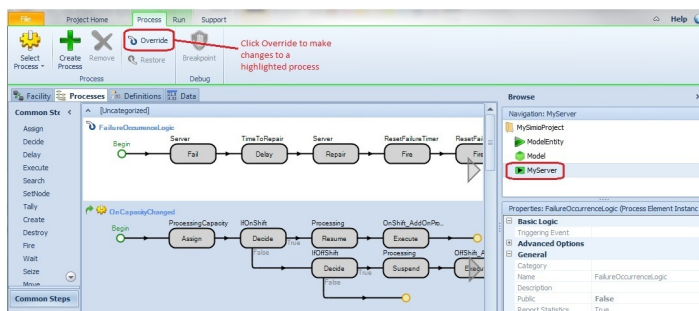
Creating an Object by Subclassing Another Object

Simio comes with a standard library of object models that are located in the **Standard Object Library** and their behavior can be enhanced with **Add-On processes**. If the desired behavior cannot be created by simply using Add On Processes, a user might need to customize the internal process logic. This is done by first subclassing an object and then overriding the processes. An object can be subclassed from within the **Project window**. Once an object is subclassed, a process must be Overridden, by selecting the Override icon in the ribbon, before any changes can be made.

Subclassing a Standard Library Server from the Project Window

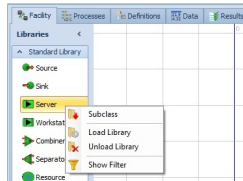


From the Project window, select which model is to be subclassed



Models can also be subclassed from the Facility window by right clicking on the object in the library window, located on the left side of the interface.

Subclassing from the Facility Window Libraries

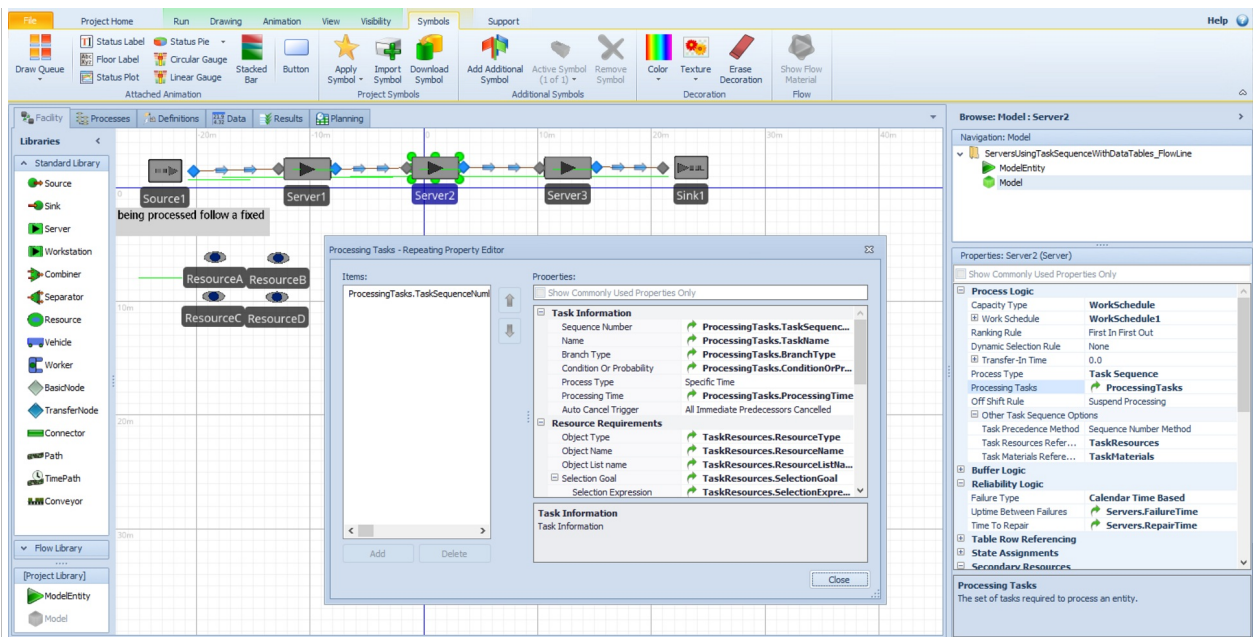


Right Click on Server

Creating an Object by Subclassing Another Object Including Default Data Using "Create Object From This" Option

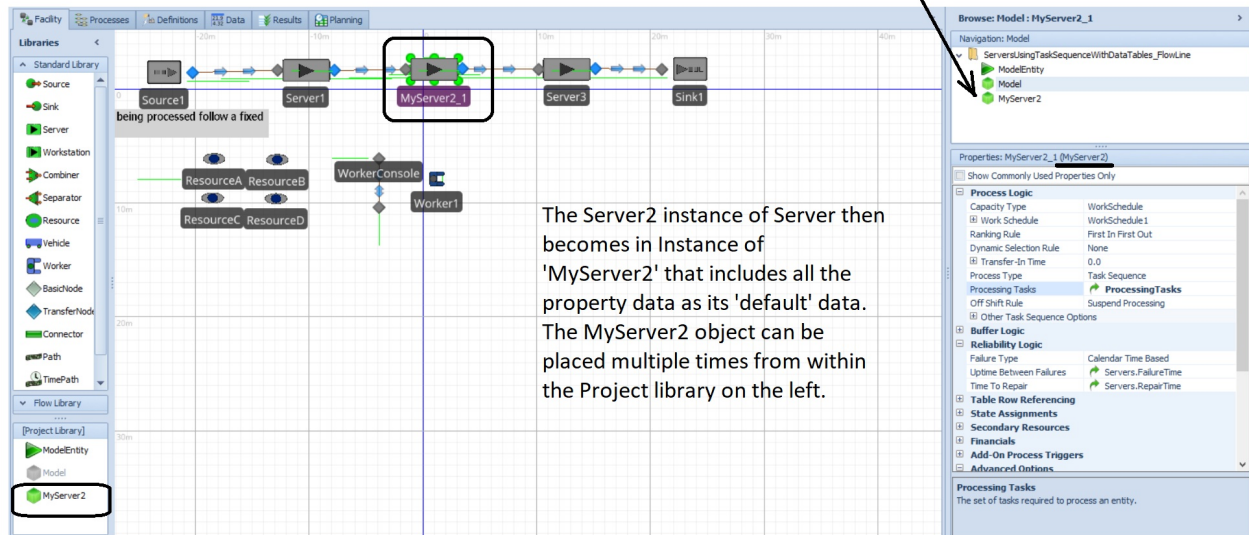
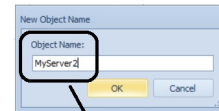
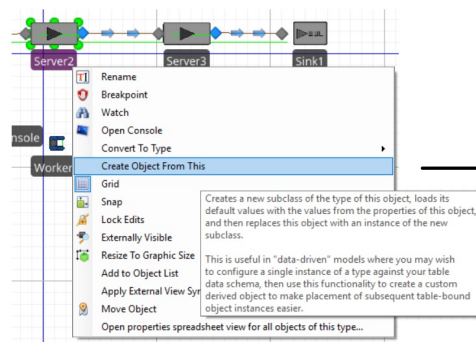
In addition to using the above methods to create a subclass object, objects include the right click option to Create Object From This object instance. When an object, such as a Server, for example, has already been placed within the Facility window and property data for that object has been changed from the original defaults, the user may wish to right click and select the Create Object From This option. This will create a subclass of the same type of object (Server in this example), load the subclass object with default values from the values of the properties of the existing object and replace the object with an instance of the new subclass. For example, the user may have an existing Server, named Server2, with a specified work schedule, multiple processing tasks including workers and materials, and reliability data.

Object within the Facility Window



If the right click option to Create Object From This is used, a subclass Server with its default data coming from the Server2 instance will be created within the project. The Server2 will then also become an instance of that subclass MyServer2 (and no longer a standard Server). Additional subclassed object instances may be placed in the same model by using the Project library.

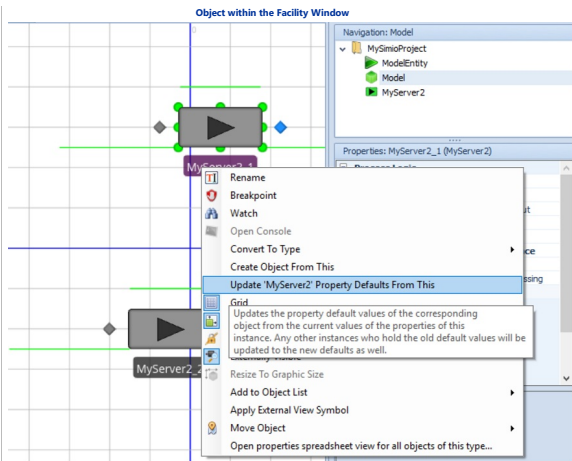
Create Object From This



This option for subclassing an object is very useful in "data-driven" models where the user wishes to configure a single instance of a type of object using table data. This functionality can be used then to create a custom derived object to make placement of table-bound object instances easier. See the Table-Based Objects (Auto-Created) page for more information. Subclass objects, such as MyServer2, will be shown in a table column list when the column is an Object Reference 'Object Type'. Thus, these subclass objects, including data, can be auto-created.

Modifying Data within a Subclassed Object Using "Update ** Object Property Defaults From This" Option

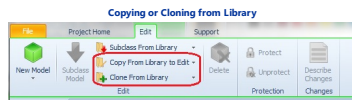
Once an object has been subclassed, either through right clicking on a library object to Subclass (object) or by using the Create Object From This (object and default data), the default property data can be automatically updated. By right clicking on a subclassed object, the option to Update ** Object Property Defaults From This will update the property information on particular object and all other similar object instances within the Facility window. For example, using MyServer2 discussed above, the user could change property data within the processing tasks and reliability areas for a placed MyServer2 instance. Right clicking that MyServer2 instance in the Facility window will give the option to Update ** Object Property Defaults From This. The property values will then be updated within the MyServer2 object, as well as all of the instances of that object in the Facility window (not just the one that was edited).



Creating an Object by Copying or Cloning an Object from the Library

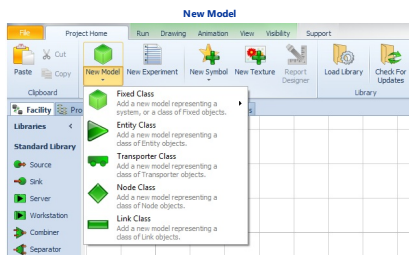
Selecting the *Copy From Library to Edit* button allows a user to choose a model from any attached library and copy it into their project, so it can then be modified. You would use this option when you want to create a new object and want to use an existing object as a starting point. The difference between this option and subclassing from the library is that when a change is made to the original object's definition, an object that was copied is not updated with the new changes in the original object's definition. Whereas a subclassed object will inherit the changes to all processes that were not "overridden".

Select the *Clone From Library* button allows a user to choose a model from any attached library and copy it into their project. This should be used when the user wants to re-arrange where models live in different libraries. If the user would like to modify the objects, they should consider *Copy from Library to Edit* or *Subclass From Library*.



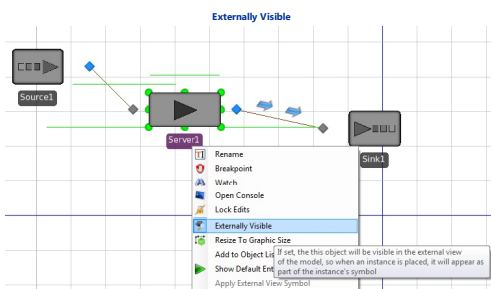
Creating an Object from an Simio Object Class

A new model is added to the project either by clicking on the New Model icon on the Project Home ribbon tab or by clicking the New Model icon from within the *Project window*. By clicking the top portion of this icon, the new model is a Fixed object class. Clicking on the bottom section of the New Model icon presents a choice of the type of object to create: Fixed, Entity, Transporter, Node or Link. The user can then add logic to their new model by adding objects in the *Facility window*, adding processes within the *Processes window*, and adding Elements, States, Properties, Events and Lists from the *Definitions window*. The external image of this new object is defined in the External panel, found in the *Definitions window*.



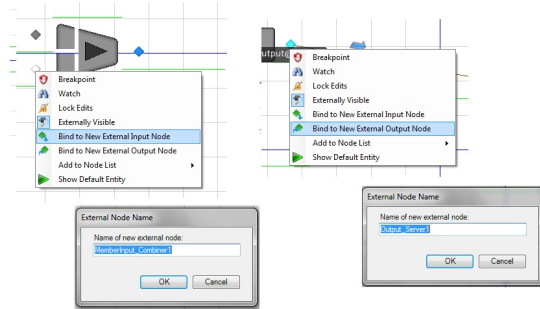
New Object Logic within the Facility Window

One approach to creating a new object's logic is to create your model using hierarchy in a Facility model. This approach can also be combined with the use of add-on processes to define custom behavior within the Facility objects. This approach is typically used for building up higher level facility components such as a work center comprised of two machines, a worker, and tooling. When the logic of the model is specified within the Facility window, you also have the option of showing the graphical representation from the Facility window in the new object. By default, each object placed in the Facility window has its 'Externally Visible' option on. This option can be viewed and changed by simply right-clicking on an object (or group of objects) in the Facility window and selecting 'Externally Visible', as seen below.



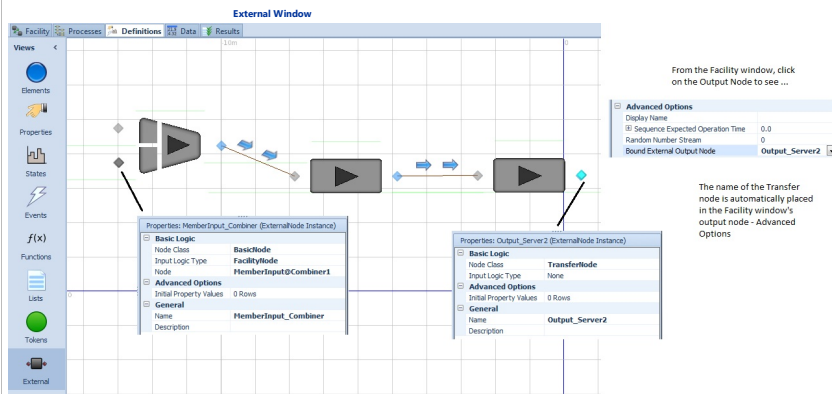
The input and output nodes for the object whose logic is in the Facility window can also be automatically added to the External view. This is done by right-clicking on the node and selecting either the 'Bind to New External Input Node' or 'Bind to New External Output Node' options, as shown below.

Binding to External Nodes



Within the Definitions tab, External window, new nodes are placed that will be used to connect into and out of the newly defined model logic in the Facility window. For external input nodes, the *Node Class Name* is specified as 'BasicNode'. *Input Location Type* as 'Node' and the *Node Name* is automatically generated. For external output nodes, the *Node Class Name* is 'TransferNode' and the *Input Location Type* is 'None'. It is important to note that with the Transfer Node that is generated by using 'Bind to External Output Node', the node name is placed in the Facility window's node Advanced Options area under the property *Bound External Output Node*. This can be seen in the example below.

IMPORTANT NOTE: An external node cannot be used as both external output node and external input node.

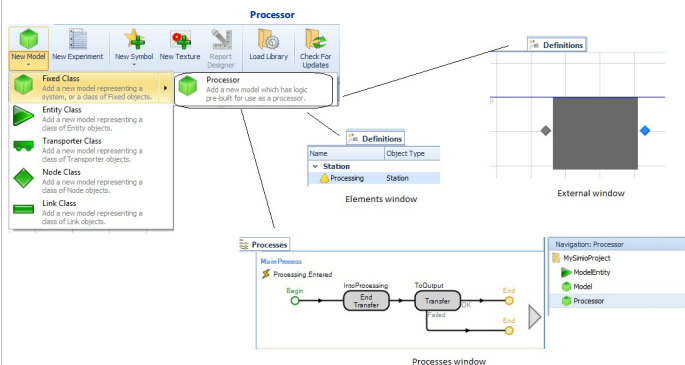


For an example of using building a new object with logic in the Facility window, refer to the SimBit [FacilityModelWithinModel](#).

New Object Logic within the Processes Window

The most flexible approach is to create the object definition behavior from scratch using a process model. This is the approach that was used to create the Standard Library.

Within Simio, we have provided a basis for new object logic within the Processes window, called a Processor. By selecting **New Model > Fixed Object > Processor** from the Project Home tab, a new Fixed model is added, and it includes a Station and Input and Output Nodes. It also includes a process for processing entities, as shown below. Additional logic would be added between the EndTransfer and Transfer steps within the Processes window.



For an example of using building a new object with logic in the Processes window, refer to the SimBit [ProcessModelWithinModel](#).

Objects created by Custom Code

Simio objects are created from Simio Processes, Steps and Elements. A user can create their own custom Steps and Elements from any .NET language and then use these to create their own Simio objects. User created Steps, Element, Add-Ins, or Dynamic Selection Rules should be placed in the (Simio Binary Directory)\UserExtensions\ directory. See the [Custom Simio Extensions](#) page and [C:\Program Files\Simio LLC\Simio\Simio API Reference Guide.chm](#) for additional information on creating custom logic with code.

Anatomy of an Object

Object Types define core functionality. Object Types include fixed, link, node, agent, entity (agent) and Transporter (entity). An object type defines a mode of behavior (e.g. provides a pathway between two nodes). An object model defines the actual behavior of the object within its class (e.g. accumulating conveyor). You, as a user, can define a wide range of behaviors within each object class.

Properties (static) pass values into the model logic for the object. The properties for the object are shown in the Properties window, located on the lower right side of the GUI. Standard Library Objects have a fixed number of properties that are important for the behavior of that object. Sub-classed objects can include the original object properties and also any user-defined properties. All of the properties for a given object can be found in the **Properties** panel in the Definitions window.

States (dynamic) change over time. States can be specified as Discrete, Level or Movement. The state names for a given model can be seen in the **States** panel of the Definitions window. Additional states can be added in this panel as well. States can be referenced during the simulation run as they dynamically change over time.

Events fire and trigger actions. Events can be added or changed within the **Events** panel of the Definitions window of an object.

External views define the object's graphical appearance within a Facility window and entry and exit points for the object. When defining a new model object or modifying a subclassed object, the external view is used to place the external nodes (entry / exit points), as well as the symbol of the object and graphical queues. The External view is defined in the **External** panel within the Definitions window. By default all objects placed in the Facility window will be automatically shown in the External view. The right-click menu when an object is selected allows the user to turn off the 'Externally Visible' option on an object by object basis.

Consoles depict the dynamic state over time. Status labels, plots and buttons can be placed in an object's console to graphically view the simulation progress and states throughout the simulation run.

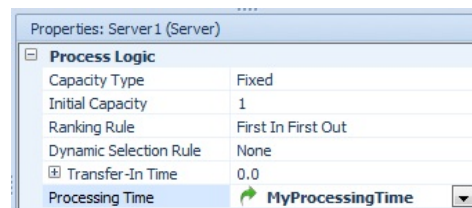
Using A Custom Object In A Model

After you've created a custom object, by either subclassing it from another object, copying it from another object or creating it from scratch, you need to ensure that it is ready to be used inside of model (or another object). You should address the following things before using the object:

Properties

The object builder should decide what type of information the object (or model) should receive from the "outside". In other words, what type of information will the object need to get from the model that it is placed into. Or, perhaps you want to allow the modeler to be able to assign the value of Processing Time or Capacity, for example. If so, [properties](#) need to be created for these parameters so that when this object is placed into another model, the user can assign a different value to that property for each instance of this object that is placed.

Creating a Property So Users Can Input Data into Submodel



Right click on the Processing Time property and select 'Set Referenced Property' and then 'Create New Referenced Property'. A user can now send a value into this new MyProcessingTime property.

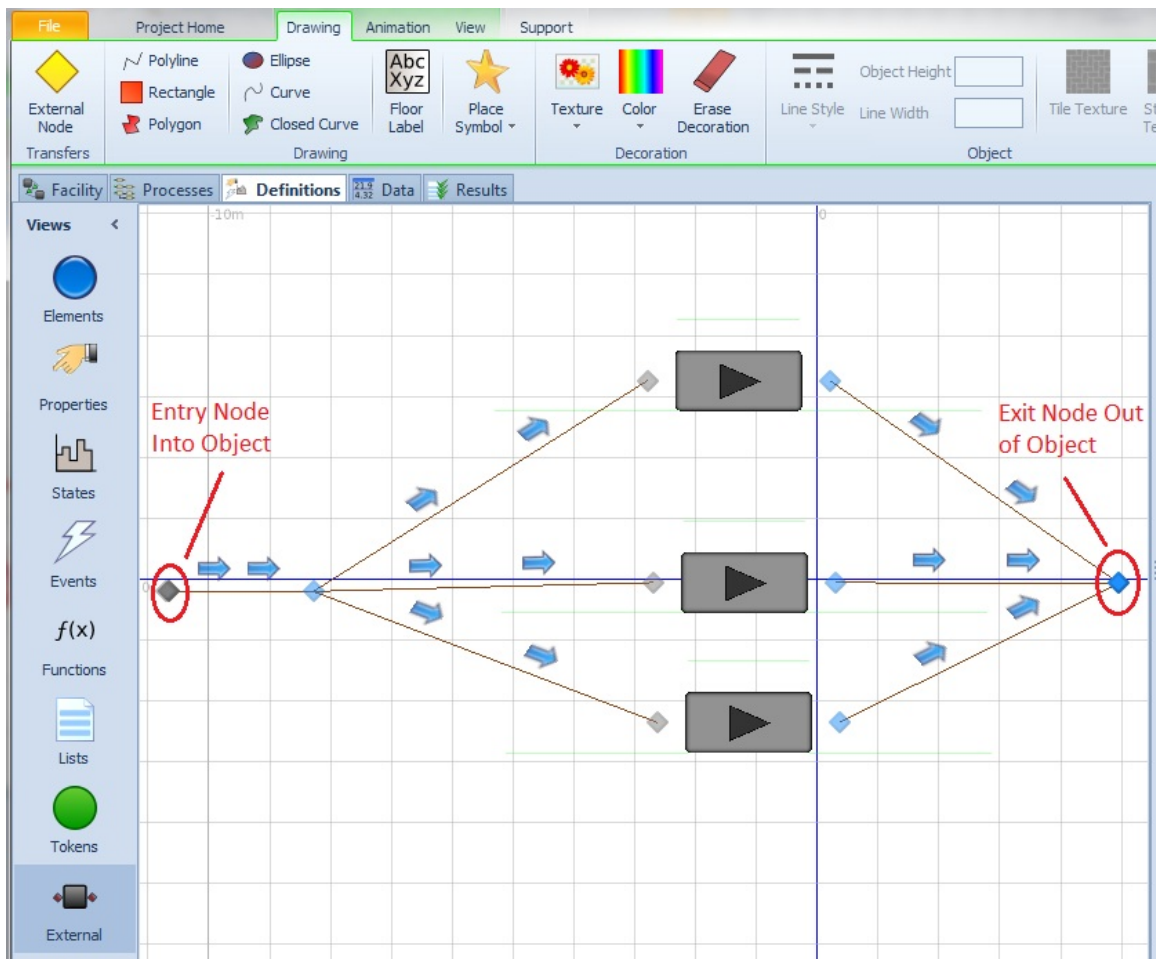
External View

The object builder should decide what this object (or model) will look like when it is placed into a model. This is controlled in the [External View](#) under the Definitions window. If the object is built with other objects, the objects will all be visible in the External View by default. To set an object to not be displayed in the external view, right click on that object in the Facility window and unselect Externally Visible. If the object is built with processes only, the object builder will need to create an External View manually and must place External Input and Output Nodes. See the Entry and Exit Points section below for additional information on this.

Entry and Exit Points

If entities or other types of objects will be entering this object (or model), the object builder should ensure that there is at least one entry point and exit point defined. If this object is built with other objects, the entry and exit points are usually nodes. A node can be set up to be an entry point by right clicking on that node in the Facility window and selecting 'Bind To External Input Node' and this will automatically create a new External Input Node where entities can enter into this object. An exit node can be defined similarly by right clicking on an existing node in the Facility window and selecting 'Bind To External Output Node'. If this object is built with only processes and no objects, the object builder will need to create External Input and Output Nodes in the External View. This is done by clicking on the External Node icon in the ribbon and placing two nodes into the External View. Make one of these nodes the entry point into the object by setting the *Node Class Name* property to 'BasicNode' and the *Input Location Type* property to 'Station'. Next, select the Station where the entities should go when they enter into this object. Make the other node the exit point out of the object by setting the *Node Class Name* property to 'TransferNode' and the *Input Location Type* to the default of 'None'.

Entry and Exit Points for Submodel



Data Tables

Starting in Simio 261, a top-level object can reference a sub-level object's Data Table. Data Tables can remain in a custom object's definition and still be accessed outside of its definition. This only works in the object hierarchy where the object above can reference information from an object below. For example, the top-level Model can reference Model's instance of SubModel and get SubModel's table information. Similar to how the top-level model could check a SubModel's State Variable or other definition information. See the [Data Tables](#) Help page for more information on how to use these table references.

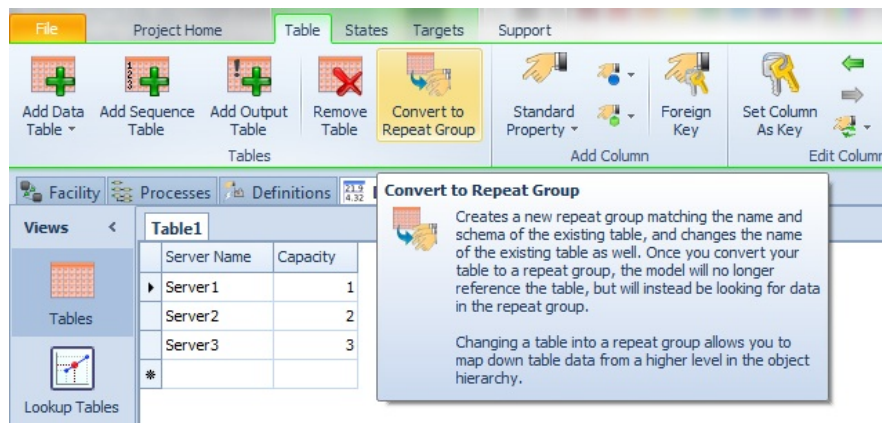
The Data Tables in the custom object are limited to information in this object definition. It cannot get information about anything else above or parallel to the object in the hierarchy, like the top model where this object might be used or other objects that live in the top-model. If the Data Table needs to reference other objects in the top-level model, like Secondary Resources used for Processing, consider moving the Data Table to the top level, and passing the information back into the custom object.

If the Data Table will need to interact with or contain information about the top-level Model, it is common practice to move this object's custom Data Table from this Object into the top Model and pass the information from the top Model back into the definition via a Repeat Group. The Repeat Group Properties will hold the spaces in the custom object's definition until the custom Object is instantiated into a Model and that instance is fed information from the top-level back to its definition logic.

[Repeat Group](#) properties will have the exact same structure as the data tables and will contain the same data. All the references in the model will then reference the new Repeat Group properties instead of the Data Tables. This will allow the user to be able to see the data and the Tables and manipulate the data or bind the tables to an outside source. By creating Repeat Group properties on the submodel and referencing these instead of the Data Tables, this allows the user to pass data down into the submodel from the top level model. (Remember that information can only be passed into an object through properties).

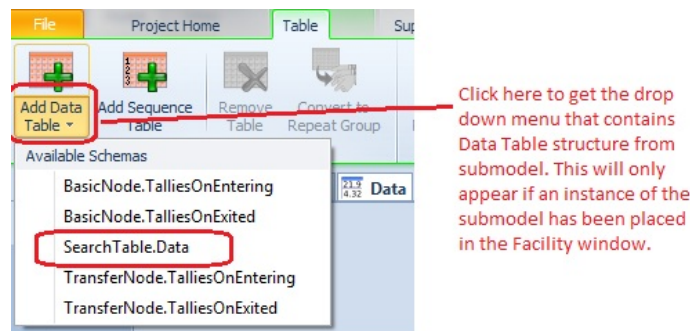
Before this object/model is placed inside another model, the user should export the data from each table to a .csv file. Next, the user should click the 'Convert to Repeat Group' icon in the ribbon to convert this table to a Repeat Group property. By clicking this, Simio automatically creates a new Repeat Group property with the same structure as the table. It also changes all the references in the model to now reference this Repeat Group, instead of the Table. And it renames the table so the _original is appended to the name. This model is no longer looking at this table for data, but instead at the Repeat Group property.

Convert To Repeat Group Icon



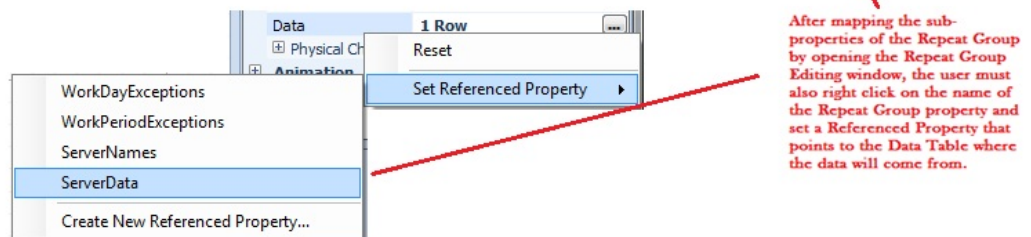
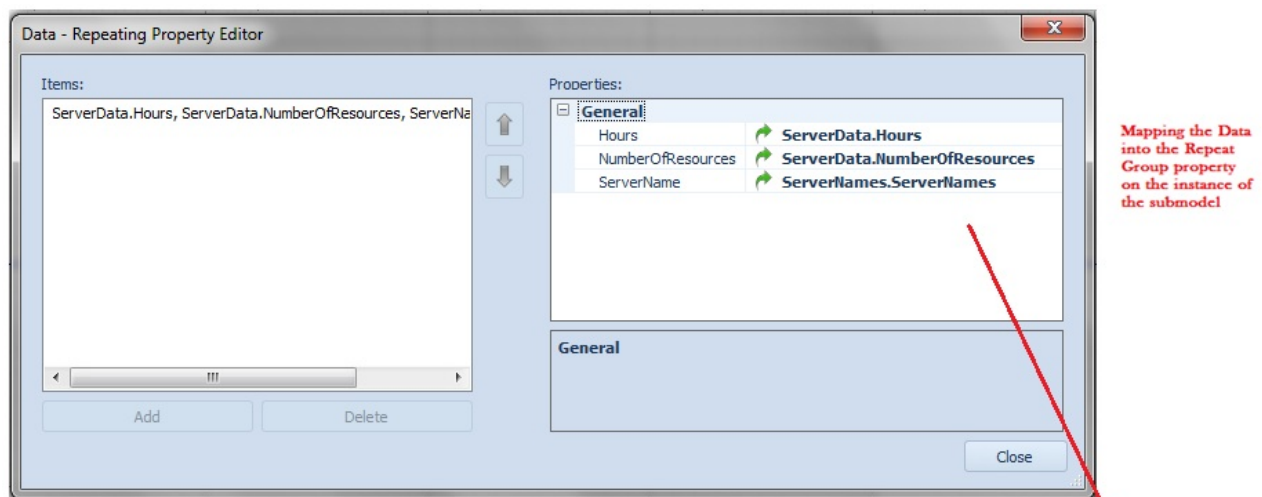
The user will go to the new project (the top level model) and load the custom object into this new project so that it appears in the [Project Library](#). The user will place an instance of the object into the Facility window of the model. The user will then create new Data Tables in this model. To assist in creating a Table with the same schema/structure as the table in the submodel, Simio gives the users the option to select available Schemas from the drop down available with the Add Data Table icon. Select the schema of the Repeat Group from the submodel that was just placed into the model. A new Data table has been created and now the data can be imported from the .csv file. Import the data by clicking the Import icon.

Uses Table Schema from SubModel



Next, the user will tell the Repeat Group property on the instance of the submodel to look at this new Data Table for data. Click on the instance of the submodel that was placed in the Facility window and find the new Repeat Group property. Enter a reference to the appropriate row in the Data Table so that it maps the data from the Table into the Repeat Group. An important step that is often missed is that the Repeat Group itself must be set to reference the Data Table, like the example below.

Map the Data Table into the Repeat Group property



This will send the data from the Data Table down into the submodel's repeat group property so that the submodel can use this information in its logic.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

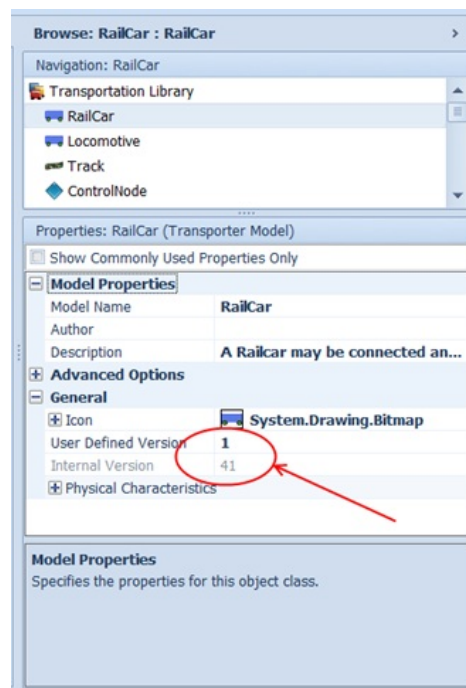
Versioning of Simio Objects

Versioning of the objects is a 2 part process; editing the version numbers as well as the description of the changes. Note that these features are available in Simio 6.100 or later.

Version Number

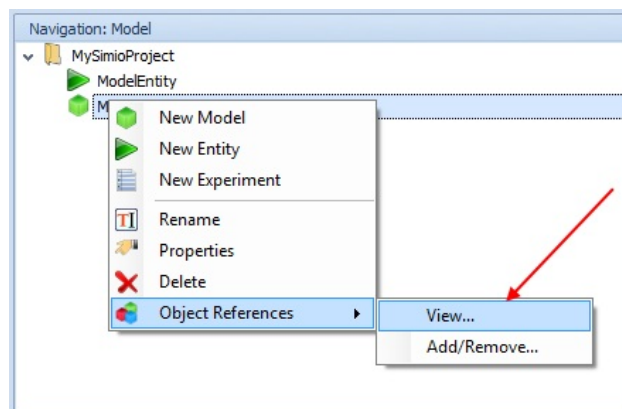
Versioning in Simio is done at the object level; libraries are not versioned. There are 2 version numbers used for a Simio object – a *User Defined Version* and an *Internal Version*. The User Defined Version is editable by the Object developer. The Internal version is not.

To modify the User Defined Version of an object, bring up that object's properties window. The *User Defined Version* (which is editable) is shown, in addition to the un-editable *Internal Version* number. The User Defined Version can be any integer or an integer combination separated by a period (ex: 21.4).

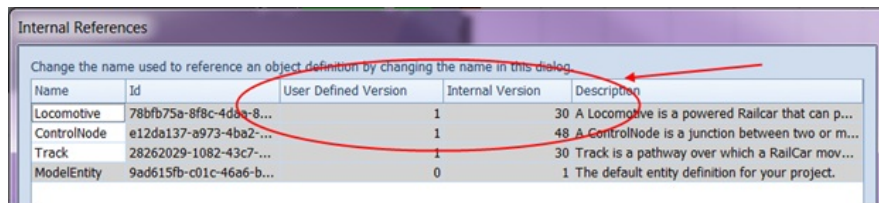


The *Internal Version* of a Simio object is automatically incremented each time a change is made to the object.

To view the existing version of an object, with an object placed in the Facility window, right click on the model and select Object References / View.



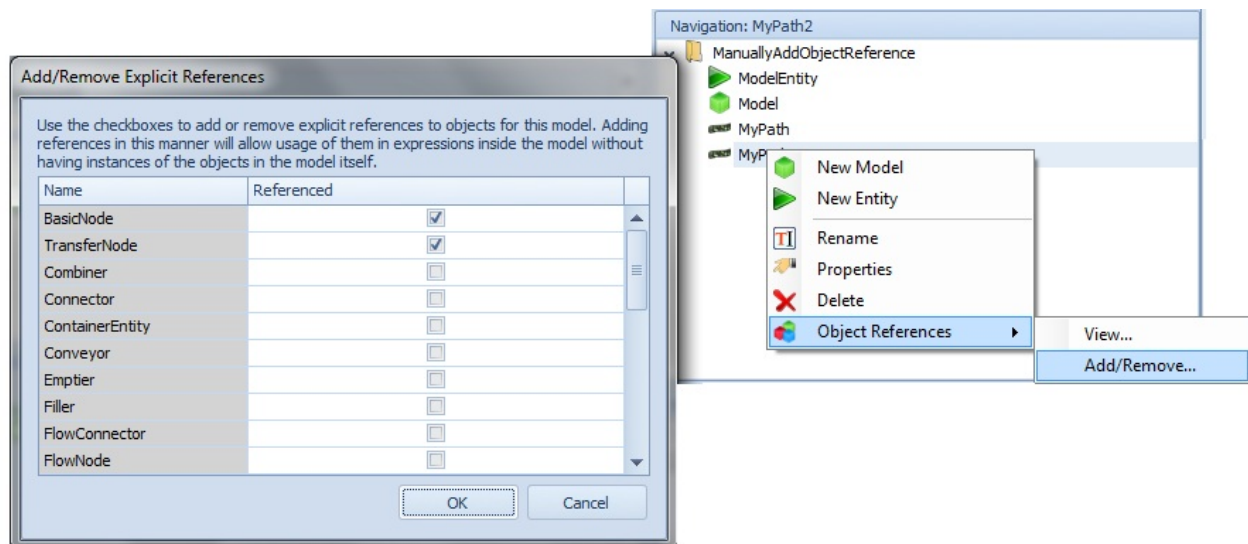
Below is an example of the Internal References window that will appear, displaying the *User Defined Version* and the *Internal Version* number.



Advanced Tip: In some unusual situations, an object in your project may behave erratically because it has gotten out of sync with the library it came from. In these cases it may help to use Shift-Click to enter this feature in edit mode. This will allow you to reset the *Internal Version* of the object to 0 and reload a fresh definition of the object. To edit the existing version of an object, with the object placed in the Facility window, right-click on Model and the shift-click on Object References. This will allow you to edit the *Internal Version*.

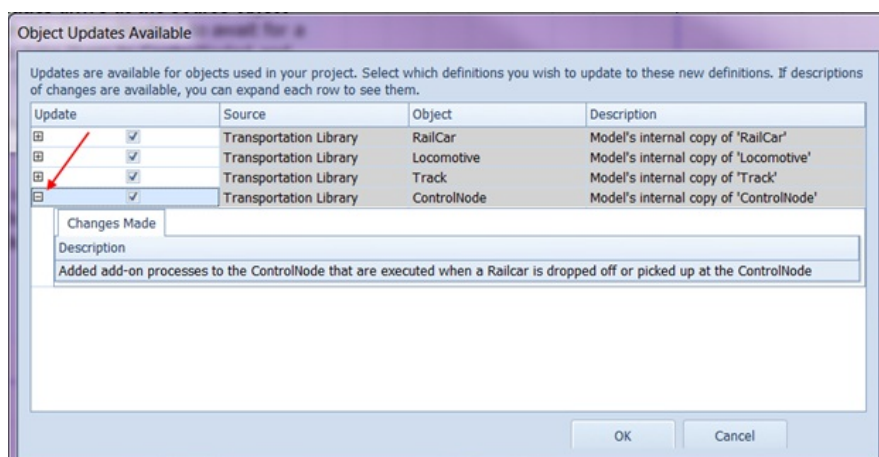
Add / Remove Explicit References

In addition to the above described internal references of a given model, the right click menu on a model also allows the option for Object Reference / Add/Remove references. The Add/Remove option will open a dialog to allow the explicit reference to another object definition, without having the instantiate an object of that type inside the model. This is particularly useful if, for example, the user is creating a custom link with StartingNode and EndingNode are going to be of type BasicNode or TransferNode. That model may then have an expression, such as 'StartingNode.BasicNode.CurrentTravelerCapacity', for example, because the BasicNode object is explicitly referenced. Note that this is only needed if you are pulling in references from other projects/libraries. Simio looks at sibling objects in the same project when resolving name and auto-referencing.

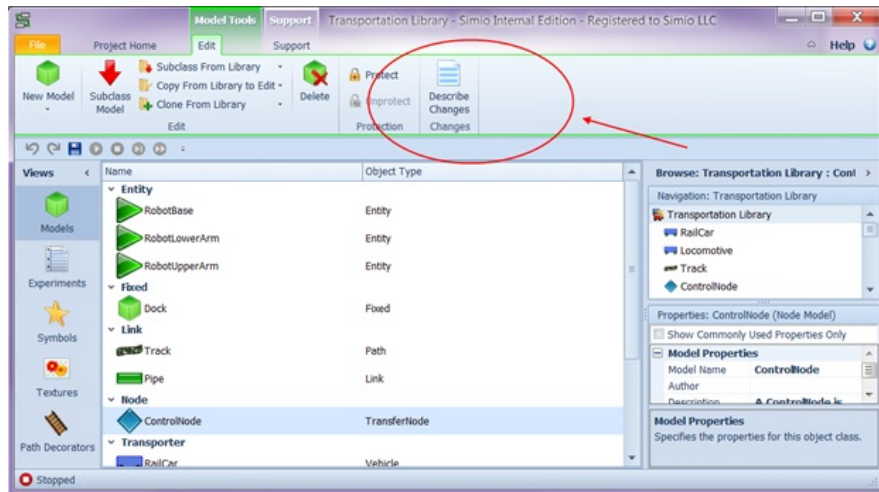


Documenting Object Changes

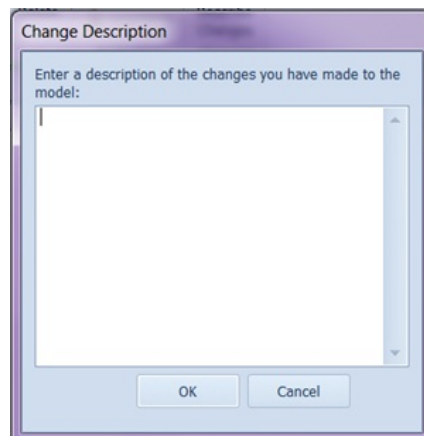
When a project that uses objects that have changed is loaded, an Object Updates Available dialog will appear. By clicking on the plus sign next to the Source library and Object name, a description of the changes made will appear.



However, in order for a user to know that what changes have been made, these changes have to be added to the object. This is done via the Describe Changes button on the Edit ribbon of the library.



Clicking on that option will bring up the Change Description dialog shown below. Note that at this time it is not possible to tie in the version number with the changes associated with that version.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Animation

Animation

In Simio, modeling and animation are done as a single step. You can layout your model with realistic spatial relations that accurately mimic your real life system. The [Facility window](#) is a 3-D drawing space in which you place objects that represent the physical components of your system. The appearance of these objects can be easily changed by altering the color of the default objects or replacing the objects with a symbol from the large library of default symbols provided by Simio. Simio provides [symbols](#) in the following categories; buildings, equipment, medical, restaurant, people, vehicles, trucks and other. But Simio does not limit the user to the [symbols](#) provided. The user can import their own images or download from the Trimble 3D warehouse or Autocad. This feature allows you to incorporate symbols that provide a more realistic feel to your model.

Simio allows the user to add [static symbols, labels and imported screen shots](#) into the model as well. The user can easily add logic to the model to [change the appearance](#) of objects and entities as they change states throughout the run. The ability to [change the view](#) and see the model from a 3D perspective provides a excellent tool for presenting the real life view of the system. See the [example models](#) to see Simio animation in use.

Status Labels, Status Plots, Status Pies, Floor Labels, Circular Gauges, Linear Gauges, Stacked Bars and Buttons can be placed in the Facility Window. They can also be attached to objects in the Facility Window. If they are attached to Entities or Transporters, they will travel with those dynamic instances. They can be placed in the External and Console views as well. To see specific information about this features, see the [Console](#) help page.

A user can create Named Views which is a saved view of the current position and orientation of the camera. These named views can then be combined into a series of camera sequences, which can then be played back by the user. The user can also record an .avi video of the animation of a model by using the Record Run Animation button. Named Views, camera sequences and .avi recording can all be found on the [View ribbon](#).

Object's attached Nodes, Queues, and Labels can respond to orientation changes of the parent object. By default, this is on, but it can be toggled on and off by right clicking on the attached object and selecting either Parent Rotation Changes Location or Parent Rotation Changes Orientation.

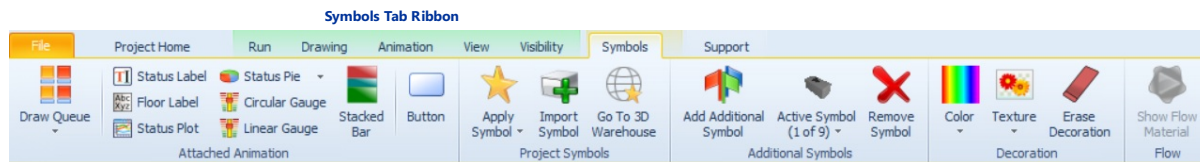
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Symbols

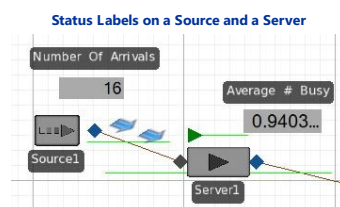
Symbols

A user can add animation components, such as attached queues and status labels, as well as download, import and create new symbols from the Symbols tab ribbon. Symbols can be added as static objects within the Facility window or they can replace the standard Simio symbol of an object. For example, to change the symbol of the Default Entity from a green triangular to a custom symbol, simply select the entity object and then either select a symbol from the Simio symbol library, import a symbol from an existing file or go to the 3D Warehouse to download a symbol for importing.

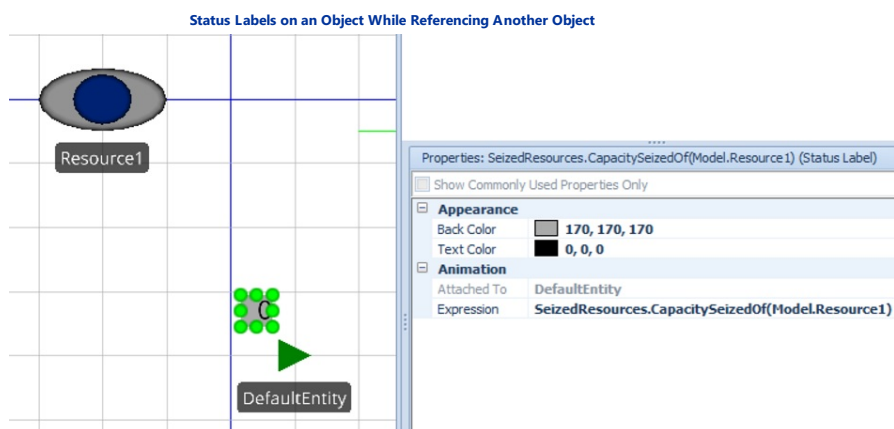


Adding Animation to a Dynamic Object

When a dynamic object, such as an Entity, Source, or Server, is selected in the Facility Window, the user has the ability to add additional animation symbols, such as an attached queue state or status label to that object. To attach a status label or queue to an object, simply highlight the object, then select the status label or queue button, and place it within the Facility Window. The queue state or expression for the status label should involve the instance which you are attaching it to. For example a label with expression 'Server1.Capacity.Allocated' on a Server named 'Server1' would animate the number of busy units of that given Server. When the dynamic object is moved, the attached queue and/or status label moves with it.



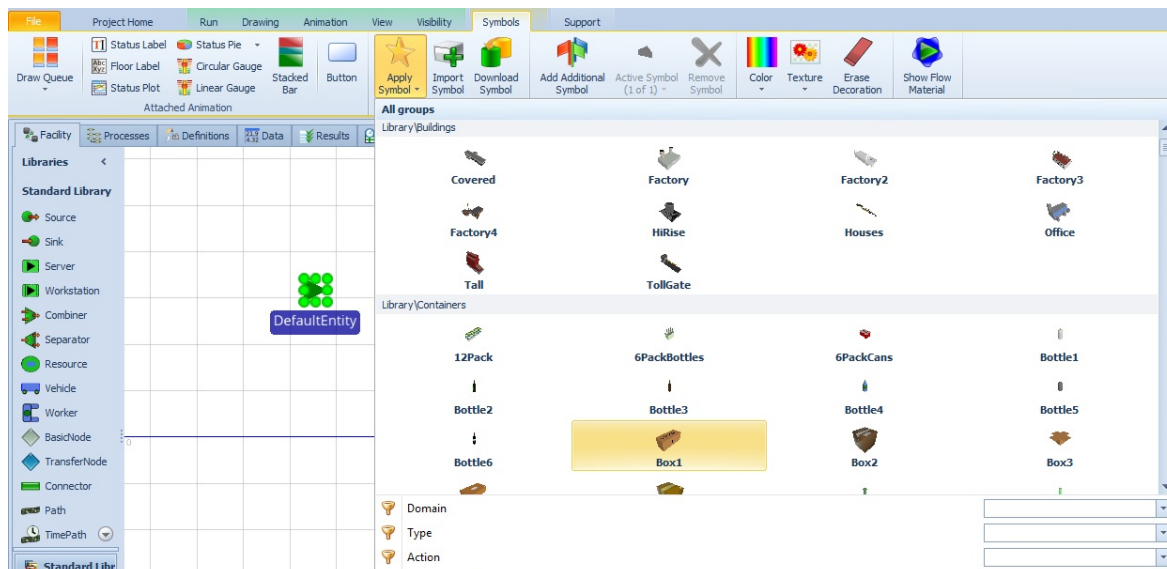
There is also a special case when you use an attached Status Label on a Dynamic Object. If the Status Label is attached to an object, but is referencing another object, the object and its parent must be specified. For example, to reference a resource in a function while the Status Label is attached to an entity (or other object), the resource should be referenced as 'Model.ResourceName' not just 'ResourceName'. See the below example screen shot. 'ModelEntity.SeizedResources.CapacitySeizedOf(Resource)' is the original expression if used in a Decide step, for example. However, because the Status Label is attached to the ModelEntity, the name of the resource must be specified as Model.ResourceName so the attached label looks like 'SeizedResources.CapacitySeizedOf(Model.ResourceName)'.



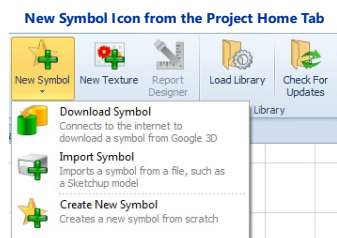
Applying a Symbol to an Existing Object and Symbol Filtering

All Simio library objects contain a default symbol that may be changed by using the Apply Symbol option on the Symbols ribbon. Highlight the symbol/object that you'd like to change and then select from among the many symbols within Simio's symbol library.

To help select from large number of symbols available, filtering may be used. Within the Symbol ribbon, when applying a symbol for an object, filtering options will help quickly get to the symbols that are of interest. This includes filtering by Domain, Type and Action. **Domain** filtering currently includes Manufacturing, Medical and Restaurant symbols. **Type** filtering includes various types of symbols such as buildings, containers, equipment, furniture, people, and trucks. **Action** filtering will distinguish between animated and static symbols.



Downloading, Importing and Creating New Symbols

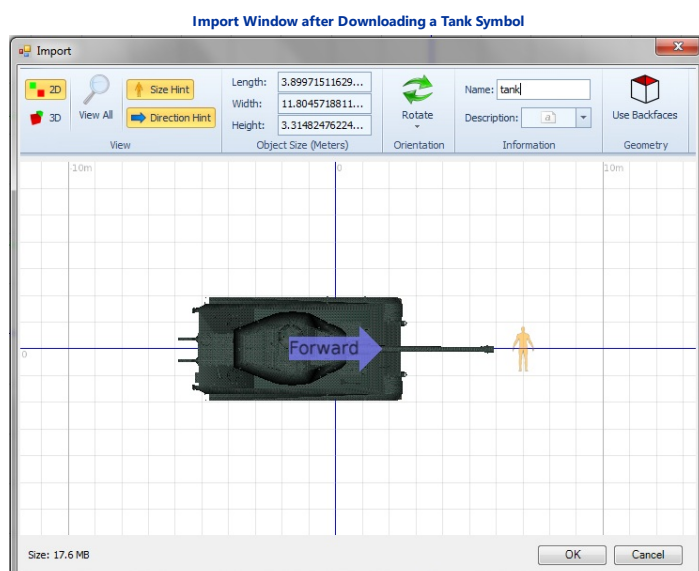


Downloading

A symbol can be **downloaded** from the Trimble 3D Warehouse. This is a site on the web that contains hundreds of thousands of 3D images that you can incorporate into your model. To download a symbol, go to the Project Home tab in the ribbon menu and click on the New Symbol icon. Select the option for Download Symbol and you will be connected to the Trimble 3D Warehouse. You must have an internet connection to access this website.

The 32 bit install of Simio has the 32 bit version of the sketchup importer, and the last version their 32 bit components support is Sketchup 2016. For all SketchUp releases after that, Sketchup used 64 bit components. In 32 bit Simio, you may have difficulty importing Sketchup files.

After the symbol has been selected from the Trimble 3D Warehouse and the user downloads it, Simio will display the new symbol in an Import pop up window.



The image is first shown in 2D but can also be viewed in 3D by clicking on the 3D icon in the top left corner of the import window. The large **blue arrow** on the image shows the user which direction Simio will consider to be the front of this image. If the user wishes to rotate the symbol, the Rotate icon provides options for rotating at 90, 10 or 1 degrees clockwise or counterclockwise.

The image of a person is displayed next to the symbol to provide a hint regarding the size of this symbol as compared with the size of a human in the Simio application. The user can resize the symbol (in terms of meters) with the length, width and height boxes in the Object Size panel.

In addition to displaying the downloaded image in Simio, the Sketchup file is also stored in the \\My Documents\\My Simio3D Warehouse Downloads directory of your computer. Trimble SketchUp (free download) can be used to customize an image (e.g. adding or removing components or changing colors).

Importing

A symbol can be **imported** from a file, such as a symbol created in Trimble SketchUp or AutoCAD (*.dxf and *.dwg). AutoCAD is a very popular standard supported by many products that features both 2D and 3D drawings. (Note: Not all products follow the standard closely so results may vary on *.dxf/*.dwg files generated from programs other than AutoCAD.) Symbols can be imported from a .jpg file and the symbol will become a 2D image. Simio supports importing from a number of 3D formats, which can be viewed by selecting the All Items button when performing the import for opening a particular type file.

To import a file, go to the Project Home tab in the ribbon menu and click on the New Symbol icon. Select the option for Import Symbol and find the file to import. The symbol will be displayed in the Import Window and can be resized and rotated similar to a downloaded symbol discussed above. Or if you'd like to import a symbol for an object, select the object and then click the Import Symbol icon in the Symbols ribbon menu. For more information on importing animation symbols (e.g. people that walk, run, jump, etc.), refer to the [Animating Moving Objects \(Walking People\)](#) page.

With regards to importing 2d and 3d symbols, *.png format is the only 2d image format imported that has translucency/alpha values in it. Please note that translucency in 3d files is different than 2d files. Importing files that include transparency depends upon whichever materials are defined in the 3d model.

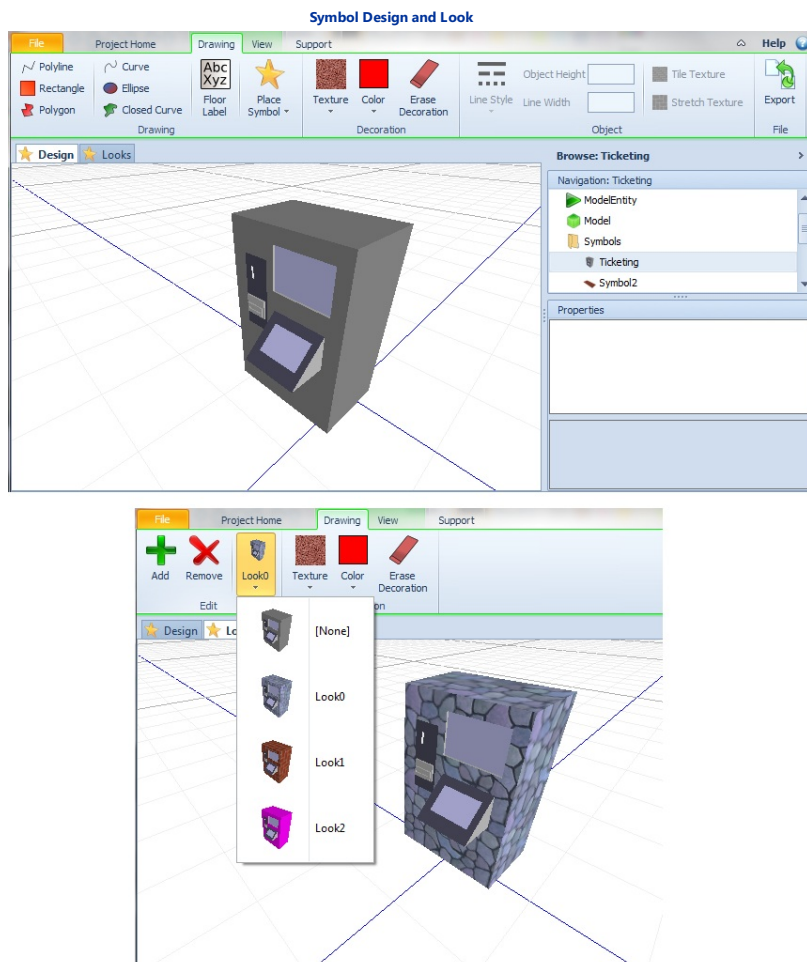
Exporting a Simio symbol and Using a Simio Symbol in another project

A symbol can be exported out of Simio. It is saved as a Simio specific format(.ssfx). The symbol can then be used in another project. In order for the symbol to be used in another project, it must be placed in the correct directory on your computer so that Simio can recognize it. You should be able to find the Symbols folder under C:\Program Files\Simio LLC\Simio. Any .ssfx files placed in this folder will be recognized by Simio and will be accessible via the Place Symbol drop down menu from the Drawing Ribbon tab.

Creating New Symbols

A new symbol can be **created by the user** using the drawing tools, colors and textures. This new symbol can be exported so that it can be reused in other projects. When it is exported, it is stored in C:\Program Files\Simio LLC\Simio\Symbols. To create a new symbol, go to the Project Home tab in the ribbon menu and click on the New Symbol icon. Select the option for Create New Symbol and a new window will appear where you can create the new symbol.

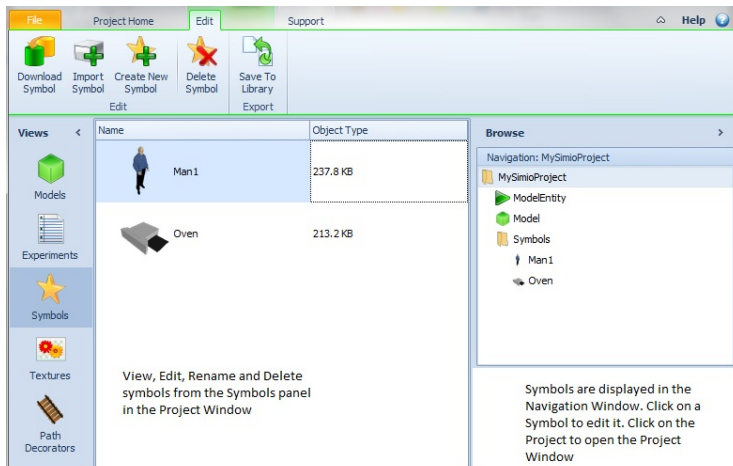
There are two windows that are available, **Design** and **Look**, as shown below. Within the Design window, any existing symbol is shown (from our library of symbols, for example). A new symbol can be added by using the various buttons on the Drawing ribbon. Within the Look window, various 'looks' can be applied to the same single symbol 3d geometry. Each look may have a different texture or color. When multiple looks are applied, the symbols then appear within the symbol library as multiple symbols to be selected. As an example, Simio's animated people are really a single 3d geometry model, but N textures applied. This saves on both disk and video memory as opposed to having X different symbols with the same geometry.



Viewing Your Symbols

Symbols will be added to your Project window. The project window is found by clicking on the name of the project within the Navigation window. Within the Project window, the Symbols panel can be selected from the icons on the left. Symbols can be copied, deleted and edited. The Project Graphics can also be viewed in the Place Symbol drop down found in the Drawing Tab.

Viewing Your Symbols



Symbols, Textures and File Size / Run Performance

When downloading and editing symbols, there is some symbol size information to keep in mind. First of all, using images with low polygon count will keep the project file size small. Additionally, using lower fidelity 3D models may make your animation run faster, but it is highly dependent on your particular graphics card and CPU. And finally, there is no difference in drawing speed for stationary or moving objects. 100 stationary servers will draw just as fast as 100 moving entities. However, the simulation itself may be slower with moving objects because of the additional logic involved.

When deciding on textures for symbols, keep the following information in mind. The advantage of .dds files is potentially significant memory savings when rendering them. The size on disk is the size in memory. Unlike something like a .jpg file where the size on disk is often significantly smaller than the size in memory. The possible disadvantage to .dds files is that they are dependent on driver support, and users may run into problems on some drivers/graphics settings. Users can force the emulation of DDS support by setting File->Settings->Graphics->Compressed Texture Hardware Support to "Disabled". To generate a *.dds file, users can download a tool, for example Paint.NET (<http://www.getpaint.net/>), which can convert image files into *.dds format.

Rotating Your Symbols

If the object has multiple Additional Symbols, and you would like to have the object respect the orientation changes of the symbol, you can use the *Current Size and Orientation Index* property. Assigning the *Current Size and Orientation Index* property with a specified value will automatically resize and orient the object to match the size and orientation of the indexed symbol. For this reason, setting the *Current Symbol Index* and *Current Size and Orientation Index* property to the same value is recommended.

If you want the symbol to appear rotated but not change the object's physical orientation, we recommend creating a new symbol copy that represents the rotation. In the Project Home ribbon, you can 'Create a New Symbol' in Simio, place the existing Symbol, then hold ctrl and click the green handles on the symbol to rotate. The project's Symbols will contain the same symbol at different degrees, for example, such that there is a symbol for Machine90Degrees, Machine180Degrees, and Machine270Degrees. These symbols would need to be applied to the object's Additional Symbols. Then, only the *Current Symbol Index* would be assigned to change look of the object to make it appear it has rotated.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

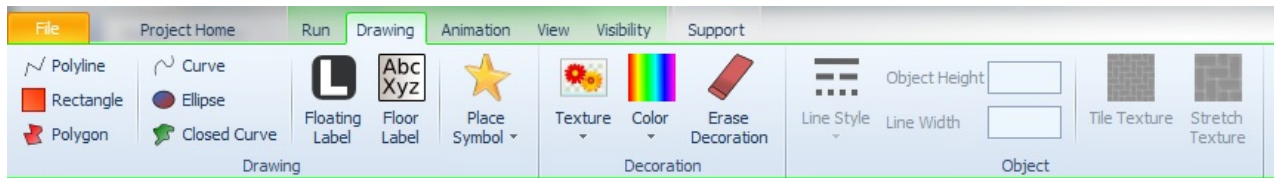
Send comments on this topic to [Support](#)

Drawing

Drawing

A user can add [symbols](#), lines, rectangles, ellipses, curves, polygons and labels to the [Facility Window](#) or [External Window](#) of their model.

Drawing Tab Ribbon (from Facility Window)



You can also change the color of any object or add texture to the object. After adding these drawings to the model, the user can change the line style, the object's height or the line width by selecting the object (i.e. curve) and viewing the Drawing tab.

With polylines and curves, additional vertices may be added or removed by using the Add Vertex and Remove Vertex buttons on the Edit ribbon. When a polyline or curve is selected and the Add Vertex button is pressed, the polyline or curve will get an extra vertex added to the end. You may click to place the vertex anywhere in the Facility window. If the new vertex is placed on the line between two other vertices, it will be added to the middle section of that line. If the new vertex is placed outside of the line, it will be added as a vertex at the end of the line. To remove a vertex, simply click on the vertex to remove and select the Remove Vertex button and the line will be redrawn.

NOTE: Multi-Select of objects, as well as Ctrl-X (Cut), Ctrl-C (Copy), and Ctrl-V (Paste), are supported for all items on the Drawing tab. Please refer to the [User Interface](#) for more details.

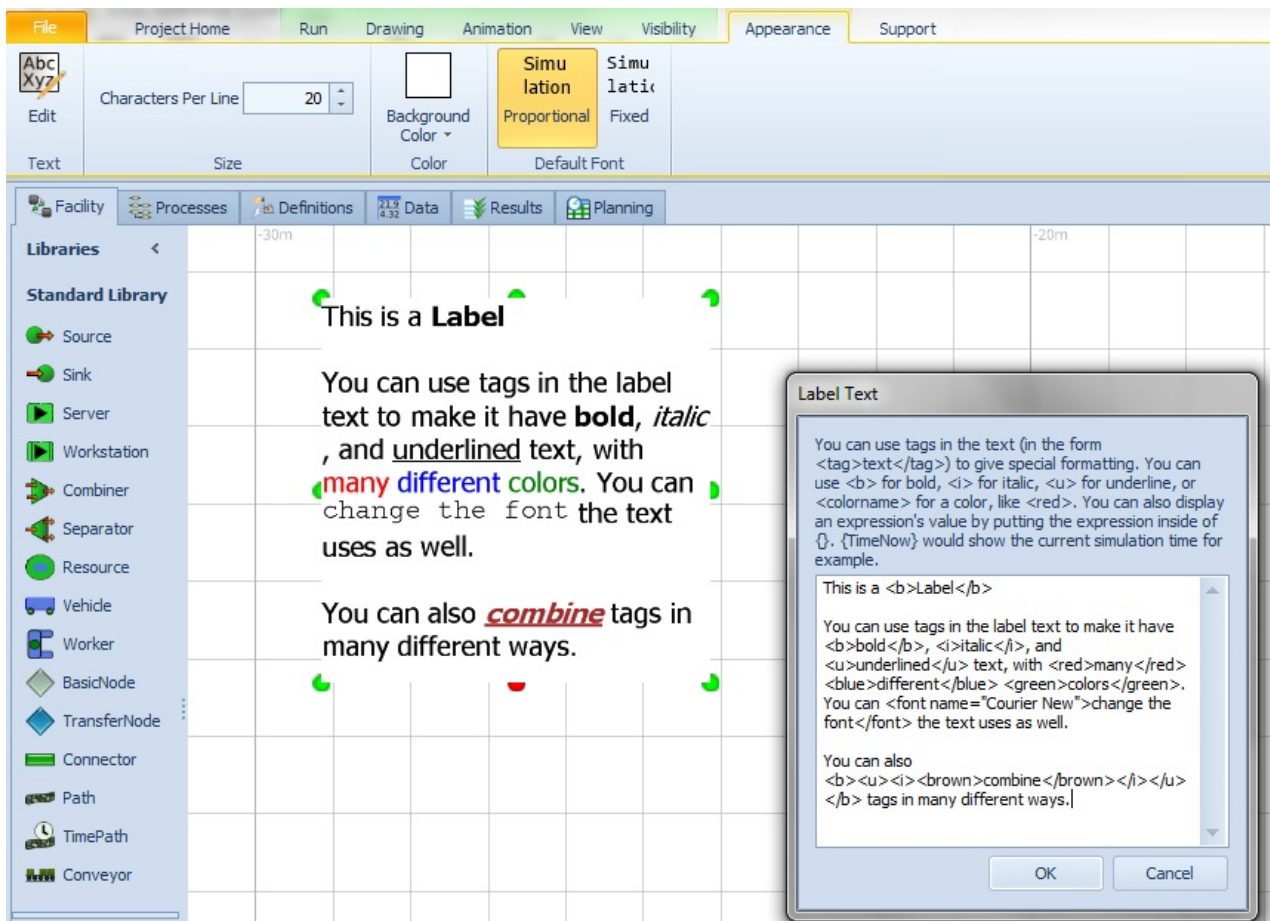
Adding Labels to the Facility Window

Simio provides the ability to add two different types of labels to the Facility Window. The Floating Label is simple, one line of text. The text size, font or color cannot be changed in this label. It is similar to the labels that are attached to the objects that are placed in the window. In both 2D and 3D view, the label will always remain faced at the screen (upright in the Z direction). When an object is instantiated in another Facility, all Floating Labels within that object's definition will be turned off and replaced with one Floating Label that follows the standard instance naming rules. Floating Labels are defined in one Facility and will not be visible if the definition is placed into another object's Facility.

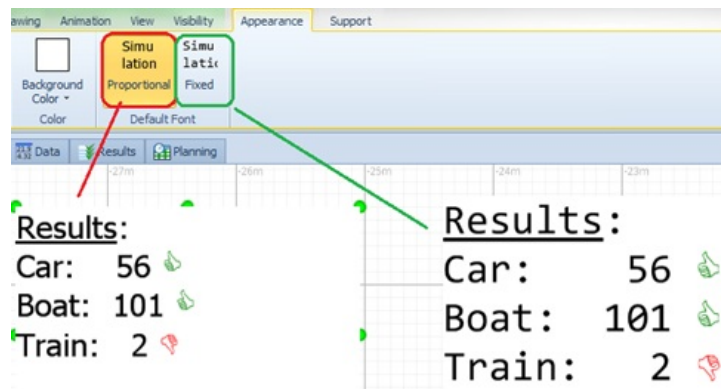
The Floor Label draws a text box on the floor of the Facility Window. Therefore, in 3D view, the label remains flush to the floor. It has more flexibility for changing the appearance of the text. Once the Floor Label is drawn in the Facility Window, select it to see the Appearance Tab in the Ribbon. The Appearance Tab allows you to control how many characters per line will appear in the label. The background color for this label can easily be changed by clicking on the Background Color icon. You can also change the text for the label by clicking the Edit button found in the Text group of the Ribbon. A Label Text dialog will appear where you may change the text. In order to change the appearance of the text, you can use the following tags:

- `` and `` will make the text **bold**
- `<i>` and `</i>` will make the text *italic*
- `<u>` and `</u>` will make the text underlined
- You can change the color of the text by putting the name of the color in a tag, such as `<red>`
- The font of the text can be changed by putting the name of the font in a tag, such as ``
- Floor Labels can also show the value of expressions. To specify an expression put it between curly braces. For example, a label with the text `"The current day is: {Math.Floor(Run.TimeNow/24)}"`, at hour 50, would show "The current day is: 2", with the 2 in bold text.

Appearance Tab Ribbon (when Floor Label is selected)



If using the Default Font (not specifying a particular font, for example ``), you can choose the Default Font 'spacing' for the floor label. The Proportional and Fixed buttons don't affect the overall text layout, only the text **not** surrounded by `` tags. The default Proportional font is "Tahoma" while the default Fixed font is "Consolas".



Adding Textures to Objects

You can add a texture to the surface of an object. Simio provides a library of a few textures, but you can also add your own textures from any .png, .bmp, .jpg, .gif or .dds file. Simply select New Texture from the drop down of the Texture icon in the Drawing or Symbols ribbon or the New Texture icon in the Project Home ribbon. Select the file from your computer and it will be imported into your project library. All project textures can be viewed from the Project window. When applying the texture to an object, you will need to decide if it should be tiled or stretched. This can be controlled by the Tile Texture and Stretch Texture icons on the Drawing ribbon. The following image was created by the user drawing a rectangle in the Facility Window and then applying a texture to that rectangle, which is the Simio Logo saved as a .jpg file.

When deciding on textures for symbols, keep the following information in mind. The advantage of .dds files is potentially significant memory savings when rendering them. The size on disk is the size in memory. Unlike something like a .jpg file where the size on disk is often significantly smaller than the size in memory. The possible disadvantage to .dds files is that

they are dependent on driver support, and users may run into problems on some drivers/graphics settings. Users can force the emulation of DDS support by setting File->Settings->Graphics->Compressed Texture Hardware Support to "Disabled". To generate a *.dds file, users can download a tool, for example Paint.NET (<http://www.getpaint.net/>), which can convert image files into *.dds format.

Texture applied to a rectangle



Adding External Nodes to the External Window

When seen from the External Window, the Drawing Tab also contains an icon called External Node. Adding this to the [External Window](#) provides an entry or exit point into and out of the model.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

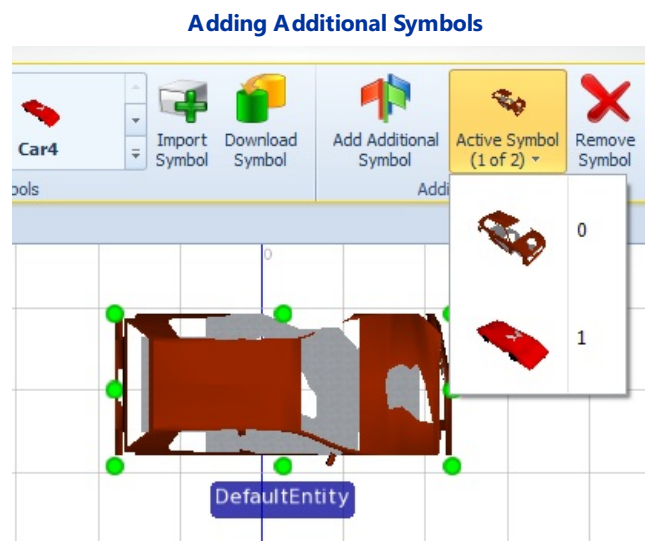
Adding Additional Symbols

Adding Additional Symbols

A user might want to add additional symbols to an object in order to have some model logic change the appearance of the object from one symbol to another while the model is running.

In order to add additional symbols to an object, first click on the object in the [Facility Window](#). With the object selected, click on the Add Additional Symbol icon in the symbols tab of the ribbon menu. Once that icon is clicked, notice that the Active Symbol icon is now active and reads (2 of 2), which indicates which symbol is currently being displayed on that selected object. The user can keep adding additional symbols by clicking the Add Additional Symbol icon again until the desired number of symbols have been created.

To change the appearance of the symbols that were just added to this object, select a symbol from the Active Symbol icon drop down menu. (The object should still be selected) You can verify that you have the correct symbol selected by reading the number (i.e. 2 of 2) in the Active Symbol icon. With the symbol selected, change the color or texture of the symbol. You can also rotate the symbol by pressing the control key and dragging the corner around so that it is in the position you desire. The user can also replace the selected symbol with a symbol from the symbol library by expanding the symbol library that is located in the Project Symbols grouping of the ribbon menu. In the example below, the current symbol that is displayed, the box, is symbol 2 of 2, with the index value of 1. If the user were to select 0 from the drop down, the symbol in the Facility View would show as a plant.



When the object is selected in the Facility Window, the user can change the properties of the object instance in the Properties window. The user can change the **Current Symbol Index** property and put an expression that will return a value that is the index in the associated symbol list - the symbols that are displayed in the Active Symbol list. Whatever value is returned by the expression in this property will determine which symbol is displayed during the run. The value returned must be an integer so that it corresponds to a value in the symbol list.

When the **Random Symbol** property is set to 'True', each new object will be randomly assigned a symbol from the associated list of symbols. Note that, since physical sizes are determined by the state variable specified as the *Current Size Index* (defaulted to ModelEntity.Picture in new model entities), the physical size of the entity will not change until that state variable does. To make the entity have both the symbol picture and size specified by the symbol, set *Random Symbol* to 'False', and make sure the Current Size Index and Current Symbol Index use the same state variable (which in new model entities is already defaulted to 'ModelEntity.Picture' for both properties), then randomly assign a value to that state variable in process logic.

In the example below, the expression in the Current Symbol Index is ModelEntity.Picture. This is a built-in state variable that can be used to hold a value for the entity symbol.

Properties Window

Properties: DefaultEntity (ModelEntity)

[-]

Travel Logic

[+]

Initial Desired Speed

2.0

Initial Network

Global

Network Turnaround Method

Exit & Re-enter

[-]

Routing Logic

Initial Priority

1.0

Initial Sequence

[-]

Population

Initial Number In System

0

Maximum Number In System

2500

Destroyable

True

[+]

Financials

[+]

Advanced Options

[+]

General

[-]

Animation

Current Symbol Index

ModelEntity.Picture

Random Symbol

False

Current Symbol Index

An expression that returns a value for the index in the associated symbol list for the current symbol to display

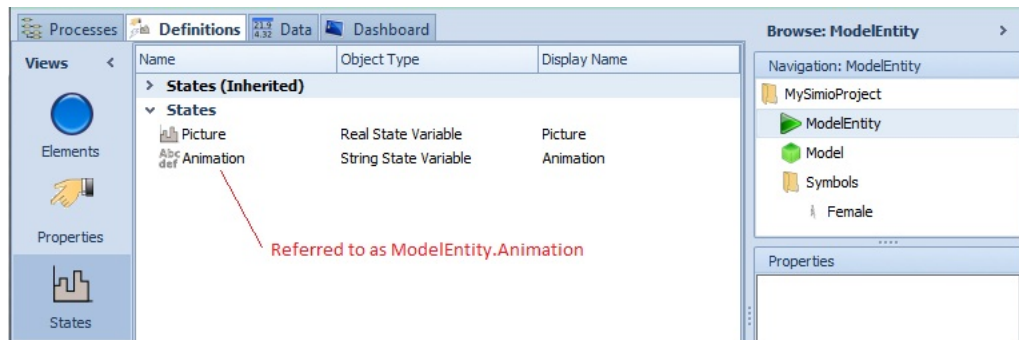
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Animating Moving Objects (Walking People)

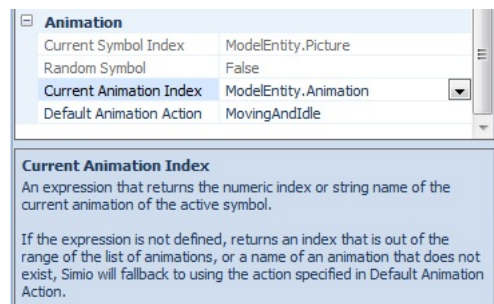
Within the ModelEntity Object in the Navigation Window

In addition to ModelEntity.Picture, the ModelEntity has a string state named Animation when it created (referenced ModelEntity.Animation).



Within the ModelEntity Placed in a Facility Window

Within the ModelEntity properties Animation section, there is a *Current Animation Index* property that is an expression returning the numeric index or string of the current animation of the active symbol. The default value for this *Current Animation Index* is the 'ModelEntity.Animation' state. Therefore, in order to use the default information, the ModelEntity.Animation should be set to a string value for an animation.



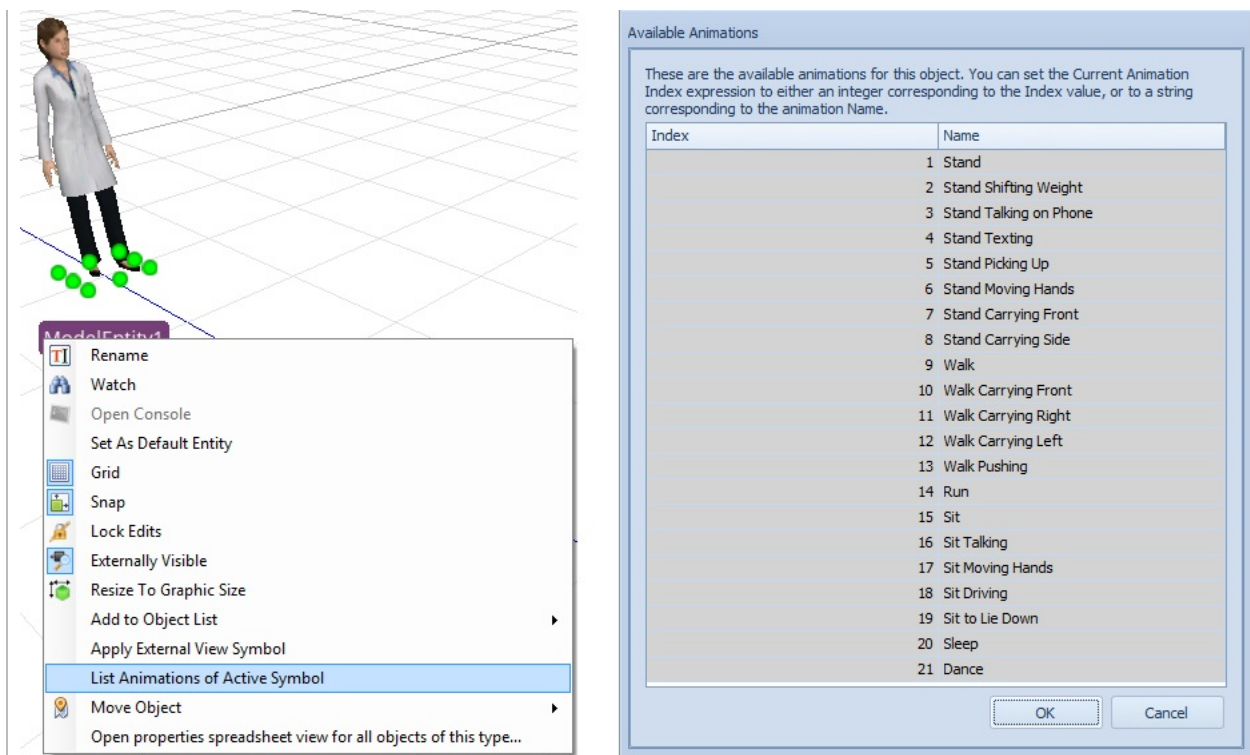
A user may use a numeric index for the animation by removing the Animation string state and adding an Animation real state OR by adding a different real state and changing the *Current Animation Index* property value appropriately.

Alternatively, the *Current Animation Index* property may be set to a string or numeric value directly in the property field itself. See the examples shown below under Current Animation Index.

Animated People and Animations for Simio's Library\Animated* Directories

Within the Project Symbols on the Symbols ribbon, there are a number of animated people that come automatically with Simio. They are located within Library\Animated* directories and include a number of male and female symbols, as well as children, elderly persons, and toons. Within a given directory, the symbols are similar, with the each having different clothing (textures). An entity symbol may be changed from the default green triangle to an animated person by selecting one of the available animated symbols within the directory.

Once placed in a model, a symbol is associated with one or more animations. To get a list of available animations for a given object, be sure the object's Active Symbol is one with animations, then right click on the object and select *List Animations of Active Symbol*, which brings up a simple dialog with a list.



Note that the animations above have both a numeric and string value. If using the default `ModelEntity.Animation` state, the string value would be assigned. If using a numeric state, the numeric value would be assigned.

For example, in an Assign step when the entity is created, you may assign `ModelEntity.Animation = Random.Discrete("Walk", .25, "Run", .5, "Sit", .75, "Dance", 1)`. In this case, 25% of the entities created would walk through the simulation, another 25% would run, another 25% would sit and the remaining would dance.

IMPORTANT NOTE: Animations in the list are case-sensitive. If the animation is not exactly as noted when assigned, the default animation will be the first one listed. All animations in the list are consistent for animated symbols (i.e., all have the same 21 animations listed above).

Current Animation Index

The *Current Animation Index* allows the animations to be listed by either name or number, so any of these values would be correct within the *Current Animation Index* property field (using the animations shown above for the woman):

- 21 – This will make the entity dance.
- "Walk Pushing" – This will make the entity walk as though they are pushing something.
- `ModelEntity.Animation` – This will look at the entity's `ModelEntity.Animation` string value to determine what the animation will look like – when an entity is created, for example, you could assign `ModelEntity.Animation = Random.Discrete("Sit Driving", .5, "Stand Texting", 1)` such that 50% of the entities will sit and appear to be driving and the other 50% will stand and appear to be texting.
- `ModelEntity.NumericProperty` – This will look at the entity's `ModelEntity.NumericProperty` integer value to determine what the animation will look like – similar to above, except that with this property, if it is a real or integer value, you may assign it to '9' for walking, '21' for dancing, etc.

Not having the expression defined, returning an invalid index (like 0 or something beyond the bounds), or an invalid name (like empty string ""), makes Simio fallback to doing the default animation, based on the *Default Animation Action* property.

Default Animation Action

The Default Animation Action property can be one of the following values:

- None - Simio will have no default animation.
- Moving - Simio will have default animation when the object is moving, and Current Animation Index did not provide an explicit animation.
- MovingAndIdle - Simio will have default animation when the object is moving or when it is idle, and Current Animation Index did not provide an explicit animation.

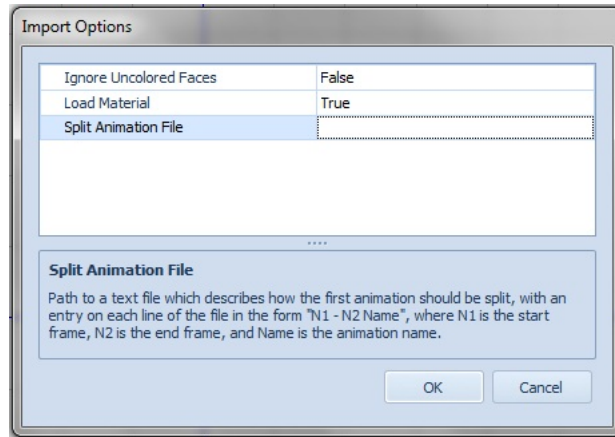
Default Animation Action and Current Animation Index will be disabled unless you have at least one symbol applied to the object that has animations in it.

Refer to the SimBit [AnimatedPeople](#) for several examples of animating moving objects.

Importing New Animations

Simio includes a number of Animated symbols (male and female), which can be found in Library\Animated\People. Users also have the capability to import their own animations.

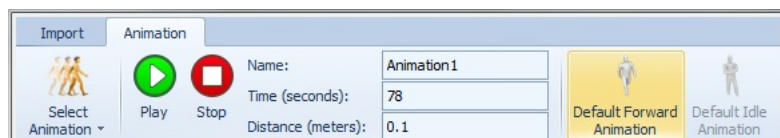
New animated symbols can be imported using either the Project Home ribbon, New Symbol button (Import Symbol) or by highlighting an existing symbol in the Facility window and selecting the Import Symbol button from the Symbols ribbon. Typical 3D files have a *.dae (digital asset exchange) filename extension and are in COLLADA (interchange file format for interactive 3D applications) file format. Once imported, there is an Import Options dialog that allows the user to specify several features.



Ignore Uncolored Faces property determines if the import will ignore any faces without color or texture information. This value is set to 'False' as a default. The *Load Material* property determines if any material settings, such as color and texture, will be considered. This value is set at 'True' as a default. The *Split Animation File* property is a path to a text file that describes how the first animation should be split, with an entry on each line of the file in the form "N1 - N2 Name", where N1 is the start frame, N2 is the end frame, and Name is the animation name.

Once the Import Options dialog is accepted, the Import dialog is displayed. This includes options to specify the size of the symbol, name, and rotations options on the Import tab.

Additionally, there is an Animation tab on the import dialog. This includes two buttons, Default Forward Animation and Default Idle Animation, as well as time / distance fields, and Play/Stop buttons. If the Animation tab did not appear, Simio did not recognize that the symbol had any animations. For .fbx files, Simio is currently set up for skinned mesh or bone and weight animations. Simio will not recognize node based animation.



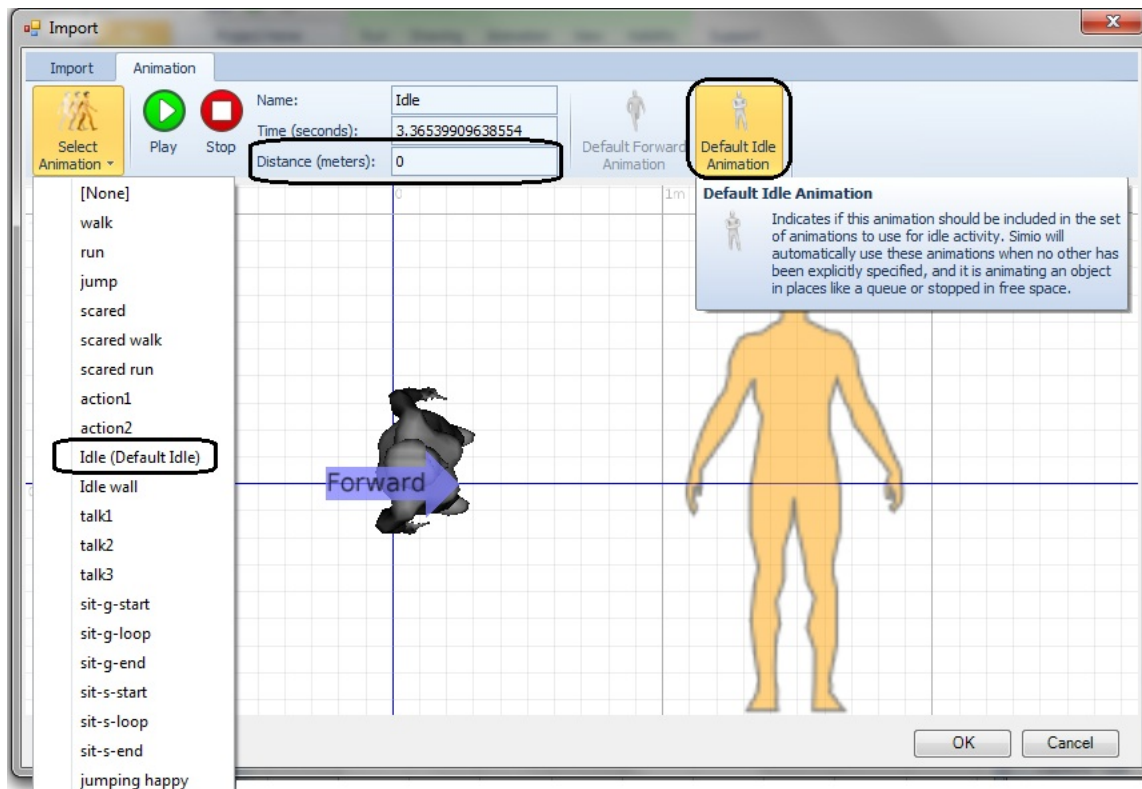
Default Forward Animation is enabled when Distance is > 0, and disabled otherwise. When enabled, it defaults to True. *Default Idle Animation* is enabled when Distance == 0, and disabled otherwise, it defaults to False.

When a selected animation has Default Forward Animation turned on, it will be included in the set of animations Simio looks at when trying to do "default animation" for some agent that is currently moving.

When a selected animation has Default Idle Animation turned on, it will be included in the set of animations Simio looks at when trying to do "default animation" for some object that is NOT currently moving.

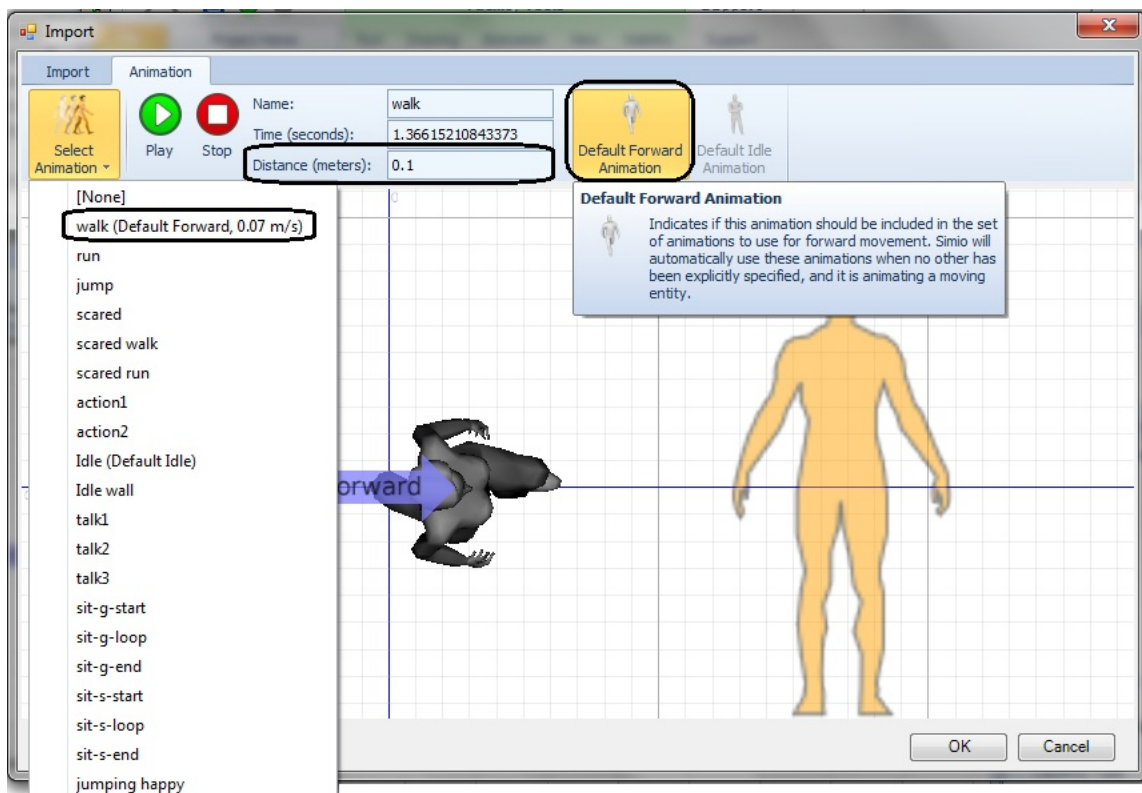
Default Idle Animation Example:

Below, you will see that the 'Idle' animation was selected (under the *Name* property) and the *Distance* is specified as '0'. The *Default Idle Animation* button is available, and in this case it is selected, as an idle animation symbol to be displayed for the idle activity. Note that when the button is selected, it will appear in () next to the animation name on the dropdown list.



Default Idle Animation Example:

Below, you will see that the 'Walk' animation was selected (under the *Name* property) and the *Distance* is specified as '0.1'. The *Default Forward Animation* button is now available instead of the *Default Idle Animation* button. The Play and Stop buttons may be used to see what the animation will look like, for example, how fast the person appears to walk. Once the *Default Forward Animation* button is selected, the () will be shown next to the animation on the dropdown and the animation will be available for use within the simulation.

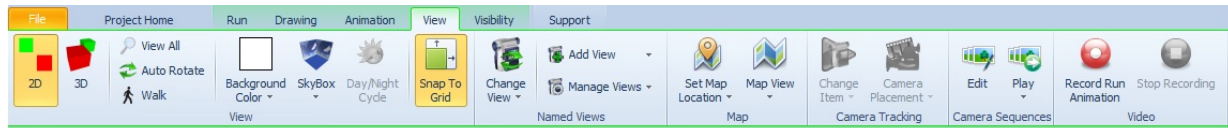


Changing the View

Changing the View

From within the [Facility Window](#), the **View Ribbon** can be used to change how the objects and the background appear in the window.

The View Ribbon, as seen from the Facility Window



The function key, **F11**, allows the capability of collapsing all of the ribbons and other windows so that the simulation animation can be seen on the full screen. Pressing the F11 or Esc key while in full screen will then put back the ribbons and windows that were collapsed. The F11 key can also be used in all other windows, such as Processes, Data (for tables), Results, and Experiments.

View

The **2D icon** changes the view to a 2D top down view. This is the default view. Left click on any empty space and drag to pan. Right click and drag up and down to zoom.

The **3D icon** changes the view to a 3D perspective view. There are many ways to move around in the 3D view including:

- Left click on any empty space and drag to pan
- Right click and drag up and down to zoom and drag left and right to rotate
- Left-click and drag any object to move it in the horizontal plane
- Ctrl and drag any object to move it on top of another object
- Ctrl and drag any object selection handle to rotate it
- Left-click to select a node object
- Ctrl+Shift and left-click on a node to start drawing a link
- Scroll wheel standard zoom in and out
- Alt+Wheel for 10x finer zoom than the standard zoom
- Ctrl+Scroll wheel changes the viewing angle of the camera
- Shift+Scroll wheel moves the camera straight up and down
- Up arrow moves forward
- Down arrow moves left
- Left arrow turns left (or moves left in top-down mode)
- Right arrow turns right (or moves right in top-down mode)
- Shift+Left arrow moves left
- Shift+Right arrow moves right
- [W] key sets the camera 1.7 meters off the ground, with the angle pointing almost straight out (i.e. "walking mode")
- Pinch-to-zoom and two-finger-rotate gestures are also supported, provided user has a touch screen

The **View All** icon brings all objects into view within the window.

The **Auto Rotate** icon starts auto-rotating the view. The user can click on any empty space to stop the rotation. Once auto-rotation is stopped, the view will remain where it currently is.

The **Walk** icon sets the camera 1.7 meters off the ground, and has the same function as the [W] key listed above.

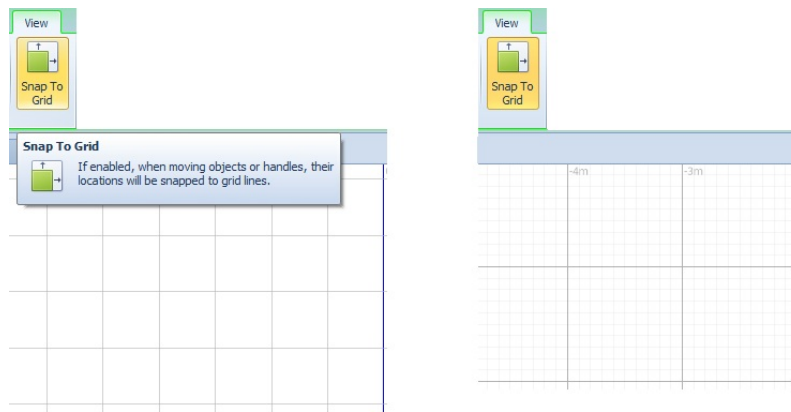
The **Background Color** icon will change the background color of the entire Facility window floor in both the 2D and 3D views.

The **SkyBox** icon opens a library of static and dynamic scenes that can be placed in the 3D animation. Dynamic scenery includes several options for clouds that move with the animation as the model runs. Typically, the Speed Factor should be set to 5 - 10 or greater to see movement. Static skybox options include many outdoor scenes. The 'None' option will remove any existing skybox. Users can create their own SkyBoxes as well. The "library" skyboxes are simply *.zip files with images inside them with the names front, back, left, right, top, bottom. They don't need to have all of those names in them, Simio will use whatever ones it finds. They are located in the Skybox folder under C:\Program Files\Simio LLC\Simio.

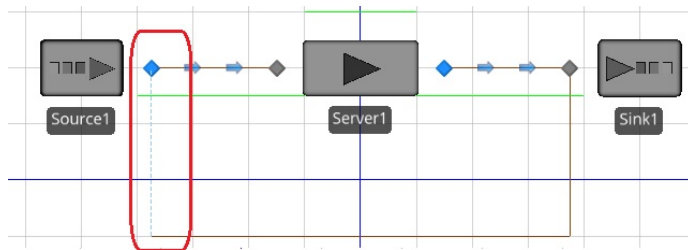
The **Day/Night Cycle** icon is available if the SkyBox is set to a dynamic skybox. This on/off icon indicates if the animation should alter the ambient light and display a moving sun and moon in accordance with a day/night cycle.

Snap to Grid will snap moving object or handle locations to grid lines to allow for object alignment. The snap locations will be dependent on the grid level. In the below diagram, the left grid shown is the default zoom location and objects will snap to those larger grid lines when placed. The right grid shown is zoomed in several times and displays more granular grid lines, in which case, the objects will snap to those smaller grid lines. Thus, zooming in within the Facility window will provide the user with more precise placement. Both Grid and Snap can be turned on/off using the right-click menu or pressing the G (Grid) or S (Snap) keys.

Additionally, Snap to Grid also snaps rotations of objects. When rotating objects with snap enabled, the object will rotate in 15 degree increments (thus 4 rotations to move the object 90 degrees).

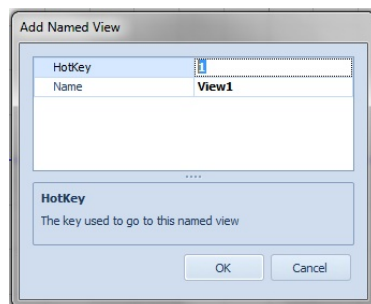


When placing objects, note that the center of the object is the location of the snap to grid. When snap is turned on and a link is being drawn, or a link vertex moved, Simio will not only snap to the grid, but snap to the closest *visible* (on the screen) node. A dashed blue line will show what node caused the snap, as shown below.



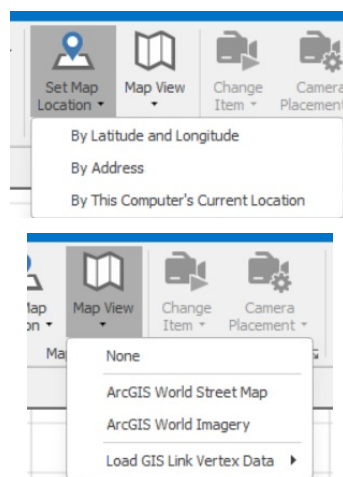
Named Views

Named Views allow you to place the camera at a location and save that view for later using **Add View**. You can either pick a specific view to go to, or cycle through them using the **Change View** option. Change or delete a specific view using the **Manage Views** option. Named views may have a name as well as a 'HotKey' that will allow users to simply press the HotKey on the keyboard and move to that particular view.



Map

There are two button options under the Maps section of the ribbon. The **Set Map Location** button provides options for setting the latitude and longitude coordinates of 0,0 in the model. The **Map View** option provides a drop down list of options for the type of map to see, from various online sources as well as Load GIS Link Vertex Data. Additional information on maps and creating links between nodes using the maps can be found on the [Maps](#) page.



Camera Tracking

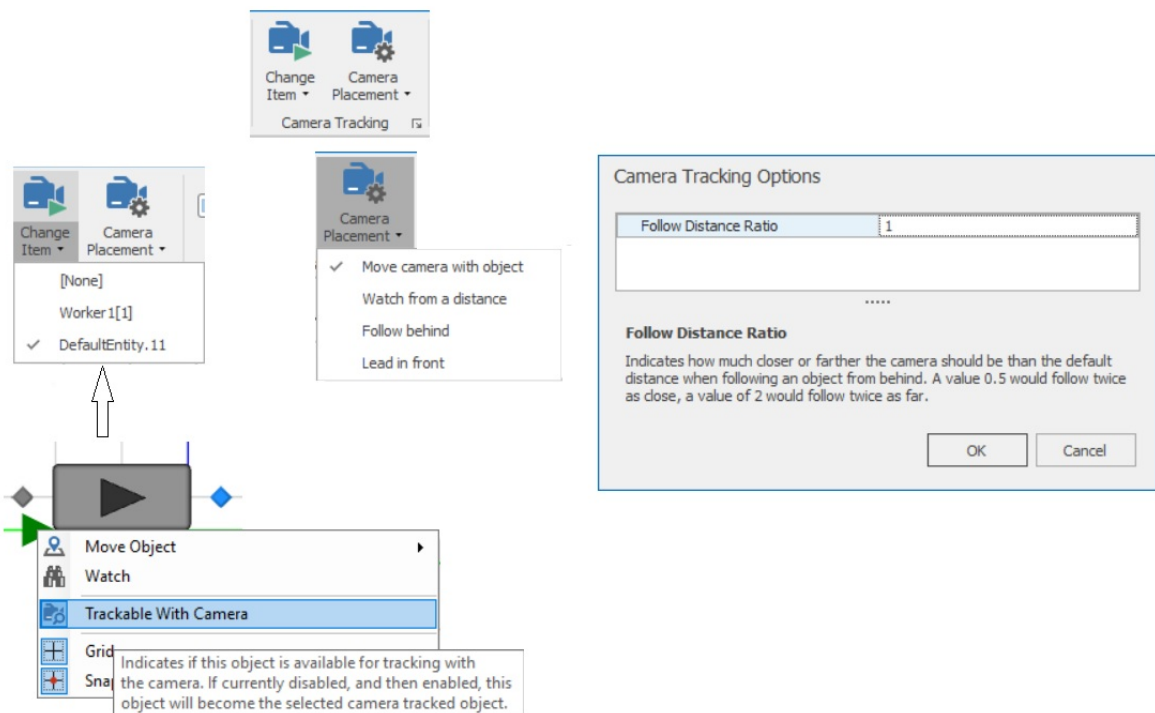
There are three buttons in the **Camera Tracking** group, including **Change Item** and **Camera Placement** buttons and a small options button, which opens a **Camera Tracking Options** dialog. These icons are disabled until you enter run mode and have at least one item for tracking. As shown below, the right-click option on any Agent, Entity or Transporter (this includes Workers / Vehicles) allows the selection of the 'Trackable with Camera' option. When selected, the camera icon will have a small square around it. Once the 'Trackable with Camera' option is selected, the agent / entity / transporter identifier will appear on the Change Item list. Selecting 'Trackable with Camera' when the item is already on the Change Item list will remove it from the tracking list. By default, all non-destroyable agents, such as Workers and Vehicles, are put onto the Change Item list for selection at the start of the simulation run. If there are no Workers or Vehicles in your model, the Change Item button will remain disabled in run mode, until you have selected 'Trackable with Camera' for a particular object such as an entity.

When running, you can click on the button part of **Change Item** to cycle between tracking different items in the model, or tracking nothing, and letting you move the camera where you wish. If you click the drop down part, you can select a specific item, or [None], meaning you want to turn off tracking.

The **Camera Placement** dropdown allows you to change the perspective of tracking and how you would like to track the object. Options include 'Move camera with object', 'Watch from a distance', 'Follow behind' and 'Lead in front', providing multiple camera angles from which to view agents. When an item you are tracking (such as an entity) gets destroyed, Simio will put you back to the view you were when you first selected the item to be tracked.

Within the **Camera Tracking Options** dialog, the *Follow Distance Ratio* property indicates how much closer or farther the camera should be than the default distance when following an object from behind. A value 0.5 would follow twice as close, whereas a value of 2 would follow twice as far. A larger distance ratio is useful to view more of the Facility window when following an entity or object.

Example of Camera Tracking buttons, as seen with dropdowns

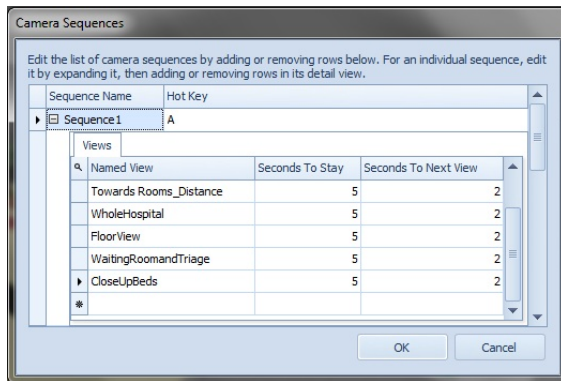


Camera Sequences

Within the **Camera Sequences** group, there are options to **Edit** and **Play**. A Camera Sequence is a sequence of instructions that can be played back either by selecting the Sequence under the Play button, or by using a hotkey associated with the sequence. Therefore, the Edit button allows you to Add / Edit / Delete an entire sequence of named views or a particular view within the sequence. To add a camera sequence, simply type a sequence name in the *Name* column and an associated key in the *HotKey* column. You will then be able to expand the sequence by pressing the + button. This will open the Views dialog for inputting the specific *Named View*, *Seconds to Stay* (at that particular view), and the *Seconds To Next View* (to change between the views).

The Play button will display the available sequences that may be selected. Once a sequence is selected, Simio will rotate through the various views. If there is currently a sequence playing, a small icon will be displayed in the lower right corner with the sequence name. The sequence will stop if the user performs any other action (i.e., such as changing the view, mouse click, etc.). Once all views have been displayed, the camera will remain at the last view specified.

Example of Camera Sequences, with multiple Named Views



Video

The **Video** group contains the **Record Run Animation** and **Stop Recording** buttons. These buttons allow the user to create an .avi recording of the model animation. Clicking on the Record Run Animation button will prompt the user to choose a name and location for their .avi file. The user will also be prompted to select a video compressor. The options that the user will be given will vary depending on what is installed on the PC. We recommend using a state of the art codec, like DivX, Xvid, x264, ffmpeg, or others. Once the user begins the interactive model run by clicking the Run button, an .avi video of each animation frame will begin. The recording stops when the model runs ends or when the user clicks the Stop Recording button.

The video will record the animation in the working space of the Facility window. If the user switches into another Simio window while the recording is occurring, whatever is present in the same defined space as the Facility window, will be recorded. For example, if the user switches over to the Data window during the recording, the tables will be recorded, but the left hand panel will not be in view, assuming the Library Panel was expanded in the Facility window. If the user collapses the Library panel before recording, the working space will be increased, thereby creating a larger area for animation capture both in the Facility window and other windows. To begin recording in full screen mode, click the Record Run Animation button, hit F11 to get to full screen mode and then hit F5 to begin the model run.

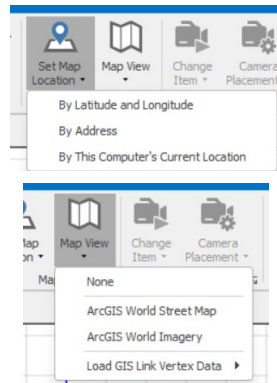
If the user switches into an application outside of Simio, whatever is shown in that same Facility window space is also recorded. This allows a user to animate a model running in the Facility window and then switch over to a spreadsheet to show external data and then move back into Simio to show more animation or possibly another Simio window, such as the tables in the Data window.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

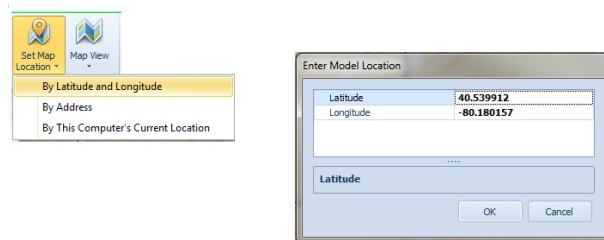
Send comments on this topic to [Support](#)

Maps

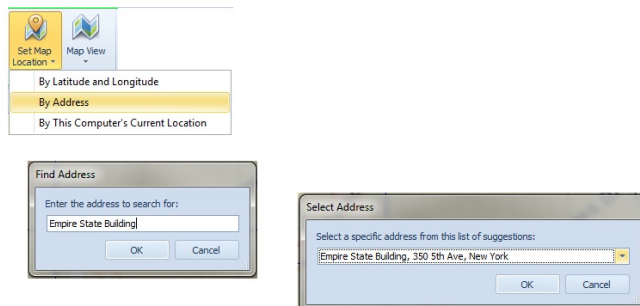
On the View ribbon, there are two button options under the Maps section. The **Set Map Location** button provides options for setting the latitude and longitude coordinates of 0,0 in the model. This setting will be used for things such as placement of the background map, if enabled. The **Map View** option provides a drop down list of options for the type of map to see, from various online sources. The option selected will set the map tile source for the current view. It is important to note that the map is only displayed with the model when the user is connected to the internet with access to the map; the map does not become a permanent part of the model itself.



The options within the Set Map Location are utilized when a map is displayed. The **By Latitude and Longitude** option always shows the current lat/long to which the map is centered.



Users can change the default latitude and longitude by using the **By Address** option. Please note that By Address will only find an address when a map is currently loaded. The Find Address dialog will appear where the user can enter a specific street address, city and state name or particular landmark (as shown below). A dialog will then ask the user to select the specific address from the list, providing a list of possible options based on the original user input. Once the location has been entered, the map center will move to the particular location and the default coordinates within the By Latitude and Longitude dialog will update.

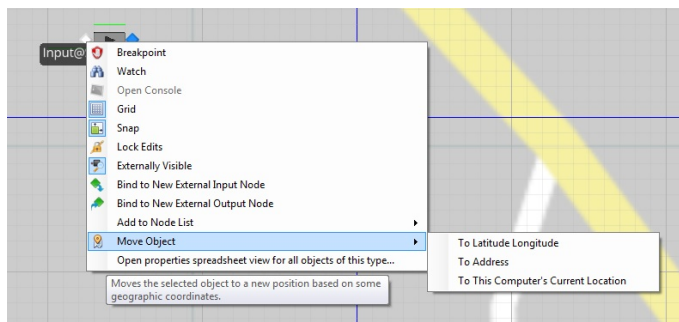


There is also an option to set the map location to the computer's current location, provided the computer graphical location is enabled. The options for setting the computer location vary by operating system.

Note that Simio uses a "Web Mercator" (https://en.wikipedia.org/wiki/Web_Mercator) projection to convert from the spherical lat/long coordinates to our 2 dimensional x,y,z coordinates. This simply means that things close to the equator are their actual size, while the further you get away, the more distorted they become. Additionally, not only does Simio use that projection, but it further scales down the size so that things close to 0,0 are much closer to their actual meter offsets from that location. If a user zooms all the way out, they may find the world's width does not match the earth's circumference.

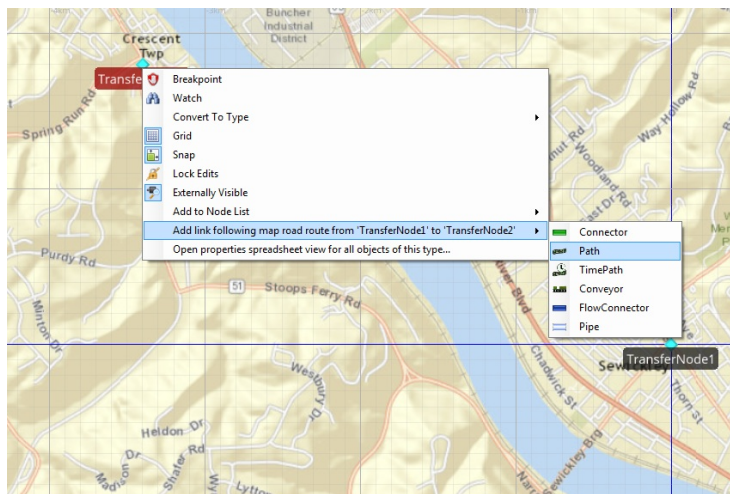
Moving an Object to a Specific Map Location

When the map is enabled, any simulation object may be moved to a given location within the map as well. This can be done by right-clicking on an object (such as a BasicNode, Server, or TransferNode, for example), selecting the Move Object option and moving the object to a latitude and longitude, address or computer's current location.

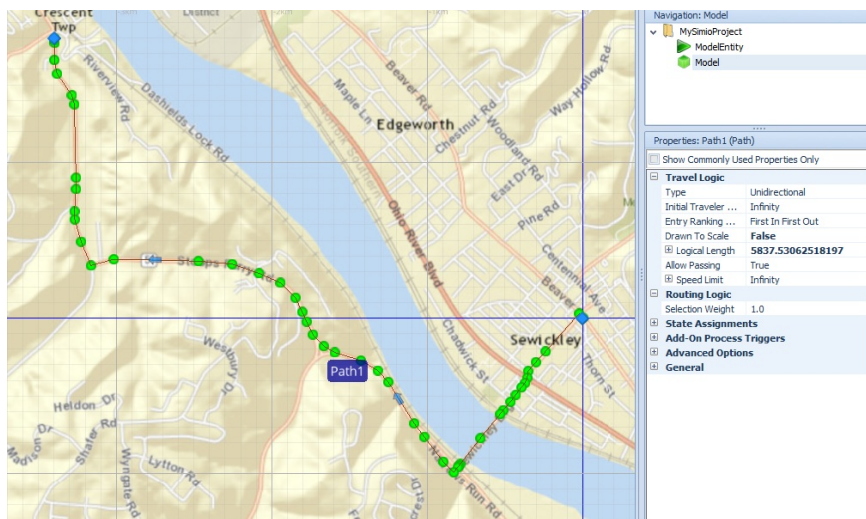


Creating a Link Based on Route Between Two Points

If **exactly** two nodes have been selected (click on a single node, then Ctrl-Click on the second node), there is an option on the right-click menu to add a link (path, connector, etc.) following a map road route from one node to the other. The order in which the nodes are selected determines the direction of the link (from the first selected node to the second selected node).



This add link option will call out to a routing service to get the road route between two geographic locations. It will set the resulting link's *Drawn to Scale* property to 'False', and set the *Logical Length* property to the measured length of the entire route. Below you see the link that was drawn from the two nodes selected in the above diagram.



Auto-create Links from Data Table on ArcGIS Roadways

If a model has a master table with an Object Instance Key column set to auto-create, with the *Object Type* for that column set to some Link type. There must also be a detail table defined for that master table that contains the vertex information for that auto-created link.

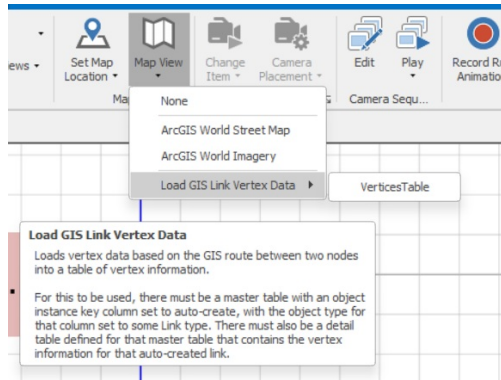
For example, load the AutoCreatePathsWithVertices SimBit. Notice its table structure. This is a similar structure to what you will need.

Object Table				
	Object Name	Object Type	X	Z
1	Source1	Source	-8.75	-3.75
2	BasicNode1	BasicNode	-4.5	-2
3	Server1	Server	0.125	0
4	TransferNode1	TransferNode	4.875	-2
5	Sink1	Sink	9.5	-2.5
*				

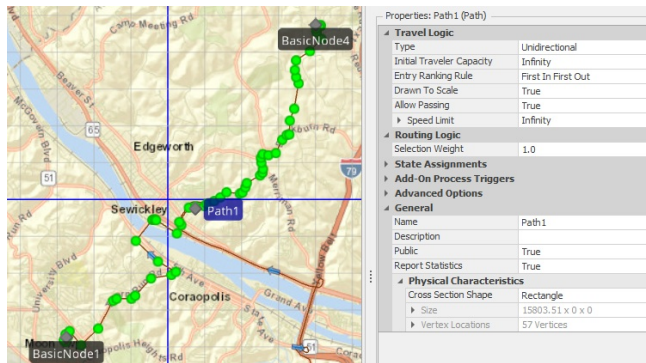
Vertices Table			
	Link Name	Vertex X	Vertex Z
1	Path1	0.5	-7
2	Path2	-0.25	3.75
3	Path3	2.375	-4.25
4	Path4	6.5	2
*			

Link Table			
	Link Name	Start Node	End Node
1	Path1	Output@Source1	BasicNode1
2	Path2	BasicNode1	TransferNode1
3	Path3	TransferNode1	Input@Server1
4	Path4	Output@Server1	Input@Sink1
*			

When going to the View ribbon, you will see the option to Load GIS Link Vertex Data.



By selecting the Vertex table, you will have the option to Reduce Vertices. Once you click OK, your vertices table will populate the ArcGIS coordinates for the links, thus autocreated the links along the roadways.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

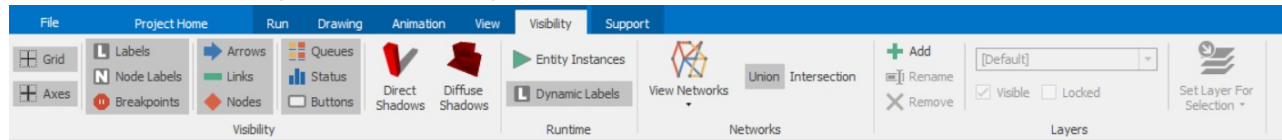
Send comments on this topic to [Support](#)

Visibility

Visibility

From within the [Facility Window](#), the **Visibility Ribbon** can be used to toggle on and off various objects, as well as view the networks.

The Visibility Ribbon, as seen from the Facility Window



When the **Grid**, **Axes**, **Labels**, **Node Labels**, **Breakpoints**, **Arrows**, **Links**, **Nodes**, **Queues**, **Status** and **Buttons** icons are highlighted, this indicates that they are visible within the window. If the user clicks on any of these icons and unhighlights it, the particular feature will not appear in the window. It is important to note that when an feature is not visible, it also is not editable. Therefore selecting, copying, deleting, etc. of groups of objects will not select anything that is not visible. An item must be made visible first before it can be edited.

The **Direct Shadows** icon shows or hides shadows from a directional light source, like the sun. The light source in Simio is fixed in the corner at a 45 degree angle. However, if Direct Shadows are used in conjunction with a dynamic Skybox and the Day/Night cycle, the shadows will be adjusted for the change in the light source. The **Diffuse Shadows** icon shows or hides soft shadows from other objects interacting with the illumination of an object. Only one of the shadows icons may be on at a time. Note: If running Simio under DirectX software fallback, shadows will not work.

The **Entity Instances** icon determines whether or not the Entity and Transporter instances will be visible while in run mode. If disabled, any instances that exist when going into run mode will be hidden and then shown again when leaving run mode. If enabled, these instances will be visible at all times including run mode. The **Dynamic Labels** icon works in conjunction with a property on Entities and Transporters (which includes Worker / Vehicle) named *Dynamic Label Text* (in the Animation category of properties). The *Dynamic Label Text* is an expression which returns a string to show a floating label below a dynamic entity at runtime. This floating label is similar to the label shown on fixed objects (Source1, Server1, Sink1, etc.). The Dynamic Labels icon on the Visibility ribbon allows users to enable or disable these labels separately from the Labels specified for other fixed objects.

The **View Networks** dropdown, highlights the links within the network(s) that are selected. Each network highlight can be individually toggled on and off. When selected, the **Union icon** shows the union of all selected networks. The link is in the union of the selected networks if it is a member of ANY of the networks. When selected, the **Intersection icon** shows the intersection of all selected networks. The link is in the intersection of the selected networks if it is a member of ALL of the networks. Networks that have Member Links defined in a data table will not appear in the dropdown.

The **Layers** subsection allows for objects, animations, and graphics to be placed on different defined Layers in the model. The **Add** icon allows you to add a new Layer to the model. The **Rename** button allows you to rename the selected Layer in the model. The **Remove** icon allows you to delete a Layer. You can toggle a Layer's **Visibility** and **Locked** via the two checkboxes below the combo box which shows the "active" Layer. The **Set Layer For Selection** dropdown allows you to assign the selected objects to a Layer.

Note that subsections of the Visibility section icons (Grid, Labels and Axes) can currently be found on the View ribbon in the Console, External, and Symbol editing ribbons.

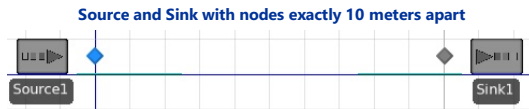
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Animating Links

Drawing and Animating Links

Links generally animate best if they are drawn approximately or exactly to scale. If they are not close to scale, then you will see entities animate with overlap or gaps between them that do not really exist in the logical model. In this example, we will place a conveyor between a Source and Sink object. Select and then drag and drop a Source and a Sink object 10 meters apart. To ensure that they are 10 meters apart, the nodes should be placed so that their centers are directly on the vertical lines. To get an idea of how nodes should be placed see the image below:

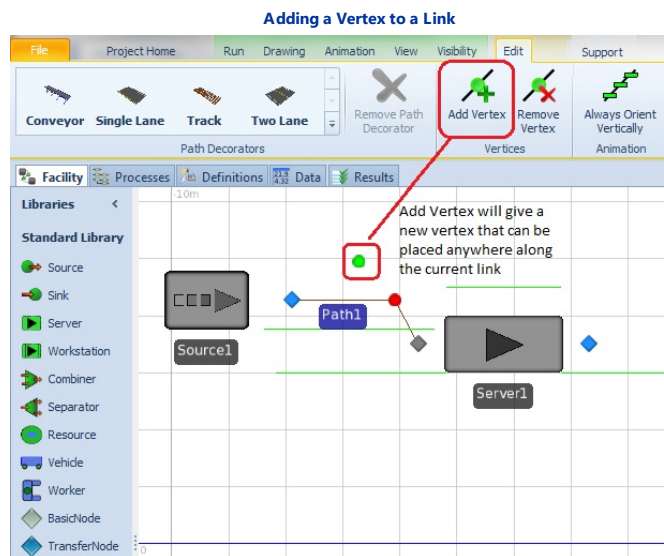


Let's add a conveyor between the Source and the Sink. Click on the Conveyor object in the Standard Library and then click on the TransferNode displayed on the Source and have the conveyor end at the BasicNode displayed with the Sink. Another way to add a Conveyor is to click on the TransferNode displayed on the Source while holding down the Ctrl+Shift keys and dragging the link to the BasicNode at the Sink. Because the type of link isn't yet determined, a small menu will be displayed to select the type of link, in this case Conveyor.

The *Drawn to Scale* option in the Properties window determines the relationship between the animated length and the logical length used in calculating movement in the model. If *Drawn To Scale* is 'False', that means that the logical length will be exactly the value specified in the *Logical Length* property. This is best to use if you know the exact link length and it is important that the model reflect that. The disadvantage of this approach is that if you later adjust the path (e.g. move the Source and Sink in our example closer or further apart) that adjustment will not be reflected in the model. If *Drawn To Scale* is set to 'True', that means that the *Logical Length* property will be ignored and instead the length used in the model logic will be determined by the length drawn in your animation. This is best to use when you are experimenting with different layouts and you want the model logic to reflect what you are currently looking at. The disadvantage of this approach is that you don't have an exact logical length. In our example, it is likely that you drew the conveyor slightly more or less than 10 meters. While this might not matter in early experimentation, it could be important in your final analysis.

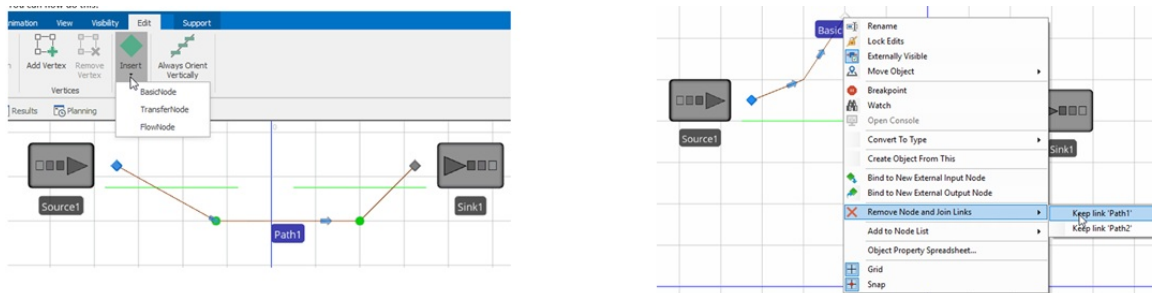
Adding Vertices or Nodes to an Existing Link

When the link is selected in the Facility window, the Edit tab is displayed in the Ribbon. This contains a library of path decorators, as well as options to add and remove vertices or insert a node to the link. Additional vertices may be added to or removed from a link by using the Add Vertex and Remove Vertex buttons. When a link is selected and the Add Vertex button is pressed, a green vertex is provided that can be placed anywhere along the link. To remove a vertex, simply click on the vertex to remove and select the Remove Vertex button and the link will be redrawn.



When an existing link is selected in the Facility window, the user may also add a node to the link by using the Insert button and selecting the type of node (i.e. BasicNode for example). If the link has one or more existing nodes that are no longer required, the user may right click on the node (as in BasicNode1 below) and select 'Remove Node and Join Links', then select the path that will remain as the current link (e.g. Path1 below).

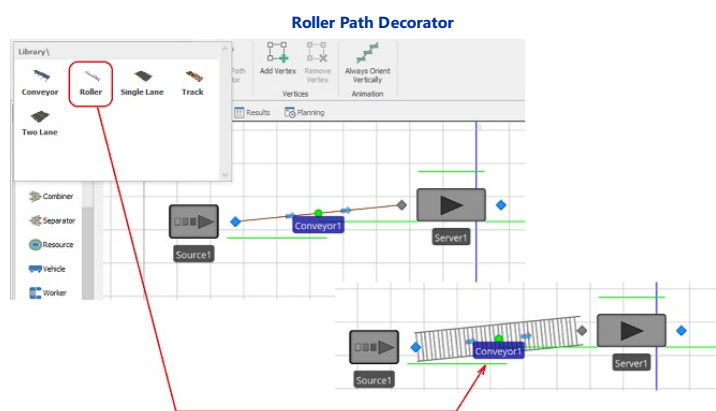
Adding a Node to a Link



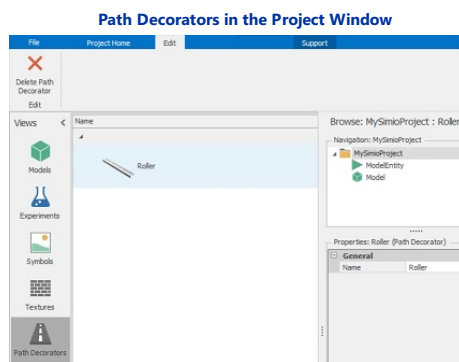
An additional option on the Edit ribbon is the Always Orient Vertically button that provides the option to have the entities' vertical orientation always remain upright regardless of the angle of the actual link that is selected. This is useful for people going up stairs, for example. Note that this option only affects animation. Logically, the entities still occupy space on the link as if they are vertically aligned at the same angle as the link itself.

Path Decorators

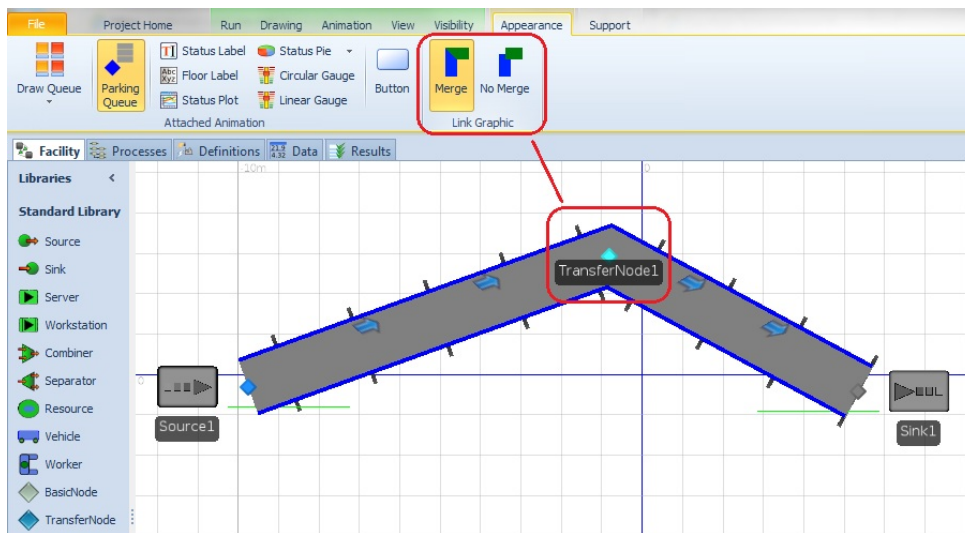
There is a library of path decorators that are provided with Simio. Simply clicking on a path decorator from that window will change the appearance of the link that is selected. In our example, we will select the roller path decorator that is found in the window.



Once the path decorator is added to an object in our project, it appears in the Path Decorators panel within the Project window. Click on the name of the Project in the Navigation window to open the Project window. Path Decorators are stored in C:\Program Files\Simio LLC\Simio\PathDecorators. Even though this example was for a conveyor type link, any type of link can have a Path Decorator displayed on it.

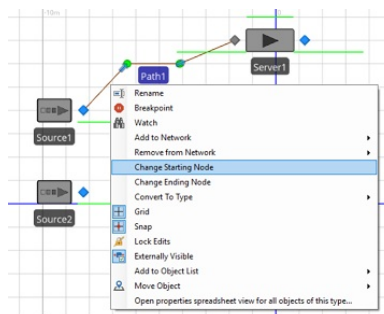


When multiple links are connected that have path decorators, the node (BasicNode or TransferNode) that connects the links may be highlighted to access the Appearance ribbon. There are Link Graphic options for Merge and No Merge that provide the user the ability to specify how the paths look when they intersect at the node. Merge will blend the paths such that they are solidly connected as shown below, whereas the No Merge draws the links such that they do not connect at the node intersection.



Changing Link Starting and/or Ending Node

The right-click menu for any type of link consists of options to add or remove the link to a network, as well as to change the starting or ending node to which the node is attached. When the user selects Change Starting Node, a new starting node may be selected and then the link will be disconnected from its current starting node. The ending node and all vertices will remain intact. The same process can be used to change the ending node for a link.

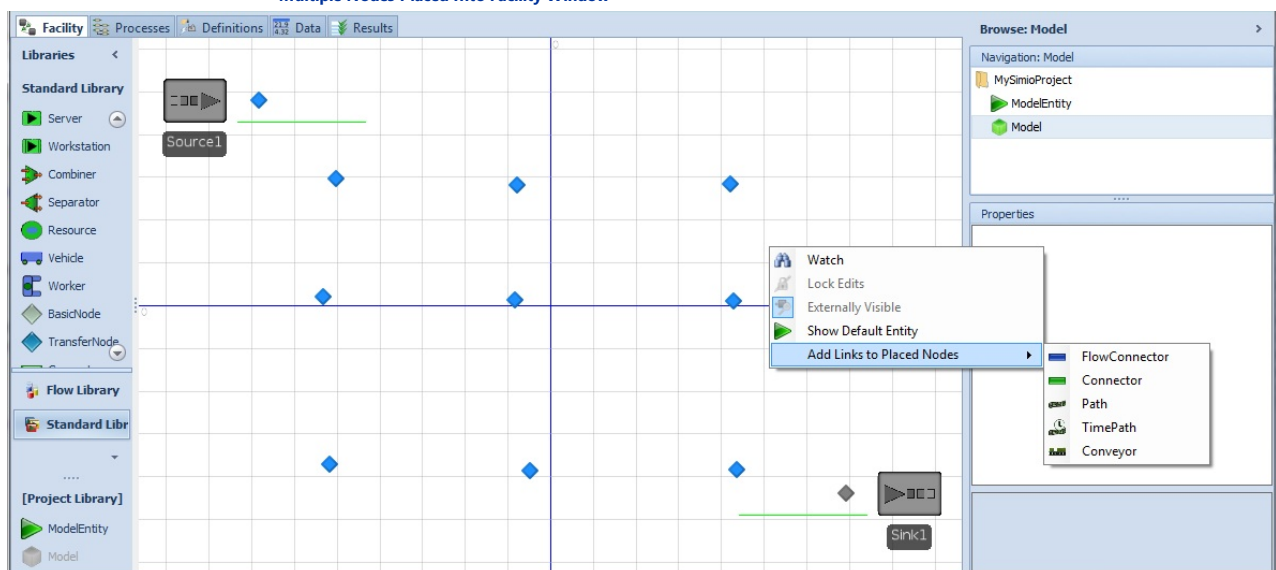


Connecting Multiple Nodes with Links

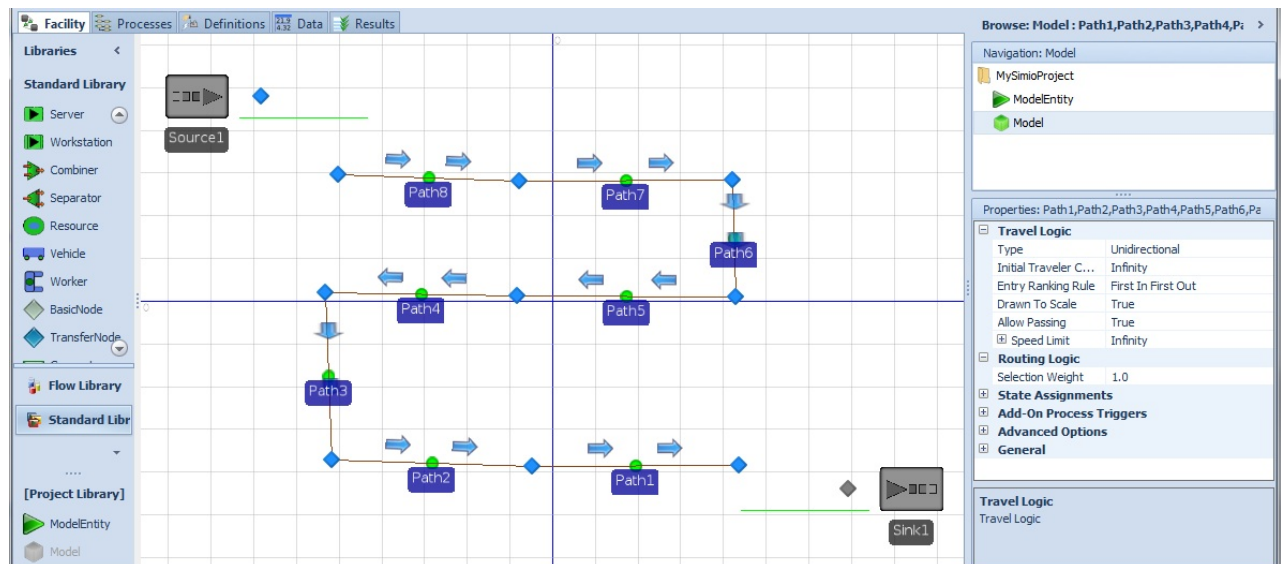
As mentioned above, you can utilize the Standard Library 'link' type objects, such as Path, Conveyor, TimePath, etc. to connect the nodes of various standard objects to one another. Alternatively, you can click on a node while holding down the Ctrl+Shift keys to start a link to another node.

When placing multiple nodes within the Facility window, there is an option for connecting those nodes automatically by using the right-click menu. If the last two or more undo-able actions were placing nodes, the right-click menu within the Facility window provides an 'Add Links to Placed Nodes' menu item that allows links of a certain type to be generated between the nodes that were just placed.

Multiple Nodes Placed Into Facility Window



Paths Automatically Connecting Nodes in the Order That They Were Placed



After the links get created, they are all selected as well, so edits can be made, if desired, within the Properties window on the right.

Note that this only shows up if the last undoable action was placing a node. So if you place 3 nodes, then go create a table, then come back to the Facility window, you *will not* see this option, as the last undoable action was create a table.

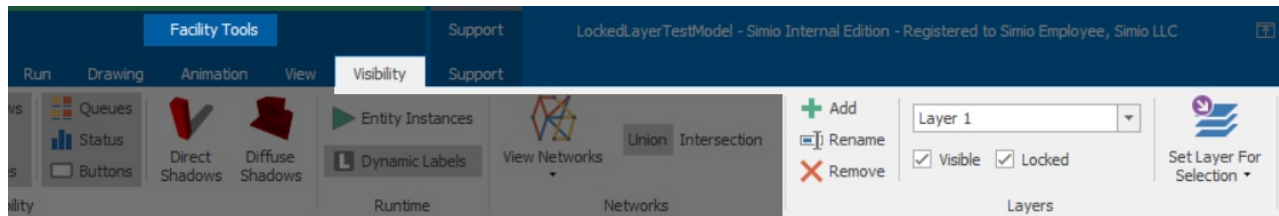
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

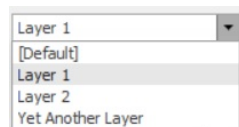
Layers

A Layer is way to display one or more groups of animations, graphics, or objects while hiding other groups with the intention of making it easier to form and view subsets of items in a model.

The main idea is that you can define multiple, named drawing layers in a 3D window and place various objects into different layers. Each object or graphic exists in exactly one Layer. Objects not in a new, named Layer remain in the "Default" Layer.



Use the "Add", "Rename", and "Remove" buttons to manage the Layers themselves. Rename and Remove act on the currently selected "active" Layer. The combo box shows which Layer is currently the "active" Layer.

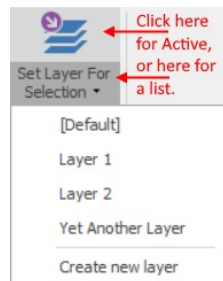


When a Layer is active, you can toggle its Visible state, which shows/hides the contents of that Layer. You can also toggle its Locked state, which is similar to toggling the "Lock for Edits" state on all individual objects in the Layer.

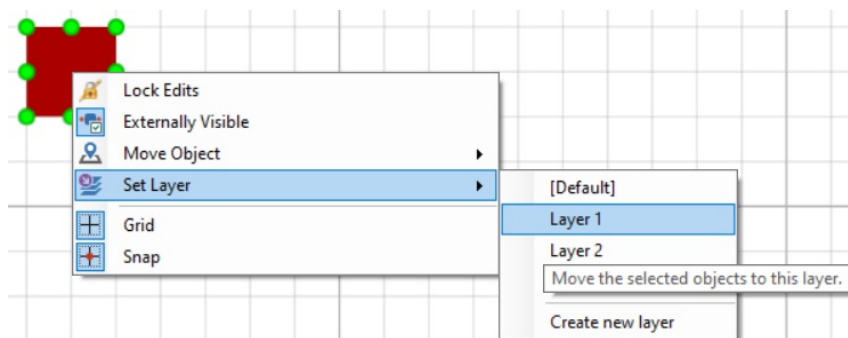
Note: These toggles do not apply to the Default Layer. Default is always visible and never locked.

Newly created objects and graphics are placed in the currently active Layer when they are created or pasted.

To move objects to a different Layer, first select them in the main window. Then use "Set Layer For Selection" to assign the selected objects to a Layer. Clicking the top part of the button moves the selected objects to the active Layer, or click the drop down part of the button to select a specific Layer.



Note: This also shows up on the context menu in the Facility window



There is also a shortcut to move the objects to a new Layer, "Create new layer", which prompts you for the Layer name, then creates the Layer and moves the selected objects to it.

Note: In order to select an object, its Layer must be both Visible and not Locked.

Expression Editor, Functions and Distributions

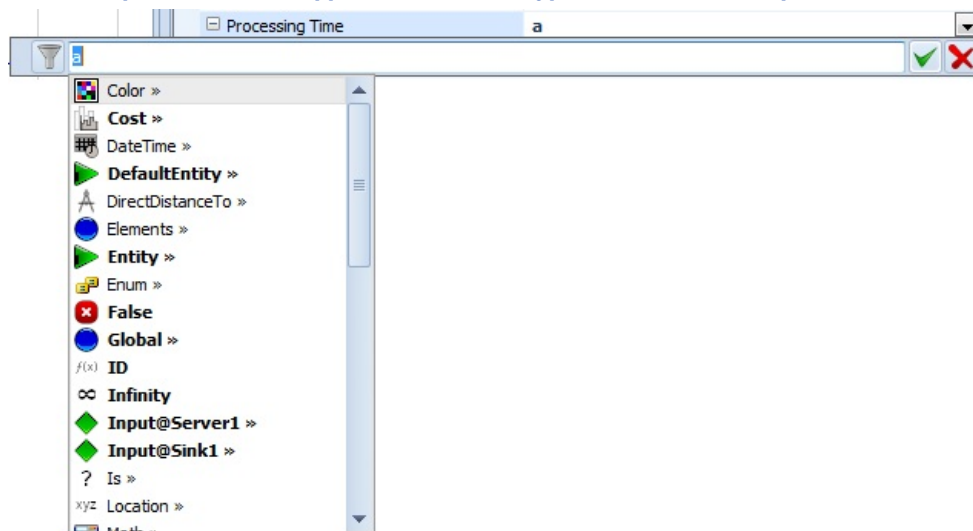
For detailed information on the available **States, Functions and Events** for each object type, see the [Functions](#) page. For detailed information on the available **random distributions**, see the [Distributions](#) page. The [Math Functions](#) page will give details on the available **Math Functions** to use in the Expression Editor. For a listing of various **operators** for use in expressions, see the [Math and Logical Operators](#) page. Various **keywords** that may be used within expressions are found on the [Keywords](#) page. The contents of the Simio **enumerations** can be found in the [Enumerations](#) page and an explanation on when to use the keyword '**Candidate**' in an expression can be found [here](#). Continue reading below for information on how to use our Expression Editor.

The Expression Editor

The Expression Editor is found in the property window for many different properties throughout the Simio product. If the property type utilizes the expression editor, it can be accessed when the user selects the property in the properties window and clicks on the down arrow that appears to the left. A horizontal input box will appear with a green checkbox and red X at the left end of the box. The expression box can be expanded (resized) by using click-drag on the left side of the expression box. Any resizing will remain for all expression editing until the user resizes again. You can also organize your expressions. Using "Ctrl + Enter" will create a new line in the expression. Using "Ctrl + Tab" will create a tab in the expression.

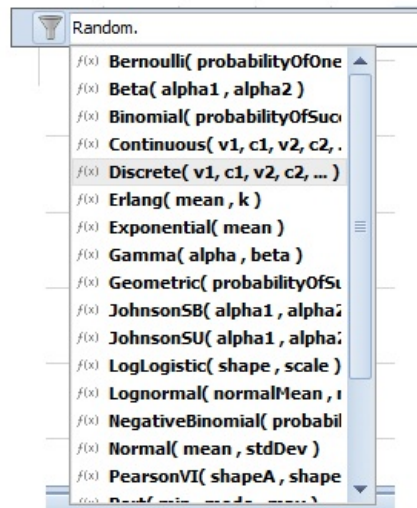
When the user types anything into this input box, a dropdown menu will appear below, showing the user a list of possible entries. There is a content filtering option on the left side of the expression editor dropdown. When the content filtering is enabled (yellow), the auto-completer will hide content that is not typically useful in the current context. For example, within most properties in the Facility window, the content filtering enabled will cause any statistics functions to be taken out of the list. However, within the Experiment window, any expression properties may typically include statistics functions. Therefore, within that context, enabling the filter will hide all content except for the statistics functions. You may enable the filter at any time during the expression building process. By default, the filter is off. The following screen shot shows the choices at the beginning of the list, when the user types an "a" into the input box (no filtering).

The top of the list that appears when a user types an "a" into the Expression Editor



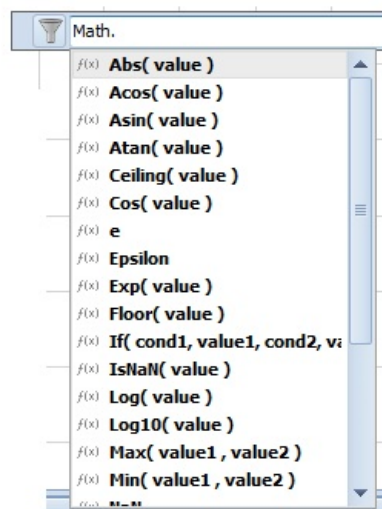
The expression editor can be used to specify a **random distribution**. The user should type in the word Random into the expression editor, followed by a period (.) and they will be given the choice of all the random distributions supported by Simio. To see a list of all distributions found in Simio, see the [Distributions](#) help page.

The top of the list that appears when a user types Random. into the Expression Editor



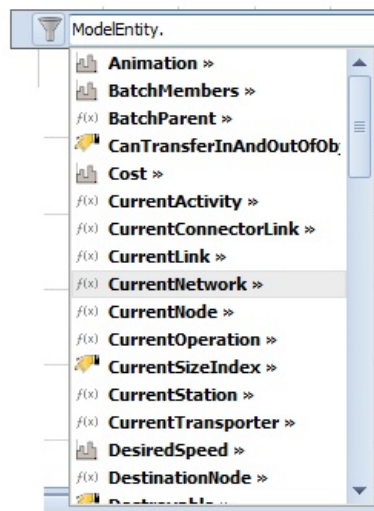
The expression editor can be used to apply **math functions** to any value. The user should type in the word Math into the expression editor, followed by a period (.) and they will be given the choice of all the [math functions](#) supported by Simio.

The top of the list that appears when a user types Math. into the Expression Editor



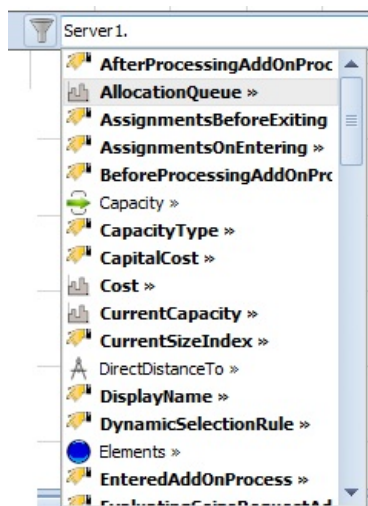
If the user should type in the **name of an object** from the model into the expression editor, followed by a period (.), they will be given the choice of all the functions, properties and states that can be referenced for this particular object. To learn more about functions that are available in Simio, see the [Functions](#) help page. You might also need clarification on what name to put into a function. See the [Object Hierarchy](#) page to learn whether to reference the object definition name, the object instance name or the object runspace (i.e. ModelEntity vs DefaultEntity).

The top of the list that appear when ModelEntity. is entered into the Expression Editor

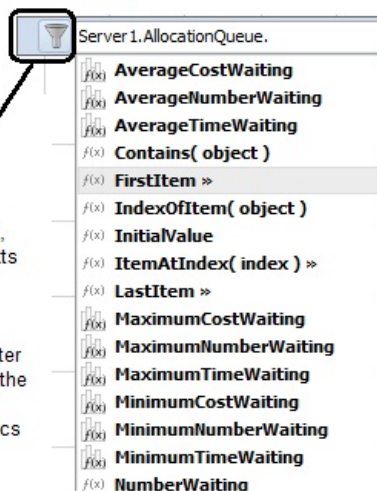


Anything that appears **bold** in the expression editor dropdown, is a valid selection that will return a value. If the selection is not in bold, you need to type a period (.) and add additional information to the expression in order to get a valid response. If a double greater than sign (>>) appears after a name listed in the Expression Editor drop down, this indicates that if this name is selected alone, it is a valid expression. However, there are other functions available with this choice. By selecting the name in the drop down and then typing another period (.), the available functions will appear. See the example below.

Understanding >> in the Expression Editor



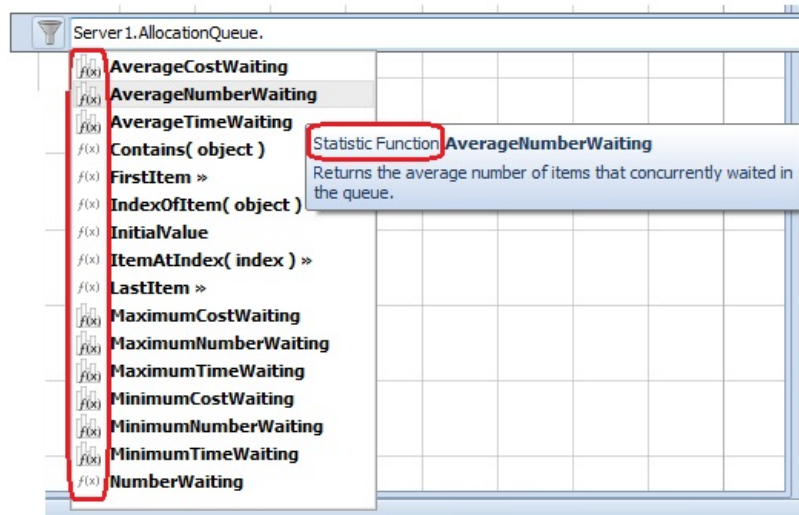
As shown in this example, Server1.AllocationQueue is a valid expression that will return the number of entities currently in the allocation queue for Server1. However, the >> next to the AllocationQueue indicates that there are additional functions that can be used, as shown below.



If this filter is 'yellow', then in many contexts within the Facility window, statistics functions will not appear. When the filter is on (yellow) within the Experiment window filters to only statistics functions.

Note that the symbol next to some of the options below indicates that these are 'statistics functions', such as AverageCostWaiting, AverageNumberWaiting, AverageTimeWaiting, MaximumCostWaiting, etc. A statistic function is any function that returns a value which is cleared whenever system statistics are cleared.

Next to each of the selections within the expression editor is a graphical symbol that indicates if the selection is a property, state, function, statistic function, and so on. The tooltip for the selection also indicates what the selection represents, as shown below.



Flexibility With Buiding Expressions

Simio supports the use of both logical expressions and numeric expressions together. Various [math and logical operators](#), as well as Simio [keywords](#), are available for use. A logical expression is one that compares two terms or expressions and results in a True or False. When used in a mixed (logical and numeric) expression, True is interpreted as 1 and False is interpreted as 0. The following example demonstrates how both types of expressions can be used together in a numeric expression field.

$(\text{PartA.Priority} > 3) * 10$

If PartA.Priority is greater than 3, then this expression will return 10, otherwise it will return 0.

Note: '&&' will only be used as a bit comparison 'And' operator. It will compare the first expression with the second and return a '0' or '1'. If you need to combine strings, use '+'. For example, 'StringState1 + "ABC" + StringState2'.

Referencing Tables within Expressions

Table values can be referenced within an expression by using 'TableName.PropertyName'. When using a table within a larger expression, i.e., 'TableName.PropertyName * XYZ', within a delay type field, the units for the PropertyName value should not be specified within the table. See [Data Tables](#) for more information.

Time Units within Expressions

When a property, that has specified time units, is used within an expression, Simio converts everything to the base time units for calculations. The base time units in Simio is always Hours. Here is an example to aid in the understanding of how Simio does calculations.

If you have a property that you've set the *Unit Type* to 'Time' and you've indicated that the *Default Units* are 'Minutes', if you use this property in an expression, such as 'MyProperty * 4', Simio will convert the value of MyProperty to the base unit of Hours for calculation, even though you've specified the Default Units of MyProperty to be 'Minutes'.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions in Simio - Automatic

Functions Available to use with Objects

There are a number of functions available for [All Intelligent Objects](#). Intelligent objects include all Fixed Resource, Entity, Link, Transporter and Node objects.

There are a number of functions available for [Fixed Resource Objects](#) (i.e. Servers, Combiners, Separators, etc.)

There are a number of functions available for [Entity Objects](#).

There are a number of functions available for [Link Objects](#).

There are a number of functions available for [Transporter Objects](#).

There are a number of functions available for [Node Objects](#). There are some functions that can be used to get state, property, or function information about the destination node's associated object versus the node itself. See the Functions for [Node Objects](#) for additional information.

Math and Random Distribution Functions Available

By typing "Math." into the Expression editor, you can see all the available Math functions to use in Simio. For more information on the available options, see the [Math Functions](#) page. By typing "Random." into the Expression editor, you can see all the available Random Distributions that are available in Simio. For more information on the available distributions, see the [Distributions](#) page.

Other Functions Available in Simio (Table Related, Network, Queue State, List States, String States, Tally Statistics, Run and Miscellaneous)

Listed below are the **Table Related** Functions available:

Function	Return Value
Entity.CurrentJobStep	Returns the entity's row index into its assigned sequence table (if any).
Entity.NumberJobSteps	Returns the total number of rows in the entity's assigned sequence table (if any)
TableName.NumericProperty.Sum	Returns the number of available rows for the table, given the current table selection context.
TableName.AvailableRowCount	Returns the sum of the numeric property column's values in a specified range of rows based on an optional true or false condition.

Listed below are the **Network** Functions available:

Function	Return Value
NetworkName.Distance(fromNode, toNode)	Returns the shortest travel distance between two nodes using this network

Functions available for the Simio state variable type **Queue State**:

Function	Return Value
NumberWaiting	Returns the current number of entities waiting in this Queue.
AverageNumberWaiting	Returns the average number of entities waiting in this Queue.
AverageTimeWaiting	Returns the average time that an entity waits in this Queue.

MaximumNumberWaiting	Returns the maximum number of entities waiting in this Queue.
MaximumTimeWaiting	Returns the maximum time that an entity waits in this Queue.
MinimumNumberWaiting	Returns the minimum number of entities waiting in this Queue.
MinimumTimeWaiting	Returns the minimum time that an entity waits in this Queue.
MinimumCostWaiting	Returns the minimum cost that an item accumulated while it waited in the queue.
MaximumCostWaiting	Returns the maximum cost that an item accumulated while it waited in the queue.
AverageCostWaiting	Returns the average cost that an item accumulated while it waited in the queue.
FirstItem	Returns a reference to the item ranked first in the queue.
LastItem	Returns a reference to the item ranked last in the queue.
ItemAtIndex(index)	Returns a reference to the item ranked at a specific index position in the queue.
IndexOfItem(object)	Returns the rank position of a specified object in the queue. If the queue does not contain the object, then the value 0 is returned.
Contains(object)	Returns True(1) if the queue contains the specified object. Otherwise the value False (0) is returned.
TimeWaiting (object)	Returns the time that a specified object has been waiting in the queue. If the queue does not contain the object then NaN is returned.

Functions available for the Simio State variable type **List States**:

Function	Return Value
AverageTime(stateValue)	Returns the average time that the object has spent in this Resource State.
NumberOccurrences(stateValue)	Returns the number of times that this object has gone into this Resource State.
PercentTime(stateValue)	Returns the percentage of the total simulation time that this object has been in this Resource State.
TotalTime(stateValue)	Returns the total amount of time that this object has been in this Resource State.

Therefore, to reference the percentage of time that an object was in a certain state, use the syntax [ObjectName].PercentTime(stateValue), where stateValue is the number referencing the state in the String List. The first value in the String List is value 0.

Functions available for the Simio **String States**:

Function	Return Value
String.Length(string)	Returns the length of a string
String.Contains(string, substring)	Returns true if substring is in string, otherwise false
String.Substring(string, startindex[, length])	Returns the string starting at the specified index of the string until the end. If the length argument is specified, then returns a string of that length starting at the index.
String.FromReal(value[,	Returns the string representation of a specified numeric value.

<code>formatString, useLocalCultureInfo)</code>	<p>The numeric value may be converted to its equivalent string representation using a specified numeric format string. The <code>formatString</code> argument should contain either a valid standard numeric format specifier (see .NET Standard Numeric Format Strings) or any combination of custom numeric format specifiers (see .NET Custom Numeric Format Strings).</p> <p>The <code>useLocalCultureInfo</code> argument indicates whether to convert the numeric value to a string using local culture-specific format information. If this argument is false or unspecified, then the formatting used will be culture-insensitive (invariant culture).</p>
<code>String.ToReal(string [, useLocalCultureInfo)</code>	<p>Returns a numeric value converted from a specified string representation of a number.</p> <p>The <code>useLocalCultureInfo</code> argument indicates whether to convert the string to a numeric value using local culture-specific format information. If this argument is false or unspecified, then the formatting used will be culture-insensitive (invariant culture).</p>
<code>String.FromDateTime(dateTime [, formatString, useLocalCultureInfo)</code>	<p>Returns the string representation of a specified numeric datetime (simulation time) value.</p> <p>The datetime value may be converted to its equivalent string representation using a specified date and time format string. The <code>formatString</code> argument should contain either a single format specifier character (see .NET Standard Date and Time Format Strings) or a custom format pattern (see .NET Custom Date and Time Format Strings).</p> <p>The <code>useLocalCultureInfo</code> argument indicates whether to convert the datetime value to a string using local culture-specific format information. If this argument is false or unspecified, then the formatting used will be culture-insensitive (invariant culture).</p>
<code>String.ToDateTime(dateTime [, useLocalCultureInfo)</code>	<p>Returns a numeric datetime (simulation time) value converted from a specified string representation of a date and time.</p> <p>The <code>useLocalCultureInfo</code> argument indicates whether to convert the string to a datetime value using local culture-specific format information. If this argument is false or unspecified, then the formatting used will be culture-insensitive (invariant culture).</p>
<code>String.Format(formatstring, arg1, arg2, ...)</code>	<p>Returns a formatted string using the specified format string with any number of parameters. The <code>formatString</code> argument should contain a format pattern that follows .NET string formatting conventions.</p>
<code>String.Compare(string1, string2 [, useLocalCultureInfo)</code>	<p>Compares two specified strings and returns 0 if the two strings are equal, -1 if the first string is less than (alphabetically before) the second string, or 1 if the first string is greater than (alphabetically after) the second string.</p> <p>The <code>useLocalCultureInfo</code> argument indicates whether to use local culture-specific information to influence the string comparison. If this argument is false or unspecified, then the comparison will be culture-insensitive (invariant culture).</p>
<code>String.CompareIgnoreCase(string1, string2 [, useLocalCultureInfo)</code>	<p>Compares two specified strings, ignoring case, and returns 0 if the two strings are equal, -1 if the first string is less than (alphabetically before) the second string, or 1 if the first string is greater than (alphabetically after) the second string.</p> <p>The <code>useLocalCultureInfo</code> argument indicates whether to use local culture-specific information to influence the string comparison. If this argument is false or unspecified, then the comparison will be culture-insensitive (invariant culture).</p>
<code>String.CompareSequenceNumbers(sequenceNumber1, sequenceNumber2)</code>	<p>Compares two specified sequence number strings and returns 0 if the two sequence numbers have no implied dependency relationship, -1 if an item assigned the first sequence number must come before an item assigned the second sequence number, or 1 if an item assigned the first sequence number must come after an item assigned the second sequence number.</p>
<code>String.ToUpper(string</code>	<p>Returns a copy of a specified string converted to uppercase.</p>

[useLocalCultureInfo])	The useLocalCultureInfo argument indicates whether to use the local culture-specific casing rules. If this argument is false or unspecified, then the casing rules used will be culture-insensitive (invariant culture).
String.ToLower(string [useLocalCultureInfo])	Returns a copy of a specified string converted to lowercase. The useLocalCultureInfo argument indicates whether to use the local culture-specific casing rules. If this argument is false or unspecified, then the casing rules used will be culture-insensitive (invariant culture).
String.Trim(string)	Returns a specified string with any leading and trailing white-space characters removed.
String.NewLine	Returns the newline constant defined for the local platform. Useful for adding a line break to form a multi-line string.
String.IndexOf	Returns the one-based index of the first occurrence of a specified substring within the string. If the substring is not found, then the value 0 is returned.
String.FromList	Returns the string representation of a specified integer constant in the specified string list.
String.Replace(string, substring, replacementString)	Returns a copy of the specified string where all instances of a specified substring are replaced with another specified string.
String.Json.Count(string, path)	Returns the number of elements in an array found in a specified JSON formatted string using the given path. If the given path is not found, returns 0. The JSON path is a string made up of property names and array indexes separated by periods, e.g. Manufacturers[0].Name.
String.Json.Value(string, path)	Returns the string representation of a value found in a specified JSON formatted string using the given path. If the given path is not found, returns an empty string. The JSON path is a string made up of property names and array indexes separated by periods, e.g. Manufacturers[0].Name.
String.RegEx.Contains(string, pattern)	Returns True (1) if a substring matching a specified regular expression pattern is found in the specified string. Otherwise, the value False (0) is returned.
String.RegEx.Replace(string, pattern, replacementString)	Returns a copy of the specified string where all substrings matching a specified regular expression pattern are replaced with another specified string.

String functions in the RegEx namespace use Regular expression patterns, which are sequences of characters used to search a string. An example pattern string is "\d", which would match the first digit (0-9) in a string. The pattern "A\d{2}" would match the strings "A02", "A16", and "A93", but not "A1" or "B26". Many different operators or constructs can be used to match strings. These are described in Microsoft's Regular Expression Quick Reference, which can be found at the link below.

String functions in the RegEx namespace use Regular expression patterns, which are sequences of characters used to search a string. An example pattern string is "\d", which would match the first digit (0-9) in a string. The pattern "A\d{2}" would match the strings "A02", "A16", and "A93", but not "A1" or "B26". Many different operators or constructs can be used to match strings. These are described in Microsoft's Regular Expression Quick Reference, which can be found at the link below.

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference>

Simio uses the default RegEx options, which are described in detail at the link below.

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-options#default-options>

Note: Strings can be combined with '+'. See String States or Math and Logical Operators for more information.

Functions available for the Simio element **TallyStatistic**:

Function	Return Value
----------	--------------

Average	Returns the average value of the observations recorded.
Maximum	Returns the maximum value from the observations recorded.
Minimum	Returns the minimum value from the observations recorded.
LastRecordedValue	Returns the value of the last observation recorded.
HalfWidth	Returns the confidence interval half width around the average of the observations recorded.
NumberObservations	Returns the total number of observations recorded.
MinimumTimeWaiting	Returns the minimum time that an entity waits in this Queue.

Functions available for the **Simulation Run**:

Function	Return Value
Run.EndingTime	Returns the time in hours that the simulation run is scheduled to end. If the run's ending time is defaulted to 'Infinite', then this function returns the largest possible floating-point value.
Run.ExperimentName	Returns the name of the experiment if running in experiment mode. Returns an empty string if running in interactive mode.
Run.Mode	Returns the mode that the model is running in. Numeric values returned by this function are Interactive Mode = 1, Experiment Mode = 2, Operational Planning Mode = 3, or Generate Training Data = 5. (Note: Operational Planning Mode is for RPS software only.)
Run.ModelName	Returns the name of the main running model.
Run.ProjectName	Returns the name of the project containing the main running model.
Run.RandomnessDisabled	Returns whether random sampling in the simulation run is disabled.
Run.ReplicationNumber	Returns the scenario replication number that the run corresponds to. In interactive mode, the default replication number is 1, but can be changed within the Advanced Options button.
Run.ReplicationsRequired	Returns the scenario replications required if running in experiment mode. Returns 0 if running in interactive mode.
Run.ScenarioName	Returns the name of the experiment scenario if running in experiment mode. Returns an empty string if running in interactive mode.
Run.TimeNow	Returns the current simulation time in hours.
Run.WarmUpPeriod	Returns the time period in hours after the beginning of the run at which statistics are to be cleared. Applies to runs in experiment mode only.
Run.ResultSetID	Returns a unique identifier for the results produced by a run. Separate identifiers exist for interactive, plan or experiment runs, each of which is assigned a new value whenever the corresponding model, plan or experiment is reset.
Run.ResultSetTimeStampUtcString	Returns the DateTime that the current ResultSetID was created. This is not a simulation date and time. It is the actual date and time, expressed as

	Coordinated Universal Time (UTC).
Run.FreezePeriod	Returns the time period in hours after the beginning of the run at which time the 'freeze period' is to be over. RPS licensing only.
Run.RunNumber	Returns a one-based run count, incremented if a run restart occurs using the RestartRun step.
Run.TimeNowUtc	Returns the current simulation time in hours, expressed as Coordinated Universal Time (UTC). For example, if the model is running using a simulation clock with time zone Eastern Time (US & Canada) and the current simulation time in hours is 1 then this function will return 5 during daylight savings time or 6 during standard time.
Run.TimeNowUtcOffset	Returns the offset in hours between the simulation clock's time zone and Coordinated Universal Time (UTC) for the current simulation time. For example, if the model is running using a simulation clock with time zone Eastern Time (US & Canada) then this function will return -4 during daylight savings time or -5 during standard time.
Functions available for DateTime :	
Function	Return Value
DateTime.Year(dateTime)	Returns the year component, expressed as an integer between 1 and 9999, for the specified numeric datetime (simulation time) value.
DateTime.Month(dateTime)	Returns the month component, expressed as an integer between 1 and 12, for the specified numeric datetime (simulation time) value.
DateTime.DayOfWeek(dateTime)	Returns the day of the week, expressed as an integer between 1 (Sunday) and 7 (Saturday), for the specified numeric datetime (simulation time) value.
DateTime.DayOfMonth(dateTime)	Returns the day of the month, expressed as an integer between 1 and 31, for the specified numeric datetime (simulation time) value.
DateTime.DayOfYear(dateTime)	Returns the day of the year, expressed as an integer between 1 and 366, for the specified numeric datetime (simulation time) value.
DateTime.Hour(dateTime)	Returns the hour component, expressed as an integer between 0 and 23, for the specified numeric datetime (simulation time) value.
DateTime.Minute(dateTime)	Returns the minute component, expressed as an integer between 0 and 59, for the specified numeric datetime (simulation time) value.
DateTime.Second(dateTime)	Returns the second component, expressed as an integer between 0 and 59, for the specified numeric datetime (simulation time) value.
DateTime.DaysInMonth(month, year)	Returns the number of days in the specified month and year. Specify the month as a number ranging from 1 to 12 and the year as a number ranging from 1 to 9999.
DateTime.SystemNow	Returns the current datetime from the computer (actual time, not simulation time), expressed in the computer's current timezone.
DateTime.SystemNowUtc	Returns the current datetime from the computer (actual time, not simulation time), expressed as Coordinated Universal Time (UTC).
DateTime.FromString(string [,	Returns a numeric datetime (simulation time) value converted from a

useLocalCultureInfo))	specified string representation of a date and time. The useLocalCultureInfo argument indicates whether to convert the string to a datetime value using local culture-specific format information. If this argument is false or unspecified, then the formatting used will be culture-insensitive (invariant culture).
DateTime.ToString(dateTime [, formatString, useLocalCultureInfo])	Returns the string representation of a specified numeric datetime (simulation time) value. The datetime value may be converted to its equivalent string representation using a specified date and time format string. The formatString argument should contain either a single format specifier character (see .NET Standard Date and Time Format Strings) or a custom format pattern (see .NET Custom Date and Time Format Strings). The useLocalCultureInfo argument indicates whether to convert the datetime value to a string using local culture-specific format information. If this argument is false or unspecified, then the formatting used will be culture-insensitive (invariant culture).

To compare only the Date value of a datetime use the function DateTime.ToString(DateTime Value, "yyyy-MM-dd"). Both values of the comparison need to be string values.

Listed below are the **Miscellaneous** Functions available:

Function	Return Value
LocationAt.Geographic (latitude, longitude)	Converts a latitude and longitude to an X,Y,Z location in the model, relative to the current latitude and longitude of the model's origin. The result of the function is a Location, with Z and X components that correspond to the latitude and longitude, respectively. The range of valid values for latitude is -90.0 to +90.0, and for longitude is -180.0 to +180.0. If either value is out of range, the result is a Location consisting of all NaN values. See also the RelocateObject step.
LocationAt.Cartesian(x,y,z)	Gets the geographic (latitude, longitude) location point for the specified Cartesian(x,y,z) coordinates. To set the geographic coordinates of the origin (0, 0, 0) in the model's world space, go to View tab -> Map group -> Set Map Location.
Color.FromRGB (R, G, B)	Returns a Color value using the specified Red, Green and Blue color channel arguments.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions, States and Events for All Intelligent Objects

Intelligent Objects Functions, Events and States

Simio is based on Intelligent Objects. The intelligent objects are built by modelers and then may be reused in multiple modeling projects. Objects can be stored in libraries and easily shared. A beginning modeler may prefer to use pre-built objects from libraries; however the system is designed to make it easy for even beginning modelers to build their own intelligent objects for use in building hierarchical models. Simio's Standard Library objects are all derived Intelligent Objects and therefore include the following functions, states and events.

Where along the X axis is this entity currently located?

ModelEntity.Location.X

Listed below are the Functions available for all Intelligent Objects:

Function	Return Value
Name	Returns the dynamic name of the object in String Format (e.g., "Path2", "Output@Source1" or "Vehicle1[1]")
ID	Returns the ID number of this object.
Location.Latitude	Returns the projected latitude of this object's current location.
Location.Longitude	Returns the projected longitude of this object's current location.
Location.Parent	Returns a reference to the parent object that contains or is associated with the object's current location.
Location.X	The current location of this object along the X axis of the Simio grid. When the object is an entity and is batched to a parent entity, the X location of the parent entity is returned by this function. When the object is an entity and is riding on a transporter, the X location of the transporter is returned by this function. When this object is a Link, the current location of the beginning node of the link is returned. When this object is a Fixed object, Simio uses the center of the object's symbol.
Location.Y	The current location of this object along the Y axis (vertical) of the Simio grid. When the object is an entity and is batched to a parent entity, the Y location of the parent entity is returned by this function. When the object is an entity and is riding on a transporter, the Y location of the transporter is returned by this function. When this object is a Link, the current location of the beginning node of the link is returned. When this object is a Fixed object, Simio uses the center of the object's symbol.

Location.Z	The current location of this object along the Z axis of the Simio grid. When the object is an entity and is batched to a parent entity, the Z location of the parent entity is returned by this function. When the object is an entity is riding on a transporter, the Z location of the transporter is returned by this function. When this object is a Link, the current location of the beginning node of the link is returned. When this object is a Fixed object, Simio uses the center of the object's symbol.
DirectDistanceTo.Object(object)	Returns the direct (straight-line) distance from this object to another specified object.
DirectDistanceTo.Location(x,y,z)	Returns the direct (straight-line) distance from this object to a specified coordinate location.
Elements.NumberItems	Returns the number of direct child elements contained within this parent object.
Elements.FirstItem	Returns a reference to the first element in the list of direct child elements contained within this parent object.
Elements.LastItem	Returns a reference to the last element in the list of direct child elements contained within this parent object.
Elements.ItemAtIndex(index)	Returns a reference to the element at a specified index position in the list of direct child elements contained within this parent object.
Elements.IndexOfItem(element)	Returns the one-based index of a specified element in the list of direct child elements contained within this parent object. If the element is not a direct child element of this parent then the value 0 is returned.
Elements.Contains(element)	Returns True (1) if the specified element is a direct child element of this parent object. Otherwise, the value False (0) is returned.
SeizedResources.NumberItems	Returns the current number of resources seized and owned by this object.
SeizedResources.FirstItem	Returns a reference to the first resource in the list of resources currently seized and owned by this object.
SeizedResources.LastItem	Returns a reference to the last resource in the list of resources currently seized and owned by this object.
SeizedResources.ItemAtIndex(index)	Returns a reference to the resource at a specified index position in the list of resources currently seized and owned by this object.
SeizedResources.IndexOfItem(resource)	Returns the index of the first occurrence of a specified resource in the list of resources currently seized and owned by this object. If the resource has not been seized then the value 0 is returned.
SeizedResources.CapacityOwnedOf(resource)	Returns the total number of capacity units of a

	specified resource that are currently seized and owned by this object.
SeizedResources.Contains(resource)	Returns True(1) if the list of resources currently seized and owned by this object includes the specified resource. Otherwise the value False (0) is returned.
SeizedResources.RequestedDestinationNodeFor(resource)	Returns a reference to the requested destination node for a specified resource in the list of resources currently seized by this object. If the resource is not found or was not requested to move when seized then the Nothing keyword is returned.
SeizedResources.AggregateEfficiency(type)	Calculates and returns an aggregate efficiency value for the list of resources currently seized by the object. The aggregate type is an integer argument. Possible values: 0 = None, 1 = Average, 2 = Count, 3 = Maximum, 4 = Minimum, 5 = Sum. Note that if the aggregate type is None or if there are no seized resources with defined efficiency values, then the value NaN is returned.

Listed below are the Functions available for all Intelligent Objects that have External Nodes (i.e.Input and Output nodes):

Function	Return Value
Input	Returns a reference to the external input node associated with the object.
Output	Returns a reference to the external output node associated with the object.

Listed below are the Functions available for Intelligent Objects that are Enabled as Resource Objects *:

Function	Return Value
Capacity	Returns the current capacity of the object.
Capacity.Initial	Returns the initial capacity of the object.
Capacity.Previous	Returns the previous capacity of the object.
Capacity.Remaining	Returns the current unallocated capacity of the object.
Capacity.Allocated	Returns the current number of capacity units of the object that are allocated (have been seized).
Capacity.Allocated.Average	Returns the average number of capacity units of the object that have been allocated during the run.
Capacity.Allocated.Minimum	Returns the minimum number of capacity units of the object that have been allocated during the run.
Capacity.Allocated.Maximum	Returns the maximum number of capacity units of the object that have been allocated during the run.
Capacity.Allocated.Total	Returns the total number of capacity units of the object that have been allocated during the run.
Capacity.AllocatedTo(owner)	Returns the current number of capacity units of this resource that are allocated to (have been seized by) a specified owner object.

Capacity.Reserved	Returns the current number of capacity units of the resource that have been reserved. Keep in mind that the function <i>ReservedTransporter</i> for entity objects is not related to the Reserve step or this function.
Capacity.ReservedTo(owner)	Returns the current number of capacity units of the resource that have been reserved for use by a specified owner object. Keep in mind that the function <i>ReservedTransporter</i> for entity objects is not related to the Reserve step or this function.
Capacity.Average	Returns the average scheduled capacity of the object during the run.
Capacity.Minimum	Returns the minimum scheduled capacity of the object during the run.
Capacity.Maximum	Returns the maximum scheduled capacity of the object during the run.
Capacity.Next	Returns the next scheduled capacity of this resource. *Note: If the resource is not following a work schedule then this function simply returns the current capacity since any future capacity changes are unknown.
Capacity.TimeOfLastChange	Returns the simulation time (in hours) that the capacity of this resource most recently changed.
Capacity.TimeOfNextChange	Returns the simulation time (in hours) of the next scheduled capacity change of this resource. *Note: If the resource is not following a work schedule then this function simply returns infinity since the timing of any future capacity changes are unknown.
Capacity.TimeSinceLastChange	Returns the elapsed time duration (in hours) since the most recent capacity change of this resource.
Capacity.TimeUntilNextChange	Returns the time duration remaining (in hours) until the next scheduled capacity of this resource. *Note: If the resource is not following a work schedule then this function simply returns infinity since the timing of any future capacity changes are unknown.
Capacity.ScheduledUtilization	Returns the percent utilization of this resource's scheduled capacity during the run, calculated by returning the value of the expression '100 * (Capacity.Utilized.Average / Capacity.Average)'.
Capacity.Utilized	Returns the current number of capacity units of this resource that are allocated and considered utilized. The resource's ResourceState variable determines which state values are considered utilized.
Capacity.IdleCost	Returns the total cost that was accrued to the cost of the resource due to idle capacity units.
Capacity.UsageCostCharged	Returns the total cost that was charged to users of the resource's capacity.
CurrentWorkSchedule.Name	Returns the name of the currently assigned work schedule.
ResourceOwners	The objects that currently own (have seized) capacity units of this resource.
ResourceOwners.NumberItems	Returns the number of objects that currently own (have seized) capacity units of this resource.

ResourceOwners.FirstItem	Returns a reference to the first owner in the list of owners that have currently seized capacity units of this resource.
ResourceOwners.LastItem	Returns a reference to the last owner in the list of owners that have currently seized capacity units of this resource.
ResourceOwners.ItemAtIndex(index)	Returns a reference to the owner at a specified index position in the list of owners that have currently seized capacity units of this resource.
ResourceOwners.IndexOfItem(owner)	Returns the one-based index of the first occurrence of a specified owner in the list of owners that have currently seized capacity units of this resource. If the owner has not seized the resource then the value 0 is returned.
ResourceOwners.Contains(owner)	Returns True(1) if the list of owners that have currently seized capacity units of this resource includes the specified owner. Otherwise the value False (0) is returned.
ResourceOwners.AllProcessingSuspended	Returns True (1) if each owner of the resource has at least one associated process token currently suspended. Otherwise, the value False (0) is returned.
ReservationOwners	Provides functions for accessing the objects that currently have reservations for capacity units of this resource.
ReservationOwners.NumberItems	Returns the number of objects that currently have reservations for capacity units of this resource.
ReservationOwners.FirstItem	Returns a reference to the first object in the list of objects that currently have reservations for capacity units of this resource.
ReservationOwners.LastItem	Returns a reference to the last object in the list of objects that currently have reservations for capacity units of this resource.
ReservationOwners.ItemAtIndex(index)	Returns a reference to the object at a specified index position in the list of objects that currently have reservations for capacity units of this resource.
ReservationOwners.IndexOfItem(owner)	Returns the one-based index of the first occurrence of a specified object in the list of objects that currently have reservations for capacity units of this resource. If the object has not reserved the resource then the value 0 is returned.
ReservationOwners.Contains (owner)	Returns True (1) if the objects that currently have reservations for capacity units of this resource include the specified object. Otherwise, the value False (0) is returned.

* Note: Standard Library Objects that are enabled as 'Resource Objects' include Node, Link, Vehicle, Operator and fixed objects such as Server, Combiner, Separator, etc.

Listed below are the Inherited States of all Intelligent Objects:

State	Description
Trace Flag	A flag to turn on/off trace information for this object. A value of 0 denotes off, all other values denote on.
Size	The current volume size of this object in cubic meters. Has state parameters Size.Length, Size.Width, and Size.Height.

Cost	The current cost of this object in the default currency units. Each intelligent object has a cost 'level' state that gets/sets the total cost that has been allocated to the object. The value of the state may increase continuously over time based on the value of its Cost.Rate parameter. It is a user-assignable state.
------	---

AllocationQueue	The queue of objects that are waiting to seize this object.
-----------------	---

CurrentCapacity	State that may be assigned to dynamically change the current capacity of this object.
-----------------	---

Listed below are the Inherited Events associated with all Intelligent Objects:

Event	Description
CapacityChanged	Fired when the resource capacity for this object has been changed by a work schedule.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions, States and Events for Entity Objects

Examples for using Functions with Entity Objects

Note: If you would like information on a dynamic object in the system, begin the function with the name of the object instance, not the object definition. For example, DefaultEntity is an example of an object instance and ModelEntity is the object definition. If you want to find out information about a specific instance, you might need to first get a token pointing to that particular instance so Simio knows which instance you are referring to. For example, first use a Search Step to find the ObjectInstance named PartA and then in the token that follows the Found segment of the Search Step, will be associated with PartA so you can use a function such as PartA.Location.X or PartA.NumberInSystem

- How many PartA type entity objects are currently in the system?

PartA.Population.NumberInSystem

- How many Customers (entity named Customer) have arrived into the system?

Customer.Population.NumberCreated

- Is this entity (LargeBox) currently on a Transporter? (returns 1 for True and 0 for False)

LargeBox.IsRiding

- Which Transporter is the entity currently riding on? (returns Nothing if the entity is not on a transporter)

PartB.CurrentTransporter (Note: This requires an 'attached' status label to the entity)

- Where is this entity (Patient) heading?

Patient.DestinationNode

IMPORTANT NOTE: Entities inherit all Functions, States and Events for Intelligent Objects, plus have additional ones listed below.

Listed below are the **Functions** available for any object derived from Simio base class - Entity object:

Function	Return Value
EntityType	Returns a reference to the entity type of this entity.
Population.NumberInSystem	Returns the current number of entities of this entity's population type in the system.
Population.NumberInSystem.Minimum	Returns the minimum number of entities of this entity's population type in the system.
Population.NumberInSystem.Maximum	Returns the maximum number of entities of this entity's population type in the system.
Population.NumberInSystem.Average	Returns the average number of entities of this entity's population type in the system.
TimeInSystem	Returns the time duration (in hours) that this entity has been in the system (is equal to Run.TimeNow - TimeCreated).
Population.TimeInSystem.Minimum	Returns the minimum time (in hours) that an entity of this entity's population type was in the system.
Population.TimeInSystem.Maximum	Returns the maximum time (in hours) that an entity of this

	entity's population type was in the system.
Population.TimeInSystem.Average	Returns the average time (in hours) that an entity of this entity's population type was in the system.
Population.TimeInSystem.NumberOfObservations	Returns the total number of time in system observations recorded for the entity's population type during the run.
TimeCreated	Returns the time that the entity was created in hours.
Population.NumberCreated	Returns the total number of entities of this entity's population type that have been created.
Population.NumberDestroyed	Returns the total number of entities of this entity's population type that have been destroyed.
Population.Capacity	Returns the sum of the current scheduled capacity for all entities of this entity's population type in the system.
Population.Capacity.Remaining	Returns the sum of the current unallocated capacity for all entities of this entity's population type in the system.
Population.Capacity.Allocated	Returns the sum of the current capacity that has been allocated for all entities of this entity's population type in the system.
Population.Capacity.Allocated.Average	Returns the average of Population.Capacity.Allocated during the run.
Population.Capacity.Allocated.Minimum	Returns the minimum of Population.Capacity.Allocated during the run.
Population.Capacity.Allocated.Maximum	Returns the maximum of Population.Capacity.Allocated during the run.
Population.Capacity.Allocated.Total	Returns the total number of capacity units for all entities of this entity's population type that have been allocated during the run.
Population.Capacity.Average	Returns the average of Population.Capacity during the run.
Population.Capacity.Minimum	Returns the minimum of Population.Capacity during the run.
Population.Capacity.Maximum	Returns the maximum of Population.Capacity during the run.
Population.Capacity.Utilized	Returns the sum of the current utilized capacity for all entities of this entity's population type in the system.
Population.Capacity.Utilized.Average	Returns the average of Population.Capacity.Utilized during the run.
Population.Capacity.Utilized.Minimum	Returns the minimum of Population.Capacity.Utilized during the run.
Population.Capacity.Utilized.Maximum	Returns the maximum of Population.Capacity.Utilized during the run.
Population.Capacity.ScheduledUtilization	Returns the percent utilization of the total scheduled capacity for all entities of this entity's population type during the run.

Population.CostPerItem.Average	Returns the average cost that was accumulated by destroyed entities of this entities's population type.
Population.CostPerItem.Minimum	Returns the minimum cost that was accumulated by destroyed entities of this entities's population type.
Population.CostPerItem.Maximum	Returns the maximum cost that was accumulated by destroyed entities of this entities's population type.
Population.CostPerItem.NumberOfObservations	Returns the total number of cost per item observations recorded for the entity's population type during the run.
Population.Index	Returns the one-based index of this entity in the population that the entity is a member of.
Population.FirstItem	Returns a reference to the first member in the current population of this entity's type.
Population.LastItem	Returns a reference to the last member in the current population of this entity's type.
Population.ItemAtIndex(index)	Returns a reference to the member at a specified index position in the current population of this entity's type.
Population.IndexOfItem(object)	Returns a one-based member index of a specified object in the current population of this entity's type. If the population does not contain the entity then the value 0 is returned.
Population.Contains(entity)	Returns True(1) if the population contains the specified entity. Otherwise the value False (0) is returned.
Sequence.CurrentJobStep	Returns the entity object's current one-based step index into the entity's assigned sequence table.
Sequence.NumberJobSteps	Returns the total number of steps in the entity's currently assigned sequence.
Sequence.NumberJobStepsRemaining	Returns the current number of steps remaining in the entity's assigned sequence, including the current job step.
Sequence.ExpectedOperationTimeRemaining	Returns the sum of the expected operation times for the remaining steps in the entity's assigned sequence, including the current job step.
Sequence.CriticalRatio	Returns an index for determining how much the entity's assigned sequence is considered to be on schedule given the due date and expected operation time remaining. A value of 1.0 is 'on schedule'. A value less than 1.0 is behind, while a value larger than 1.0 is ahead of schedule. This function returns a value equivalent to the expression $\frac{(\text{Entity.Sequence.DueDate} - \text{Run.TimeNow})}{\text{Entity.Sequence.ExpectedOperationTimeRemaining}}$.
Sequence.SlackTime	Returns the expected amount of time that would be left until due date after completing the entity's assigned sequence, if the sequence's remaining operation time was started now. This function returns a value equivalent to the expression $(\text{Entity.Sequence.DueDate} - \text{Run.TimeNow}) - \text{Entity.Sequence.ExpectedOperationTimeRemaining}$.

Sequence.SlackTimePerOperation	Returns the average slack time per remaining operation for the entity's assigned sequence. This function returns a value equivalent to the expression 'Entity.Sequence.SlackTime / Entity.Sequence.NumberJobStepsRemaining'.
Sequence.DueDate	Returns a due date for the entity to complete its assigned sequence. The 'Due Date Expression' property for the entity type is used to get the due date value.
Sequence.ModifiedDueDate	Returns a modified due date value for the entity's assigned sequence, which is the highest of either the due date or the expected completion date if the sequence's remaining operation time was started now. This function returns a value equivalent to the expression 'Math.Max(Entity.Sequence.DueDate, Run.TimeNow + Entity.Sequence.ExpectedOperationTimeRemaining)'.
Sequence.DestinationNodes	The destination nodes in the entity's assigned sequence.
Sequence.DestinationNodes.NumberItems	Returns the number of destination nodes in the entity's assigned sequence.
Sequence.DestinationNodes.FirstItem	Returns a reference to the first destination node in the entity's assigned sequence.
Sequence.DestinationNodes.LastItem	Returns a reference to the last destination node in the entity's assigned sequence.
Sequence.DestinationNodes.ItemAtIndex(index)	Returns a reference to the destination node at a specified step index position (or job step) in the entity's assigned sequence.
Sequence.DestinationNodes.IndexOfItem(node)	Returns the one-based step index (or job step) of the first occurrence of a specified destination node in the entity's assigned sequence. If the entity does not have an assigned sequence then the value 0 is returned.
Sequence.DestinationNodes.Contains(node)	Returns True(1) if the entity's assigned sequence contains the specified destination node. Otherwise the value False (0) is returned.
SlipSpeed	The entity's speed on a conveyor with possible slippage based on a slower traffic entity in front.
TotalDistanceTraveled	Returns the total distance that has been traveled by the entity, irrespective of how the travel movements were performed (free space or on a network).
IsParked	Returns true if entity is currently parked.
IsTravelingInReverse	Returns true if the entity is currently traveling in reverse. Otherwise, the value false is returned. This function will return true if an entity is traveling in forward orientation on a network and then does a turnaround on a bidirectional link using the <i>Network Turnaround Method</i> 'Reverse'. It will also return true if an entity is traveling in free space with a negative movement rate.
IsBatchMember	Returns True if the entity is currently a batch member in another entity's BatchMembers queue.
BatchParent	If the entity object is a batch member, then this function

	returns a reference to the parent entity owning the batch. Otherwise, the Nothing keyword is returned.
IsRiding	Returns true if the entity is currently riding on a transporter.
ReservedTransporter	<p>If the entity object currently has made a reservation to ride on a transporter, then this function returns a reference to that transporter. Otherwise, the Nothing keyword is returned. This value is set to the reserved transporter until the entity is actually loaded and has moved into the transporter's RideStation.</p> <p>Keep in mind that this function is not related to the Reserve and UnReserve steps or the <i>Capacity.Reserved</i> and <i>Capacity.ReservedTo(owner)</i> functions.</p>
CurrentTransporter	If the entity object currently is currently riding on a transporter, then this function returns a reference to that transporter. Otherwise, the Nothing keyword is returned.
DestinationNode	Returns a reference to the entity object's current destination node, or the Nothing keyword if the entity does not have a destination.
CurrentNetwork.Distance(fromNode, toNode)	Returns the shortest travel distance between two nodes using the entity's current network.
NetworkDistanceTo.Node(node)	Returns the shortest travel distance from the entity object's current location to a specified node using the entity's current network.
NextEntityAheadOnLink	If the entity object's leading edge is on a link, then this function returns a reference to the next entity on the same link whose trailing edge is ahead of that leading edge. If there is no entity ahead then the Nothing keyword is returned.
NextEntityBehindOnLink	If the entity object's trailing edge is on a link, then this function returns a reference to the next entity on the same link whose leading edge is behind that trailing edge. If there is no entity behind then the Nothing keyword is returned.
NetworkDistanceTo.NextEntityAheadOnLink	If the entity object's leading edge is on a link, then this function returns the distance from the entity's leading edge to the trailing edge of the next entity ahead on the same link. If there is no entity ahead then the value Infinity is returned.
NetworkDistanceTo.NextEntityBehindOnLink	If the entity object's trailing edge is on a link, then this function returns the distance from the entity's trailing edge to the leading edge of the next entity behind on the same link. If there is no entity behind then the value Infinity is returned.
CurrentNode	If the entity object's leading edge is currently at a node or the entity is located in a station owned by a node, then this function returns a reference to that node. Otherwise the Nothing keyword is returned.
PreviousNode	Returns a reference to the most recent node at which the entity object was located, not including its current node if presently located at one.
CurrentLink	If the entity object's leading edge is currently on or at the end

	of a link, then this function returns a reference to that link. Otherwise the Nothing keyword is returned.
LastLinkUsed	Returns the most recent link used by the entity to complete a travel or instant transfer (in the case of a connector link) into a new node location. The value returned by this function is updated whenever the entity transfers from the end of a link into a node, or transfers across a connector from one node into another node.
TrailingLink	If the entity object's trailing edge is currently on or at the end of a link, then this function returns a reference to that link. Otherwise the Nothing keyword is returned.
CurrentConnectorLink	If the entity object is currently using one or more connector links to transfer between two node locations, then this function returns a reference to the forwardmost connector being used. Otherwise the Nothing keyword is returned. Note that a connector link does not have a logical length and thus is not considered to be physically occupied by an entity that is using it. To get a reference to the link that an entity's leading edge is currently physically located on, use the 'CurrentLink' function.
CurrentStation	If the entity object is currently located in a station, then this function returns a reference to that station. Otherwise the Nothing keyword is returned.
CurrentRoutingGroup	If the entity object is currently waiting in a route request queue, then this function returns a reference to the routing group being used. Otherwise, the Nothing keyword is returned.
Queuing.IsWaitingResourceAllocationQueue	Returns True (1) if the entity is currently waiting in the AllocationQueue of a resource. Otherwise, the value False (0) is returned.
Queuing.IsWaitingMaterialAllocationQueue	Returns True (1) if the entity is currently waiting in the AllocationQueue of a material. Otherwise, the value False (0) is returned.
Queuing.IsWaitingStationEntryQueue	Returns True (1) if the entity is currently waiting in the EntryQueue of a station. Otherwise, the value False (0) is returned.
Queuing.IsWaitingLinkEntryQueue	Returns True (1) if the entity is currently waiting in the EntryQueue of a link. Otherwise, the value False (0) is returned.
Queuing.IsWaitingNodeEntryQueue	Returns True (1) if the entity is currently waiting in the EntryQueue of a node. Otherwise, the value False (0) is returned.
Queuing.IsWaitingRidePickupQueue	Returns True (1) if the entity is currently waiting in the RidePickupQueue of a node to be picked up by a transporter. Otherwise, the value False (0) is returned.
Queuing.IsWaitingRouteRequestQueue	Returns True (1) if the entity is currently waiting in the RouteRequestQueue of a routing group element to be assigned a destination. Otherwise, the value False (0) is

	returned.
Queuing.IsWaitingBatchLogicParentQueue	Returns True (1) if the entity is currently waiting in the ParentQueue of a batch logic element to collect a batch of other entities. Otherwise, the value False (0) is returned.
Queuing.IsWaitingBatchLogicMemberQueue	Returns True (1) if the entity is currently waiting in the MemberQueue of a batch logic element to be added as a member to a batch. Otherwise, the value False (0) is returned.
CurrentStation	If the entity object is currently located in a station, then this function returns a reference to that station. Otherwise the Nothing keyword is returned.
Orientation.Direction.X	Returns the X component of the unit vector indicating the current direction the object is facing.
Orientation.Direction.Y	Returns the Y component of the unit vector indicating the current direction the object is facing.
Orientation.Direction.Z	Returns the Z component of the unit vector indicating the current direction the object is facing.
Orientation.Roll	Returns the current roll angle in degrees, measured clockwise from the +Y, of the object in the X-Y plane.
PlannedPath.Links.Contains(link)	Returns True (1) if the entity's planned network path contains the specified link. Otherwise, the value False (0) is returned.
PlannedPath.Links.FirstItem	Returns a reference to the first link on the entity's planned network path.
PlannedPath.Links.IndexOfItem(link)	Returns the one-based index of a specified link on the entity's planned network path. If the entity does not have a planned network path then the value 0 is returned.
PlannedPath.Links.ItemAtIndex(index)	Returns a reference to the link at a specified one-based index position on the entity's planned network path.
PlannedPath.Links.LastItem	Returns a reference to the last link on the entity's planned network path.
PlannedPath.Links.NumberItems	Returns the number of links on the entity's planned network path.
StockRequirements.Quantity	The total quantity in the entity's list of stock requirements to putaway or remove.
StockRequirements.UnreservedQuantity	The total remaining quantity in the entity's list of stock requirements for which to reserve storage bin capacity or stock.
StockRequirements.Reserved	Indicates whether sufficient storage bin capacity or stock has been reserved for putaway or removal to fully satisfy the entity's list of stock requirements.
StockRequirements.HitCount	Returns the total unreserved quantity in the entity's list of stock requirements or the total quantity that can currently be added to or removed to or removed from a specified storage area or bin given the specified hitcount type, whichever is

	smaller.
StockRequirements.NumberItems	The number of requirements in the entity's list of stock requirements.
StockRequirements.FirstItem	The first requirement in the entity's list of stock requirements.
StockRequirements.LastItem	The last requirement in the entity's list of stock requirements.
StockRequirements.ItemAtIndex	Returns the requirement at the specified one-based index position in the entity's list of stock requirements.
StockRequirements.IndexOfItem	Returns the one-based index of the specified requirement in the entity's list of stock requirements, if found. Otherwise, returns 0.
Listed below are the Inherited States of any object derived from Simio base class - Entity object:	
State	Description
Picture	A utility state to hold a picture index. This state can then be referenced in an expression for Current Symbol Index.
Size	The current volume size of the object in cubic meters. Has state parameters Size.Length, Size.Width, and Size.Height.
Movement	The total linear distance that this object has traveled in free space or on its current network link. Has state parameters Movement.Rate, Movement.X, Movement.Y, Movement.Z, Movement.Pitch, Movement.Heading, and Movement.Roll.
DesiredSpeed	The desired movement rate of this object. Non-zero does not mean the object is currently moving.
Priority	The current priority value for this object.
CellRange	The cell alignment distance for this object.
VisitRequestQueue	The queue of destination visit requests for this object.
BatchMembers	The queue of batch members that this object owns.
Volume	The current logical volume of the object. At runtime, the ratio between the logical volume and the Size implicit volume will remain constant.
Weight	The current logical weight of the object.
CurrentNetwork	The entity object's currently assigned network.
CurrentTravelMode	The current travel mode for the entity. Possible values include FreeSpaceOnly (0), NetworkOnly (1), or NetworkIfPossible (2).
Listed below are the Inherited Events associated with any object derived from Simio base class - Entity object:	
Event	Description
Destroyed	The Destroyed event provides notification that the entity has been destroyed.
Engaged	The Engaged event provides notification that the entity has been engaged (locked) to a location on a link.

Transferring	The Transferring event provides notification that the entity has started transferring into another object or a station location.
Transferred	The Transferred event provides notification that the entity has completed transferring into another object or a station location.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions, States and Events for Transporter Objects

Examples for using Functions with Transporter Objects

Note: Transporters are a dynamic population of moveable unit resources. Both Workers and Vehicles are considered Transporter Objects. Within each of these, an Initial Number In System may be defined. For each of the initial number in system, the specific unit may be referenced by using the instance name followed by brackets, such as Vehicle1[1] to refer to the first of the Vehicle1 objects. To refer to the entire set of objects, you may use just the name of the object in the function, such as Vehicle1.Size.Length.

- How do I animate the resource state of a particular vehicle?

`VehicleName[1].ResourceState`

- How many entities are riding on the second of my two Vehicle1 objects?

`Vehicle1[2].NumberRiders`

- What is the length size of the vehicle group called Vehicle1?

`Vehicle1.Size.Length`

- What is the destination node of the third worker in my group of workers called Operator?

`Operator[3].DestinationNode`

IMPORTANT NOTE: Transporters inherit all [Functions, States and Events for Intelligent Objects](#), as well as [Functions, States and Events for Entities](#), plus have additional ones listed below.

Listed below are the [Functions](#) available for any object derived from the Simio base object class - Transporter:

Function	Return Value
<code>RideStation.Capacity.Remaining</code>	Returns the remaining ride capacity of the transporter.
<code>RideStationLoad</code>	Returns the current 'load' on the station location that holds the entities riding on the transporter. The ride station 'load' is defined as the sum of current entities reserving space on and intending to ride the transporter plus the current number of riders.
<code>RideStationOverload</code>	Returns the current difference between the load and capacity values (a positive difference indicating an 'overload') for the station location that holds the entities riding the transporter. The ride station 'load' is defined as the sum of current entities reserving space on and intending to ride the transporter plus the current number of riders. This function returns the difference between the load and current ride capacity (Overload = Load - Capacity).
<code>NumberRiders</code>	Returns the number of riders currently on the transporter.
<code>NumberRiders.Loading</code>	Returns the current number of riders being loaded onto the transporter by a Pickup step.
<code>NumberRiders.Unloading</code>	Returns the current number of riders being unloaded onto the transporter by a Dropoff step.

Listed below are the [Inherited States](#) available for any object derived from the Simio base object class - Transporter:

State	Return Value
HomeNode (on the standard Vehicle object)	The name of the Node that is set as the Home node for this object. This state is assignable.
ResourceState (on the standard Vehicle object)	The current resource state of this object (numeric values for this state are Idle = 0, Busy = 1, Failed = 3, OffShift = 4).

Listed below are the Inherited Events associated with any object derived from Simio base class - Transporter object:

Event	Description
RiderLoaded (on the standard Vehicle object)	The RiderLoaded event provides notification to the vehicle's OnVisitingNode process that a rider has been loaded.
RiderUnloaded (on the standard Vehicle object)	The RiderUnloaded event provides notification to the vehicle's OnVisitingNode process that a rider has been unloaded.
CalendarTimeBasedFailuresReset (on the standard Vehicle object)	An occurrence of this event will reset the 'CalendarTimeBasedFailures' timer.
EventCountBasedFailuresReset (on the standard Vehicle object)	An occurrence of this event will reset the 'EventCountBasedFailures' timer.
Allocated (on the standard Vehicle object)	The Allocated event provides notification to the vehicle's OnVisitingNode process that capacity of this Vehicle object has been allocated.
Released (on the standard Vehicle object)	The Released event provides notification to the vehicle's OnVisitingNode process that capacity of this Vehicle object has been released.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions, States and Events for Objects Enabled as Resource Objects

Examples for using Functions with Fixed Resource Objects

- How many units of Server3 are currently allocated to some object in the system?

Server3.Capacity.Allocated

- How many units of capacity of Workstation1 are currently available to be seized/allocated?

Workstation1.Capacity.Remaining

- Is this object based on a Fixed object class? (returns 1 for True and 0 for False)

Is.Fixed

- How do I determine the ID of the object Resource1?

Resource1.ID

- Can I manually change the capacity of an object, Server1, using an Assign step?

Yes, by assigning the state of the object, Server1.CurrentCapacity, in an Assign step. This will change the value that is returned by the Server1.Capacity function. NOTE: This is an alternative to specifying a schedule for an object.

IMPORTANT NOTE: Entities inherit all **Functions, States and Events for Intelligent Objects**, plus have additional ones listed below.

Listed below are the **Functions** available for Fixed Resource Objects, such as the standard Servers, Separators, Combiners, Resources and Workstations (Deprecated):

Function	Return Value
NumberInSystem	Returns the current number in system.
NumberInSystem.Minimum	Returns the minimum number in system.
NumberInSystem.Maximum	Returns the maximum number in system.
NumberInSystem.Average	Returns the average number in system.
Is	Used with '.ClassName' or '.InstanceName' for logic based on object characteristics.
Inventory(material)	Returns a reference to the inventory holding the specified material at this fixed object. If there is no such inventory defined, then the Nothing keyword is returned.
ExpectedSetupTimeFor(entity)	Returns an estimate of the expected setup time for a specified entity at the fixed object. NOTE: Refer to the <i>ExpectedSetupTimeExpression</i> property in Advanced Options for the fixed object to specify the expression used to estimate a setup time for an entity.
ExpectedOperationTimeFor(entity)	Returns an estimate of the expected operation time for a specified entity at the fixed object.

NOTE: Refer to the *ExpectedOperationTimeExpression* property in the Advanced Options of the fixed object to specify the expression used to estimate an operation time for an entity.

HasActiveChangeover	Returns True (1) if there is an active changeover occurrence at the resource. Otherwise, the value False (0) is returned.
---------------------	---

Listed below are the Inherited States of any object derived from Simio base class - Fixed object:

State	Description
ResourceState	The current resource state of this object (numeric values for this state are dependent on the type of object and can be found within the List States page).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions, States and Events for Node Objects

Examples for using Functions with Node Objects

- How many travelers are on their way to BasicNode1 (i.e. have their destination node set to this node)?

BasicNode1.NumberTravelers.RoutingIn

- How many entities are currently located at the node named "WaitingArea"?

WaitingArea.NumberTravelers

For any external input node, a new set of functions are now provided to return some basic information on the node's associated object, such as the capacity, load, or overload. Such information on the input locations of input nodes is often useful or required to implement dynamic routing logic. For example, in the TransferNode object in the Standard Library, when setting an entity's next destination using the Select From List option, the new functions are useful in selecting the best destination node from a list of candidates.

- I'd like to know the capacity of the InputBuffer of Server3, by using a function for Input@Server3.

Input@Server3.AssociatedStation.Capacity

- How many entities are en route to Input@Combiner1, intending to enter the stations of Combiner1, plus the current entities already arrived to the node but still waiting to enter the stations, plus the current entities occupying the stations.

Input@Combiner1.AssociatedStationLoad

Functions available for any object derived from the Simio base object class Node:

Function	Return Value
IsExternalNode	Returns whether the node is an external node.
IsInputNode	Returns 'True' if this node is an external input node for entering another object.
IsOutputNode	Returns 'True' if this node is an external output node for exiting another object.
Nearest.Node	Returns the node that is the shortest straight-line distance from this node. Note that, if this node is an external node, then any other external nodes attached to the same associated object are excluded from consideration as the nearest node.
Nearest.InputNode	Returns the external input node that is the shortest straight-line distance from this node. Note that, if this node is an external node, then any other external nodes attached to the same associated object are excluded from consideration as the nearest node.
Nearest.OutputNode	Returns the external output node that is the shortest straight-line distance from this node. Note that, if this node is an external node, then any other external nodes attached

	to the same associated object are excluded from consideration as the nearest node.
NumberTravelers	Returns the current number of travelers occupying this node.
NumberTravelers.RoutingIn	Returns the current number of travelers with their destination set to this node.
NumberTravelers.RoutingInToEnterAssociatedObject	For an external input node, this function returns the current number of travelers with their destination set to this node and which can enter the node's associated object.
InboundLinks.NumberItems	Returns the number of drawn inbound links into this node.
InboundLinks.FirstItem	Returns a reference to the first link in the collection of drawn inbound links into this node.
InboundLinks.LastItem	Returns a reference to the last link in the collection of drawn inbound links into this node.
InboundLinks.ItemAtIndex(index)	Returns a reference to the link at a specified index position in the collection of drawn inbound links into this node.
InboundLinks.IndexOfItem(link)	Returns the one-based index of a specified link in the collection of drawn inbound links into this node. If the link is not an inbound link into this node, then the value 0 is returned.
InboundLinks.Contains(link)	Returns True(1) if the collection of drawn inbound links into this node contains the specified link. Otherwise the value False (0) is returned.
OutboundLinks.NumberItems	Returns the number of drawn outbound links into this node.
OutboundLinks.FirstItem	Returns a reference to the first link in the collection of drawn outbound links into this node.
OutboundLinks.LastItem	Returns a reference to the last link in the collection of drawn outbound links into this node.
OutboundLinks.ItemAtIndex(index)	Returns a reference to the link at a specified index position in the collection of drawn outbound links into this node.
OutboundLinks.IndexOfItem(link)	Returns the one-based index of a specified link in the collection of drawn outbound links into this node. If the link is not an outbound link into this node, then the value 0 is returned.
OutboundLinks.Contains(link)	Returns True(1) if the collection of drawn outbound links from this node contains the specified link. Otherwise the value False (0) is returned.
OutboundLinks.SelectItemFor(entity)	For a specified entity, returns a reference to a drawn outbound link from the node using the node's outbound link selection-related properties to make the selection decision.
AssociatedObject	For an external node, returns a reference to the associated object that may be entered or exited using the node.

AssociatedStation	<p>For an external input node, this function returns a reference to the immediate station location inside the node's associated object that may be entered using the node. If there is no station location, then the Nothing keyword is returned.</p> <p>Example: For a Server 'Input' node, Input@Server1.AssociatedStation.Capacity function returns the current capacity of either the server's input buffer or processing station (if the server has no input buffer).</p>
AssociatedStationLoad	<p>For an external input node, this function returns the current 'load' on the station locations inside the node's associated object that may be entered using the node</p> <p>The associated station 'load' is defined as the sum of current entities en route to the node intending to enter the stations, plus the current entities already arrived to the node but still waiting to enter the stations, plus the current entities occupying the stations.</p>
AssociatedStationOverload	<p>For an external input node, this function returns the current difference between the load and capacity values (a positive difference indicating an 'overload') for the station locations inside the node's associated object that may be entered using the node.</p> <p>The associated station 'load' is defined as the sum of current entities en route to the node intending to enter the stations, plus the current entities already arrived to the node but still waiting to enter the stations, plus the current entities occupying the stations. This function returns the difference between that load and the current station capacity (Overload = Load - Capacity).</p>
AssociatedStationHasSpaceFor(Entity)	<p>For an external input node, this function returns True (1) if the node's associated station has space available for the specified entity. Otherwise, the value False (0) is returned. Note that the value returned by this function is determined by looking at the current 'load' on the station locations inside the node's associated object that may be entered using the node. The associated station 'load' is defined as the sum of current entities en route to the node intending to enter the stations, plus the current entities already arrived to the node but still waiting to enter the stations, plus the current entities occupying the stations. This function returns True (1) if that load is interpreted to be less than the current station capacity values. Otherwise, the value False (0) is returned.</p>

Note: The functions AssociatedStationOverload, AssociatedStationLoad, and AssociatedStationHasSpaceFor(Object) all include the capacity of the AssociatedStation and the capacity of the stations it redirects to, if any.

Inherited States available for any object derived from the Simio base object class Node:

State	Description
EntryQueue	The queue of entities that are waiting to enter this node.
RidePickupQueue	The queue of entities that are waiting at this node to be picked up by and ride on a transporter.
CurrentTravelerCapacity	State used to get or set the current maximum number of travelers that may simultaneously occupy this node.

Inherited Events available for any object derived from the Simio base object class Node:

Event	Description
Entered	The Entered event provides notification that a traveler has entered the crossing point of the node. (on the standard TransferNode and BasicNode)
Exited	The Exited event provides notification that a traveler has exited the crossing point of the node.(on the standard TransferNode and BasicNode)
RiderWaiting	The RiderWaiting event provides notification that a traveler has entered the RidePickupQueue of the node waiting for a ride. (on the standard TransferNode)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Functions, States and Events for Link Objects

Listed below are the Functions available for any object derived from Simio base class - Link object:

Function	Return Value
NumberTravelers	Returns the current number of travelers on the link.
NumberTravelers.Minimum	Returns the minimum number of travelers that have been on the link.
NumberTravelers.Maximum	Returns the maximum number of travelers that have been on the link.
NumberTravelers.Average	Returns the average number of travelers that have been on the link.
NumberTravelers.Entered	Returns the total number of travelers that have entered the link.
NumberTravelers.Exited	Returns the total number of travelers that have exited the link.
NumberTravelers.Accumulated	<p>Returns the total number of travelers that are accumulated on the link.</p> <p>Note: An entity is counted as an 'Accumulated' traveler on a link if the entity has reached the end of the link and has been stopped there without being engaged to the link OR if the entity's leading edge has collided with the trailing edge of an entity in front of it on the link, and the entity has accumulated behind that entity without being engaged to the link. Therefore, the NumberTravelers.Accumulated function is tracking accumulation anywhere on the link, not just from the end. Once flagged as 'Accumulated', the entity will continue to be considered 'Accumulated' until either its leading edge leaves the link or the collision situation is cleared. If a link is a 'Connector' (i.e., drawn to scale with logical length = 0) then entities are never considered to be on the link and NumberTravelers and NumberTravelers.Accumulated are always 0.</p>

Accumulated Entities

NumberTravelers.Accumulated = 6 (the accumulated entities are marked as 'A')



NumberTravelers.Accumulated.Minimum	Returns the minimum number of accumulated travelers on the link.
NumberTravelers.Accumulated.Maximum	Returns the maximum number of accumulated travelers on the link.
NumberTravelers.Accumulated.Average	Returns the average number of accumulated travelers on the link.
TimeOnLink.Minimum	Returns the minimum time that a traveler was on the link.
TimeOnLink.Maximum	Returns the maximum time that a traveler was on the link.
TimeOnLink.Average	Returns the average time that a traveler was on the link.
CurrentDirection	Returns the current direction of traffic on the link.
StartingNode	Returns a reference to the node located at the drawn starting point of

	this link.
EndingNode	Returns a reference to the node located at the drawn ending point of this link.
CrossSectionalArea	Returns the cross-sectional area of the link in square meters.
Vertices.NumberItems	Returns the number of vertices in the link. Note that these are the "interior vertices", meaning the end points of the link are excluded, since those are already available via the nodes at each end. Therefore, a straight link between two nodes would return 0 for NumberItems.
Vertices.LocationOfItemAtIndex(index).X	Returns the current location of the specified vertex along the X axis of the Simio grid.
Vertices.LocationOfItemAtIndex(index).Y	Returns the current location of the specified vertex along the Y axis (vertical) of the Simio grid.
Vertices.LocationOfItemAtIndex(index).Z	Returns the current location of the specified vertex along the Z axis of the Simio grid.
PlannedPathLoad	PlannedPathLoad Returns the current number of entities with a planned path that includes this link.

Listed below are the Inherited States of any object derived from Simio base class - Link object:

State	Description
TransferInState	State indicating whether an entity is currently transferring onto this link (Idle = 0, Busy = 1).
DesiredSpeed	The desired movement rate of the 'Mark' on this link, which is a reference point along the link's length whose movement represents the link's movement. Non-zero does not indicate that the link 'Mark' is currently moving.
Movement	The position of the 'Mark' on this link, which is a reference point along the link's length whose movement represents the link's movement. This state variable has a parameter that returns the current Rate (speed) of the mark.
EntryQueue	The queue of entities that are waiting to enter this link.
Contents	State used to access the queue of entities that are physically located on the link. * Note that they are ranked in order of <i>entry</i> onto the link.
DistanceFromFirstEdgeToEnd	The distance from the leading or trailing edge of the first entity on this link to the end of the link.
DesiredDirection	The desired traffic direction on this link. Assignable values are Enum.TrafficDirection.Either, Enum.TrafficDirection.Forward, Enum.TrafficDirection.Reverse, or Enum.TrafficDirection.None.
CurrentVolumeFlowIn	State used to get the current total volume flowed into this link. The 'rate' from this state is also available.
CurrentVolumeFlowOut	State used to get the current total volume flowed out of this link. The 'rate' from this state is also available.
CurrentWeightFlowIn	State used to get the current total weight flowed into this link. The 'rate' from this state is also available.

CurrentWeightFlowOut	State used to get the current total weight flowed out of this link. The 'rate' from this state is also available.
AvailableVolumeFlowInRate	Gets the current potential volume inflow rate for the link if ideal case where there are no link-related flow constraints. Note that this is a private variable primarily intended for flow link object builders (and thus will only be visible inside a link object's process view).
AvailableWeightFlowInRate	Gets the current potential weight inflow rate for the link if ideal case where there are no link-related flow constraints. Note that this is a private variable primarily intended for flow link object builders (and thus will only be visible inside a link object's process view).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Obsolete Functions

Obsolete Functions

Listed below are the Functions that are obsolete, meaning replaced by current functions, but may still be used in older models.:

Obsolete Function	Replaced by Current Function
ActualDurationFor(entity)	CurrentBatchDurationFor(entity)
ActualSetupTime	ExpectedSetupTimeFor(entity)
BatchesRequiredFor(entity)	NumberBatchesRequiredFor(entity)
BatchParentID	BatchParent.ID
Capacity.UnitsOwned	Capacity.AllocatedTo(owner)
Capacity.CurrentOwners	ResourceOwners
Capacity.CurrentOwners.NumberItems	ResourceOwners.NumberItems
Capacity.CurrentOwners.FirstItem	ResourceOwners.FirstItem
Capacity.CurrentOwners.LastItem	ResourceOwners.LastItem
Capacity.CurrentOwners.ItemAtIndex(index)	ResourceOwners.ItemAtIndex(index)
Capacity.CurrentOwners.IndexOfItem(owner)	ResourceOwners.IndexOfItem(owner)
CurrentActivity	
CurrentOperation	
CurrentLinkID	CurrentLink.ID
CurrentNodeID	CurrentNode.ID
DestinationNodeID	DestinationNode.ID
DisableRandomness	Run.RandomnessDisabled
FrontTraffic	EntityAheadOnLink != Nothing
FrontTraffic.ID	EntityAheadOnLink.ID
FrontTraffic.Distance	Math.If(EntityAheadOnLink != Nothing, Max(EntityAheadOnLink.Movement – EntityAheadOnLink.Length – Movement, 0), CurrentLink.Length – Movement)
FrontTraffic.Speed	Math.If(EntityAheadOnLink != Nothing, EntityAheadOnLink.Movement.Rate, DesiredSpeed)
InputCapacity	InputLocation.Capacity

InputLocation	AssociatedStation For example, InputLocation.Capacity is replaced with AssociatedStation.Capacity
InputLocation.Load	AssociatedStationLoad
InputLocation.OverLoad	AssociatedStationOverLoad
InputLocation.NumberInLocation	AssociatedStation.Contents
InputLocation.NumberWaitingEntry, InputNumberWaiting	AssociatedStation.EntryQueue.NumberWaiting
InputLocation.CapacityRemaining, InputRemainingCapacity	AssociatedStation.Capacity.Remaining
IsExternalInputNode	IsInputNode
IsWaitingRide	Queuing.IsWaitingRidePickupQueue
MaterialCostCharged	UsageCostCharged
NumberParked	ParkingStation.Contents
NumberOccupiedLinks	
NumberOccupiedNodes	
NumberRouting	NumberRoutingIn
NumberRoutingIn	NumberTravelers.RoutingIn
NumberTokensInProcess	TokensInProcess.NumberItems
NumberTransportersRouting	NumberRoutingIn.Transporters
NumberRoutingIn.Transporters	*Use Load/Overload functions
NumberRoutingIn.CanEnterAssociatedObject	*Use Load/Overload functions
ParentRunSpaceID	Location.Parent.ID
Population.Name	EntityType.Name
RideID	ReservedTransporter.ID or CurrentTransporter.ID
SeizedResources.CapacityUnitsOwned(resource)	SeizedResources.CapacityOwnedOf(resource)
SequenceTableRow	Sequence.CurrentJobStep
SequenceTableRowCount	Sequence.NumberJobSteps
TableRow	Sequence.CurrentJobStep
TaskInfo	Task
TaskInfo.IDNumber	Task.IDNumber
TaskInfo.Name	Task.Name

TaskInfo.SeizedResources.NumberItems	Task.SeizedResources.NumberItems
TaskInfo.SeizedResources.FirstItem	Task.SeizedResources.FirstItem
TaskInfo.SeizedResources.LastItem	Task.SeizedResources.LastItem
TaskInfo.SeizedResources.IndexOfItem(resource)	Task.SeizedResources.IndexOfItem(resource)
TaskInfo.SeizedResources.ItemAtIndex(index)	Task.SeizedResources.ItemAtIndex(index)
TaskInfo.SeizedResources.Contains(resource)	Task.SeizedResources.Contains(resource)
TaskInfo.SeizedResources.CapacityOwnedOf(resource)	Task.SeizedResources.CapacityOwnedOf(resource)
TaskInfo.TimeStarted	Task.TimeStarted
TimeNow	Run.TimeNow

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Distributions

Distributions

Simio supports a number of distributions used to draw random samples. Simio supports an infinite number of random number streams. To specify a particular stream for a distribution to use, list the number of the stream directly after the last parameter proceeded by a comma. For example, `Random.Triangular(.1, .2, .3, 2)` to use stream number 2. When running a model from an experiment, replications of the same scenario will use different portions of the stream and different scenarios will start from the same point in the stream.

To access these distribution from within the [Expression Editor](#), type in the word `Random`, followed by a period (`.`) to see the list of distributions.

- [Bernoulli Distribution](#)
- [Beta Distribution](#)
- [Binomial Distribution](#)
- [Continuous Distribution](#)
- [Discrete Distribution](#)
- [Erlang Distribution](#)
- [Exponential Distribution](#)
- [Gamma Distribution](#)
- [Geometric Distribution](#)
- [JohnsonSB Distribution](#)
- [JohnsonUB Distribution](#)
- [LogLogistic Distribution](#)
- [LogNormal Distribution](#)
- [NegativeBinomial Distribution](#)
- [Normal Distribution](#)
- [PearsonVI Distribution](#)
- [Pert Distribution](#)
- [Poisson Distribution](#)
- [Triangular Distribution](#)
- [Uniform Distribution](#)
- [Weibull Distribution](#)

NOTE: Simio will produce an error if a delay time expression returns a negative value. This can occur if you use an unbounded distribution that might produce negative values, such as a Normal Distribution, within a Delay.

Simio produces an error to alert you that your distribution would have a spike at time 0.0 and the mean would be higher than what you specified. If you want to use an unbounded distribution and convert all negative values to 0.0 (ignoring the above problems), then use an expression like: `Math.Max(0.0, Random.Normal(5.0, 2.0))`

Examples

1. How do I generate values for which:

- 10% of the time I get values between 0 and 25
- 35% of the time I get values between 25 and 50
- 55% of the time I get values between 50 and 100

`Random.Discrete(x, .1, y, .45, z, 1)` where `x = Random.Uniform(0, 25)`, `y = Random.Uniform(25,50)`, `z=Random.Uniform(50,100)`

In other words: `Random.Discrete((Random.Uniform(0, 25)), .1, (Random.Uniform(25,50)), .45, (Random.Uniform(50,100)), 1)`

Distribution Fitting Software

The following are recommended solutions for statistical distribution fitting (e.g., Input Analysis):

Stat::Fit from Geer Mountain Software (<http://www.geerms.com/>) has direct support for Simio built into their latest

version. They also have a free student version available.

ExpertFit from Averill Law and Associates (<http://www.averill-law.com/>) has a comprehensive tool kit and direct support for Simio in their latest version.

Because the above packages have direct support for Simio (e.g., their output exactly matches and can be cut and pasted into a Simio expression) they are the easiest to use. The following packages will provide a similar data analysis, but the results must be manually translated to match Simio's required inputs:

BestFit is part of the @Risk Excel-based toolkit from Palisade Software (<http://www.palisade.com/>).

Note, here is a conversion chart for BestFit submitted by Barry Nelson:

<u>If BestFit gives...</u>	<u>Use this in Simio</u>
Erlang(k, β)	Erlang($\beta*k$, k)
Binomial(n, p)	Binomial(p, n)
LogLogistic(γ , β , α)	LogLogistic(α , β) + γ
Lognorm2(μ , σ)	LogNormal(μ , σ)
Negbin(n, p)	NegativeBinomial(p, n)
BetaGeneral(α_1 , α_2 , a, b)	a + (b-a)*Beta(α_1 , α_2)

EasyFit from Mathwave (<http://www.mathwave.com/en/home.html>) has both standalone and Excel versions.

A product named R has both free and commercial versions available. One source is <http://www.r-project.org/>.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Bernoulli

Random.Bernoulli(ProbabilityOfOne)

The Bernoulli distribution is a discrete distribution that returns a value of 1 with a specified probability, and 0 otherwise. The parameter for this distribution is the probability of returning 1.

The Expected Value of the Bernoulli distribution is $E(x) = p$.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

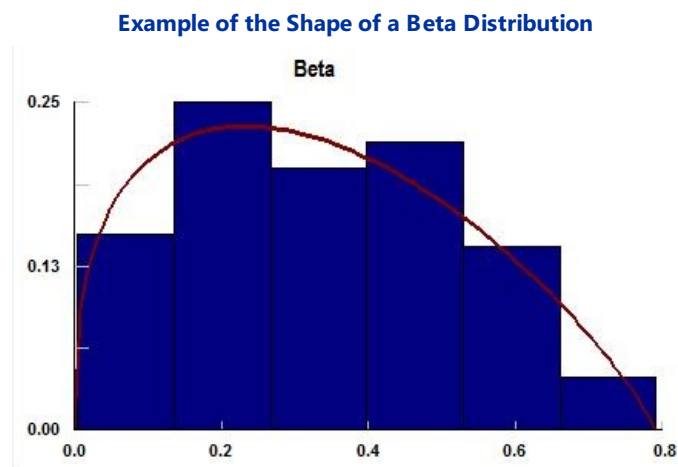
Send comments on this topic to [Support](#)

Beta

Random.Beta(alpha1, alpha2)

The Beta distribution has a range from 0 to 1 and can take a wide variety of shapes based on the two shape parameters (alpha1 and alpha2). The [Pert distribution](#) is a special case of the Beta and is often used as a rough model in the absence of data.

The Expected Value of the Beta distribution is $E(x) = \text{Alpha1}/(\text{Alpha1}+\text{Alpha2})$.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

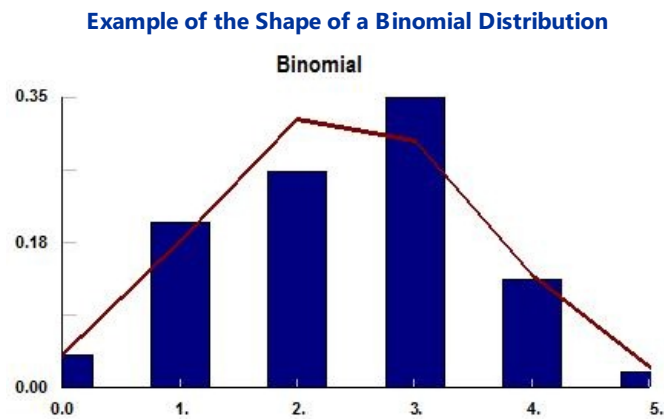
Send comments on this topic to [Support](#)

Binomial

Random.Binomial(ProbabilityOfSuccess, NumberTrials)

The Binomial distribution is a discrete distribution representing the number of successes in a specified number of independent trials. The parameters are the probability of success for each trial, and the total number of trials.

The Expected Value of the Binomial distribution is $E(x) = \text{ProbabilityOfSuccess} * \text{NumberTrials}$.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Continuous

Random.Continuous($v_1, c_1, v_2, c_2, \dots, v_N, c_N$)

This is an empirical distribution defined by a set of value and cumulative probability pairs(v_i, c_i) that define a piecewise linear cumulative distribution function for a continuous random variable over the defined range of values.

This implementation has an implied V_0 of 0. So C_1 represents the probability of a value between 0.0 and V_1 . If you want your first range to start at other than 0 (e.g. at V_1), then specify C_1 as 0.0.

This distribution may be used as an alternative to a theoretical distribution using empirical data to define the points on the cumulative distribution function. Note that there is always an even number of parameters since the parameters are defined in pairs (value, cumulative probability).

The Expected Value of the Continuous distribution is

$$E(x) = (C_1 V_1) / 2 + \sum_{k=2}^N (C_k - C_{k-1}) ((V_k + V_{k-1}) / 2)$$

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Discrete

Random.Discrete($v_1, c_1, v_2, c_2, \dots, v_N, c_N$)

This is an empirical distribution defined by a set of value and cumulative probability pairs (v_i, c_i) that define a stepwise linear cumulative distribution function for a discrete random variable. This distribution defines a set of N possible discrete values, where N is the number of value and cumulative probability pairs. Note that the number of parameters for this distribution is $2*N$. This distribution is typically used to make random discrete assignments to a state.

The Expected Value of the Discrete distribution is

$$E(x) = C_1 * V_1 + \sum((C_i - C_{i-1}) * V_i)$$

Examples

Example 1 - Assigning a Single Value

Suppose you want to assign a value of 1, 2, 3 or 4 to an entity priority, based on a certain percentage to each value. 40% of the entities will have a priority of 1, 35% of the entities have a priority of 2, 10% have a priority of 3 and the remaining 15% a priority of 4. The discrete distribution would be specified as `Random.Discrete(1,4,2,.75,3,.85,4,1)` where the values are specified in the distribution along with the cumulative probability.

Example 2 - Assigning a String State

You may wish to define an entity string state, named `PartType`, to a value based on a discrete distribution. Suppose you wish for 50% of the entities created to be "TypeA", 20% are "TypeB" and the remaining 30% are "TypeC". Your discrete distribution would then be `Random.Discrete("TypeA",.5,"TypeB",.7, "TypeC",1)`

Example 3 - Assigning a Value from Various Distributions

Suppose you wanted a processing time to be taken from one of two distributions. You can specify the value parameters within the discrete distribution based on an expression. Let's say that 20% of the time, the value should be longer processing time due to unforeseen circumstances, while 80% of the time, the processing time is shorter. The discrete distribution can be specified as `Random.Discrete(Random.Normal(50,3),.2,Random.Normal(10,.5),1)`

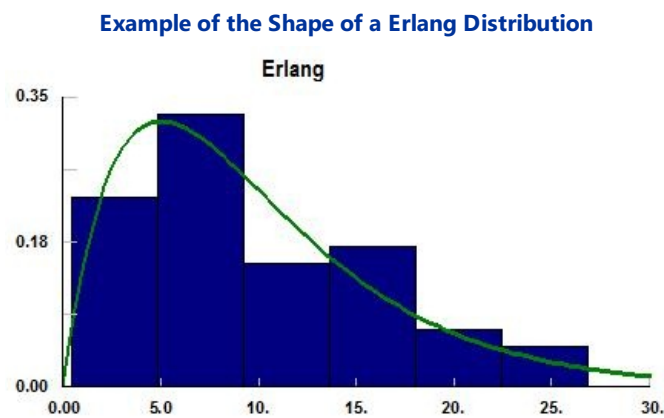
For examples of using the Discrete distribution, please refer to the SimBits [CombineMatchingMembers](#), [StringStates](#), [ImportExportTables](#) and [AnimatedPeople](#) .

Erlang

Random.Erlang(mean, K)

The Erlang distribution models an n-phase activity where the time for each phase is exponentially distributed. The Erlang has parameters that specify the mean and number of integer phases (K).

Erlang is the sum of K exponentials, each with an expected value of Mean/K . The Mean parameter is the mean of the Erlang, not the mean of the individual exponentials.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

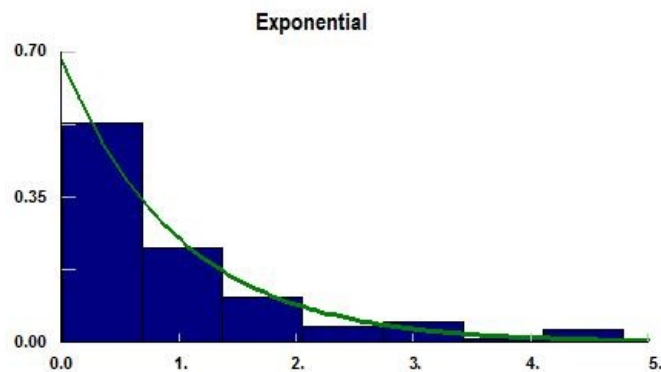
Exponential

Random.Exponential(mean)

The Exponential distribution is often used to model inter-arrival times based on a Poisson process. This distribution has a single parameter that specifies the mean. This distribution is generally not appropriate for modeling process delay times.

The Expected Value of the Exponential distribution is $E(x) = \text{mean}$.

Example of the Shape of a Exponential Distribution



For examples of using the Exponential distribution, please refer to the Source object's *Interarrival Time* property in many of the SimBits and Examples.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

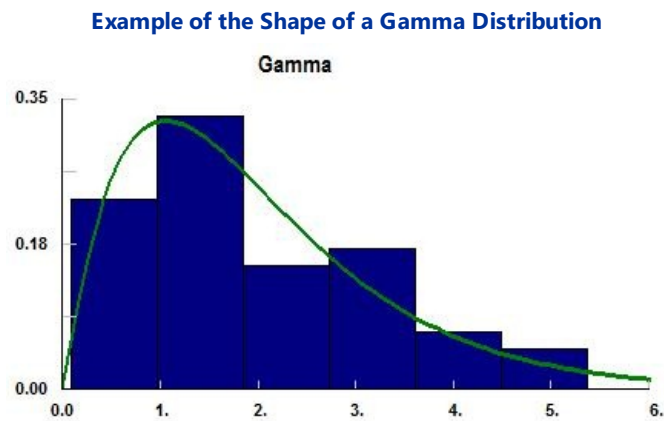
Send comments on this topic to [Support](#)

Gamma

Random.Gamma(shape, scale)

The Gamma distribution has a shape and scale parameter. The mean of the distribution is the product of these parameters. The [Erlang](#) is a special case of the Gamma with an integer shape parameter.

The Expected Value of the Gamma distribution is $E(x) = \text{shape} * \text{scale}$.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

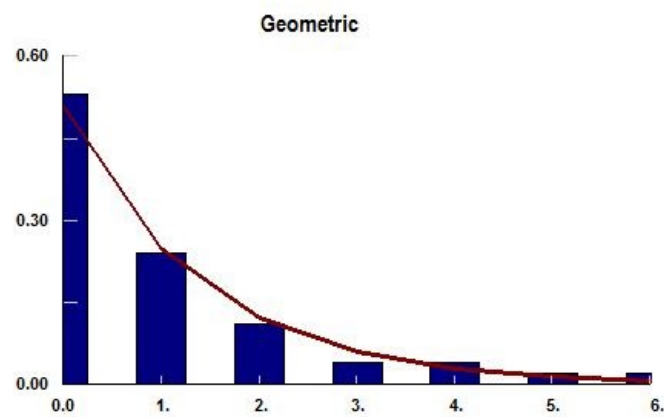
Geometric

Random.Geometric(ProbabilityOfSuccess)

The Geometric distribution is a discrete distribution and represents the number of failures before the first success. The parameter for the distribution specifies the probability of success for each independent trial.

The Expected Value of the Geometric distribution is $E(x) = (1 - \text{ProbabilityOfSuccess}) / \text{ProbabilityOfSuccess}$.

Example of the Shape of a Geometric Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

JohnsonSB

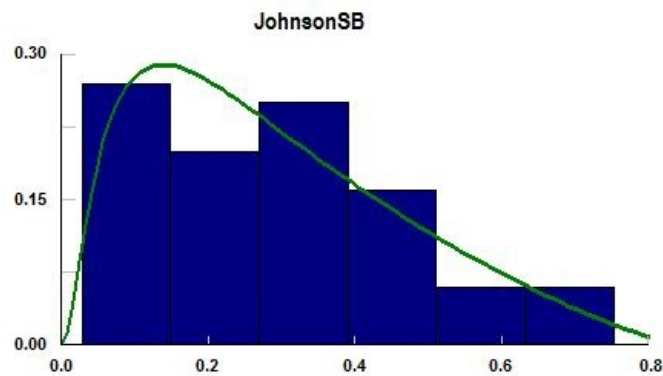
Random.JohnsonSB(alpha1, alpha2, min, max)

The JohnsonSB is a bounded four-parameter distribution that can take a wide range of shapes based on the two shape parameters (alpha1 and alpha2) and the minimum and maximum values.

The Expected Value of the JohnsonSB distribution is

$$E(x) = \min + (\max - \min) * 1 / (1 + e^{\alpha_1/\alpha_2})$$

Example of the Shape of a JohnsonSB Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

JohnsonSU

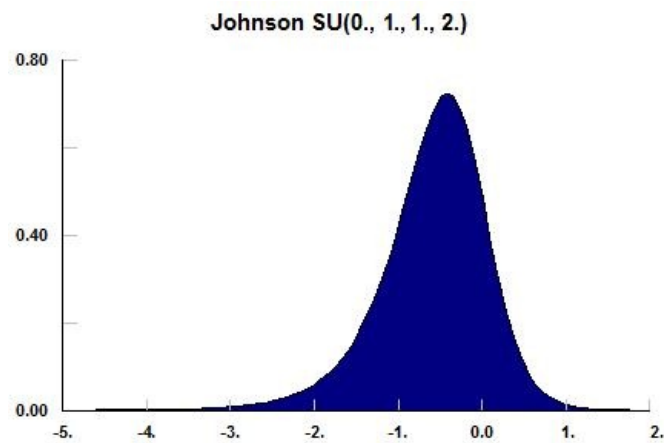
Random.JohnsonSU(alpha1, alpha2, location, scale)

The JohnsonSU is an unbounded four parameter distribution that can take a wide range of shapes based on the two shape parameters (alpha1 and alpha2), the location parameter a, and scale parameter b.

The Expected Value of the JohnsonSU distribution is

$$E(x) = \text{location} - \text{scale} * e^{(1 / (2 * \alpha_2 * \alpha_2)) * \sinh(\alpha_1 / \alpha_2)}$$

Example of the Shape of a JohnsonSU Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

LogLogistic

Random.LogLogistic(shape, scale)

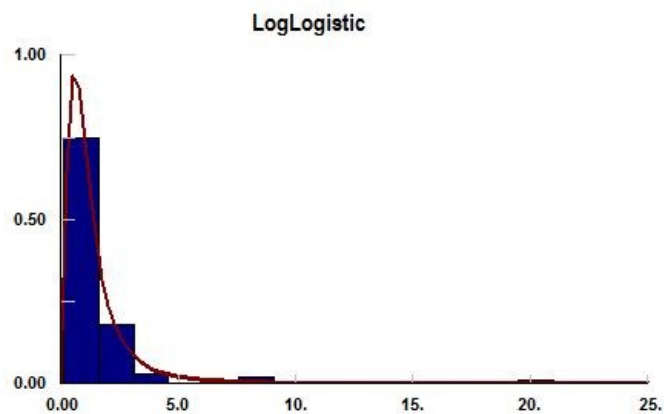
The LogLogistic distribution has one shape and one scale parameter, both of which must be non-negative. This distribution has a range from 0 to infinity and is typically used to model a task time.

The Expected Value of the LogLogistic distribution is

$$E(x) = \text{scale} * \Theta / \sin(\Theta)$$

where $\Theta = \pi / \text{shape}$

Example of the Shape of a LogLogistic Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Lognormal

Random.LogNormal(normalMean, normalStdDev)

A sample from the Lognormal distribution is generated by first generating a sample N from the normal distribution, and then converting this to a Lognormal sample, LN , using the following equation:

$$LN = e^N$$

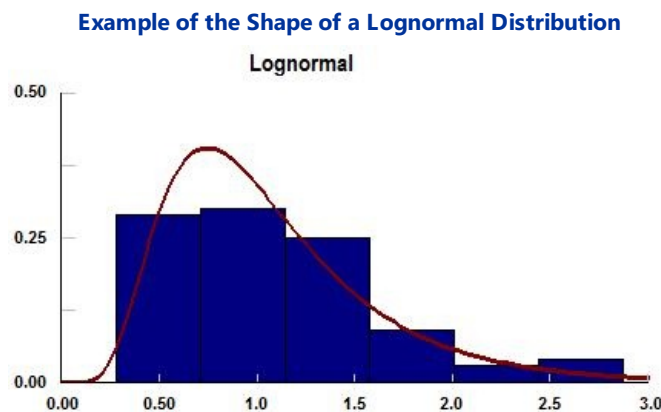
Note that the parameters specified for the Lognormal are the mean and standard deviation of the underlying normal before transformation to the Lognormal distribution. In some cases you may know the mean and standard deviation of the Lognormal distribution and need to convert these to the corresponding mean and standard deviation of the normal for use in specifying the distribution. Let LogNormalMean be the mean LogNormalStdDev be the standard deviation of the Lognormal. You can calculate the mean (normalMean) and standard deviation (normalStdDev) of the underlying normal distribution using the following formulas.

$$\text{normalMean} = \ln(\text{LogNormalMean}^2 / \sqrt{\text{LogNormalMean}^2 + \text{LogNormalStdDev}^2})$$

$$\text{normalStdDev} = \sqrt{\ln(1 + \text{LogNormalStdDev}^2 / \text{LogNormalMean}^2)}$$

You then use the normalMean and normalStdDev as the parameters of the Lognormal distribution to generate samples with mean and standard deviation given by LogNormalMean and LogNormalStdDev .

The Expected Value of the Lognormal distribution is $E(x) = \text{Lognormal Mean} = e^{(\text{normalMean} + \text{normalStdDev}^2 * \text{normalStdDev}/2)}$.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

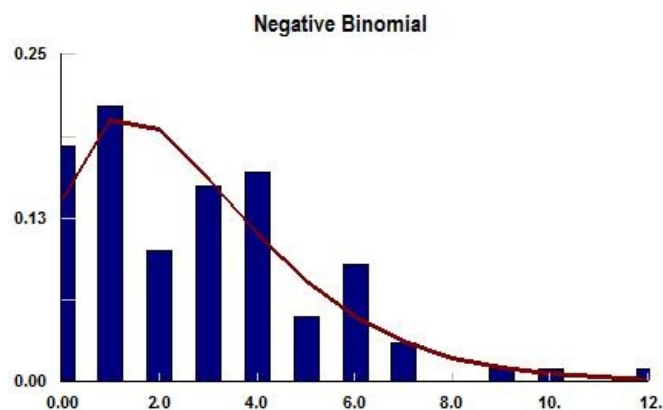
NegativeBinomial

Random.NegativeBinomial(ProbabilityOfSuccess, NumberOfSuccesses)

The NegativeBinomial distribution is a discrete distribution and represents the number of failures before reaching the specified number of successes. The parameters of the distribution are the probability of success for each trail, and the number of required successes.

The Expected Value of the NegativeBinomial distribution is $E(x) = (\text{ProbabilityOfSuccess} * (1 - \text{ProbabilityOfSuccess})) / \text{ProbabilityOfSuccess}$.

Example of the Shape of a Negative Binomial Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

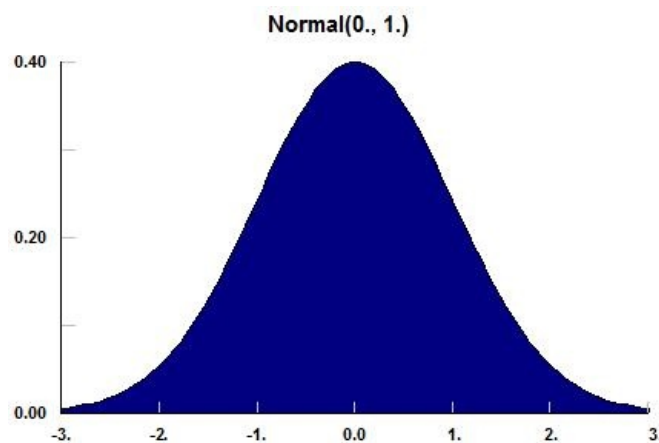
Send comments on this topic to [Support](#)

Normal

Random.Normal(mean, standard deviation)

The Normal distribution has parameters that specify the mean and standard deviation. The Normal Distribution is often used in situations where a value is the sum of several other values and by the central limit theorem can be approximated by a normal distribution.

The Expected Value of the Normal distribution is $E(x) = \text{mean}$.

Example of the Shape of a Normal Distribution

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

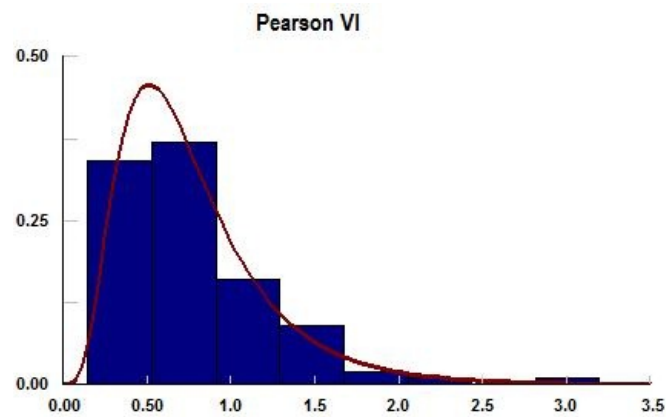
PearsonVI

Random.PearsonVI(shapeA, shapeB, scale)

The PearsonVI distribution has two shape parameters and one scale parameter. Both shape parameters and the scale parameter are non-negative. This distribution has a range from 0 to infinity and is typically used to model a task time.

The Expected Value of the PearsonVI distribution is $E(x) = (\text{scale} * \text{shapeA}) / (\text{shapeB} - 1)$ for all $\text{shapeB} > 1$.

Example of the Shape of a PearsonVI Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Pert

Random.Pert(minimum, mode, maximum)

The Pert distribution (also known as the Beta-Pert) is a special case of the beta distribution where the shape parameters are computed from the minimum, mode (most likely), and maximum parameters. This is a useful distribution for modeling expert data based on these three parameters. Unlike the triangular distribution the pert distribution provides a smooth curve which can closely approximate the normal or lognormal distributions. The mean of the pert distribution is equal to $(\text{minimum} + \text{maximum} + 4 * \text{mode}) / 6$.

The Expected Value of the Pert distribution is $E(x) = (\text{minimum} + \text{maximum} + 4 * \text{mode}) / 6$.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

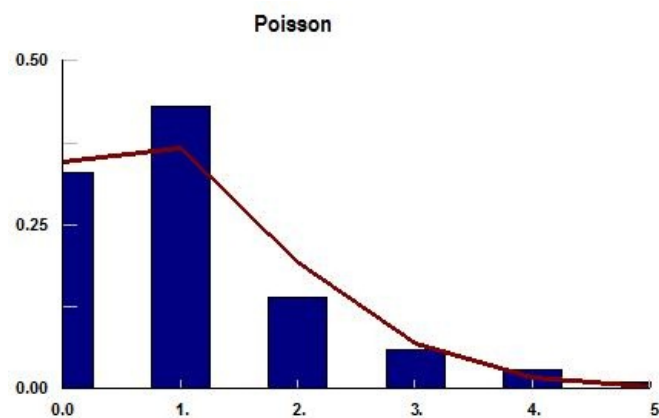
Poisson

Random.Poisson(mean)

The Poisson distribution has a single parameter that specifies the mean. This distribution is a discrete distribution that models the number of occurrences in an interval of time when the events are occurring at a constant rate according to a Poisson process. The time between each event is exponentially distributed and the number of events in a specified time is Poisson distributed. The parameter of the distribution is the rate of event occurrence (events per unit time), and must be a non-negative value. Note that for means ≥ 40 , the distribution will be calculated using the Normal distribution.

The Expected Value of the Poisson distribution is $E(x) = \text{mean}$.

Example of the Shape of a Poisson Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

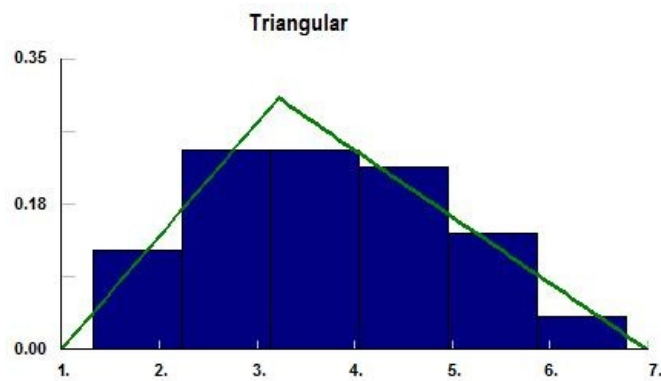
Triangular

Random.Triangular(minimum, mode, maximum)

The Triangular distribution has parameters that specify the minimum, mode (most likely), and maximum values. This distribution is often used as a rough model in the absence of data. The [Pert Distribution](#) may also be used in this situation.

The Expected Value of the Triangular distribution is $E(x) = (\text{minimum} + \text{mode} + \text{maximum})/3$.

Example of the Shape of a Triangular Distribution



Copyright 2006-2025, Simio LLC. All Rights Reserved.

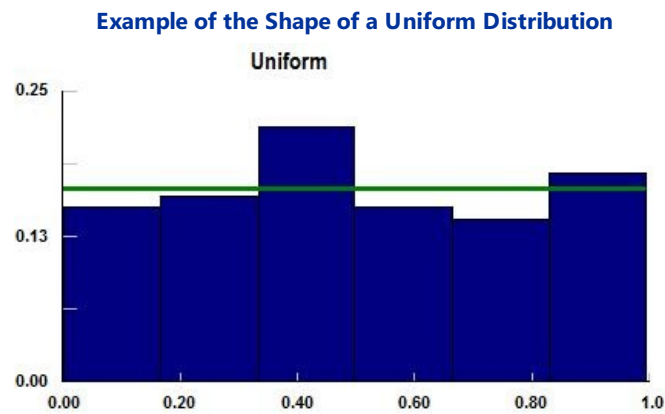
Send comments on this topic to [Support](#)

Uniform

Random.Uniform(minimum, maximum)

The Uniform distribution has parameters that specify the minimum and maximum values. All values within this range have an equal probability of occurring.

The Expected Value of the Uniform distribution is $E(x) = (\text{minimum} + \text{maximum})/2$.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

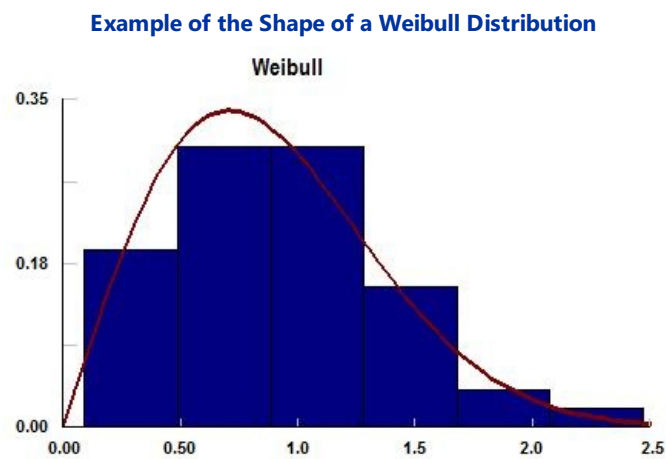
Send comments on this topic to [Support](#)

Weibull

Random.Weibull(shape, scale)

The Weibull distribution has a shape parameter and scale parameter. This distribution has a range from 0 to Infinity and is skewed to the left. If the system has a large number of components that are all required for the system to function and each component fails independently then the time between failures can be approximated by this distribution.

The Expected Value of the Weibull distribution is $E(x) = (\text{scale}/\text{shape}) * \Gamma(1/\text{shape})$ where Γ is the Gamma function.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Math Functions

Math Functions

Simio supports a number of math functions that can be used in an expression. To access these math functions from within the [Expression Editor](#), type in the word Math, followed by a period (.) to see the list of functions.

- **Math.Abs(value)**
 - Returns the absolute value of the specified number. The absolute value of a number is its unsigned magnitude.
- **Math.Acos(value)**
 - Returns the arc cosine of the specified number (in radians).
- **Math.Asin(value)**
 - Returns the arc sine of the specified number (in radians).
- **Math.Atan(value)**
 - Returns the arc tangent of the specified number (in radians).
- **Math.Ceiling(value)**
 - Returns the smallest integer not less than the specified number.
- **Math.Cos(value)**
 - Returns the cosine of the specified number (in radians).
- **Math.e**
 - Returns the value of the mathematical constant e , which is approximately 2.718.
- **Math.ExpectedValue(randomExpression)**
 - Deterministically evaluates the specified random expression and returns the expected value. For example, the `Math.ExpectedValue(Random.Uniform(0,1))` will return the value 0.5.
 - If the `randomExpression` is a Table Value Input Parameter, the expected value will be the average of the observations from the table.
- **Math.Epsilon**
 - Represents the smallest real value greater than zero. (Note: The standard `BasicNode` and the standard `TransferNode` objects do a delay for `Math.Epsilon`, in order to guarantee the correct order of events. The Delay step actually interprets `Math.Epsilon` as a delay of 0.0, but it forces the event to be put at the end of the current event list.)
- **Math.Exp(value)**
 - Returns the value of the mathematical constant e , to the power of the specified number.
- **Math.Floor(value)**
 - Returns the largest integer not greater than the specified number.
- **Math.If(test1, value1, test2, value2, test3, value3, ..., otherwise value)**
 - Returns the value specified in `valueN` if the `testN` is True. Returns the value specified in `otherwiseValue` if all the `testN` are False. For example, `Math.If(ModelEntity.Priority == 2, 1, 3)` would return the value of 1 if the expression `ModelEntity.Priority == 2` is True and would return the value of 3, otherwise.
- **Math.IsNaN(value)**
 - Returns the value of 0 true and 1 if false, based on the value that is evaluated. If the value is NaN (which means Not a Number), then the function will return a "True" or 1. If the value is not NaN, it will return a "False" or 0. This function is used in conjunction with the `Math.NaN` function.
- **Math.Log (value)**
 - Returns the natural logarithm (base e) of the specified value.
- **Math.Log10 (value)**
 - Returns the logarithm (base 10) of the specified value.
- **Math.Max (value1, value2, value3, ...)**

-
- Returns the value which is the maximum between the multiple specified values (i.e. expressions).
 - **Math.Min (value1, value2, value3, ...)**
 - Returns the value which is the minimum between the multiple specified values (i.e. expressions).
 - **Math.NaN**
 - Returns the value of NaN, which means Not a Number. It may be used in some special cases with Math.If(), for example, to return a value indicating that nothing was valid within the Math.If() so there isn't a value. Math.NaN can be used in conjunction with the Math.IsNaN() function. One of the special rules of NaN is that it does not equal anything, even itself. So the expression "Math.NaN == Math.NaN" would be false. In order to test if a value is NaN, then you need to use Math.IsNaN(), so the expression "Math.IsNaN(Math.NaN)" is true.
 - **Math.Pi**
 - Returns the value of Pi, which is approximately 3.1415.
 - **Math.Pow (value, power)**
 - Returns the result of the specified value to the specified power. For example, Math.Pow(2,3) returns 8.
 - **Math.Remainder (value1, value2)**
 - Returns the remainder, which is defined as $\text{value1} - (\text{math.floor}(\text{value1}/\text{value2}) * \text{value2})$
 - **Math.Round (value[, digits])** where digits is the number of significant digits (defaults to 0)
 - When used without optional argument, returns the closest integer to the specified number. This function rounds down from .50 and rounds up from .51. For example, Math.Round(4.5) will return 4 and Math.Round(4.51) will return 5.
 - If you would like to round a number to a specific number of significant digits, specify the number of digits in the optional second argument. For example, if you have the number 1.2345 and you'd like to round it to 2 significant digits after the decimal, you would use the expression, Math.Round(1.2345, 2) to get 1.23.
 - You can also use this with expressions. For example if you would like to take a number in hours (like Run.TimeNow) and display it in minutes to the nearest tenths, you would use the expression Math.Round(Run.TimeNow*60, 1). A time of 1.5123 hours would be displayed as 90.7 minutes.
 - **Math.Sin (value)**
 - Returns the sine of the specified number (in radians).
 - **Math.Sqrt (value)**
 - Returns the square root of the specified number.
 - **Math.SumOfSamples (randomExpression, numberOfSamples)**
 - Independently samples a specified random expression for a specified number of times and returns the total sum of the samples.
 - The expected value returned will be the expected value of the random expression, multiplied by the number of samples. For example, if the Random Expression is specified as 'Random.Uniform(1,2) and the Number of Samples is specified as '100', then the expected sum of the samples is $1.5 \times 100 = 150$. Therefore, if you enter the expression 'Math.SumOfSamples(Random.Uniform(1,2),100)' and then disable randomness in the model, you will see that expression always returns the value 150.
 - If the specified number of samples is not a positive integer, then the function returns the value of 0.0
 - **Math.Tan (value)**
 - Returns the tangent of the specified number (in radians).
-

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Math and Logical Operators

Logical Operators Available in Simio

The following logical operators are available for use throughout Simio:

`==` (Equal To)

`>` (Greater Than)

`<` (Less Than)

`>=` (Greater Than or Equal To)

`<=` (Less Than or Equal To)

`&&` (And)

`||` (Or)

`!` (Not) , such as `!Is.Transporter` (is not a Transporter)

`!=` (Not equal)

Boolean (True/False) - A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value. When using an expression to result in true/false(1/0), parentheses must be used, for example `(State1==1)*(State2>3)`.

When doing a logical comparison with NaN, it will always be false. A special rule of NaN is that it does not equal anything, even itself. So the expression `Math.NaN == Math.NaN` would be false. In order to test if a value is NaN, then you need to use `Math.IsNaN()`. See the "Math Functions" page for more information.

When comparing strings, Simio evaluates the first characters based on the ASCII table with a cultural sensitive sorting `a < A < b < B < ... < z < Z`. If the characters are the same, the next character in each string is evaluated and so on. Characters are case sensitive. If you want to compare them ignoring the case, use the functions `String.ToLower(string)`, `String.ToUpper(string)`, or `String.CompareIgnoreCase(string1, string2)`.

To compare only the Date value of a datetime use the function `DateTime.ToString(DateTime Value, "yyyy-MM-dd")`. Both values of the comparison need to be string values.

Note: Starting within Simio Sprint 101, a complex expression using `&&` (And) or `||` (Or) will be 'short circuited'. For example, if the first part of `&&` is False, then the entire `&&` operation returns False, and we don't evaluate the second part of the expression. Similarly, if the first part of the `||` is True, then the entire `||` operation is True, and we don't evaluate the second part of the expression. Prior to Simio Sprint 101, the entire expression was evaluated. See the Release Notes for Simio Sprint 101 and/or the Advanced Compatibility Settings section of [Running the Model](#) help for additional information on this feature before/after Simio Sprint 101.

Note: `'&&'` will only be used as a bit comparison 'And' operator. It will compare the first expression with the second and return a '0' or '1'. If you need to combine strings, use `+`. For example, `'StringState1' + "ABC" + StringState2'`.

Math Operators Available in Simio

The following math operators are available for use throughout Simio:

`+` (Addition)

`-` (Subtraction)

`*` (Multiplication)

`/` (Division)

`^` (Power), such as `(2^3)` results in 8

Send comments on this topic to [Support](#)

Keywords

Keywords Available in Simio

The following keywords are available for use throughout Simio:

Infinity - A keyword representing the numeric value of infinity. This keyword is always interpreted and returned as `double.PositiveInfinity`. See [Comptability Notes](#) for change in Infinity interpretation in Simio sprint 69+.

False - A keyword used to denote a False condition. This has a numeric value of 0.

True - A keyword used to denote a True condition. This has a numeric value of 1.

Nothing - A keyword used to denote a null object or element reference. May be used in a comparison statement to check whether an object or element reference exists (e.g., `ObjectReferenceValue == Nothing`).

ParentObject - A keyword used to specify a non-ambiguous reference to the parent object in a function argument or comparison statement (e.g. `Entity.SeizedResources.CapacityOwnedOf(ParentObject)`).

Is - The 'Is' keyword followed by a class or type name returns the value True (1) if the item is of that class or type. Otherwise, the value False (0) is returned. Note that this keyword can follow the 'Candidate' keyword to determine if the candidate object being evaluated is an instance of the specified class or is a specific instance.

For example, 'Is.PartA' would result in a 0 (False) if the object was not a 'PartA' type of object instance and a 1 (True) if it was. This may be used in a Decide step, for example, or as the Selection Weight of a path.

Similarly, a Decide step upon entering a node may have a Condition-based expression, 'Is.Vehicle', which would evaluate to True for incoming vehicles, but false for incoming model entities.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

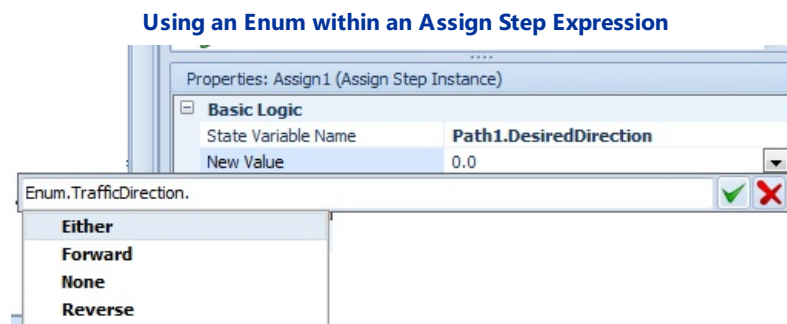
Send comments on this topic to [Support](#)

Enumerations

Enumerations

Simio has a number of built in Enumeration types that are used throughout the product and can be referenced in an expression. To access these enumerations from within the [Expression Editor](#), type in the word Enum, followed by a period (.) to see the list.

An example of using an Enum within an expression is shown below. This is a scenario where a user is changing the direction of travel on a Path and can choose from a list of possible values from Enum.TrafficDirection



Built in Enumeration Types in Simio

- **Activity Range** - The range of activities that a secondary resource is required for when seized for an operation. Found in the standard library [Workstation](#) object and within the [Operation](#) element.
 - Enum.ActivityRange.All
 - Enum.ActivityRange.AllButSpecific
 - Enum.ActivityRange.Specific
- **Array Dimension** - The choices for defining the dimension of an [Array](#). Enum.ArrayDimension.One refers to the Rows dimension and Enum.ArrayDimension.Two refers to the Columns dimension. This can be found in the properties of a Discrete State when the Dimension property is set to either Vector or Matrix.
 - Enum.ArrayDimension.One
 - Enum.ArrayDimension.Two
- **Batch Categories** - The choices for defining whether or not the parent entity or the member entity is executing the [Batch Step](#).
 - Enum.BatchCategories.Member
 - Enum.BatchCategories.Parent
- **Blocked Routing Rule** - The choices for defining the action the entity should take if the destinations in the list of candidate nodes are blocked (Blocked Destination Rule). Found in the standard TransferNode when *Entity Destination Type* is set to 'SelectFromList'.
 - Enum.BlockedRoutingRule.PreferAvailable
 - Enum.BlockedRoutingRule.SelectAny
 - Enum.BlockedRoutingRule.SelectAvailableOnly
- **Blocking Modes** - The choices for defining the action to take when an entity cannot immediately initiate a transfer because it is blocked. Found in the *Blocked Action* property of the [Transfer Step](#).
 - Enum.BlockingModes.Fail
 - Enum.BlockingModes.Wait
- **BOM Action Rules** - The choices for defining the method to be used to Produce materials or Consume Materials. Found in the *Production Type* property of the [Produce Step](#) and the *Consumption Type* property of the [Consume Step](#).
 - Enum.BOMActionRules.BillOfMaterials
 - Enum.BOMActionRules.Material
 - Enum.BOMActionRules.OperationSpecified

- **Capacity Type** - The choices for defining the capacity of a Resource type object, such as the Server, Combiner, Resource, etc. Found in the *Capacity Type* property of each of these standard library objects.
 - Enum.CapacityType.Fixed
 - Enum.CapacityType.WorkSchedule
- **Create Type** - The choices for defining the creation method, used within a Create Step. 'New Object' creates new objects of the specified instance type, 'CopyParentObject' creates a copy of the parent object and 'CopyAssociatedObject' creates a copy of the object that is associated with the token that is executing the [Create Step](#).
 - Enum.CreateType.NewObject
 - Enum.CreateType.CopyParentObject
 - Enum.CreateType.CopyAssociatedObject
- **Cross Direction** - Used in the [Monitor Element](#) for Monitors with *Monitor Type* set to 'CrossingStateChange'. The Monitor's event will fire if the state variable value crosses the threshold value in this crossing direction.
 - Enum.CrossDirection.Negative
 - Enum.CrossDirection.Positive
- **Decide Type** - Used in the [Decide Step](#) to indicate whether the decision is based on a probability or a condition.
 - Enum.DecideType.ConditionBased
 - Enum.DecideType.ProbabilityBased
- **Destination Modes** - A choice of methods to use for determining the entity's destination node. This is used in the *Destination Type* property of the [SetNode step](#). 'By Sequence' requires that the entity object has been assigned a sequence table. The destination node value will be set to the next destination in the sequence.
 - Enum.DestinationModes.Specific
 - Enum.DestinationModes.BySequence
 - Enum.DestinationModes.Expression
- **Destroy Type** - Used in the [Destroy Step](#) to indicate which object is to be destroyed; the parent object or the object associated with the token.
 - Enum.DestroyType.ParentObject
 - Enum.DestroyType.AssociatedObject
- **Duration Type** - Used in an [Activity element](#) and the standard Workstation object to provide the choices available to determine how the per batch time duration for the activity or operation is specified. 'Specific' dictates that the duration of the activity will be a specific value that is evaluated as a user entered expression. 'ChangeDependent' is used to specify an activity who's duration will be determined by one expression if the *Comparison Expression* is the same for the current entity as it was for the previous entity and a different expression if the value is different than the previous entity. 'SequenceDependent' is used to specify that the activity or operation should reference a ChangeOver matrix to determine its duration.
 - Enum.DurationType.Specific
 - Enum.DurationType.ChangeDependent
 - Enum.DurationType.SequenceDependent
- **Element Scope** - Found in the properties of all [Elements](#). This is used to define the element as either a public or private element, hence determining whether this element is publicly accessible outside of its containing object.
 - Enum.ElementScope.Private
 - Enum.ElementScope.Public
- **Entity Alignment** - Used for aligning travelers on a link. If 'Any Location', then a traveler can engage at any location on the link. If 'Cell Location', then the link length has a discrete number of cell locations, and the leading edge of an engaged traveler must align with a cell boundary. A standard Conveyor object is the only link that displays the Entity Alignment enum property.
 - Enum.EntityAlignment.AnyLocation
 - Enum.EntityAlignment.CellLocation
- **Entity Travel Mode** - Used for the initial travel mode for ModelEntity, Worker and Vehicle. 'Free Space Only' indicates that the entity is required to perform travel movements in free space. 'Network Only' indicates that the entity is required to perform travel movements using its currently assigned network. 'Network If Possible' indicates a preference for the entity to perform travel movements using its currently assigned network, but if no followable network path exists to its next destination then travel in free space is allowed.
 - Enum.EntityTravelMode.FreeSpaceOnly
 - Enum.EntityTravelMode.NetworkOnly
 - Enum.EntityTravelMode.NetworkIfPossible

- **Execute Step Action** - Found in the [Execute Step](#) and used to determine whether or not the token that is executing the Execute step waits until the new process is completed before continuing on in the current process or if the token continues in the current process while the new process is executed at the same time.
 - Enum.ExecuteStepAction.WaitUntilCompleted
 - Enum.ExecuteStepAction.Continue
- **Extended Selection Goal** - Found in the [Ride Step](#) and therefore the standard TransferNode when *Ride On Transporter* property is set to 'True'. This enumeration populates the *Selection Goal* property and allows the user to rank the transporter preference when multiple candidates are available to ride on.
 - Enum.ExtendedSelectionGoal.PreferredOrder
 - Enum.ExtendedSelectionGoal.SmallestValue
 - Enum.ExtendedSelectionGoal.LargestValue
- **Insert Object Type** - Provides the choice of Associated Object or Parent Object, for use in the [Insert Step](#). This allows the user to specify which object will be inserted into a storage queue.
 - Enum.InsertObjectType.AssociatedObject
 - Enum.InsertObjectType.ParentObject
- **Interrupted Process Action** - Found in the [Interrupt Step](#) as a choice for how the interrupted process delay will be handled. EndProcess will immediately end the current delay. However, the choice of ResumeProcess will cause the interrupted delay to resume for the remaining delay time once the interruption process is finished (i.e. the new token created by the Interrupt Step to handle the interruption occurrence ends its processing)
 - Enum.InterruptedProcessAction.EndProcess
 - Enum.InterruptedProcessAction.ResumeProcess
- **Link Selection Preference** - Used by an entity to select an outbound link from this location to the next. This enumeration is found in the *Outbound Link Preference* property of the standard [TransferNode](#).
 - Enum.LinkSelectionPreference.Any
 - Enum.LinkSelectionPreference.Available
- **Link Selection Rule** - Used by an entity to select an outbound link from this location to its next destination. This enumerator is found in the *Outbound Link Rule* property of the standard TransferNode. Shortest Path is only used when the entity already has a destination set. It will determine the shortest path between the current location and the destination of the entity. If the entity does not have a destination set (Entity.DestinationNode = Nothing), then ByLinkWeight will be used for routing. ByLinkWeight tells Simio to look at the *Selection Weight* properties of the outbound links to determine the routing for the entity.
 - Enum.LinkSelectionRule.ByLinkWeight
 - Enum.LinkSelectionRule.ShortestPath
- **Link Type** - The choices found in the *Type* property of all Link objects. It is only visible on the standard library Path and TimePath. The other Link objects have this property but it is not visible, by default. A bidirectional link allows entities to move in both directions on the link and a unidirectional link only allows movement in one direction at a time. By default, this is the direction in which the link was drawn in the Facility window (i.e. the direction the arrows are pointing).
 - Enum.LinkType.Bidirectional
 - Enum.LinkType.Unidirectional
- **Match Type** - This enumeration is found in the *Matching Rule* property of the [Batching Logic](#) element and therefore the standard [Combiner](#) object, since it uses the Batching Logic element. This determines how the entities will be batched together. 'Any Entity' indicates that the first members of the specified batch quantity will be attached to the first entity in the parent queue. There is no specific matching criteria. 'Match Members' indicates that members must be matched together using a specified expression. 'Match Members and Parent' indicates that parents must also be matched with members according to a specified expression.
 - Enum.MatchType.AnyEntity
 - Enum.MatchType.MatchMembers
 - Enum.MatchType.MatchMembersAndParent
- **Material Options** - This enumeration is found in the *Material Consumption* property and *Material Production* property of the [Activity Element](#) and the standard [Workstation](#), since this object uses the Activity Element. It determines if material will be consumed or produced by this activity (per batch). 'BillOfMaterials' indicates that the Material(s) to be consumed/produced can be found as part of a Bill Of Materials. 'Material' is indicating that a specific Material element will be consumed or produced. Either of these two choices (BillOfMaterials or Material) will switch on the *Consumed Material Name* and *Consumed Quantity* properties or *Produced Material Name* and *Produced Quantity* properties.

- Enum.MaterialOptions.BillOfMaterials
- Enum.MaterialOptions.Material
- Enum.MaterialOptions.None
- **Monitor Type** - This enumeration is found in the *Monitor Type* property of the Monitor element. This determines if the Monitor will fire an event whenever the value of the State Variable changes (DiscreteStateChange) or if it will only fire an event when the State Variables value crosses a specified threshold value in the crossing direction specified.
 - Enum.MonitorType.DiscreteStateChange
 - Enum.MonitorType.CrossingStateChange
- **Node Input Logic Type** - Indicates the location type within the object that an entity will transfer into when entering from an external node. These choices are found in the *Input Location Type* property on an External Node Instance, found when an external node is placed in the External View of an object. If an object's external node has *Input Location Type* set to 'Station', the entity will be transferred into a station when it enters the object. If the object's external node has *Input Location Type* set to 'Node', the entity will be transferred to a node that is defined within the Facility window of this object. Finally, if the object's external node has *Input Location Type* set to 'Container', the entity will be transferred to the specified container within the Facility window.
 - Enum.NodeInputLogicType.None
 - Enum.NodeInputLogicType.Station
 - Enum.NodeInputLogicType.Node
 - Enum.NodeInputLogicType.Container
- **Node Outbound Travel Mode** - Indicates whether to assign a new travel mode to entities attempting to transfer from this node to either free space or an outbound link. 'Continue' indicates that there is no assignment. The entities are to continue using their currently assigned travel modes. 'Free Space Only', 'Network Only' and 'Network If Possible' will assign the current travel mode to the specified value.
 - Enum.NodeOutboundTravelMode.Continue
 - Enum.NodeOutboundTravelMode.FreeSpaceOnly
 - Enum.NodeOutboundTravelMode.NetworkOnly
 - Enum.NodeOutboundTravelMode.NetworkIfPossible
- **Node Selection Goal** - These choices are found in the *Selection Goal* property of the Route Step and therefore also in the standard TransferNode since this object uses a Route Step when its *Entity Destination Type* property is set to 'SelectFromList'. The enumeration is used to decide which node to select from the nodes in the specified Node List.
 - Enum.NodeSelectionGoal.SmallestDistance
 - Enum.NodeSelectionGoal.LargestDistance
 - Enum.NodeSelectionGoal.PreferredOrder
 - Enum.NodeSelectionGoal.Cyclic
 - Enum.NodeSelectionGoal.Random
 - Enum.NodeSelectionGoal.SmallestValue
 - Enum.NodeSelectionGoal.LargestValue
- **Object Release Order** - Found in the *Release Order* property of the [Release Step](#), this indicates the order in which to release objects that are owned by the specified owner.
 - Enum.ObjectReleaseOrder.FirstSeized
 - Enum.ObjectReleaseOrder.LastSeized
- **Object Seize Type** - Found in the *Object Type* property of the [Seize Step](#), this indicates the method for specifying the object(s) to be seized. 'FromList' will trigger the user to select an object list from the *Object List Name* property. 'Specific' will allow the user to enter the name of a specific object in the *Object Name* property of the Seize step. And 'ParentObject' will indicate that the Seize step should seize capacity of the parent object of the token that is executing this Seize step.
 - Enum.ObjectSeizeType.FromList
 - Enum.ObjectSeizeType.ParentObject
 - Enum.ObjectSeizeType.Specific
- **Owner Type** - Found in the *Owner* property of the [Seize Step](#), this determines which object will be allocated capacity of the seized object. In other words, which object will perform the seize - the parent object of the token or the associated object of the token.
 - Enum.OwnerType.ParentObject
 - Enum.OwnerType.AssociatedObject
- **Park Unpark Type** - Found in both the *Park Type* and *UnPark Type* properties of the [Park Step](#) and the [UnPark Step](#).

These properties are used to determine which object to park or unpark; the parent object of the token or the associated object of the token.

- Enum.ParkUnParkType.ParentObject
- Enum.ParkUnParkType.AssociatedObject
- **Queue Ranking** - This enumeration is found in multiple places throughout Simio. Most notably, it is used in the *Ranking Rule* property of almost all of the standard library objects. It provides choices for the static ranking rule of the queues of these objects. For example, the Server uses these choices in its *Ranking Rule* property to order the requests waiting to be allocated capacity of the object. The TransferNode uses it to order travelers waiting to cross through the node.
 - Enum.QueueRanking.FirstInFirstOut
 - Enum.QueueRanking.LastInFirstOut
 - Enum.QueueRanking.SmallestValueFirst
 - Enum.QueueRanking.LargestValueFirst
- **Remove Object Type** - This enumeration is found in the *Removal Type* property of the [Remove Step](#). It specifies whether the step is removing the parent object, the object associated with the token or and object at a specific rank within the queue.
 - Enum.RemoveObjectType.ParentObject
 - Enum.RemoveObjectType.AssociatedObject
 - Enum.RemoveObject.AtRankIndex
- **Reservation Method** - This enumeration is found in the *Reservation Method* property of the [Ride Step](#) and therefore also in the standard TransferNode since this object uses a Ride Step in its logic. This property specifies which method to use to select and reserve a transporter to ride on. ReserveClosest will find the transporter that is closest on a Network or physically closest (if no network) to the TransferNode. If there are two that are of equal distance, the node's *Selection Goal* property will be used to determine which transporter to select. ReserveBest simply looks to the *Selection Goal* and *Selection Criteria* properties and finds the Transporter that best fits the goal, regardless of the vehicle's current location. The FirstAvailableAtLocation will cause the entities to wait for a Transporter to arrive to the node. This method should only be used when the vehicle has a *Routing Type* of 'Fixed Route' or when the user specifically moves the vehicle to an entity location. Otherwise, this method doesn't request the vehicle to move to the entity location.
 - Enum.ReservationMethod.ReserveClosest
 - Enum.ReservationMethod.ReserveBest
 - Enum.ReservationMethod.FirstAvailableAtLocation
- **Search Collection Type** - These choices are found in the *Collection Type* property of the [Search Step](#). It indicates where the Search Step will perform its search.
 - Enum.SearchCollectionType.ObjectInstance
 - Enum.SearchCollectionType.ObjectList
 - Enum.SearchCollectionType.ObjectsSeizedByParentObject
 - Enum.SearchCollectionType.ObjectsSeizedByAssociatedObject
 - Enum.SearchCollectionType.QueueState
 - Enum.SearchCollectionType.TableRows
- **Search Type** - This enumeration is found in the *Search Type* property of the [Search Step](#). It indicates what type of Search should be performed. 'Forward' indicates that the collection will be searched starting at the value set in the *Starting Index* property of the Search Step or at the beginning of the collection, by default. The collection is then searched moving forward. 'Backward' indicates that the collection will be searched starting at the value set in the *Starting Index* property of the Search Step or at the end of the collection, by default. The collection is then searched moving backward. 'MinimizeReturnValue' will search the collection and return the minimum value that it finds for the criteria listed in the *Return Value* property. 'MaximizeReturnValue' will search the collection and return the maximum value that it finds for the criteria listed in the *Return Value* property.
 - Enum.SearchCollectionType.ObjectInstance
 - Enum.SearchType.Forward
 - Enum.SearchType.Backward
 - Enum.SearchType.MinimizeReturnValue
 - Enum.SearchType.MaximizeReturnValue
- **Seize Request Move Type** - This enumeration is found in the *Request Move* property of the [Seize Step](#). It indicates whether a move to a specified location will be requested from the seized object(s). 'None' indicates that a move is not required. 'ToNode' indicates that the seized object must arrive to the specified node before the token will be released from the Seize Step.

- Enum.SeizeRequestVisitType.None
- Enum.SeizeRequestVisitType.ToNode
- **Seize Selection Goal** - This enumeration is found in the *Select Goal* property of the [Seize Step](#). It determines the criteria that the Seize step will use to determine which object to seize.
 - Enum.SeizeSelectionGoal.SmallestDistance
 - Enum.SeizeSelectionGoal.LargestDistance
 - Enum.SeizeSelectionGoal.PreferredOrder
 - Enum.SeizeSelectionGoal.SmallestValue
 - Enum.SeizeSelectionGoal.LargestValue
 - Enum.SeizeSelectionGoal.Random
- **SetRow Object Type** - This enumeration is found in the *Object Type* property of the [SetRow Step](#). It specifies whether the step is assigning a table reference to the parent object, executing token, object associated with the executing token, or a specific object or element.
 - Enum.SetTableObjectType.ParentObject
 - Enum.SetTableObjectType.AssociatedObject
 - Enum.SetTableObjectType.Token
 - Enum.SetTableObjectType.SpecificObjectOrElement
- **Set Value Object Type** - This enumeration is found in the *Object Type* property of the [Set Node Step](#). It specifies whether the step is setting the destination node value of the parent object or of the object associated with the executing token.
 - Enum.SetValueObjectType.ParentObject
 - Enum.SetValueObjectType.AssociatedObject
- **Statistic Confidence Interval Type** - This enumeration is found in the *Confidence Interval* property of the [State Statistic Element](#) and the [Tally Statistic Element](#). It specifies what confidence interval to use to calculate a confidence interval half width statistic for the average of the recorded values. 'Default' will indicate that the confidence interval used will be either 'None' if running interactively, or will be the experiment's specified confidence level if running an experiment scenario for a single replication.
 - Enum.StatisticConfidenceIntervalType.None
 - Enum.StatisticConfidenceIntervalType.Default
 - Enum.StatisticConfidenceIntervalType.Point90
 - Enum.StatisticConfidenceIntervalType.Point95
 - Enum.StatisticConfidenceIntervalType.Point98
 - Enum.StatisticConfidenceIntervalType.Point99
- **Suspend Resume Type** - This enumeration is found in both the *Suspend Type* property of the [Suspend Step](#) and the *Resume Type* property of the [Resume Step](#). It specifies what type of behavior or movement to suspend or resume. 'ParentObjectMovement' suspends or resumes the movement (Movement.Rate) of the token's parent object. 'AssociatedObjectMovement' suspends or resumes the movement (Movement.Rate) of the token's associated object. And 'Process' will suspend or resume the token that is executing the specified process.
 - Enum.SuspendResumeType.ParentObjectMovement
 - Enum.SuspendResumeType.AssociatedObjectMovement
 - Enum.SuspendResumeType.Process
- **Tally Value Type** - This enumeration is found in the *Value Type* property of the [Tally Step](#). It specifies what type of observation to record. 'Expression' will record the value of the specified expression into the Tally Statistic. 'TimeBetween' records the time between arrivals to the Tally Step.
 - Enum.TallyValueType.Expression
 - Enum.TallyValueType.TimeBetween
- **Timer Interval Type** - These choices are found in the *Interval Type* property of the [Timer Element](#). It determines the mode that is used by the Timer Element to determine the time interval between two successive timer events. 'Time' indicates that the time between events is specified in the *Time Offset* and *Time Interval* properties. 'TimeInState' indicates that the Timer will fire an event based on the amount of time spent in a certain state. The Timer determines the State from the specified value in the *State Variable Name* property and the *State value* property. 'RateTable' tells the Timer Element to fire events according to the specified Rate table, which defines how the rate of timer events changes over time (the event occurrences follow a non-stationary poisson process). 'EventCount' will trigger an event according to the number of *Triggering Event* occurrences. 'ArrivalTable' indicates that the Timer will fire events according to the times listed in the numeric table property (i.e. table column) of the specified table.
 - Enum.TimerIntervalType.Time
 - Enum.TimerIntervalType.TimeInState

- Enum.TimerIntervalType.RateTable
- Enum.TimerIntervalType.EventCount
- Enum.TimerIntervalType.ArrivalTable
- **Traffic Direction** - This enumeration is found in the *Initial Desired Direction* property of all the standard Link objects, such as the Path. When the *Type* property is set to 'BiDirectional', this property is switched on to allow the user to specify which direction the traffic should travel on this link. 'Either' indicates that traffic can travel in both directions, both forward and reverse. 'Forward' restricts traffic to travel in the forward direction, or the direction that matches the direction in which the arrows are pointing. 'Reverse' restricts traffic to travel in the opposite direction. When Traffic Direction is set to 'None', entities cannot travel on this link.
 - Enum.TrafficDirection.Either
 - Enum.TrafficDirection.Forward
 - Enum.TrafficDirection.Reverse
 - Enum.TrafficDirection.None
- **Traffic Direction Rule** - This enumeration is found in the *Traffic Direction Rule* property of all the standard Link objects, such as the Path. When the *Type* property is set to 'BiDirectional', this property is switched on to allow the user to specify the rule that is used to direct traffic entry onto the bidirectional link.
 - Enum.TrafficDirectionRule.FirstInEntryQueue
 - Enum.TrafficDirectionRule.MatchDesiredDirection
 - Enum.TrafficDirectionRule.PreferDesiredDirection
- **Transfer From Type** - This enumeration is found in the *From* property of all the [Transfer Step](#). This indicates the location type that the entity object is to be transferred from.
 - Enum.TransferFromType.FreeSpace
 - Enum.TransferFromType.CurrentStation
 - Enum.TransferFromType.CurrentNode
 - Enum.TransferFromType.EndOfLink
- **Transfer Object Type** - This enumeration is found in the *Object Type* property of all the [Transfer Step](#). This specifies which entity object is to be transferred, either the parent object or the object associated with the executing token.
 - Enum.TransferObjectType.ParentObject
 - Enum.TransferObjectType.AssociatedObject
- **Transfer To Type** - These choices are found in the *To* property of all the [Transfer Step](#). This indicates the location type that the entity object is to be transferred to.
 - Enum.TransferToType.AssociatedObject
 - Enum.TransferToType.FreeSpace
 - Enum.TransferToType.Node
 - Enum.TransferToType.OutboundLink
 - Enum.TransferToType.ParentExternalNode
 - Enum.TransferToType.Station
- **Transporter Type** - This enumeration is found in the *Transporter Type* property of the [Ride Step](#) and therefore also in the standard TransferNode since this object uses a Ride Step in its logic. This property specifies where to look to reserve a transporter to ride on - either to a specific transporter or select from a list of transporters.
 - Enum.TransporterType.Specific
 - Enum.TransporterType.FromList
- **Trip Selection Goal** - This enumeration is found in the *Task Selection Strategy* property of the standard [Vehicle](#) object. It specifies the strategy used by the Vehicle object to select its next transport task.
 - Enum.TripSelectionGoal.SmallestDistance
 - Enum.TripSelectionGoal.LargestDistance
 - Enum.TripSelectionGoal.SmallestPriority
 - Enum.TripSelectionGoal.LargestPriority
 - Enum.TripSelectionGoal.FirstInQueue

Referencing Candidate Objects

Use of the 'Candidate' Keyword in an Expression Property

Some expressions in Simio are referred to as candidate reference expressions because they may include references to objects that are not the token's associated object.

For example, in the [TransferNode](#) object in the Standard Library, if *Entity Destination Type* is specified as 'Select From List' and *Selection Goal* is 'Smallest Value' or 'Largest Value', then a *Selection Expression* property is available where an expression is evaluated in the context of each of the candidate destination nodes in the list ('Candidate.Node.AssociatedStationOverload' is the default value). In such an expression, to include an object reference to a candidate object in the collection that is being evaluated, prefix the reference with the keyword '**Candidate**'. For example, in the TransferNode's *Selection Expression* property described above, to enter an expression for selecting between downstream nodes based on input capacity, you can use the reference **Candidate.Node.AssociatedStation.Capacity** in the expression. The 'Candidate' keyword clearly declares to Simio's expression parser that the reference is the value of the AssociatedStation.Capacity function for each candidate node being evaluated in the collection, rather than a value of the token's associated object.

The 'Candidate' keyword can also be used in the [Search step](#) for searching a list, the [Ride step](#) for selecting a transporter, and the [Seize step](#) for selecting an object to seize. 'Candidate' keyword may be used in other property instances in Standard Library objects, as noted in help and in the property description in the software.

Specifying an Expression Property Definition to Allow Candidate References

When adding an Expression property to a model, specify the *Candidate References* property as 'True' to allow the use of 'Candidate' keyword in the expression.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

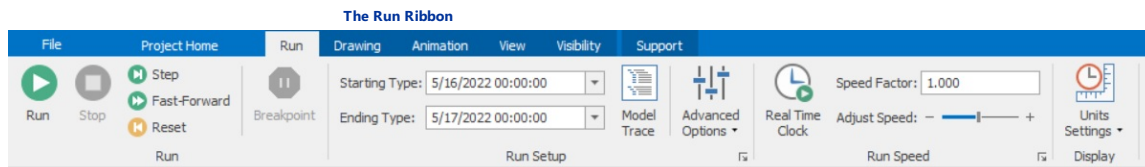
Running the Model

Running the Model

A Model can be run either via an Experiment or Interactively.

An [Experiment](#) allows the user to run multiple scenarios, multiple replications, add controls and get responses. An Experiment is run in batch mode, without animation, so it is the faster way to run a model. If the user would like to add a warm up period for the model, this can be specified in the properties of an experiment.

The Run Tab is where a user can run a model interactively. Running a model interactively allows the user to make certain changes to the model during the run. Certain tools are available while running the model interactively, such as the ability to view or hide the [Trace](#) window, the ability to add [Breakpoints](#) to the model and the ability to [Watch](#) states, functions and child elements that are associated with an object. The [Start and End time options](#) are specified here. The Run Ribbon also has a [Animation Speed](#) slide bar which allows the user to control the speed of the animation during the interactive run. The Time Units in the Run Ribbon control the time units that are displayed in the Trace window, in the run status bar along the bottom of the interface, in the reports, results pivot grid and watch window.



There is also a small run icon and stop icon in the Quick Access Toolbar at the very top left of the Simio application. This can be accessed from anywhere in the application and allows the user quick access to start, pause or stop a run. Note that a run can begin by the user hitting F5 (and paused with F5) and can be stopped with Shift-F5.

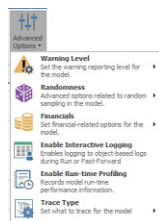


Stepping through a Model

The Step button in the ribbon is used to step from one point in the simulation logic to the next. However, Stepping works differently depending on which window is active when the Step button is clicked. Stepping in the Facility window stops the simulation when an entity moves from one object to another. Yet, stepping in the Processes window stops the simulation as a token moves from Step to Step in a process.

Advanced Options

Within the Advanced Options button are [Warning Level](#), [Randomness](#) and [Financials](#), which allow the user to turn various warnings on and off, as well as disable randomness and/or run a particular replication. Financials deals with the costing within the model. There are also options for [Interactive Logging](#) (Simio Professional and RPS Editions only), [Run-time Profiling](#) and [Trace Type](#). Stop At End Of Run is an RPS Edition Only feature that will automatically "stop" the model when the run is complete without requiring user interaction. See each of those help sections for more details.



Real Time Clock is an RPS Edition only toggle that indicates whether the simulation clock should advance at the same rate as a real clock if running the model interactively.

Advanced Compatibility Settings

The small icon to the bottom right of the Advanced Options button will open the Advanced Compatibility Settings dialog. This will allow the user to specify particular advanced settings related to running Simio.

Advanced Compatibility Settings

Automatically Return Zero Instead of NaN In Expressions	False
Do Not Perform Unit Conversions For State Parameters	False
Maximum Expression Stack Size	20
Ensure Foreign Nodes Cast To Non Node Types Are Associated Nodes	False
Always Allow 'Infinity' As Link Selection Weight Value	False
Assume Input Node Entries Are To Associated Object If No Drawn Outbound Links	True
Ignore Short Circuit Evaluation In Expressions	False
Ignore Primary-Foreign Key Relationships When Evaluating Table Property 'RandomRow' Function	False
Throw Error If Index Out Of Range When Evaluating An 'ItemAtIndex' Function	True
Use Performance-Optimized Steps	True
Default List State Results Classification Data Item String	Time\$(StateValueLabel)
Prevent Monitor Triggering During State Initialization	True
Harvest Available Rows Via Union Across All Rows Referenced From Original Table	False
Always Associate Dynamic Selection Rule Evaluations With The Objects Making The Selections	True
Set CurrentJobStep When Setting Row On Sequence Destination Table	False
Use Consistent Table Reference Searches	True
Ignore Table References When Resolving Schedule Exceptions	False
Include Context Object In Table Reference Searches	True
Always Allocate Resources To First Eligible Seize Requests	True
Ignore Table References When Resolving Table States Indexed By Row	False
Automatically Clear Entity Destination Node First Before Executing Route Step	False
Use Executing Token To Evaluate Route Step Selection Condition Expression	True
Node.AssociatedStationHasSpaceFor() Function Considers All Possible Redirect Stations	True
Allow Route Request Queue Bypassing	False
Run OnEvaluatingRouteRequest As A Regular Process	False
Run Resource OnCapacityAllocated Process After Seize Step On Seized Process	True
Run Resource OnCapacityReleased Process After Release Step On Released Process	True
Schedule Late Current Event To Try Allocating When Resource Capacity Increased	True
Token.IsSuspended Function Considers Only Suspensions Applied Explicitly To The Token	False
Routing Group As Context On Evaluating Token	True
Schedule Late Current Event To Try Batching When Batch Queue Item Inserted	True
Exclude Unrelated Rows In Timer Arrival Tables	True
Report Material Consumption Cost As Usage Cost Charged	True
Allow Material Allocation Queue Bypassing	False
Schedule Late Current Event To Try Allocating When Material Produced	True
Evaluate Expression Functions In Expressions Using Units Of Expression	False
Always Do Full Checks Of Batch Logic Parent Queues	True
Always Use Seize Selection Goal When Checking Resource Allocation Queue	True
Automatically Adjust Timer's Target Triggering Event Count If Zero To One	False
Auto Destroy Zero Volume Entities In Containers With Zero Rate Inflow	True
Allow Indexing Past Related Rows For Table State Indexing	False
Search Associated Objects Of Candidate Objects For Table References	True
On Seized Process Compatibility Mode	False
On Released Process Compatibility Mode	False
Search Step Copies Over Table Row References When Searched Table Is Implicitly Referenced	True
Auto Cancel Move Request If Resource Released	True
Search Step Uses Explicit Copy Over Table Row References	True
Always Check Specific Run Space Context When Resolving Table Row References	True
Exclude Entities Transferring Out Of Child Stations From Resource Utilization	True

Automatically Return Zero Instead of NaN In Expressions is initially set to False. If it is True, then for any element or state function with no parameters that is used in an expression, the engine will automatically adjust a NaN result returned by the function to zero. This was implicitly true in Sprint 85 and older.

Do Not Perform Unit Conversions For State Parameters is initially set to False. If it is True, then the engine will not properly handle conversions of state parameters such as Weight or Volume parameters of a queue that is holding entities. This was implicitly True in Sprint 89 and older.

Maximum Expression Stack Size indicates the size of the stack expressions to evaluate before throwing a runtime error. For example, if expression A calls into expression B, which calls into expression C, a stack size of at least 3 would be necessary. The default value is set to '20'. These options have been put into the software for compatibility reasons, so that when a changes was made internally to the Simio software, it would not automatically 'update' a user's model such that their results are different.

Ensure Foreign Nodes Cast To Non Node Types Are Associated Nodes is set to False by default. If True, Simio will give a runtime error if a node is referenced via the Foreign/Candidate/Owner keywords, the cast given is not a Node type, and the node is not associated with an object as an external input or output node. This option was implicitly True in Sprint 90 and older.

Always Allow 'Infinity' As Link Selection Weight Value is an option that recommended to be set to False. This indicates whether 'Infinity' is always allowed as a possible value for a link selection weight.

Assume Input Node Entries Are To Associated Object If No Drawn Outbound Links is initially set to True. If it is True, then if an entity is entering a node with no drawn outbound links and that node is an external input node to enter another object, the engine will assume the entity's objective is to enter the input node's associated object and thus execute the node's standard 'OnEnteredAssociatedObject' process (if specified). Prior to Sprint 99, a runtime error would be given if the node had no outbound links.

Ignore Short Circuit Evaluation In Expressions is initially set to True. If it is True, then Simio will not 'short circuit' evaluation of expressions using && or || in them. For example, 'MyRefState != Nothing && Random.Uniform(MyRefState.MyEntity.MyState, 100) < 5' will normally 'short circuit' evaluation if it sees that MyRefState is Nothing and will not evaluate the latter part of the && statement. If the setting is True, Simio will evaluate both parts no matter what. This option was added to the software within Sprint 101.

Ignore Primary-Foreign Key Relationships When Evaluating Table Property 'RandomRow' Function is initially set to False. If using TableName.PropertyName.RandomRow function syntax in an expression and this setting is set to True, then the value returned by the RandomRow function will ignore any foreign-primary key relationships and always return a random row index into the entire table, regardless of whether or not the active token or object is referencing a row selection in the table. This option was added in Simio Sprint 102.

Throw Error If Index Out Of Range When Evaluating An 'ItemAtIndex' Function is initially set to True. If using Collection.ItemAtIndex(index) function syntax in an expression and this setting is set to True, then when evaluating the function, a runtime error will be thrown if the index argument is out of range. Otherwise, an undefined or null value will be returned without any error. This option was added in Sprint 105.

Use Performance-Optimized Steps is initially set to True. If this setting is True, then at runtime Simio will attempt to use

performance-tuned versions of certain steps where possible. The recommended value for this setting is True.

Default List State Results Classification Data Item String represents the default string used as the Data Item field for classifying results reported by list states in the model. Note that you may insert a state variable label into the Data Item classification string by using the special parameter `$(StateValueLabel)`. When a list state is reporting statistics for an individual state value, if that parameter occurs within the Data Item classification string then it will be replaced with the actual state value label. The default value for this is `'Time$(StateValueLabel)'`.

Prevent Monitor Triggering During State Initialization is initially set to True. Indicates whether monitors in the model will be prevented from triggering during the value initialization of their monitored state variables.

Harvest Available Rows Via Union Across All Rows Referenced From Original Table is initially set to False. Indicates whether calculation of available rows will result from the union of all row relations into tables referenced via foreign key from the table explicitly selected. This was implicitly True in Sprint 134 and older.

Always Associate Dynamic Selection Rule Evaluations With the Objects Making The Selections is initially set to True. Indicates whether expression evaluations when using a dynamic selection rule are always associated with the objects making the selections. The recommended value for this setting is True. For models built in Sprint 134 or older, this value should be changed to True to utilize the dynamic selection rules correctly.

Set CurrentJobStep When Setting Row On Sequence Destination Table is initially set to False. Indicates if, when executing a SetRow on a table, and that table contains sequence destinations, and a previous table selection was made with foreign keys into that table, if the CurrentJobStep should be set to the row indicated in the SetRow step. The recommended value for this setting is False. This was implicitly True in Sprint 134 and older.

Use Consistent Table Reference Searches is initially set to True. Indicates how, when resolving a table reference, multiple references in the context will be searched. If True, then the object or element containing the property that is referencing the table will be searched first, then the active token, then the active token's associated object, then the active token's context object (if including the context object in the search). The recommended value for this setting is True.

Ignore Table References When Resolving Schedule Exceptions is initially set to False. Indicates if, when resolving the exception repeat groups for a work schedule on a resource, the table references set on the resources will be ignored. The recommended value for this setting is False.

Include Context Object In Table Reference Searches is initially set to True. Indicates whether, when resolving a table reference, references to the active token's context object will be included in the search. The recommended value for this setting is True.

Always Allocate Resources To First Eligible Seize Requests is initially set to True. Indicates whether to always allocate a resource's available capacity to the first eligible seize request(s) waiting in its allocation queue. The recommended value for this setting is True. This option was added in Simio Sprint 146.

Ignore Table References When Resolving Table States Indexed By Row is initially set to False. Indicates when resolving table states in the form of `TableName[RowNumber].StateName` that existing table references on the object should be ignored. The recommended value for this setting is False. This option was added in Simio Sprint 150.

Automatically Clear Entity Destination Node First Before Executing Route Step is initially set to False. Indicates whether to automatically clear an entity's reference to a destination node first before executing a Route step. This option was added in Simio Sprint 156.

Node.AssociatedStationHasSpaceFor() Function Considers All Possible Redirect Stations is initially set to True. Indicates whether the `AssociatedStationHasSpaceFor()` function provided by an external input node considers all possible redirect stations in addition to the node's most immediate associated station location. For example, if the node is the input node of a Server with an input buffer, this setting determines whether that node's `AssociatedStationHasSpaceFor()` function considers space in both the input buffer and processing station locations of that Server or only its input buffer. This option was added in Simio Sprint 156.

Allow Route Request Queue Bypassing indicates whether an entity executing the Route step is allowed to immediately be assigned a destination node bypassing the route request queue. The recommended value for this setting is False (but is set to True for models prior to Sprint 157).

Run OnEvaluatingRouteRequest As A Regular Process indicates whether the `OnEvaluatingRouteRequest` process of a `RoutingGroup` element is executed as a 'regular' process instead of a 'decision' process. This option was added to Simio Sprint 156 and has a recommended setting of False.

Run Resource OnCapacityAllocated Process After Seize Step on Seized Process When a resource is seized, indicates whether the resource's `OnCapacityAllocated` process is executed after the `On Seized Process` optionally specified by the Seize step. This option was added in Simio Sprint 160 and has a recommended setting of True (set to False for old models).

Run Resource OnCapacityReleased Process After Release Step on Released Process When a resource is released, indicates whether the resource's `OnCapacityReleased` process is executed after the `On Released Process` optionally specified by the Release step. This option was added to Simio Sprint 160 and has a recommended setting of True (set to False for old models).

Schedule Late Current Event To Try Allocating When Resource Capacity Increased Indicates whether to schedule a late priority current event to try allocating whenever the capacity of a resource has been increased. Otherwise, allocation will be immediately attempted before the execution of any other simulation logic in the system. The recommended value for this setting is True.

Token.IsSuspended Function Considers Only Suspensions Applied Explicitly To The Token Indicates whether the `IsSuspended` function provided by a token considers only suspensions applied explicitly to the token, not any suspensions applied to the token's parent process. This option was added to Simio Sprint 162 and has a recommended setting of False.

Routing Group As Context On Evaluating Token Indicates whether when evaluating the *Destination Node List Name* and *Destination Blocked Condition* for a `RoutingGroup` element, if the routing group should be set as the 'context' element for the executing token. This option was added to Simio Sprint 166 and has a recommended setting of True.

Schedule Late Current Event To Try Batching When Batch Queue Item Inserted Indicates whether to schedule a late priority current event to try batching whenever a new item has been inserted into the parent or member queue of a `BatchLogic` element. Otherwise, batching will be immediately attempted before the execution of any other simulation logic in the system. This option was added to Simio Sprint 167 and has a recommended setting of True (set to False for old models).

Exclude Unrelated Rows In Timer Arrival Tables Indicates whether to exclude unrelated rows if an arrival table used to schedule timer events has primary key/foreign key relationships. The recommended value for this setting is True. This option was added to Simio Sprint 170.

Report Material Consumption Cost as Usage Cost Charged Indicates whether the total cost charged to consumers of a specific material is reported in the results as 'UsageCostCharged' instead of 'MaterialCostCharged'. The recommended value for this setting is True. This option was added to Simio Sprint 176.

Allow Material Allocation Queue Bypassing Indicates whether an object executing a Consume step is allowed to immediately consume material bypassing the material allocation queue. The recommended value for this setting is False. This option was added to Simio Sprint 176.

Schedule Late Current Event To Try Allocating When Material Produced Indicates whether to schedule a late priority current event to try allocating to waiting consumption requests whenever material has been produced. Otherwise, allocation will be immediately attempted before the execution of any other simulation logic in the system. The recommended value for this setting is True. This option was added to Simio Sprint 176.

Evaluate Expression Functions in Expressions Using Units of Expression Indicates whether expression functions used in expressions will be evaluated using the units specified for the expression itself. The recommended value for this setting is False.

Always Do Full Checks of Batch Logic Parent Queues If this setting is False, then depending on the matching rule used, the batch logic element may stop further checking of entities waiting in its parent queue if the entity waiting at the front of the queue cannot collect its required batch quantity. The recommended value for this setting is True so that further checking is always allowed.

Always Use Seize Selection Goal When Checking Resource Allocation Queue Indicates whether a Seize step's selection goal should be used not only at the initial seize attempt, but also whenever any resource allocation queue check occurs - if waiting for resources is necessary - regardless of the Must Simultaneously Seize option setting. The recommended value for this setting is True. This option was added to Simio Sprint 186.

Automatically Adjust Timer's Target Triggering Event Count If Zero To One If this setting is True and an event count-based timer's Triggering Event Count expression property has returned the value zero, then the target triggering event count for that timer will be automatically adjusted to one. The recommended value for this setting is False.

Auto Destroy Zero Volume Entities In Containers With Zero Rate Inflow If this setting is False, entities in a container that fall to zero volume with an existing inflow, where the inflow rate is zero will not automatically be destroyed. Doing so may constrain upstream flows. The recommended value for this setting is True to remove the potential upstream constraint.

Allow Indexing Past Related Rows For Table State Indexing If this setting is True, when explicitly indexing into a table to get a table state value (in the form TableName[Index].TableState) if the table has foreign keys into a master table for which the table selections have a row set, and the specified index goes beyond the set of 'detail' rows for that selection, then the index will instead be used as a raw index into the overall table, instead of a runtime error being thrown. The recommended value for this setting is False.

Search Associated Objects Of Candidate Objects for Table References If this setting is True, when using table references in the form Candidate.TableName (such as Candidate.TableName.ColumnName), if the Candidate object has an Associated object, such as the Input Node of a Server having an associated Server object, then that associated object's table references will also be used to find a value if the value isn't found using the Candidate's table references. The recommended value for this setting is True. This option was added to Sprint 191.

On Seized Process Compatibility Mode Indicates whether to execute an On Seized Process of a Seize step using token associated object and token context object references that are compatible with Simio version Sprint 190 or earlier.

On Released Process Compatibility Mode Indicates whether to execute an On Released Process of a Release step using token associated object and token context object references that are compatible with Simio version Sprint 190 or earlier.

Search Copies Table References When Searched Table Is Implicitly Referenced If this setting is True, when a Search step is used to search a table with rows that are only implicitly referenced by the incoming token to the Search step (for example, searching a 'detail' table of a master-detail relationship with the reference to the 'master' table is already set on the incoming token), then all table references on the incoming token copied to the 'found' token. If this setting is False, then those references will not be copied in that case. This option was added to Sprint 194.

Auto Cancel Move Request If Resource Released Indicates whether to automatically cancel a resource move request if that resource is released. The recommended value for this setting is True.

Search Step Uses Explicit Copy Over Table Row References Indicates whether the copying of table row references over from an original token executing a Search step to new tokens associated with found items is based solely on the step's Copy Over Table Row References property value, removing all restrictions that were required by previous Simio versions. Enabling this option also ensures that the table row references for the original token are always included in match condition and search expression evaluations. The recommended value for this setting is True. This option was added to Sprint 198.

Always Check Specific Run Space Context When Resolving Table Row References Indicates whether the specific run space context is always checked when resolving table row references. The recommended value for this setting is True.

Exclude Entities Transferring Out of Child Stations From Resource Utilization Indicates whether capacity allocated to entities transferring out of child station locations of a resource are excluded from that resource's utilization calculations. The recommended value for this setting is True. This option was added to Sprint 201.

Minimal Automatic Route Request Queue Checking Indicates whether routing groups will automatically check their route request queues only when space at a possible destination node has become available or when a planned or unplanned primary resource downtime has ended. Otherwise, if this setting is False, then routing groups will automatically check their route request queues when any station or resource state change occurs at a possible destination node, resulting in a greater number of automatic (and likely unnecessary) route request queue checks. The recommended value for this setting is True. This option was added to Sprint 205.

All Changeover Logic Expression Evaluations Allow Resource References Indicates whether all expression evaluations by a Changeover Logic element allow references to the resource. The recommended value for this setting is True. This option was added to Sprint 205.

Search Related Rows Only Option Always Looks For Best Foreign Key Relationship Indicates whether the Search Related Rows Only option of a Search step always looks for the best foreign key relationship in the searched table. The recommended value for this setting is True. This option was added to Sprint 206.

Allow Token-Based Table Row References In Queue Ranking Expressions Indicates whether table row references set for process tokens are allowed in queue ranking expressions. For example, whether table row references set for process tokens executing Seize steps are allowed in the resource Ranking Expression properties used to rank allocation queue items. The recommended value for this setting is True. This option was added to Sprint 208.

Schedule Constraint Logic Related Queue Checks As Latest Current Events Indicates whether to schedule constraint logic-related queue checks as latest priority current events, thereby allowing queue checks for any other reasons to happen first. The recommended value for this setting is True. This option was added to Sprint 211.

Record State Variable Value Whenever State Statistic Function Called For any State Statistic element, indicates whether to record and log the monitored state variable's current value and parameters whenever one of the state statistic's functions is called (specifically the Average, Minimum, Maximum, LastRecordedValue, or HalfWidth functions). The recommended value for this setting is False. This option was added to Sprint 215.

Include Static Agent Object In Table Reference Searches Indicates whether, when resolving a table reference, references held by the associated object's static instance (if the object is an agent) will be included in the search. The recommended value for this setting is True. This option was added to Sprint 215.

Allow Cross-Node Comparisons When Using Routing Group Dynamic Selection Rules Indicates whether routing groups are allowed to compare dynamic selection rule results across destination nodes, in order to determine the best overall entity destination assignments. The recommended value for this setting is True. This option was added to Sprint 216.

Always Use Property Parent Table References Indicates if the table references of a parent object of the property whose value is being resolved should always be considered. The recommended value for this setting is True. This option was added to Sprint 221.

Animation of Active Agents Should Prevent Run Ending Indicates if animation of active agents in the system should prevent a run from stopping when the end time of the run is unspecified. The recommended value for this setting is False. This option was added to Sprint 242.

Use Process Element Table Row References Indicates whether to use a Process Element's table row references when retrieving property values. The recommended value for this setting is True. This option was added to Sprint 244.

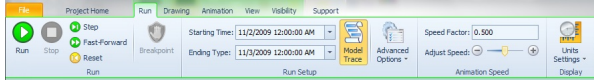
Always Consider Parent Token Table Row References When Resolving Row Counts In Search Step Indicates whether to consider the parent token's table row references when resolving Search step expressions involving table row counts. This setting could affect the output of table expressions such as AvailableRowCount and RandomValue. The recommended value for this setting is True. This option was added to Sprint 246.

Trace

Trace

The Trace window can be displayed by clicking on the Trace Window icon in the Run Tab of the ribbon menu. The Trace window will appear at the bottom of the application and will read Trace - Trace On. If the user clicks the Trace Window icon again, the window will remain but it will read Trace - Trace Off. To close the window, the user can click the X in the top right corner of the Trace Window. This window can be moved around the application by clicking and dragging. It is important to note that if a project was saved with the Trace Window on, Simio will automatically open the Trace Window when opening a project to avoid running with the trace accidentally turned on. Excluded Steps and Processes are not shown in the Trace.

The Run Tab with the Model Trace Clicked On



If the Trace has been on during the model run, the contents will also be written to a .csv file. This file can be located in the same location where the project file is saved. This file can be loaded into Excel so it can be filtered and sorted for ease in troubleshooting and understanding of the exact processing steps of the model. There is a limit to the size of the file so if the model is run for a long period of time, the .csv file might not contain all of the contents of the trace.

The time units that are displayed within the Trace Window can be changed by selecting the desired units from the Units Settings button, Time property drop down, found on the Run Ribbon.

A user can filter the Trace window by any of the column headers. There is a filter icon located to the right of each column name. This allows the user to hide sections of the trace so it is easier to analyze the sections of that are of interest. For instance, if you would like to only trace the activities of one particular entity or one particular process, you can filter so that you only view the activity of that entity or process in the Trace window. No other activity is displayed in the Trace until the filter is removed. To set a filter, select the filter icon within a given column and select the entity (or object, process, token, step, etc.) of interest from the drop down. Simio allows for more than one filter at a time. The filters that are set are displayed at the bottom left of the Trace window. Filters can be removed by selecting the 'X' next to the filter on the bottom left of the Trace window, by selecting (All) within a given filtered column or by clicking on the 'X' next to the filter within the column filter dropdown. Custom filters may also be specified (see below custom filter dialog).

Any runtime errors that occur within a simulation model run will automatically be displayed within the Trace window, regardless of whether the Trace Window button is enabled. This will allow users to more easily view the Entity, Object, Process and Step associated with the error that occurred.

To further investigate process logic, the Process and Step columns in Trace have addition functionality. You can right-click on a Trace entry's Process or Step cell and use the corresponding 'Go To...' option, if it is available. This will bring you to that exact Process or Step. The Processes for that object must be viewable for the option to be available. For example, the process logic inside the Model you are building and running will be available, as it will take you to the Processes tab in this model. Any other object definitions in this project, objects in the Navigation Pane, will also be available to navigate to. If you cannot navigate to the Process Logic window of that object in this project, the 'Go To...' options will be disabled for those Trace entries.

Filtering Trace by Entity (Customer.823) and Starting to Add an Additional Filter by Object

Date Time	Time (Hours)	Entity	Object	Process	Token	Step	Action
1/10/2011 01:27:42	1.46182785115305	Customer.823	Model	Output_Source1_Entered	141	[Assign] New Picture	Assigning state variable 'Customer.823.Picture' the value '4.549752944614...
						[Transfer] To Line	Entity 'Customer.823' transferring from '[FreeSpace] Model' into node 'Basic...
			BasicNode2	OnEntered	15	[Begin]	Process 'BasicNode2.OnEntered' execution started.
						[Fire] EnteredEvent	Firing event 'BasicNode2.Entered'.
						[Transfer] ToOutboundLink	Entity 'Customer.823' waiting at '[Node] BasicNode2' to transfer onto link 'Pa...
1/10/2011 01:27:42	1.46190718198696		Path3	OnEntered	138	[Begin]	Entity 'Customer.823' transferring from '[Node] BasicNode2' onto link 'Path3'.
							Process 'Path3.OnEntered' execution started.
						[Assign] OnEnteringAssign...	Assigning state variable 'Customer.823.OrderSize' the value '3'. Previous val...
						[Assign] EntityMovement...	Assigning state variable 'Customer.823.MovementRate' the value '5400' Me...

Example Custom Filter for Time Column

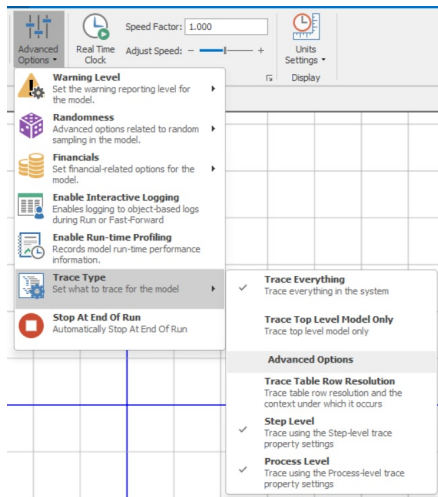
Filtering Trace

You can also right-click on the column header and select "Filter Editor" for a filter builder. When creating a filter in the Custom Filter or Filter Editor dialog, you can choose from different filter expressions such as "Equals", "Contains", or "Begins with". Most filter expressions use the exact specified string to filter on. For example, when searching for all the Delay steps in Trace, the filter expression might look like: **[Step] Begins with [Delay]**

However, the filter expressions "Is like" and "Is not like" can use special characters. These filters accept wildcards such that "." will replace a single character and "%" can replace any number of characters. These filters also use "[" and "]" to escape special characters. Square brackets are natively used in Trace items. If you need to filter with "Is like" or "Is not like" and you are looking for something with square brackets, you can wrap the first square bracket in square brackets, so it is not considered a special character. For example, again if I am searching for all the Delay steps, the filter expression might look like this: **[Step] Is like ([Delay])%**

Trace Type

Additionally, there are three types of trace that can be shown, determined by the Advanced Options > Trace Type within the Run ribbon. The default is to *Trace Everything*, which includes trace actions taken by tokens and elements at any level in the system. A second alternative is *Trace Top Level Model Only*, which will only display trace actions taken by tokens that are at the top level model. This can be useful with some debugging scenarios where the amount of trace generated by the system is large and the debug target is an item in the top level model (for example, a process). The third option is *Trace Table Row Resolution*. This trace option will generally increase the trace size substantially and thus should be used only when running to debug a particular table(s) reference issue. The existing table references for a token, associated object and other objects are displayed when resolving a table row reference. Additionally, you can toggle Step Level and Process Level depending on your desire to use process or step level tracing based on their respective property setting.



Step Level and Process Level Trace

These Trace Types determine whether the Step or Process properties will be used when deciding if these steps should appear in Trace when the model is run.

When enabled, these Trace levels might limit the visible steps in Trace. When debugging and needing to see all steps in trace, consider disabling the Step Level and Process Level Trace options.

The Process Level Trace setting decides if Trace uses the Process's *Allow Step Trace* property setting. When Process Level Trace is enabled, the *Allow Step Trace* property will be respected. When Process Level Trace is disabled, all processes will be considered as if the Process's *Allow Step Trace* property setting is set to 'True', even if set to a different setting.

The Step Level Trace setting decides if Trace uses the Step's *Trace Level* property setting. When Step Level Trace is enabled, the *Trace Level* property will be respected. When Step Level Trace is disabled, all steps will be considered as if the Step's *Trace Level* is 'Default', even if set to a different setting.

When a step uses a 'Default' *Trace Level*, it will defer whether it is visible based on its Process's *Allow Step Trace* setting and then the overall Trace setting.

Important: the decision to show a step in Trace is decided on a per step basis. A step's setting will supersede a process's setting.

For example, a Process with *Allow Step Trace* set to 'false' typically does not have any steps appear in Trace. But if a Step in that process had a *Trace Level* set to 'Always', the step's decision to always show is prioritized and that step will be seen in Trace even if the rest of its process will not.

Below are the tables outlining the different settings and when you can expect to see the Step in the Trace window. Recall that the Trace window can be open, and Tracing can be turned off, but steps can still appear in the Trace window.

Table 1: Trace setting Process Level is TRUE. Trace setting Step Level is TRUE.

Process's <i>Allow Step Trace</i>	Step's <i>Trace Level</i>	Tracing is ON – Step seen in Trace?	Tracing is OFF – Step seen in Trace?
True	Default	True	False
True	Always	True	True
True	Never	False	False
False	Default	False	False
False	Always	True	True
False	Never	False	False

Table 2: Trace setting Process Level is FALSE. Trace setting Step Level is TRUE.

Process's <i>Allow Step Trace</i>	Step's <i>Trace Level</i>	Tracing is ON – Step seen in Trace?	Tracing is OFF – Step seen in Trace?
True	Default	True	False
True	Always	True	True
True	Never	False	False
False	Default	True	False
False	Always	True	True
False	Never	False	False

Table 3: Trace setting Process Level is TRUE. Trace setting Step Level is FALSE.

Process's <i>Allow Step Trace</i>	Step's <i>Trace Level</i>	Tracing is ON – Step seen in Trace?	Tracing is OFF – Step seen in Trace?
True	Default	True	False
True	Always	True	False
True	Never	True	False
False	Default	False	False
False	Always	False	False
False	Never	False	False

Table 4: Trace setting Process Level is FALSE. Trace setting Step Level is FALSE.

Process's <i>Allow Step Trace</i>	Step's <i>Trace Level</i>	Tracing is ON – Step seen in Trace?	Tracing is OFF – Step seen in Trace?
True	Default	True	False
True	Always	True	False
True	Never	True	False
False	Default	True	False
False	Always	True	False
False	Never	True	False

Trace Table Row Resolution

This Trace Type option turns on an in-depth tracing of the resolution that occurs for table referencing. This shares the table row references for the token, associated object, and other objects taking part in the resolution process.

This can be turned on via the Run ribbon > Advanced Options > Trace Type > Trace Table Row resolution.

Note that with certain models using a large amount of table logic, enabling this feature could increase the trace size substantially. It is recommended to only turn it on after running to the simulation time you wish to debug something, then turn it back off again.

Understanding the types of row references

There are explicit, implicit, and potential row references provided in the Trace Table Row resolution. Explicit references are purposefully set rows. Implicit and potential rows are obtained when setting an explicit row where a Table Relationship is present.

With Relational Tables, the direction of the relationship is going to impact the explicit, implicit, and potential row references a Token/Entity may have. For example, ParentKeyTable has a Master Key column which corresponds to Foreign Key column in ForeignKeyTable. If a row reference is set to ParentKeyTable [1], there will be an explicit reference to ParentKeyTable [1], and a set of potential rows in ForeignKeyTable. The potential rows in ForeignKeyTable are the related rows in the child table.

[SetRow] SetRow1	Table reference for object or element 'DefaultEntity.12' set to row index '1' in table 'ParentKeyTable'.
[Assign] Assign1	Resolving row for table 'ParentKeyTable' for action 'GetPropertyGroupInstance'...
	Associated object (DefaultEntity.12) references:
	Table references: ParentKeyTable[1]
	Explicit selection: ParentKeyTable[1]
	Potential rows: ForeignKeyTable

If the reverse is done, and a row reference is set to a row in ForeignKeyTable, you will have an explicit row reference to ForeignKeyTable [1] and an implicit row reference to ParentKeyTable[1]. The child knows the exact Parent row it belongs to. There are no potential rows since the exact explicit and implicit rows are known between this table relationship. This means that there are no related rows, so functions that rely on related rows will not work. However the exact row reference is known, so Table Reference should work where needed without further work.

[SetRow] SetRow1	Table reference for object or element 'DefaultEntity.9' set to row index '1' in table 'ForeignKeyTable'.
[Assign] Assign1	Resolving row for table 'ParentKeyTable' for action 'GetPropertyGroupInstance'...
	Associated object (DefaultEntity.9) references:
	Table references: ForeignKeyTable[1], ParentKeyTable[1]
	Explicit selection: ForeignKeyTable[1]
	Potential rows: [None]

Copyright 2006-2025, Simio LLC. All Rights Reserved.

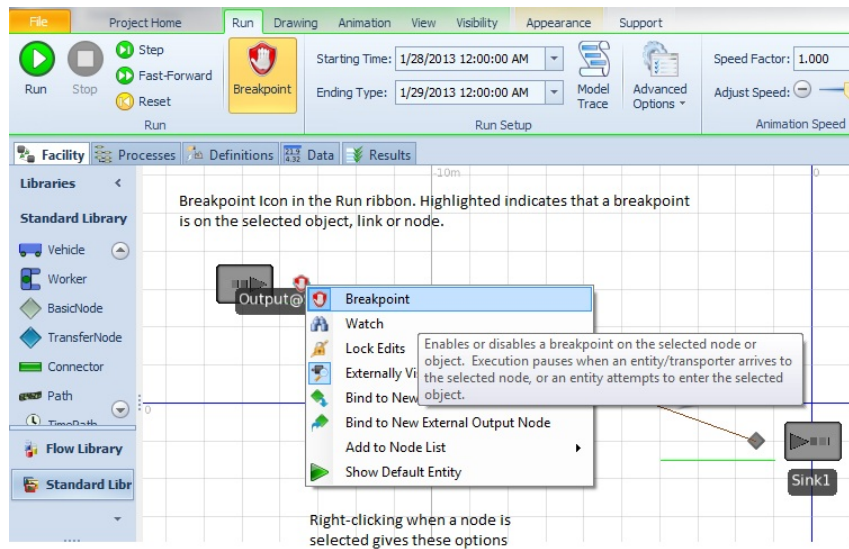
Send comments on this topic to [Support](#)

Breakpoints

Adding a Breakpoint

A Breakpoint can be added to an object, node or link in the Facility window, or to a step within the Processes window. It causes the execution of the model to pause when an entity or a transporter arrives to the specified object, node or link or when a token arrives to the specified step.

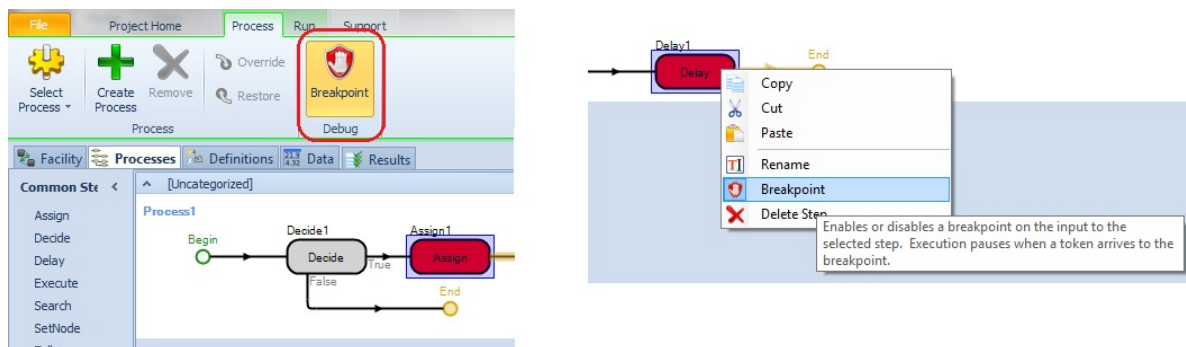
A Breakpoint can be added in two ways in the Facility window. A breakpoint may be added by clicking the Breakpoint icon in the Run ribbon when the object, node, or link is selected. Alternatively, if an object, node or link is selected in the Facility window, right-click will bring up a small menu that includes Breakpoint.



Similarly, within the Processes window, a breakpoint may be added to a step in several ways. First, the Breakpoint icon on either the Run ribbon or Process ribbon can be selected when the desired step is highlighted. Alternatively, if a step is selected, the right-click menu provides a Breakpoint option to turn the breakpoint off and on. Steps with breakpoints are then shown in red.

*NOTE: A breakpoint that is set on a step within a decision type process (i.e., OnEvaluating**) or OnRunEnding may be used to step through a decision process.

NOTE: If a Step is not going to enact any logic, the breakpoint on the step will be ignored. For example, a Delay step with time '0' or an Assign step with no assignments will be skipped and no operation performed. Any breakpoints on these steps will also be skipped.



Steps in Red indicate that a breakpoint has been set

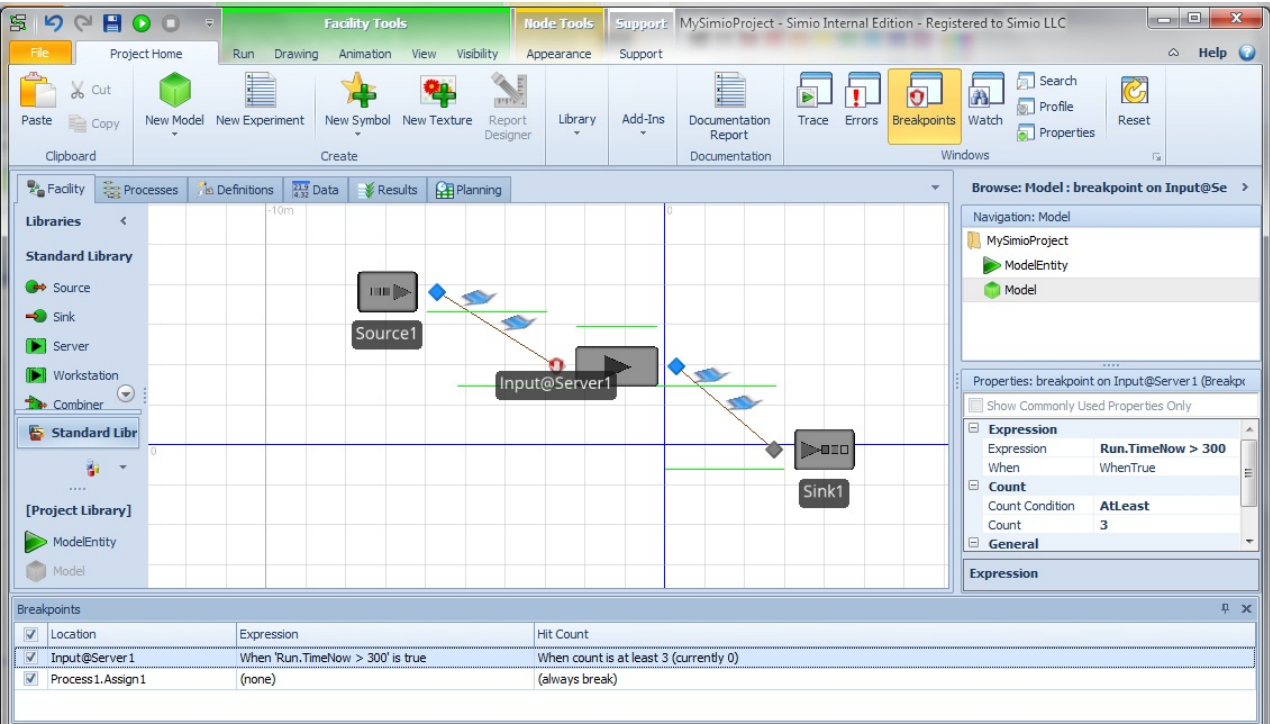
The Breakpoints Window

The Project Home ribbon provides an icon for opening the Breakpoints window. This window will display all of the breakpoints that have been set in the model, in both the Facility and Processes windows. Within this window, double-clicking on a given breakpoint will take you there, either in the Facility or Processes window. Additionally, there are several options for evaluating the breakpoint, including an Enable toggle, as well as Expression and Hit Count conditions.

The Enabled option allows users to toggle on/off the various breakpoints. If a particular breakpoint is disabled, the simulation run will not stop at that object or step. All breakpoints can be disabled by using the top row header toggle.

The Expression and Hit Count conditions, if specified, are displayed within the Breakpoints window. To add these conditions, simply highlight a particular breakpoint, and input the Expression and/or Hit Count information in the properties window to the right. The Expression and Count are combined; that is, the Expression is evaluated first, and if it evaluates to true, Simio will then check the Count criteria. Therefore, the hit count only begins if the expression (if any) is true to cause the break to occur.

Right-click within the Breakpoints window will open a context menu that can be used to remove the currently highlighted breakpoint or remove all breakpoints that have been set in the model.



Breakpoint Properties

Listed below are the properties of **Breakpoints**:

Property	Description
Expression	The expression to be evaluated.
When	The option for determining when to break on a specified expression. WhenTrue will break when the expression specified is evaluated to true. WhenChanged will break when the expression specified has changed from the previously evaluated value when that breakpoint object or step was reached.
Count Condition	The option for a count-based break, based on the number of times the breakpoint has been 'hit'. Always will not track the count, EqualTo will break when the hit count is equal to the Count value, and AtLeast will break when the hit count is equal to or greater than the Count value.
Count	The value associated with the Count Condition for breaking either EqualTo or AtLeast a certain value.

Start and End Time Options

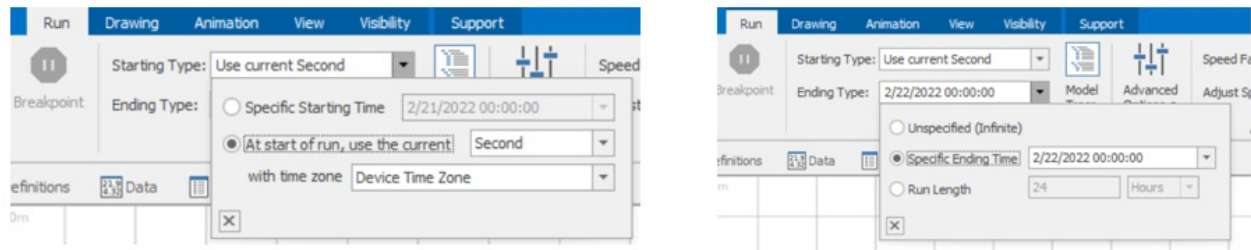
Start and End Time Options

The starting and ending time of the run is specified on the Run ribbon in the Run Setup area.

There are two choices of Starting Type: Specific Starting Time or At start of run, use the current *time unit*. When indicating a specific starting time, the time can either be changed directly in the text box or by selecting a date and time using the calendar feature available with the drop down arrow. When using the At start of run, use current *time unit* option, you may specify any time units, including second, minute, hour, day, week, month or year. You can also specify the time zone the model should be run in. IMPORTANT NOTE: when using this option, if your model includes Arrival Tables or Schedules (where the actual date/time is specified), your results may vary from one simulation run to another if you use small time units.

There are three choices of ending type: Unspecified (infinite), a specific ending time of a date and time, or a run length in hours, minutes, seconds, days or weeks.

Starting and Ending Types



Simio also provides the user with the ability to specify a stopping condition for the model run, which is done in the properties of both a StateStatistic and a TallyStatistic. See [Elements](#) for information on these statistic elements.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

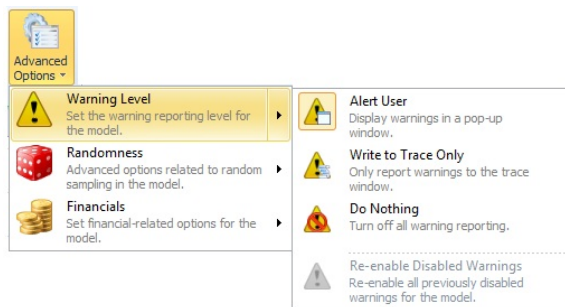
Warning Level

Warning Level

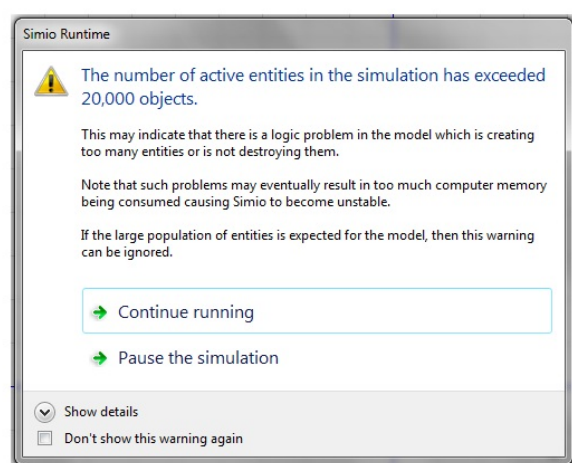
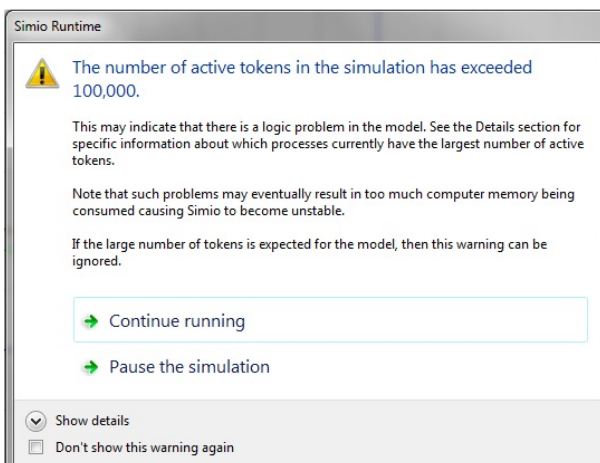
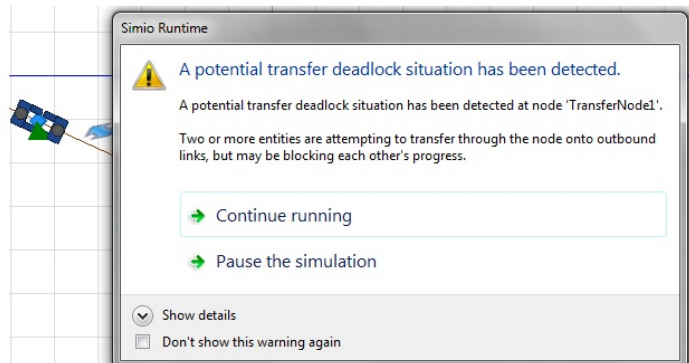
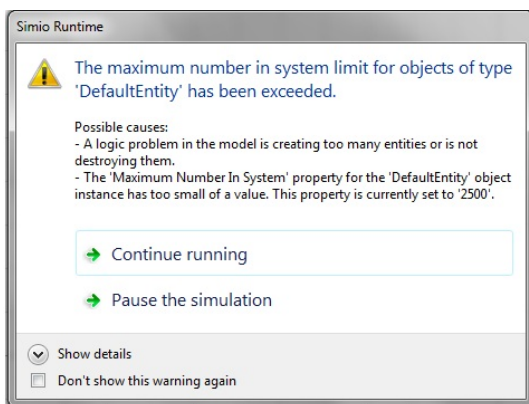
The Warning Level sets the warning reporting level for the current model. There are three levels of warnings:

- **Alert User** - Display warnings in a pop-up window. If this option is selected, the runtime warnings for the current model will be written to the Trace window and will also appear in pop-up message boxes during an interactive simulation run. The user will have the option of preventing individual warnings from appearing in pop-ups.
- **Write to Trace Only** - Report warnings to the Trace window. If this option is selected, the runtime warnings for the current model will be written only to the Trace window.
- **Do Nothing** - Turn off warning reporting.

When a warning pop-up is displayed, the user will be given the option to turn off future warnings of the same type. In the bottom left corner of the pop-up, a box may be checked that indicates "Don't show this warning again". Within the Advanced Options, Warning Levels area, there is an option to Re-enable Disabled Warnings, which will re-enable all previous disabled warnings for the current simulation model.



There are several automatic warnings that have been implemented within Simio. These include when the limit of a given object has been exceeded (as defined in the object's *Maximum Number in System* property), when a potential deadlocking situation has occurred at a node (when two bidirectional links intersect at a node), when the number of active tokens has exceeded a given value and when the number of active agents or entities in the model is exceeded (the last two possibly due to logic issues and potentially resulting to memory issues).



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Randomness

Randomness

Within the Advanced Options button on the Run ribbon within the Facility window and the Processes window, the options to Disable Randomness and specify the Random Number are available.

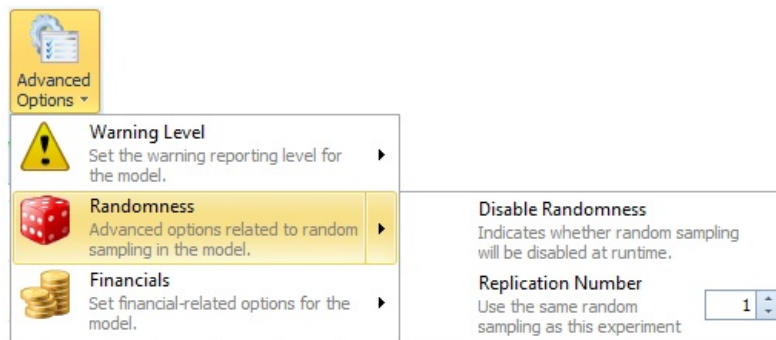
Disable Randomness

Disabling randomness is often useful to facilitate verification and debugging, and for generating deterministic schedules, but is not appropriate for most typical simulation applications. If the randomness is disabled, any random distribution reference will return the expected value instead of a random variate. Additionally any probabilistic selection in the logic (e.g., probability based Decide step, link selection weights, the table property RandomRow function, Random selection goal in Route step) will always select the choice that includes the 0.5 (or "50%") cumulative probability. When randomness is disabled, failures are also disabled.

There is a **Run.RandomnessDisabled** function that is available to determine during the simulation run whether the randomness is turned off or on. This function may be useful in process steps to change the logic or data in your system if the DisableRandomness button is 'True' or 'False'. For example, you may wish to have a Decide step that is 'ConditionBased' and evaluates the Expression 'DisableRandomness==False', meaning that distributions will be used. The True exit may have a Delay step that delays for Random.Normal(10,1), where the average delay is 10 minutes; however, the False exit may have a Delay step may delay for 11 minutes, using a 10% 'buffer' since there are no distributions utilized.

Replication Number

The Replication Number option is for use in interactive model only. This option may be used to generate the same sequence of random samples during an interactive run as would be generated for a specific replication when running the model in experiment mode. All random number streams will be initialized using seed values corresponding to the specified replication number. This is useful if the user detects an error, warning or problem with output in one of the simulation runs and would like to interactively view a particular replication.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Financials

Financials

Financials can be added to a model by either adding a [Cost Center element](#) manually and then assigning a value to the CostCenter.Cost or the CostCenter.Cost.Rate with an Assign Step, or by using the Financial properties on each Standard Library object. See the Financials category of the properties of each Standard Library object to see where to assign costs such as Capital Cost, Cost Per Use, Holding Cost and Transportation Costs.

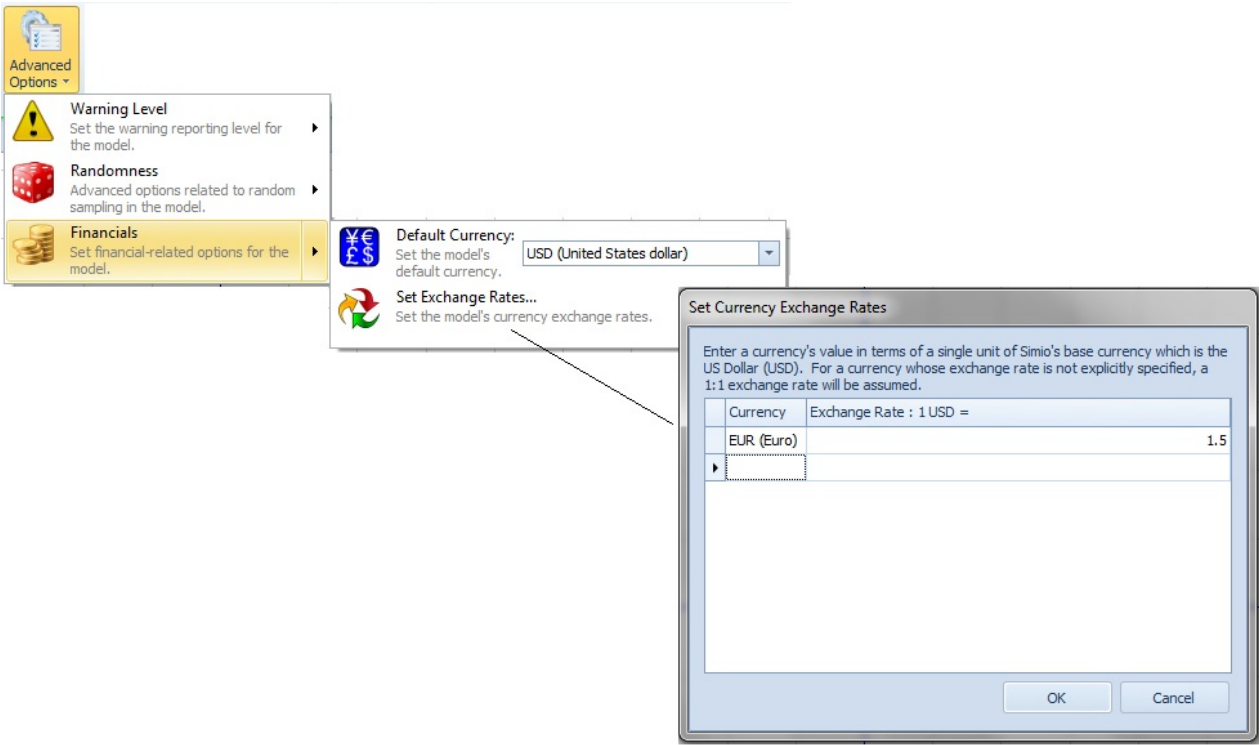
Within the Advanced Options button on the Run ribbon within the Facility window and the Processes window, the options to specify Financial information, including the default currency type and exchange rates, is available.

Default Currency

The default currency option sets the default currency units assumed by the simulation model for financial data entry and reporting purposes. The value is initially USD (United States dollar). Options to change the default currency can be found on the CostCenter element, as well as on any currency type state, such as Cost. The level state, Cost, is available on all intelligent objects and may be assigned by using the Assign step.

Set Exchange Rates

This option will set the exchange rates to be used by the model for conversion between any currency and Simio's base currency which is the US Dollar (USD). For a currency whose exchange rate is not explicitly specified, a 1:1 exchange rate will be assumed. When this option is selected, the dialog shown below is displayed. Under the currency column, a drop-down list of over 100 world-wide currency types is available. The exchange rate column requires an integer or real value.

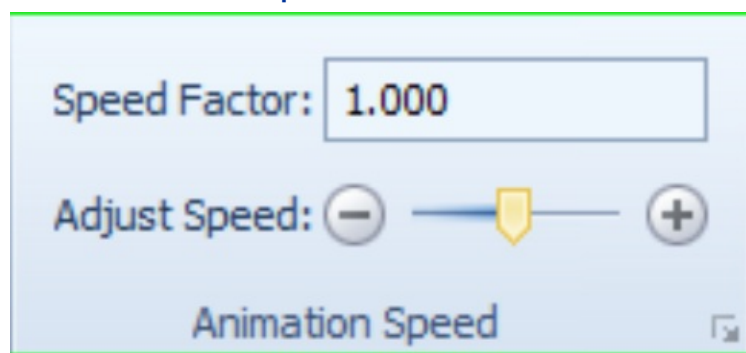


Animation Speed

Speed Factor

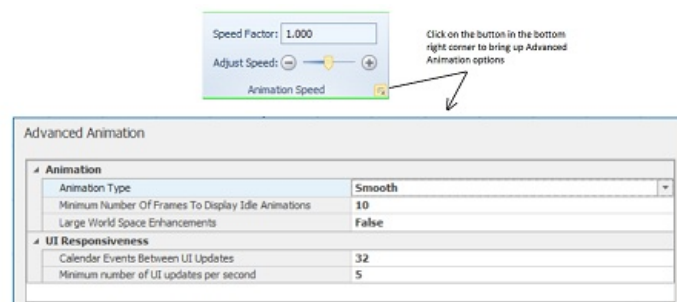
The Speed Factor represents the amount of simulation time between each animation frame. Lower values produce slower, more fluid animation. Higher values produce faster, more coarse animation. The time between animation frames can be adjusted by using the Adjust Speed slider or by typing in a new value into the Speed Factor text box. This is just an arbitrary scale. It will adjust how smooth the animation is by scheduling more or less frames to render. The speed of the animation will vary from machine to machine, depending on your machine's performance. The scale is in tenths of a second, so "10" will schedule a frame render for every 1 second of animation time, and "600" will schedule a frame render for every minute of simulation time.

Animation Speed controls on the Run Ribbon



Simio provides an Advanced Animation dialog that allows for several additional animation settings.

Advanced Animation Options



Within the Animation grouping, there are several options. *Animation Type* can be set to either Smooth or Coarse. Smooth animation will draw every animated frame, according to the speed factor. Coarse animation will draw less frames, which produces faster animation that is less smooth. We recommend using Smooth animation when recording a video with the .avi recording features.

Also under Animation, *Minimum Number of Frames to Display* is used as an advanced animation option. If a full cycle of a default idle animation will last less than this many animation frames, then Simio will instead just show the first frame of the idle animation. This gives the effect of faster animation speeds showing static poses (instead of very jerky animation), and slower animation speeds showing the full idle animation.

Large World Space Enhancements enables some enhancements to allow for displaying and interacting with things more correctly at larger distances from the origin (several hundred km). The enhancements sacrifice some drawing performance for correctness. The project will need to be reloaded to see this take effect.

Within the UI Responsiveness grouping, *Calendar Events Between UI Updates* indicates how many calendar events should execute before the UI is updated (including repainting the view for animation). A lower number may provide smoother animation and more responsive UI, a higher number will increase simulation speed.

Also within the UI Responsiveness grouping is the *Minimum Number of UI Updates Per Second*. For the interactive runs (including fast-forward and plan), this indicates the minimum number of UI updates (including repainting the view for

animation), that should be done per second. A larger number may provide more responsiveness UI, a lower number will increase simulation speed a small amount. Setting a value to zero here indicates to only use a count of executed calendar events to indicate when to update the UI.

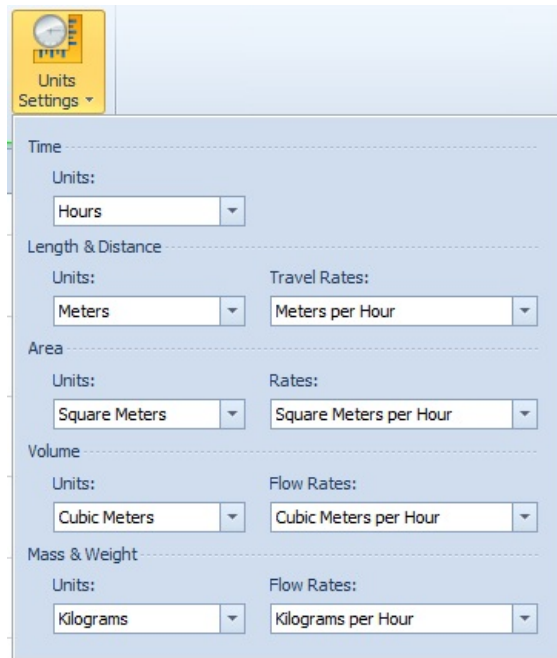
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Units Settings

The Unit Settings button provides a list of nine (9) combo boxes for various unit types in the simulation. The dialog allows the user to set the units used to display time, length and distance travel rate, area and rate, volume and volume flow rate, and mass and weight flow rate.

These units are displayed in the the reports, pivot grid, watch window, status bar and trace window. Note that Status Labels in the Facility window are always shown in the default unit setting.



The screenshot shows the 'Units Settings' dialog box in Simio. It features a yellow 'Units Settings' button with a gear icon in the top left corner. The dialog is organized into sections for different unit types, each with a 'Units' and a 'Rates' dropdown menu. The sections are: Time (Units: Hours), Length & Distance (Units: Meters, Travel Rates: Meters per Hour), Area (Units: Square Meters, Rates: Square Meters per Hour), Volume (Units: Cubic Meters, Flow Rates: Cubic Meters per Hour), and Mass & Weight (Units: Kilograms, Flow Rates: Kilograms per Hour). All dropdown menus are currently set to their default values.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

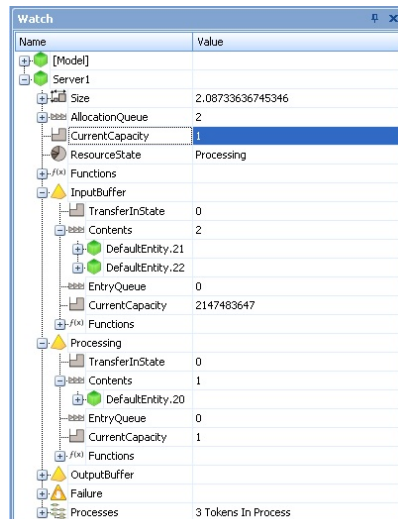
Send comments on this topic to [Support](#)

Watching an Object

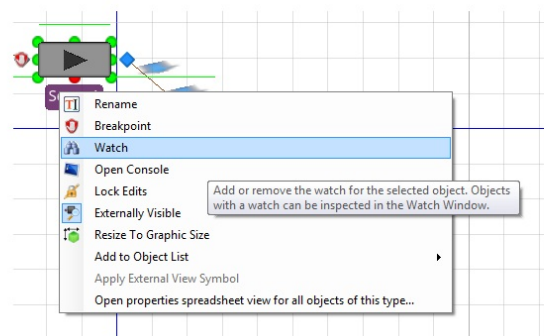
Watching States, Functions and Elements During an Interactive Run

The Watch window allows the user to view the values of an object's states, functions and elements during the run. The Watch window also includes Processes and their active tokens under their parent object, as well as child object information within a hierarchical parent object. A dynamic object, such as an entity, can also be watched but the Watch only lasts as long as the entity is in the system, i.e. until it is destroyed or the run ends. Watches added to instances persist across runs. It is important to note that Watches are not saved with the project.

You can explicitly show and hide the Watch window by using the Watch button in the Windows group on the Project Home tab. To add a Watch on an object, simply right click on the object in the Facility Window and select Watch. To remove a Watch from an object, right click on the object and select Watch again or select the row for the object in the Watch window and hit the Delete key. When the model is run, the values associated with that object will appear in the Watch window under the object's name. The values are updated only when the run is paused or each time the user hits the Step button from the Run group. The Watch window can be moved around the interface by clicking the top part of the window and dragging it to a new location. If it is moved to the middle of the interface, it can be resized to meet your needs.

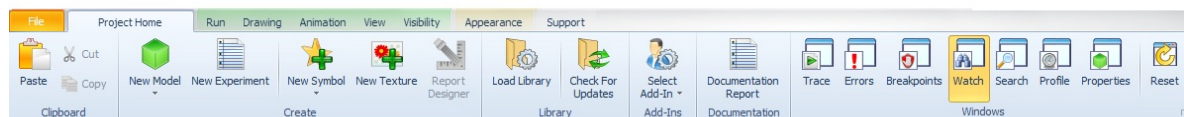


Name	Value
[Model]	
Server 1	
Size	2.08733636745346
AllocationQueue	2
CurrentCapacity	1
ResourceState	Processing
Functions	
InputBuffer	
TransferInState	0
Contents	2
DefaultEntity.21	
DefaultEntity.22	
EntryQueue	0
CurrentCapacity	2147483647
Functions	
Processing	
TransferInState	0
Contents	1
DefaultEntity.20	
EntryQueue	0
CurrentCapacity	1
Functions	
OutputBuffer	
Failure	
Processes	3 Tokens In Process



Right-click on an object to select Watch

A standard Server object in the Watch window



The Windows group in the Project Home Ribbon contains the Watch button to show or hide the window

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Debugging the Model

One of the most common tools for debugging a simulation model is the animation. Simio includes both 2D and 3D animation graphics that allow you to quickly and easily build a model and watch it run. By watching the animation, you can visually see the paths that entities take, the number of entities in various queues, vehicle movements and much of what is going on in the system. You may find issues that cause you to investigate further why things are happening as they are and Simio has a number of tools to help.

Trace Window

A very important tool for debugging a simulation model is the model trace. The model trace allows you to see exactly what is occurring during the simulation run with each step that occurs. This enables you to slowly “walk through” the simulation events one step at a time to determine whether the logic that has been incorporated is working as intended.

The Trace window can be displayed by clicking on the Trace Window icon in the Run tab of the ribbon menu. General information for turning on/off debugging tools can be found in the [Trace, Breakpoint, Running the Model](#) and [User Interface](#) sections of help.

More specifics about Trace can be found in the below section titled Trace Window Details.

Run Tab Options

There are several options available on the Run tab for running a model interactively. These include the Run/Pause, Stop, Step, Fast-Forward, Reset, and Reset Variables buttons. Typically when first running a model, you may like to set the Starting Time and Ending Type to the appropriate values (8 hours, 1 day, 10 days, etc.) and run the simulation. If your model runs and entities proceed through the objects as you’d expect them to, this may be all you are likely to do. However, to get a closer look at what is happening or to ‘debug’ a potential or known problem, you will need more information from the simulation run.

Run / Pause

The Run button is useful for running the simulation model without specific pauses between objects or for particular logic. Notice that when you click on the Run button, it becomes a Pause button that you can press to simply Pause the model at the current simulation time. The Pause button then reverts back to a Run button to continue the simulation.

Step

The Step button is extremely useful when tracing through the simulation run. When the active window is the **Facility window**, the Step button will stop the simulation when an entity moves from one object to another. However, stepping when the **Processes window** is the active window will stop the simulation as a token moves from step to step in a process. This includes the steps within the Standard Library objects.

Step in Facility window

For example, let’s say you have a simple simulation model of Source1 – Server1 – Sink1, connected by paths. If the Facility window is active, then using Step, the simulation model will pause and entity will be seen moving between the following objects with each step:

- Source1 (in OutputBuffer.Contents queue)
- Output@Source1 Transfer Node then moves along path to
- Input@Server1 Basic Node
- Server1 (in InputBuffer.Contents queue to the left of the Server)
- Server1 (in Processing.Contents queue above the Server)
- Server1 (in OutputBuffer.Contents queue)
- Output@Server1 Transfer Node then moves along path to
- Input@Sink1 Basic Node

Step in Processes window

If the Processes window is active, the simulation model will pause much more frequently, as it stops at each step within the processes window (including those built-in steps within a Standard Library object).

Many times it is also helpful to be able to view the animation while stepping through at a detailed level. This can be done by right-clicking on the Processes tab and selecting New Vertical Tab Group. This will make two windows visible within the main viewing area, where you may have both the Facility window and Processes window visible simultaneously. Making sure to have the Processes window active (highlighted), using the Step button in the same example above will stop at each individual step that occurs in the simulation. It is very useful at this time to have the Model Trace button activated and the Trace window open. The steps that occur within the above example are too numerous to detail, however, the processes that are occurring and the logic that is being executed are displayed in the Trace window.

Fast-Forward

The Fast-Forward button is useful when you would like to move quickly through a portion of the simulation model without viewing the animated objects in the Facility or Console windows. This is useful when you have a Breakpoint set and would like to get to that logic quickly. It may also be useful to run until a certain time, which we will discuss in the Running Until a Given Time section below.

Reset / Reset Variables / Stop

The Reset button can be used to re-initialize the model to its starting conditions. When Reset is used, all trace shown in the Trace window is cleared. The Reset Variables button is used to reset the model and clear the state columns without running the model’s initialization logic and automatic imports. The Stop button is also used to stop the simulation run. One difference to note is that the Trace shown in the Trace window is not cleared when using the Stop button. Trace can be viewed even when the model is stopped. Using the Stop button will cause the model to re-initialize upon using the Go or Step buttons.

Running Until a Given Time

When debugging a model, there may be situations where you want to run the simulation for a given amount of time before pausing the model to review everything in more detail. You may do this by specifying, through the Run tab, and Ending Type of Run Length with a given run length specified. Once the model has run (or fast-forwarded) for that time, this Run Length can be changed and you may continue to step through or run the model. For example, if you want to run the model for 2 hours then start stepping through the model, set the Run Length of 2 hours, and run the model (or fast-forward). It will appear as though the model has run to completion (i.e., End of Run shown on bottom left run status bar). You may now change the Run Length to 3 hours and the run status bar at the bottom left will change to “Paused”, where you may now wish to step.

Running until a given time is very useful if you have a modeling error at a specific time and would like to get back to that error quickly, or if you have a particular time during the simulation where you would like to observe in detail what is happening.

Running Until a Breakpoint

A breakpoint can be added to an object, node or link in the Facility window, as well as to a step within the Processes window. A breakpoint will cause the execution of the model to pause when an entity or transporter arrive to the specified object, node or link or when a token arrives at the specified step.

Within the Facility window, the breakpoint can be added to an object by highlighting the object and clicking on the Breakpoint icon on the Run tab or by right-clicking on the object and selecting Breakpoint. Within the Processes window, the breakpoint can be added by selecting the appropriate step and clicking on the Breakpoint icon in either the Process or Run tab.

Running until a breakpoint can be especially useful when defining a process within Add-On Process Triggers in the Standard Library objects. With a breakpoint set, the model will run until that particular step is executed. If the Processes window is active, you may wish to Step through the various steps in the process to make sure that the logic you added is working as intended.

Creating a Single Entity Through the System

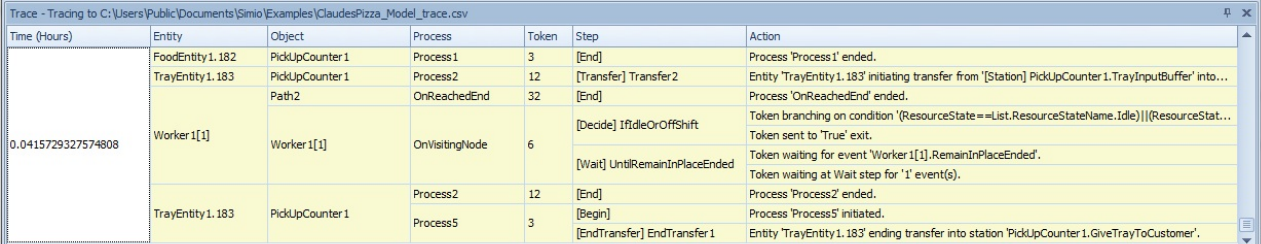
Depending upon the complexity of your system and reason for debugging, it may be useful to simplify the number of entities through the system. If you have a system with a lot of object interactions and/or you have an error occurring at a specific time with multiple entities, then this option would not work for you. However, if you are trying to determine why an entity is going a particular route or whether a specific process is working correctly, it may be very useful to allow only a single entity go through the system at a time.

What benefit does that provide? When tracing through specific logic or stepping to see an object move through the system, allowing only one entity at a time into the system may allow you to more quickly determine what the problem is without having interaction and movement of other objects in the system. Within the Stopping Conditions section of the Source, the Maximum Arrivals property allows you to specify the number of entities to create; for example, 1 for a single entity.

Trace Window Details

The Trace window is opened by clicking on the Model Trace button on the Run tab. Once the Trace window is opened, the actual model trace can be turned on/off by using the same Model Trace button. You will notice that within the Trace window, there are several columns to provide you with much information about what is happening in the simulation model. These columns include:

- **Time** – The current simulation time, in time units as specified on the Run tab. These time units may be changed during the simulation run and the change will be reflected in the Trace window.
- **Entity** – The current 'Associated Object' executing the logic. This is typically an Entity and includes ModelEntity objects, as well as Vehicles and Workers (since they are also "entities"). Entity references include the name of the ModelEntity instance, as well as a defined number, which is assigned by the simulation and will remain with the entity as it moves through the system until it is destroyed. Vehicles and Workers will include the name and dynamic object instance (i.e., [1], [2], etc.). Other references that are not Entities include timers and failures, for example, in which case you will see the 'Associated Object' for that timer or failure.
- **Object** – The name of the object where the current entity is evaluating logic. This may include names of fixed objects, such as Source1, Server1 or Resource1. It may also include nodes, or links such as Input@Server1 or Path1 or dynamic objects, such as Vehicle1[1] or Worker1[2]. Within the Process steps, this object is typically referred to as the 'Parent Object'.
- **Process** – The process in which the entity's associated token is executing. This may include a user defined process such as Server2.Processed or a Standard Library object process such as OnEnteredProcessing.
- **Token** – The token number that is executing a given step. This token is automatically created when the process execution is initiated and will exist only while in the steps in a given process. The token can refer to both the Associated Object and the Parent object.ok
- **[Step]StepName** – The internal name of the step that the token is executing; for example, Assign, Delay, Decide or Transfer followed by the user-defined name of the Step. When you are writing your own processes the user-defined name can be very helpful in understanding and debugging your model.
- **Action** – The action that is taken based on the step that is executed. For example, it may say 'Delaying Token for '0.32' hours until time '1.35' hours' or 'Token branching on condition 'Is.RedOnes, Token sent to False exit.' These actions are very useful in determining exactly what is happening in your model on a step by step basis.



Time (Hours)	Entity	Object	Process	Token	Step	Action
	FoodEntity1.182	PickUpCounter1	Process1	3	[End]	Process 'Process1' ended.
	TrayEntity1.183	PickUpCounter1	Process2	12	[Transfer] Transfer2	Entity 'TrayEntity1.183' initiating transfer from '[Station] PickUpCounter1.TrayInputBuffer' into...
		Path2	OnReachedEnd	32	[End]	Process 'OnReachedEnd' ended.
0.0415729327574808	Worker1[1]	Worker1[1]	OnVisitingNode	6	[Decide] IfIdleOrOffShift	Token branching on condition '(ResourceState=#List.ResourceStateName.Idle)'''(ResourceStat...
					[Wait] UntilRemainInPlaceEnded	Token sent to 'True' exit.
						Token waiting for event 'Worker1[1].RemainInPlaceEnded'.
						Token waiting at Wait step for 'i' event(s).
	TrayEntity1.183	PickUpCounter1	Process2	12	[End]	Process 'Process2' ended.
			Process5	3	[Begin]	Process 'Process5' initiated.
					[EndTransfer] EndTransfer1	Entity 'TrayEntity1.183' ending transfer into station 'PickUpCounter1.GiveTrayToCustomer'.

As you can see in the above picture, the model trace can be quite extensive. This trace output is automatically saved to the file 'ModelName_Model_Trace.csv' in the folder where the model is located. You may wish to later view the file, rename it or search the file for specific information.

Within the Trace window, the trace file can also be filtered to include only information you wish to view. Each column within the Trace window has a filter button to the right of the column name. For example, perhaps you would like to view the specific trace information for a given entity instead of all entities in the system. Within the entity column, click on the filter button and select the entity you would like to view. Filters can later be removed to view all of the trace content by selecting (All) from a specific column's filter list. The existing filters are shown at the bottom left of the Trace window.

Any runtime errors that occur within a simulation model run will automatically be displayed within the Trace window, regardless of whether the Trace Window button is enabled. This will allow users to more easily view the Entity, Object, Process and Step associated with the error that occurred.

To further investigate process logic, the Process and Step columns in Trace have additional functionality. You can right-click on a Trace entry's Process or Step cell and use the corresponding 'Go To...' option, if it is available. This will bring you to that exact Process or Step. The Processes for that object must be viewable for the option to be available. For example, the process logic inside the Model you are building and running will be available, as it will take you to the Processes tab in this model. Any other object definitions in this project, objects in the Navigation Pane, will also be available to navigate to. If you cannot navigate to the Process Logic window of that object in this project, the 'Go To...' options will be disabled for those Trace entries.

Reviewing Simulation Data via Animation objects or Watch window

Other tools are also available when debugging a simulation model. These include the ability to view certain pieces of information during the simulation model, from the number in a given queue to the value of particular function or statistic.

Animation in the Facility or Console windows

Within the Facility and Console windows, you may graphically animate any number of variables or functions by using the tools on the Animation tab, such as status labels, plots or pies. For example, a status label may be useful for displaying the value of a user defined state variable, 'inventory', that you increase / decrease throughout the model. Status symbols can be stand alone or can also be attached to a particular object. This can be done by highlighting the object, such as a Server or Node and selecting the animation symbol from the Symbol tab. See the following functions, states and events pages for [entity](#), [node](#), [transporter](#), [resource](#) and [link](#) objects for a listing of expressions you may wish to animate.

As just mentioned, animation can be attached directly to an object. This can be a stationary object, such as a Server, but can also be a dynamic object, such as an Entity, Worker or Vehicle. Viewing information, such as an property or state of an entity, as it moves through the system can provide invaluable information. Additionally, floor labels can include embedded expressions that can be very useful for debugging. See three SimBits, [Examples of Functions, Static Objects.spfx](#), [Examples of Functions, Dynamic Objects.spfx](#) and [Overflow WIP.spfx](#) to see many of the functions displayed in the Facility window and attached to objects.

Watch Window

The Watch window is another very useful tool for viewing information during the simulation run. Within this window, you can view the values of various states and functions for any number of objects in the system. To add a Watch on a particular object, right click on the object in the Facility window and select Watch. This will automatically open the Watch window at the bottom portion of the project. The Watch button on the Project Home tab can be used to turn the Watch window off and on. When the model is run, any objects that are being watched will be displayed with a "+". Clicking on the "+" will expand the object to show any functions, variables and processes associated with it. Refer to the [Watching an Object](#) page for more details.

Additionally, the model itself has a section containing any user defined state variables as well. The values are updated when the run has paused or when the Step button is used to move through the model. More detailed information about the user interface can be found in on the Watch window section of help. By watching the values of functions and states on specific objects and the model, this will allow you to make sure that values are being updated as you expect.

Summary

Simio includes many interactive tools for helping in the verification phase of your simulation project. Whether you have an error that has occurred in your model or simply are stepping through the model to be sure that the logic you have implemented is working correctly, using a combination of the Trace window, Watch window and/or animation within the Facility window can help provide great insight into the processes that are being run and decisions that are being made within your simulation model.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Simio Event Calendar

The event calendar is the central core of any simulation engine. Its purpose is to maintain an ordered list of events that need to be executed at the current and future times. These events correspond to token/entity movements through the system (e.g. an entity reaching the end of a link). In large models there could be many thousands of events scheduled on the event calendar to occur at different times. Having an efficient mechanism for handling a large number of scheduled events is crucial to fast execution of the model.

The Simio engine runs a model by executing the following event loop:

While (simulation is running)

{

Remove the next event from the event calendar

Advance Run.TimeNow to the time of this event

Execute the event logic for this event

}

Note that within this simple loop Simio is repeatedly removing the next event, advancing Run.TimeNow to the time of this event, and then executing the logic for that event. The event logic will in turn often schedule additional events to occur, either at the current time (Run.TimeNow), or after a time delay (Run.TimeNow + DelayTime).

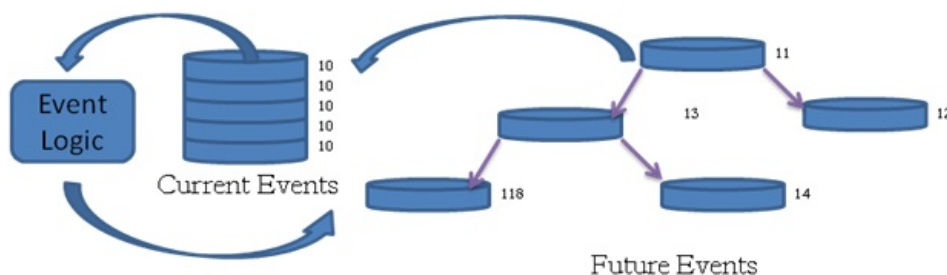
The most common event that is executed by Simio in this loop is a "token arrived to a step" event. Each step has a method that is called when this event executes. This method processes the logic for the step, making state changes as necessary to the arriving token, associated or parent object, and/or one or more elements in the model.

In many cases the step arrival event executes in zero simulated time and the arriving token can immediately continue to the next step without returning to the event calendar. However if a time delay occurs within the step (e.g. a Delay step) then the token arrival to the next step in the sequence is scheduled on the calendar to occur at the end of the delay.

Some of the steps in Simio have more than one token that departs the step at the same simulated point in time. In most cases the token departing the primary exit point is processed immediately and the token exiting the secondary exit point is scheduled to arrive to its next step at the current time (Run.TimeNow). Hence the arriving token and secondary token depart the step at the same simulated time, but one token executes its logic before the other. Each step with multiple departing tokens has a defined order for processing the departing tokens as documented in the Simio Reference Guide.

Two primary actions that repeatedly take place during the simulation run are scheduling of a calendar event to occur at either the current time or a later time, and removing the next event item from the calendar. These actions will take place millions of times in a typical simulation. In addition to these it is also sometimes necessary to cancel a scheduled event (i.e. remove it from the calendar even though it's not the next event), or adjust the scheduled event time. All of these actions must be very efficient in execution.

The scheduled events in Simio are maintained internally using two connected data structures illustrated below: an ordered list and a minimum heap. The ordered list holds all current events that are scheduled to execute at the current time (Run.TimeNow). The minimum heap is a tree structure that holds events that are scheduled to occur in the future. The scheduled event time for each node in the tree is always less than or equal to the event times for the two nodes that are immediately below it in the tree (if any). For example the top most event is scheduled to occur at time 11, and this is less than the event times (13, 12) immediately below it in the tree. As time advances events are moved from the future events heap to the current events list.



Each scheduled event item (either on the current events list or future events heap) maintains the following information:

Event time: The time that this event is to be executed.

Event method: A method that is to be called to execute the logic associated with the event.

Event object: An object (e.g. an entity) that is associated with the event.

Event index: An index that is incremented and assigned to each event item when it is scheduled.

Event type: An event type (Normal, Early, Late) that is used to break ties between events.

The events that are on the current events list are all scheduled to occur at the current time (Run.TimeNow). The times on the future events heap will be greater than or equal to Run.TimeNow. The algorithm executes each of the events on the current events list until it is empty. Note that during this period time does not advance. It then advances time to the time of the next event on the future events heap (always the top item on the heap) and continues removing all events from the heap that are scheduled at that time and places them on the current events list for execution.

The process of handling ties (i.e. events scheduled to occur at the same time) is very important in simulation engine design. Although a minimum heap is very efficient in handling large numbers of events, it is not very good at dealing with ties. In Simio the relative position of tied events are ignored when on the heap, but then sorted properly once moved to the current events list. This sorting is done first by event type to make events scheduled as *Early* to execute first, events scheduled as *Normal* to execute next, and finally followed by events scheduled as *Late*. Within an event type the events are ordered by event index; i.e. the order in which they were scheduled. An example of an *Early* event is an "initialization" event for the simulation. An example of a *Late* event is the "end of run" event that occurs at the end of the simulation. The vast majority of events are *Normal* events.

The heap is maintained during the simulation using operations that "bubble" an event up or down the tree. A bubble down operation compares an event item to its left and right child. If it is less than or equal to both it is in its proper location and the bubble operation is complete. Otherwise it trades places with its child with the smallest event time and this process continues until the event is bubbled down to its correct location in the tree. A bubble up operation is slightly faster because we are comparing an event to its parent, and trading with the parent if the event time for the parent is greater. Hence a bubble down requires two comparisons at each step where a bubble up requires only one.

Whenever we add a new event to the future event heap we add it to the next position in the bottom row of the heap and bubble it up to its correct location in the tree. Whenever we remove the next event from the calendar we remove the top most event node from the tree, and then move the far right event from the bottom row to the newly open position in the top row, and then bubble it down.

Note that if a scheduled event is canceled it can be removed from the tree and replaced with far right event from the bottom row which is then bubbled up or down as necessary. Likewise an event that changes its scheduled event time can remain on the heap and simply bubble up or down as necessary.

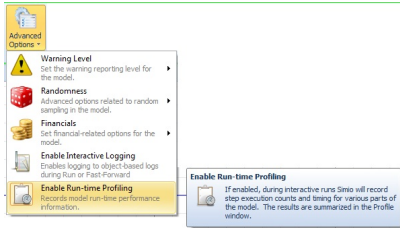
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Run-time Profiling

Run-time Profiling

The ability to Enable Run-time Profiling can be found in the Advanced Options button on the Run ribbon. The Profile window can be activated by selecting the Profile button in the Project Home ribbon under Windows.



Run-time profiling, if enabled, will record step execution counts and timing for various parts of the model to provide run-time performance information.

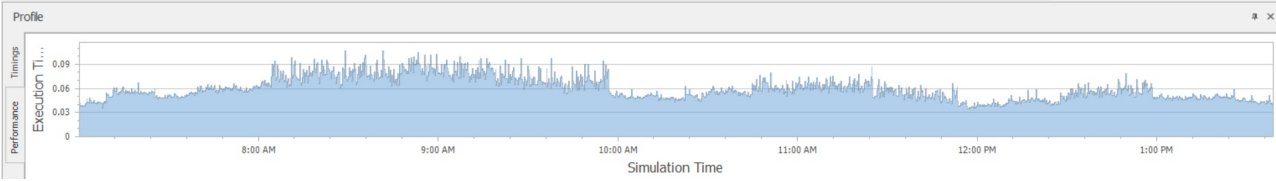
Example of a Timings Profile for the HospitalEmergencyDepartment Model

Profile						
	Item	% Time in Parent	% Time Overall	Count	Total Time (seconds)	Plot
Timings	[Animation Update]	99.65	99.6516	86400	447.4356055	
	[Calendar Event]	0.35	0.3483	86209	1.5639053	
Performance	Worker.OnCreated.OnCreatedAssignments [Assign]	0.00	0.0000	7	4.87E-05	
	Vehicle.OnCreated.OnCreatedAssignments [Assign]	0.00	0.0000	1	2E-05	
	Vehicle.OnRunInitialized.AtInitialNode [Park]	0.00	0.0000	1	1.48E-05	
	Worker.OnRunInitialized.AtInitialNode [Park]	0.00	0.0000	7	1.34E-05	
	Resolve Display Name (Nurse)	0.00	0.0000	316	1.21E-05	
	Resolve Display Name (Doctor)	0.00	0.0000	237	7.8E-06	
	Resolve Display Name (Billing)	0.00	0.0000	108	5.1E-06	
	Resolve Display Name (Room3)	0.00	0.0000	108	4.9E-06	
	Resolve Display Name (Bed6)	0.00	0.0000	114	4.8E-06	
	Resolve Display Name (Bed1)	0.00	0.0000	114	4.5E-06	
	Resolve Display Name (Bed4)	0.00	0.0000	114	4.2E-06	
	Processing.InitalCapacity (Expression Evaluation)	0.00	0.0000	18	4.1E-06	
	Resolve Display Name (WaitingArea_PostTriaged)	0.00	0.0000	108	4.1E-06	
	Vehicle.OnCreated.IfInvalidCapacity [Notify]	0.00	0.0000	1	4E-06	
	Resolve Display Name (WaitingAreaBeforeTriaged)	0.00	0.0000	108	3.9E-06	
	ParkingStation.InitalCapacity (Expression Evaluation)	0.00	0.0000	59	3.5E-06	
	MinimumOverTimeExpirationTimer.MaximumEvents (Expression Evaluation)	0.00	0.0000	8	3.5E-06	
	Resolve Display Name (Bed5)	0.00	0.0000	114	3.4E-06	

Within the Profile window, there are several columns that can provide information on run-time performance. The Object.Process.step information is specified under the "Item" column. The "% Time in Parent" column shows the step's run-time contribution to the parent object as displayed by the ">" symbol (expand to get more details on the steps beneath that ">"). The "% Time Overall" column shows what this step's run-time contribution is to the overall run, rather than just to its immediate parent. The "Total Time (seconds)" column shows the step's run-time contribution to the parent object as the actual real time in seconds. The "Count" column is how many times this particular step was hit. The columns can be sorted, and we default to what you see above. Note that sorting is within each row's parent row, rather than across all rows.

If you want to review the performance of the model over simulation time per walk-clock time, you can toggle to the Performance Profile to see a chart of this. Note: This Profiler only updates when the model is complete or paused.

Example of a Performance Profile for the HospitalEmergencyDepartment Model



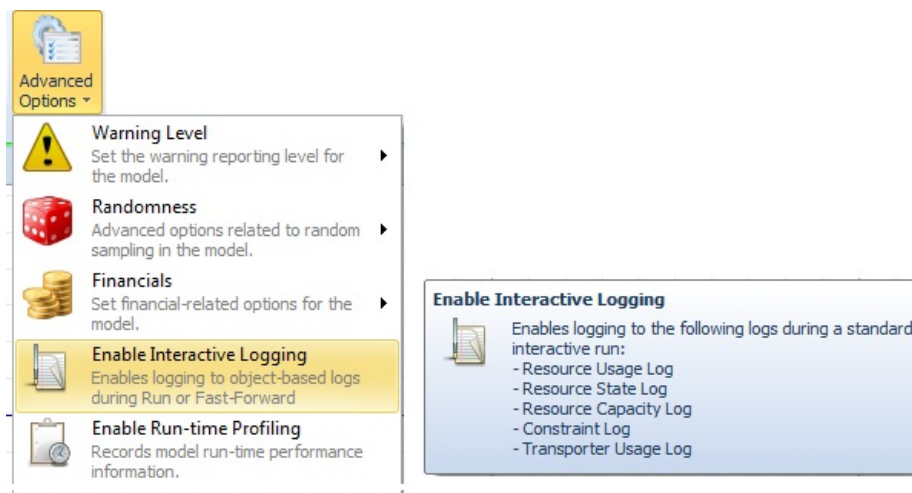
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Interactive Logging

Interactive Logging

Interactive Logging can be enabled by selecting the appropriate option within the Advanced Options dialog. This is only available in the Simio Professional and RPS Editions of Simio.



This option will enable logging to the following logs during an interactive run, including the [Resource Usage Log](#), [Resource State Log](#), [Resource Capacity Log](#), [Constraint Log](#) and [Transporter Usage Log](#). See the associated help topics for details on each of these log reports.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Simulation Replications

Simio Replications

Simulation results are generally based on replicating a simulation model multiple times to generate identically distributed and independent (IID) output samples. For example we might replicate a simulation model 25 times, and use the average waiting time recorded in each of the 25 replications as an estimate of the true but unknown value for the expected waiting time. Based on the IID assumption we can then use the 25 samples to generate a confidence interval on the expected waiting time.

Each replication will automatically initialize both the system state and statistics. The model state is initialized by initializing all objects within the model to their default starting state, and running the `OnRunInitialized` process for each object to allow user-customization of the initialized state. The model statistics are initialized by zeroing out all statistics values (e.g. the average time in system). If a warm up time is specified for the replication the statistics are initialized again at the end of the warm up time without reinitializing the system state. In other words the initial statistics recorded during the warm up period are discarded, but the simulation continues running from its current state.

Simio makes it easy to setup an experiment for a model that is comprised of one or more scenarios. Each scenario may have one or more controls (i.e. properties that supply inputs to the model), and one or more responses (i.e. a value such as average waiting time that is recorded at the end of each replication). Simio will then automatically run replications across each specified scenario, changing the values for the controls, and recording the values for the responses. The built-in SMORE plots will then display the results of a specified "primary" response for each scenario that combines error (confidence interval) and risk (percentiles) measures, and optionally plot individual samples and/or histograms of the individual response samples that are recorded at the end of each replication.

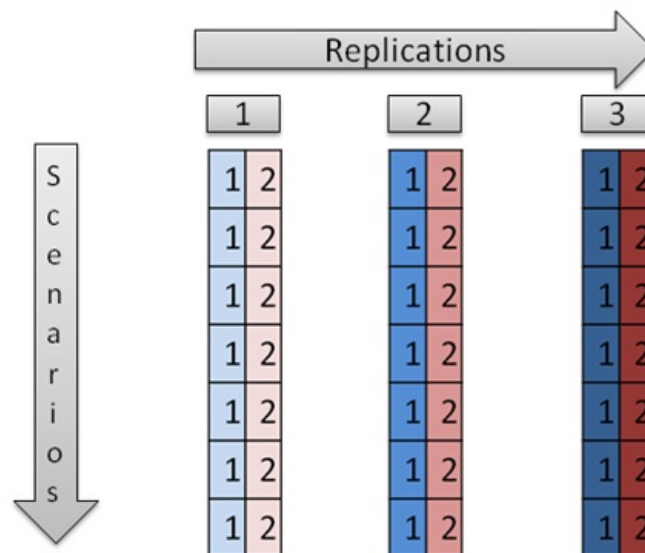
Each replication typically employs random numbers for generating samples for the time between arrivals, processing times, repair times, etc. These random numbers are automatically generated by Simio using the built-in Mersenne twister pseudorandom number generator. This state of the art generator has an extremely long period ($2^{19937} - 1$ or about 10^{6001}) that means from a practical sense the sequence will never repeat within a Simio simulation scenario.

The random numbers within a model are generated from independent streams, and you can employ as many streams within your model as you desire. The initialization seeds for each stream are automatically set by Simio. By default all random samples are taken from internal stream number 0, however you can specify a different stream by appending the optional stream parameter for each distribution. For example `Random.Uniform(3,5,1)` returns a random sample from a uniform distribution between 3 and 5 using random stream number 1. As we will discuss below the ability to select a specific stream is important when using a method called common random numbers (CRN) to reduce the variability in our estimators when comparing multiple scenarios.

When replicating a simulation model each replication samples from a different but defined starting point within the stream. These starting points are set very far apart in the stream so that there is no practical risk of overlap. Since the random numbers in each replication are different, each replication produces a statistically independent estimate for each of the output responses.

Although the random numbers that are used across replications are different (because they have a different but defined starting point in the stream), they are identical for each replication within each stream across scenarios. Hence rerunning a simulation experiment with the same set of controls will always produce exactly the same results for each replication. For example the random numbers generated for stream number 3 and replication number 8 will always be the exactly the same sequence of numbers, regardless of which scenario is being evaluated. This is important for two reasons: first it makes our simulation experiments repeatable, and second it makes it possible to use common random numbers across scenarios.

The random sampling from streams across replications and scenarios is illustrated in the following example with 3 replications, 2 streams, and 7 scenarios. Stream 1 is represented by shades of blue and stream 2 is indicated by shades of red. The different shades of blue and red indicate that a different segment of random numbers within each stream is being used across the three replications. Hence although the three replications sample from the same streams, the random values returned from those three streams are different because we start of different positions within those streams. Hence the results across replications are independent samples. However if we look down across scenarios we see that the random samples by stream and replication number are identical across scenarios. Hence we have the potential to have highly correlated results across the scenarios.



This pattern of random sampling supports the concept of common random numbers to reduce the variability of our estimators when comparing across scenarios. The basic idea of common random numbers is that we would like the differences between scenarios to largely reflect the differences in control settings and not the particular set of random values that were used for a particular scenario. In other words we would like to subject all the scenarios to the same random pattern of arrivals, process times, etc., and in doing so create correlation across the scenarios, while keeping independence across the replications. This will improve the underlying results for many statistical analysis procedures that compare alternatives. For example both the built in Simio subset selection procedure and the Nelson and Kim add in for selecting the best system benefit from the use of common random numbers.

To make effective use of common random numbers it is important to create as much correlation as possible across the scenarios. This is done by ensuring (to the extent possible) that the common random numbers that are generated across scenarios are used in the same way. Note that letting all random variables sample from the default stream (0) will typically not achieve this because the controls on the model (e.g. number of servers), will change the sampling pattern from the stream, and therefore a random sample that was used for a processing time may now be used for an inter-arrival time, etc. Ensuring the exact same usage of random numbers is not always convenient to do in all cases throughout the model, but typically very easy to do for arrival processes. By using a separate random number stream for all arrival processes we can guarantee that we get the exact same sequence of arrivals across all scenarios.

A good practice is to always use stream 1 for inter-arrival times, stream 2 for task times, and then default everything else to stream 0. This is very simple and effective way to induce correlation across scenarios and improving our ability to draw conclusions from the results.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Experiments

Experiments

Experiments are used to define a set of scenarios to be executed using the model. They are executed in batch mode. This is typically done (once a model has been validated) to make production runs that compare one or more variations of the system.

Each scenario has a set of [control variables](#) (e.g. size of each input buffer) and [output responses](#) (e.g. the throughput, waiting times, etc.). The control variables are the values assigned to properties of the associated model. Before adding an experiment you typically add [properties](#) to your main model to serve as the controls that you want to change for each scenario. Since most models contain random components (e.g. service times, failures, etc.) replications of the scenario are required to allow the computation of confidence intervals on the results.

For every data table with 2 or more data bindings in the model, there is a column within the Experiment Design view of an experiment. The header of a column is the table name and the cell values are a drop down of the named bindings for that table. For a particular scenario, a user can set the various bindings to be used per table. The default value of the cell for a new scenario will be the currently active databinding for that table. When using the OptQuest add-in, Simio also uses the default active binding for each table. The default active binding is set on the data table in the "Active Import Binding" or "Active Export Binding". See the [Data Connectors](#) page for more information.

Note: Even data tables with a *Binding Option* of 'Manual' will be treated as an 'Automatic' binding in an Experiment. An 'Automatic' binding imports data at the beginning of the run without any user interjection.

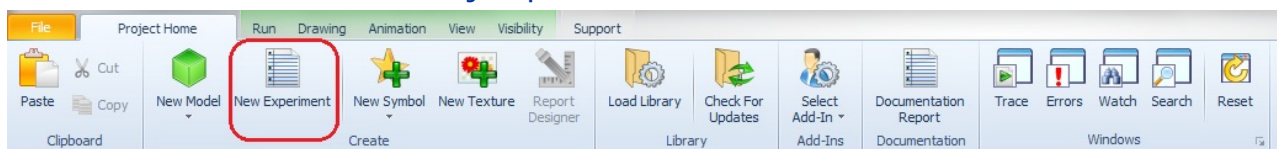
An experiment is not owned by a model. Rather, an experiment references and uses a model (is "bound" to a model). This means that, if a model is removed from a project, any experiments that referenced and used that model may remain in the project to preserve experiment results. Users should be able to distribute experiments between projects.

Within the [Analysis](#) section of the Experiment properties window, a user can specify a warmup period for the run, as well as a confidence level.

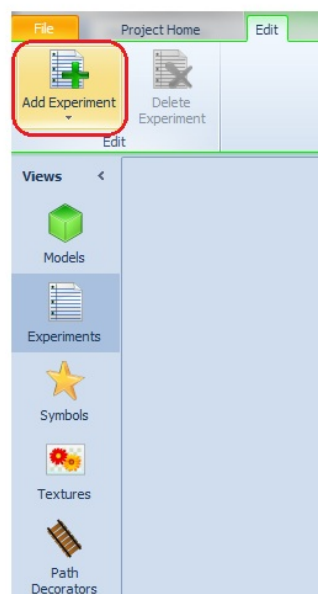
Creating a New Experiment

An Experiment can be added to a project in two different ways. It can be added from within the Project Home ribbon by selecting the New Experiment icon. A fixed type model must be active (highlighted in the Navigation window) and the new Experiment is bound to this active model. An Experiment can also be added from within the Experiments panel in the Project window. The drop down menu of the Add Experiment icon in the ribbon gives the user a choice of all the fixed type models that exist in the project. The user must select a model for the experiment to be associated with. The experiment can then be renamed, edited, duplicated and deleted from the Experiments panel.

Adding an Experiment



Add an Experiment from Project Home Ribbon



Add an Experiment from the Experiments Panel in the Project Window

Adding Scenarios to an Experiment

To add additional scenarios to an experiment, either click in the checkbox to fill in the columns with the default information, or manually type in the Scenario Name and all the default information will be populated for that scenario. Make any necessary changes to the inputs and then click anywhere outside of the new row to add this new scenario to the experiment. A checkbox appears to the left of each scenario. Only scenarios that are checked will be enabled. This provides a way to do additional replications on selected scenarios. Using Ctrl-C to copy a particular scenario and Ctrl-V to paste into a new scenario works as well.

Adding a Scenario to an Experiment

Scenario		Replications		Controls	
<input checked="" type="checkbox"/>	Name	Status	Required	Completed	ServerCapacity
<input checked="" type="checkbox"/>	Scenario1	Idle	10	0 of 10	1 Random.Normal(2, 1)
<input checked="" type="checkbox"/>	Scenario2	Idle	10	0 of 10	2 Random.Normal(2, 1)
<input checked="" type="checkbox"/>	Scenario3	Idle	10	0 of 10	0 0.0
<input type="checkbox"/>					

Either click on the checkbox to automatically add a scenario or start typing in the Scenario Name. Either way, the default property information is displayed in the Controls.

The control values for a given scenario may be copied into the Model properties in the Facility window. See Controls for additional information.

The control values that are currently set in the Model properties in the Facility window may be copied into a Scenario by right clicking on the Scenario and selecting "copy the model's control values to this scenario".

Sorting and Filtering the Experiment Grid

The columns of the Experiment grid are categorized for easier reading. Columns and categories can be reordered. The Experiment Grid can be filtered and sorted. Clicking in the upper right hand corner of a column will bring up the filtering menu. This allows the user to select a value to filter by or allows the user to create a custom filtering expression. The grid can also be sorted by clicking once or twice on the column header to be sorted. Multiple sorts can be added with shift-click.

Additionally, the columns of the experiment grid can be rearranged. The first row of column headers (Scenario, Replications, Controls) can be dragged around to change the order of the columns. For example, a user might want to have the Controls columns appear first in the grid and therefore they would drag the Controls column header to the left of Scenario. Similarly, a user can rearrange the second row of column headers, but only within the umbrella of the parent column header. For example, a user can rearrange the order of the individual Controls columns by dragging the columns around underneath the larger Controls column.

Filtering the Contents of a Column within an Experiment

Scenario		Replications		Controls	
<input checked="" type="checkbox"/>	Name	Status	Required	Completed	ServerCapacity
<input checked="" type="checkbox"/>	Scenario1	Idle	10	0 of 10	Random.Triangular(.1,.2,.3) 1
<input checked="" type="checkbox"/>	Scenario2	Idle	10	0 of 10	Random.Triangular(.1,.2,.3) 2
<input checked="" type="checkbox"/>	Scenario3	Idle	10	0 of 10	Random.Triangular(.3,.4,.5) 1
<input checked="" type="checkbox"/>	Scenario4	Idle	10	0 of 10	Random.Triangular(.3,.4,.5) 2

Click in the upper right hand corner of a column for filtering

Reset Scenario Results

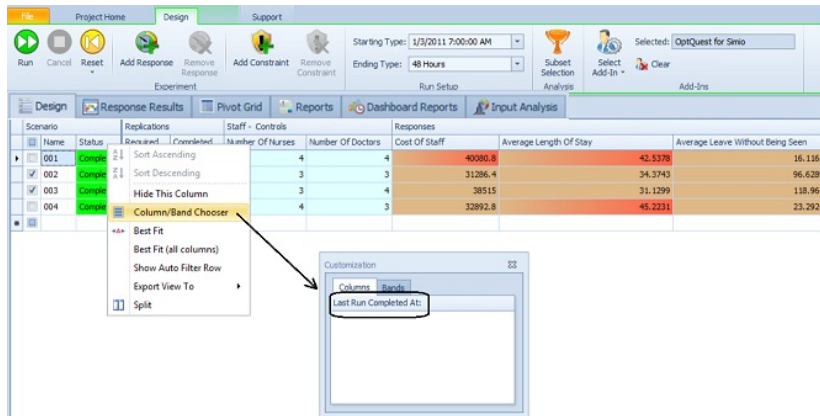
The Reset grouping of options allows users to reset all scenario results or to selectively reset only a smaller grouping of checked scenarios. The Reset Selective Scenarios is especially useful if a user only wants to re-run a few new scenarios with new control values while keeping the other scenario results.

Reset Options from the Design Ribbon

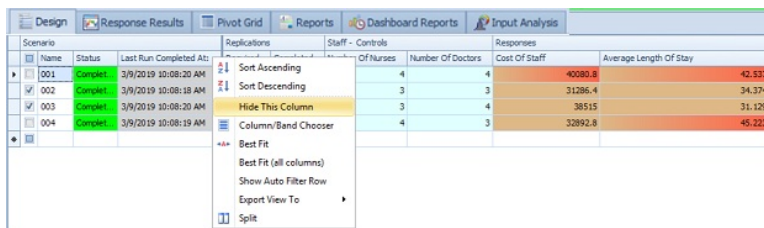
File		Project Home		Design		Support	
	Run		Cancel		Reset		Add Response
	Remove Response		Add Constraint		Remove Constraint	Starting Time: 1/3/2011 7:00:00 AM	
						Ending Type: 48 Hours	
						Run Setup	
						Input Analysis	
Scenario		Replications		Controls			
<input type="checkbox"/>	Name	Status	Required	Completed	ServerCapacity		
<input type="checkbox"/>	001	Idle	10	0 of 10	3	3	31286.4
<input checked="" type="checkbox"/>	002	Idle	7	7 of 7	3	4	38515
<input checked="" type="checkbox"/>	003	Idle	10	10 of 10	3	4	38515
<input type="checkbox"/>	004	Idle	5	5 of 5	4	3	32871.5

Viewing Scenario Timestamp

Users may wish to view the Scenario timestamp column that displays when the experiment scenario was last run. To view this column, right Click on a column header (such as Status, Required, or any column header) in the Experiment View. Select 'Column/Band Chooser' and a small box appears. In the box, double click 'Last Run Completed At'.



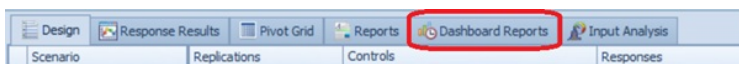
It now appears next to Status column. To hide the column, Right Click 'Last Run Completed At' and select 'Hide This Column'.



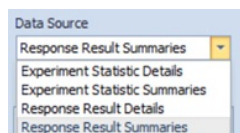
To unhide any hidden columns, again right click the column header and go to the 'Column/Band Chooser'. The Customization window will contain the hidden column name(s). The column name can be double clicked to automatically add the column to the experiment, or it can be dragged and dropped to a specific column position. Currently viewable columns can also be hidden by dragging the column into the Customization Window.

Experiment Dashboard Reports

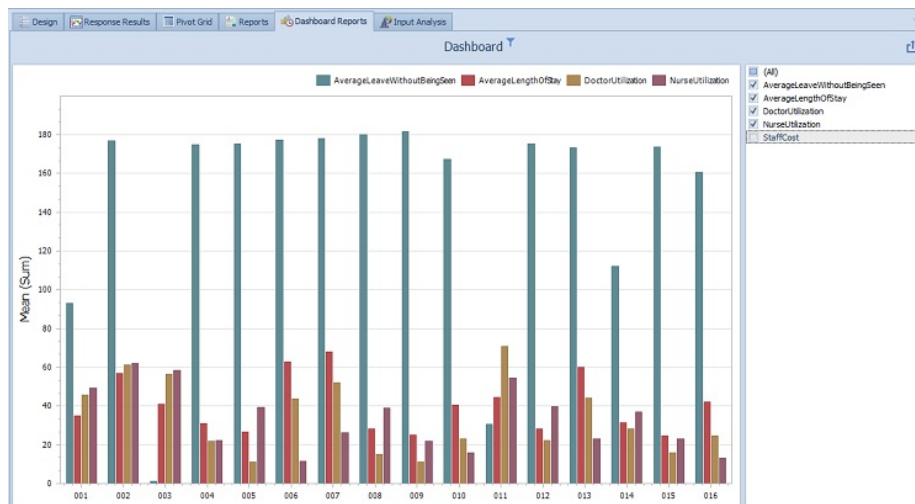
We have added the ability to generate Dashboard Reports within the Experiments of a project. This can be done by using the new Dashboard Reports tab:



This tab opens a view like the Dashboard Reports view for interactive runs. Under Experiments, however, there are a different set of data sources, as shown below, where the "Summaries" data sources include average/half-with/min/max across replications, and the "Details" data sources have the individual replication observations.



In the below graph, the HospitalEmergencyDepartment example was used, with additional Responses to capture ScheduledUtilization statistics for the Nurse and Doctor workers. Once the experiment scenarios were run, the Response Results Summaries were used to generate a 'Chart' type dashboard. Within the Chart's Data Items, the 'ResponseName' is used within the Series, the 'ScenarioName' is specified under Arguments and the 'Mean' under the Values section. A Filter Element (List Box) was used for the filtering section on the right side of the dashboard below ('ResponseName' under Dimensions Data Items).



Batch Means and HalfWidth Calculations

The method of batch means attempts to obtain a sequence of independent samples (batch-means) by aggregating h successive observations of a steady state simulation. For a given single replication, data points are divided into batches and the average value of the data points within each batch are the batch means. These batch means are then used to calculate the stochastic process' half width, which is a measure of how far the system tends to stray from its average value.

This batch means method is automatically enabled for TallyStatistics and StateStatistics for experiments with a single replication. Simio's standard library has a TallyStatistic (in Sink). All other automatic statistics within the Standard Library objects are custom generated and do not use StateStatistics or TallyStatistics elements. For users that would like to see a batch means calculation on 'utilization', for example, would need to manually add a StateStatistic to mimic our internal custom statistic.

For the interactive simulation run, Simio does not utilize batch means, and therefore, no half-width calculations are made.

For the experiment, if there is more than one replication, Simio does not generate the batch means half-width. However, if there is one replication, the confidence level from the element (or from the experiment if the element says 'Default') is used to calculate the batch means half-width.

Write Step in Experiments

When a Model contains Write Steps in its Process Logic, and an Experiment is run, the output files will be merged for improved data portability, provided in addition to a file per Scenario per Replication.

To merge the files, set *Auto Merge Write Step Files For Experiment* to 'True' on the File Element.

To make the most use of the merged file created from Write steps via Experiments, consider adding Run.ScenarioName and Run.ReplicationNumber as Items so record entries can be associated with the corresponding information.

Exporting Logs for Each Replication

To export various Logs for each Replication of an Experiment, toggle on *Export At End Of Experiment Replication Run*. It is recommended to add additional columns denoting ScenarioName and ReplicationNumber to help distinguish between Replications. Note: This will create a large data set that may slow the model down.

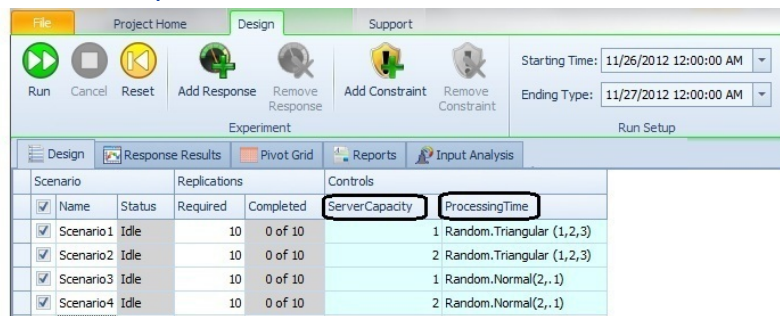
Controls

Control Variables

The control variables are the values assigned to properties of the associated model. Before adding an experiment you typically add [properties](#) to your main model to serve as the controls that you want to change for each scenario. Once a property exists in your model, it will appear as a column in the experiment table in the Experiment Design Window. The property's *DisplayName* is what will appear as the column name. The user can create multiple scenarios, each with a different value in the control variable.

Even though there are default values for the properties (controls), the experiment will use the values that are specified in the Experiment Design table. Using the example below, if the default value of the ServerCapacity property is set to 3, the experiment will ignore this and use the values specified in the table (1, 1, 2, 2), for each scenario, respectively. Therefore, do not assume that the default property values will be used if the field in the Experiment Design table is left empty. The experiment will use the value of 0.

Experiment with Multiple Scenarios of Different Controls

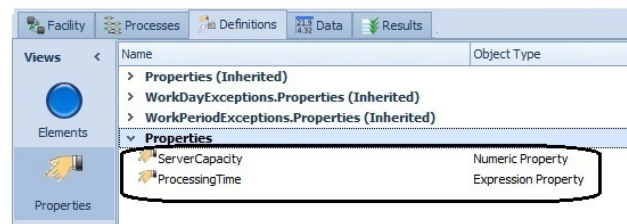


Scenario		Replications		Controls	
Name	Status	Required	Completed	ServerCapacity	ProcessingTime
Scenario1	Idle	10	0 of 10	1	Random.Triangular (1,2,3)
Scenario2	Idle	10	0 of 10	2	Random.Triangular (1,2,3)
Scenario3	Idle	10	0 of 10	1	Random.Normal(2,, 1)
Scenario4	Idle	10	0 of 10	2	Random.Normal(2,, 1)

Experiment Window - Controls



Facility Window - Using Properties in the Server



Definitions Window / Properties Panel - Defining the Properties

Copying the Control Values of a Scenario to a Model

There may be times when you'd like to use the control values specified in one or more scenarios within the interactive model run. This can be easily done by right clicking on a given scenario and selecting "Copy this scenario's control values to the model." This will take all the control values for the specific scenario selected and push them into the model's property values, as seen in the screenshot below.

FileProject HomeDesignSupport

RunCancelResetAdd ResponseRemove ResponseAdd ConstraintRemove Constraint

Starting Time: 10/19/2009 7:00:00 AM

Ending Type: 4 Hours

Subset Selection

Select Add-InClear

Selected: OptQuest for Simio

ExperimentRun SetupAnalysisAdd-Ins

DesignResponse ResultsPivot GridReportsInput Analysis

Scenario		Replications		Controls				Responses		
Name	Status	Required	Completed	Checkin Kiosks	Checkin Clerks	ID Check	Scanner Stations	Revenue	Profit	Cost
001	Idle	5	5 of 5	5	5	1	2	32111.2	27681.2	4430
002	Idle	5	5 of 5	1	1	1	2	11642.4	7532.4	4110
003	Idle	5	5 of 5	8	8	1	2	37312	32642	4670
004	Idle	5	5 of 5	3	3	1	2	27834.4	23564.4	4270
005	Idle	5	5 of 5	6	6	1	2	34962.4	30452.4	4510

Copy this scenario's control values to the model.

Browse: Model\Experiment1

Navigation: Model\Experiment1

AirportTerminal

ModelEntity

Model

Experiments

Experiment1

Browse: Model : Model

Navigation: Model

AirportTerminal

ModelEntity

Model

Experiments

Experiment1

Properties: Model (Fixed Model)

Show Commonly Used Properties Only

Controls

General

Checkin Kiosks	8
Checkin Clerks	8
ID Check	1
Scanner Stations	2

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Responses and Constraints

Often times a system will have user defined limitations – they could be putting a limit on the total number of cashiers at a grocery store, setting a maximum time a customer is allowed to be waiting on hold at a call center, or making sure that the total cost of implementing a new system is within budget. These limitations are essential for correctly modeling a process and must be represented by your simulation because without them, your model can return infeasible results. Simio allows you to easily incorporate these limitations into your model through experimentation. If you would like to determine if the result of a Scenario is acceptable then you can set an Upper or Lower Bound on a **Response**. If your limitation involves Controls “interacting” with each other – such as the sum of the lengths of a set of conveyors has to be less than a certain value – then you can represent this using a **Constraint**.

Responses

A Response specifies an expression that typically includes statistics recorded during the run, such as average waiting time or throughput, etc. The response is recorded at the end of each replication of each scenario.

A value can be defined for a Lower Bound and an Upper Bound for this response. Simio will color the response red if it fall below the lower bound or falls above the upper bound. Simio does not do anything else with the bounds except for changing the color in the interface. These bounds will be used in the future with Experiment Add-Ins. Similarly, the response property named *Objective* will be used in the future for Experiment Add-Ins.

To add a Response, simply click the *Add Response* button in the Experiment section of the Experiment Design Ribbon. The resulting Response column can either act as a purely informative display of a certain statistic of interest or an objective function that you are trying to optimize. You can customize a Response column type with the information entered in the following properties:

- Name – The name of the response, which is reflected in the column header.
- Description – The description of this response.
- Expression – The expression to be evaluated at the end of the simulation run.
- Objective – Indicates if larger or smaller values of this response are preferred. This is available for use with Add-Ins, such as the *OptQuest Add-In*. It determines how OptQuest is to use the response information when determining the optimal solutions. Objective can either be:
 - None – Just an informative response and not used to make decisions
 - Maximize– OptQuest drives the Experiment to achieve the greatest values for the objective function
 - Minimize– OptQuest drives the Experiment to achieve the lowest value for objective function
- Upper Bound – Any values for this response above this value are considered infeasible
- Lower Bound – Any values for this response that are below this value are considered infeasible

The response can be removed from the experiment by clicking the Remove Response icon in the Design tab of the ribbon menu.

Adding a Response to an Experiment

The red color indicates that the Response result is outside of the Lower and Upper Bound

Scenario	Replications	Controls	Responses
Name	Status	Required	Completed
Scenario1	Completed	5	5 of 5
Scenario2	Completed	5	5 of 5
Scenario3	Completed	5	5 of 5
Scenario4	Completed	5	5 of 5

Properties: NumberProcessed (Response)

General	
Name	NumberProcessed
Display Name	NumberProcessed
Expression	Server1.Processing.NumberDepartures
Unit Type	Unspecified
Objective	None
Lower Bound	60
Upper Bound	200

Constraints

If your limitation involves Controls “interacting” with each other – such as the sum of the lengths of a set of conveyors has to be less than a certain value – then you can represent this using a Constraint column. Constraints are added by clicking the *Add Constraint* button in the Experiment section of the Design Ribbon. Constraints are designed to identify any out of limit controls that have been manually defined as well as to limit automatic creation (by OptQuest) of scenarios with controls that would be out of limits. The values displayed in a constraint column will be the value of the expression defined in the *Expression* property. When using OptQuest, this column will only show values that are within the constraint bounds – this only works for linear constraints (no multiplication or division). All scenarios that violate input constraint bounds are discarded before running in Simio thus will never show up on the screen. List and Enum type controls cannot be part of a Constraint expression, as they are not linear. Boolean controls evaluate to '1' (True) or '0' (False), so may be included within a Constraint expression.

Responses Vs Constraints

If all members of your system limitation expression can be represented by a Control – use a **Constraint**.

If any member of your limiting expression needs to run a Scenario in order to be evaluated – use a **Response**.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

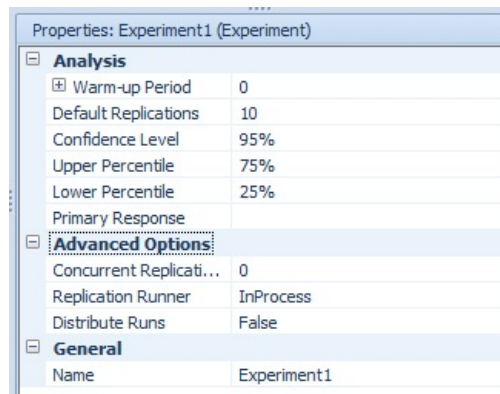
Send comments on this topic to [Support](#)

Analysis - ExperimentProperties

Experiment Properties Window

The Experiment Properties window is where a user can specify a **Warm-up Period** for the run, as well as the **Confidence Level** for the experiment. This is done in the Properties window of the [Experiment](#). To access the properties window, right click on the Experiment within the [Navigation window](#) and select Experiment Properties.

Specifying the Warm-up Period and Confidence Level



Listed below are the properties of the **Experiment**:

Property	Valid Entry	Description
Warm-up Period	Real	Time period after the beginning of a run at which statistics are to be cleared. Warm-up periods are useful for removing the effects of atypical system conditions from the statistics collection. State variables also include an <i>Auto Reset When Statistics Cleared</i> property that indicates whether to reset the variable to its initial value after this specified warm-up period.
Default Replications	Integer	Specifies the default number of replications for newly created scenarios. When creating new scenarios, this value will be used to initialize the Replications Required property.
Confidence Level	90%, 95%, 98%, 99%	The confidence level used to calculate confidence interval half-width statistics for result averages across replications.
Upper Percentile	75%, 80%, 90%, 95%, 99%	The percentile of data which is the upper bound of the box in the Experiment Response Chart (SMORE plot).
Lower Percentile	1%, 5%, 10%, 20%, 25%	The percentile of data which is the lower bound of the box in the Experiment Response Chart (SMORE plot).
Primary Response	Valid Response	Specifies which response is considered the most important. This is available for use by add-ins, such as the OptQuest Add-In .
Concurrent Replication Limit	Integer	Specifies the maximum number of replications that will be run simultaneously. The default value of 0 allows Simio to determine this value. Note: Simio RPS and Simio Professional allow Replication Runner and include up to 16 concurrent replications simultaneously. Additional replication runner licenses can be

purchased.

Replication Runner	InProcess, External, None	Specifies if replications should be run in the Simio process, or in a separate process. Choose InProcess for smaller models, or External for the largest models. Choose None to only use the Distribute Runs option.
---------------------------	---------------------------	--

Distribute Runs	True, False	Specifies if the replications should be distributed to other computers on your local network. This feature is available in Professional and RPS Editions of Simio.
-----------------	-------------	--

Note: Considering the waiting time of entities during the warm-up period could bias the results of the run and Simio takes the position of excluding that bias. Therefore, the average waiting time statistic within a replication is not dependent on what happened during the warm-up period.

Note: An experiment's Confidence Level setting may also be used to calculate runtime confidence interval half-widths on TallyStatistic or StateStatistic element averages in the model. See the ConfidenceLevel property of the [TallyStatistic](#) and [StateStatistic](#) elements for more information.

OptQuest for Simio Parameters

This section within the experiment properties will appear when the [OptQuest for Simio Add-In](#) is selected. This is where you define properties for that Add-In, which will aid in finding an optimal solution for your model.

Select Best Scenario using KN Parameters

This section within the experiment properties will appear when the [Select Best Scenario using KN Add-In](#) is selected. This is where you define properties for that Add-In, which will aid in selecting the best scenario from a list of candidate scenarios.

Select Best Scenario using GSP

This section within the experiment properties will appear when the [Select Best Scenario using GSP Add-In](#) is selected. This is where you define properties for that Add-In, which is a scenario selection algorithm for large-scale problems. This add-in is similar to the add-in described in [Select Best Scenario using KN Add-In](#), but may provide improved performance when doing larger scale models that require evaluation of many scenarios in a parallel processing environment.

This add-in was created by Sijia Ma as part of his PhD work at Cornell University. You can read his thesis here: <https://ecommons.cornell.edu/handle/1813/64955>

bi-PASS Scenario Elimination

This Experiment add-in iteratively runs replications of scenarios and attempts to eliminate non-competitive Scenarios using the sample mean and sample variances of the Responses across all Scenarios. You can read more about it here: <https://pubsonline.informs.org/doi/10.1287/opre.2022.2343>

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

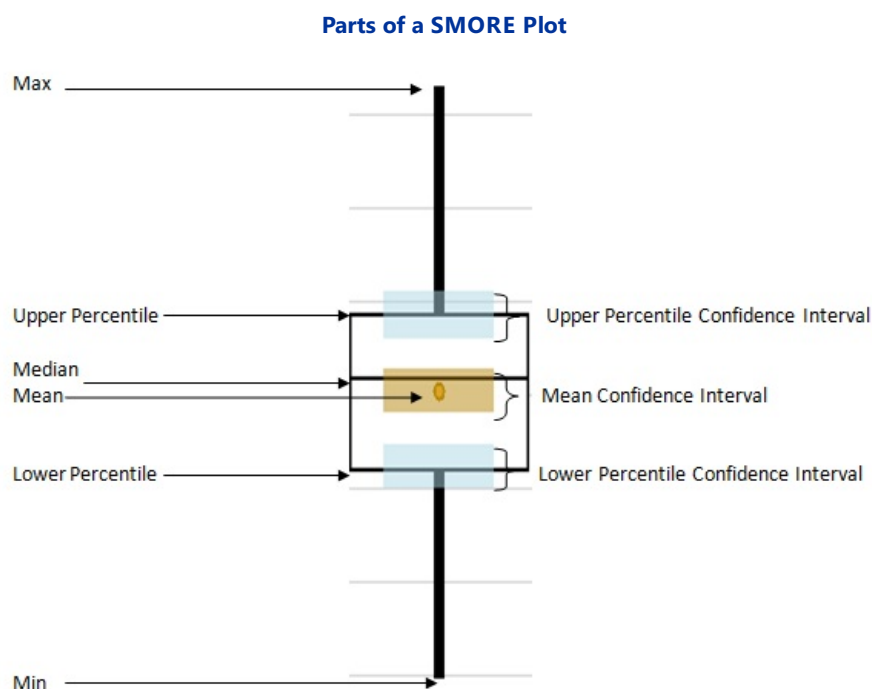
Experiment Response Chart

Graphing an Experiment

The Response Results window of an Experiment creates a Simio Measure of Risk & Error (SMORE) plot using the Response value that is selected in the Response pull down menu. The plot displays results across replications, for each scenario. The Response Results tab can be moved so that the graph can be seen plotting as the Design Window steps through replications and scenarios, in real time.

Simio Pivot Tables and Reports provide an estimate of the population mean and confidence interval based on multiple replications. While this is exactly what is needed in some situations, in others it provides an inadequate amount of information required to make a decision taking into account risk. As Dr. Nelson puts it in his paper discussing the concepts of Measure of Risk & Error (MORE) plots (Nelson 2008), "A baseball player's batting average is a meaningful historical statistic. However, a simulation is not trying to create history; instead it is trying to say something about what will happen in the future and whether we can live with it."

A Simio Measure of Risk & Error (SMORE) plot displays both the estimated expected value of a scenario and multiple levels of variability behind the expected value. A SMORE plot consists of a Mean, Confidence Interval for the Mean, Upper Percentile Value, Confidence Interval for the Upper Percentile Value, Lower Percentile Value, Confidence Interval for the Lower Percentile Value, Median, Maximum Value, and Minimum Value (sample illustrated below):



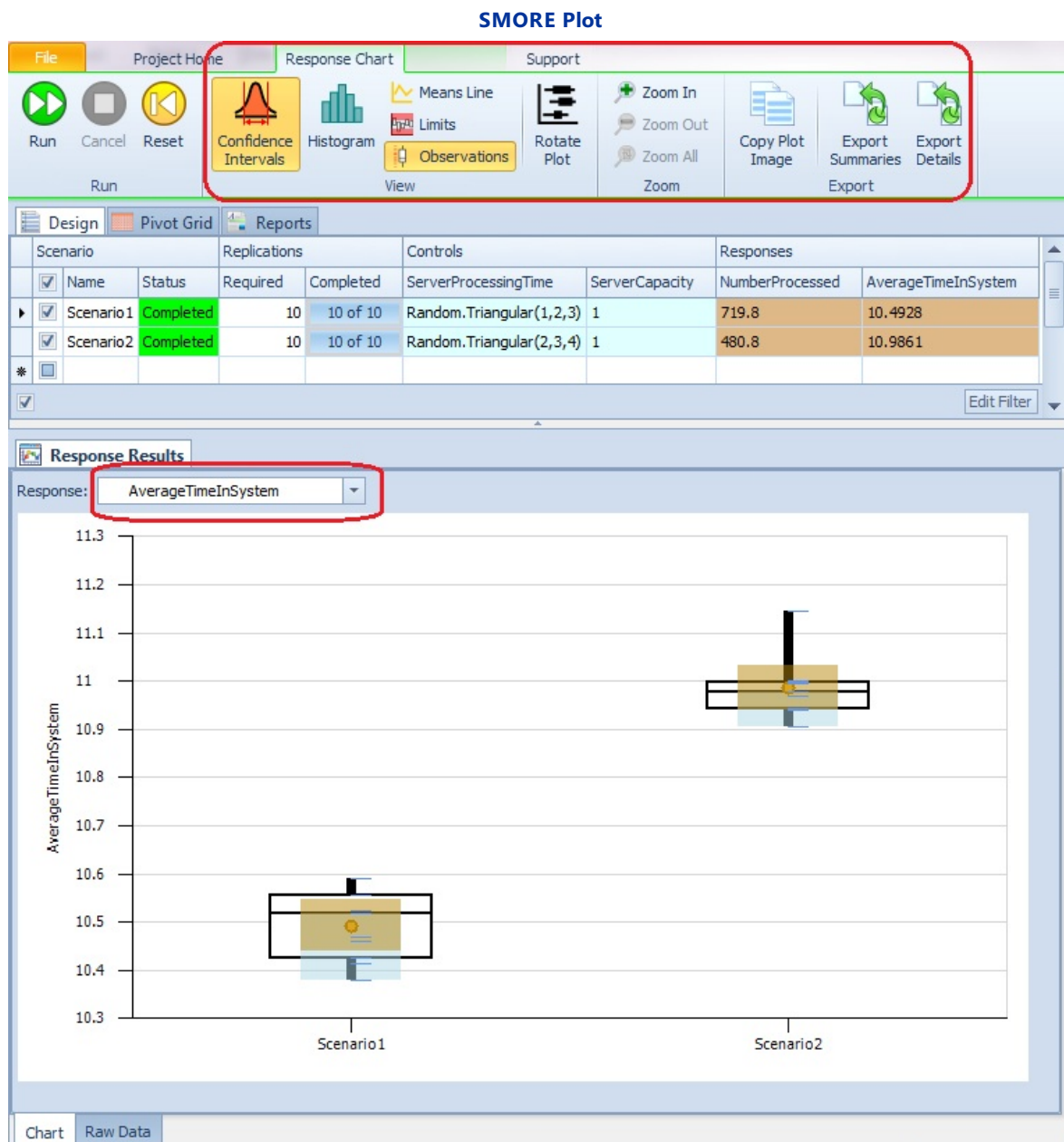
In the Properties Window of an Experiment, the user is able to set the Upper and Lower percentiles (Upper Percentile will be set to 75 and Lower to 25 by default), Confidence level used to calculate Confidence Intervals (95% by default).

Experiment Properties

Properties: Experiment1 (Experiment)

Analysis	
Warm-up Period	0
Default Replications	10
Confidence Level	95%
Upper Percentile	75%
Lower Percentile	25%
Primary Response	
Advanced Options	
Concurrent Replicati...	0
Replication Runner	InProcess
Distribute Runs	False
General	
Name	Experiment1

The accuracy and usability of this plot depends on having a large amount of data, which means a large number of repetitions. All aspects of the plot will not be displayed fully unless we have a sufficient amount of data. Simio always displays the Mean, Max, and Min. Once the scenario has at least six replications, the Median, Upper and Lower Percentiles, and all Confidence Intervals will be displayed. The Ribbon now has new buttons allowing the user to be able to turn on/off Confidence Intervals, Observations, Upper/Lower Bounds, and Lines Between Means.



On the bottom left-hand corner of the Response Chart Window is a Raw Data tab. This tab allows the user to view the

values that are used to construct the SMORE plots for each Response including the Mean, Minimum Value, Maximum Value, Mean Confidence Interval Half Width, Upper and Lower Percentile values, the bounds for Percentile Confidence Intervals. *Copy Plot to Clipboard*, *Export Summaries*, and *Export Details* buttons have been added to the Ribbon. Exporting Summaries allows the user to export what they see in the Raw Data tab, while Export Details provides users with the exact Response Value for each unique Scenario-Response-Repetition combination. Users can use this data to create their own plots if they wish. Note that this is different than exporting the detailed results from the Experiment Design window.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Response Sensitivity and Sample Size Error Analysis

Simio incorporates two types of input data analysis that relate the **Input Parameters** of a model to the **Responses** defined in the experiment. Both of these are found in the Experiment window under the Input Analysis tab. **Response Sensitivity** analysis is used to determine the sensitivity of the Responses to changes in the Input Parameters. Response Sensitivity analysis is very useful in situations where the input parameters have been specified in the absence of real world data and shows which Input Parameters have the biggest influence on the Responses. Response Sensitivity analysis can be used early in a modeling project to help guide decisions on devoting time/money to initially collect real-world observations for input parameters. In contrast **Sample Size Error** analysis is performed once some real-world data has been collected and is used to determine the overall impact of estimating the Input Parameters (based on real-world data) on the uncertainty in the Responses, to determine which estimated Input Parameters make the largest contribution to this uncertainty, and to identify the Input Parameters for which additional real-world observations would have the greatest benefit in reducing the uncertainty in the Responses.

Response Sensitivity analysis is automatically performed by Simio whenever the number of replications for a scenario is greater than the number of Input Parameters. Note that no additional simulation experiments are required to perform this analysis; it is performed by Simio whenever the minimum number of replications have been made in estimating the Responses. Note that uncertainty in the Responses is created whenever we estimate our Input Parameters based on real-world data, and Sample Size Error analysis is used to answer the basic question “do we have a large enough sample size for our real-world data to have confidence in our results, and if not which Input Parameters should we focus on for collecting additional real-world data?” Sample Size Error analysis is a more complex undertaking than Response Sensitivity analysis and requires an additional set of special simulation replications. Hence while Response Sensitivity analysis is automatically run by Simio, Sample Size Error analysis is an optional extra analysis that can be initiated by the user. Once initiated this complex analysis is fully automated within Simio.

Input Parameters and Experiment Responses

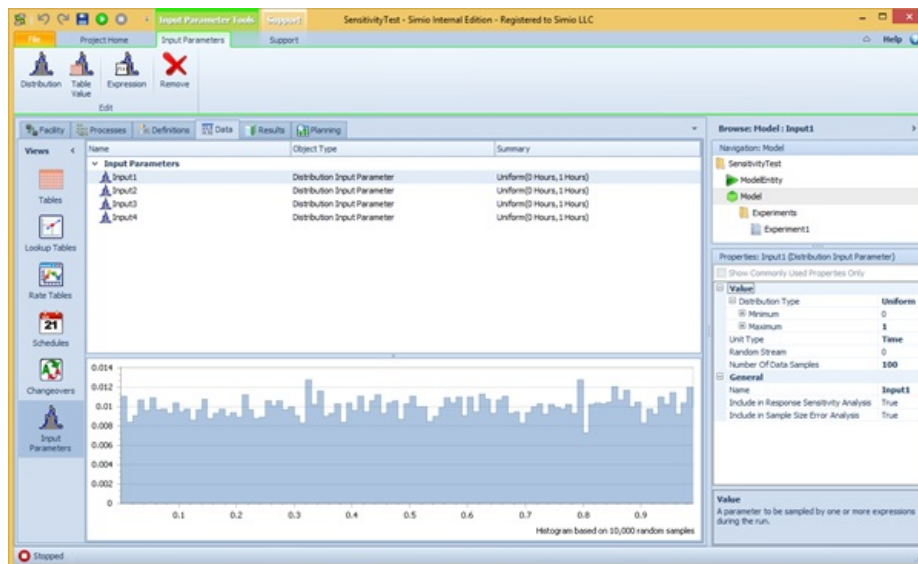
Both Response Sensitivity analysis and Sample Size Error analysis compute relationships between the Input Parameters in the model and the Responses that are defined for an Experiment. Hence the analysis can only be performed if at least one Input Parameter and one Experiment Response has been defined.

Input Parameters are generally used to define named parameters which can then be referenced anywhere within your model. For example you might define Input Parameters named *InterarrivalTime* and *ProcessingTime*, and then reference these names on the Source and Server objects in your model. There are several advantages in using Input Parameters in place of directly entering these expressions in the model.

1. Input Parameters can be shared across multiple objects; e.g. if five servers all share the same Process Time, they can all reference the same Input Parameter.
2. It is more convenient to enter distributions using Input Parameters since the distribution parameters are individually defined as properties of the Input Parameter and a histogram based on 10,000 samples is provided to show the shape of the distribution for the parameters.
3. Input Parameters enable both Response Sensitivity and Sample Size Error analysis.

Input Parameters are defined by selecting the Data window tab, and then selecting Input Parameters on the left panel. There are three different types of Input Parameters – Distributions, Table Values, and Expressions. A Distribution is used to specify a random distribution. The parameters for the distribution are specified in the property grid, and the histogram based on 10,000 samples of that distribution is displayed in the bottom panel of the window. In addition to the distribution parameters you also define a Boolean flag that specifies if this Input Parameter is to be active in the Response Sensitivity analysis, and a second Boolean flag that specifies if the Input Parameter is to be included in the Sample Size Error analysis. If it is included in the Sample Size Error analysis you also specify the number of real-world observations that were used in fitting this distribution. The second type of Input Parameter is a Table Value and is used whenever the actual observational data is randomly sampled in place of fitting the data to a distribution and then sampling from that distribution. This is typically the preferred approach when actual data is available. In this case you specify the table column name that holds the real-world observations. The third type of Input Parameter is an Expression. In this case you can specify any mathematical expression, including random distributions and model properties. Although this is the most general form of the Input Parameter one limitation is that it cannot be used in the Sample Size Error analysis.

Specifying an Input Parameter of Type Distribution



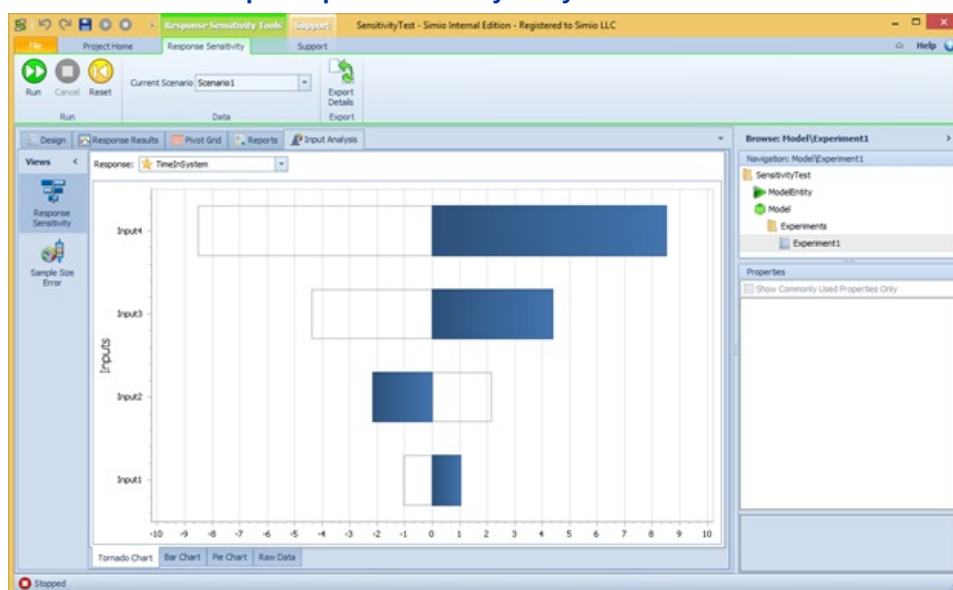
Response Sensitivity Analysis

Response Sensitivity analysis calculates the sensitivity of each experiment *Response* to the model's Input Parameters. It is often used in the absence of real-world input data. The calculated sensitivities are a measure of how each Response in the model is changed by changes in the Input Parameters. This is an indication of which Input Parameters have the most impact on the Response. The analysis is performed independently for each active scenario in the experiment.

The basic calculation that is performed is a linear regression that relates each Response variable to the set of Input Parameters. To perform this calculation the number of replications for the scenario has to be greater than the number of Input Parameters included in the analysis. Hence if 10 replications are made, 9 or fewer Input Parameters can be considered in the analysis.

The sensitivities are shown in the *Response Sensitivity* panel view for the Input Analysis tab in the Experiment window. The user can view the sensitivity coefficients of each Input Parameter for each Response as either a tornado chart, stacked bar chart, or a series of pie charts. These alternate views can be selected using the lower tabs in the Response Sensitivity view. The user can also view the raw data in tabular form. Note that the Tornado chart shows a single selected Response at a time, whereas the bar and pie charts show all responses simultaneously. The tornado chart displays the coefficients from smallest to largest with a solid bar for the coefficient, and an outline for the negative of the coefficient. The outline is useful for comparing the magnitude of the sensitivities. The following shows an example of a tornado chart with four input parameters. Note that Input2 has a negative coefficient which implies that increases in the input parameter will decrease the expected response.

Sample Response Sensitivity Analysis (Tornado Chart)



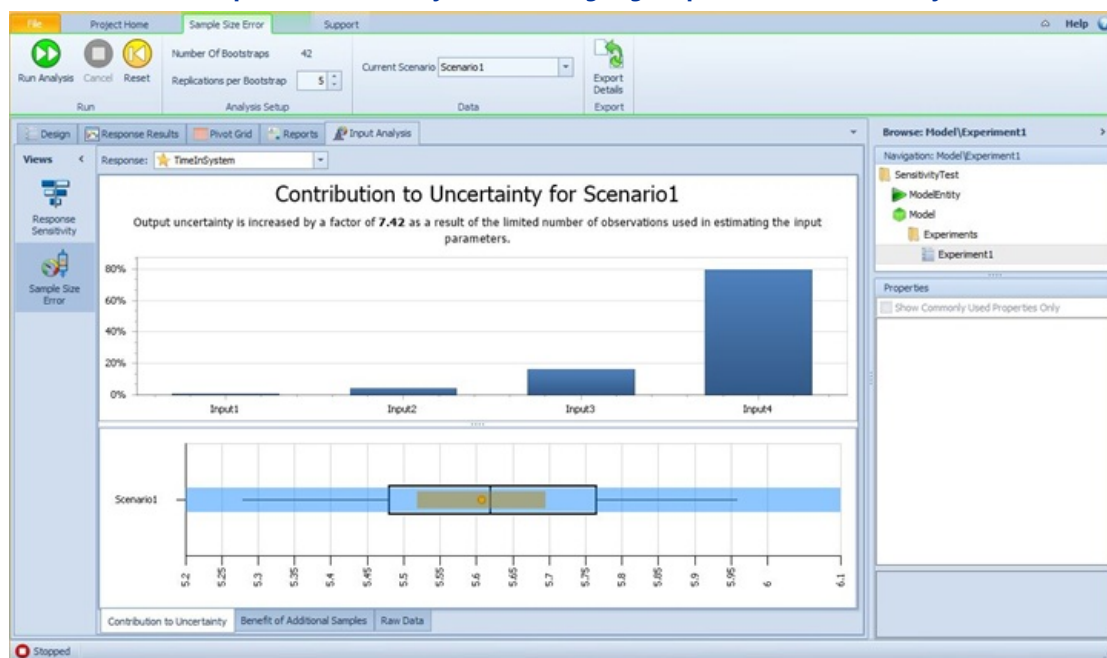
Sample Size Error Analysis

Sample Size Error analysis is used once real world data has been collected to determine the overall impact of estimated Input Parameters (based on real world samples) on the uncertainty in the Responses. This analysis is performed in the Sample Size Error panel of the Input Analysis tab contained within the Experiment Window. Note that for an Input Parameter to be included in the analysis you must specify the number of data samples used to estimate the Input Parameter. The uncertainty resulting from input parameter estimation is expressed in terms of a *Half Width Expansion Factor*, which expands the standard confidence interval on a response to account for uncertainty resulting from estimation of the input parameters.

The method for computing the Half-Width Expansion Factor is based on a procedure developed by Song and Nelson at Northwestern [Quickly Assessing Contributions to Input Uncertainty, in review] and requires a special set of “bootstrap” experiments. The number of bootstrap experiments is automatically determined by the procedure, but you can specify the number of replications per bootstrap (≥ 1) on the Sample Size Error ribbon under *Analysis Setup*. You can initiate the analysis by clicking the Run Analysis button on this ribbon.

The following shows a sample output from the Sample Size Error analysis. The results displays the Half Width Expansion Factor along with a chart showing the percentage of contribution from each input parameter. Below this chart is an abbreviated SMORE plot that shows the standard confidence interval for the mean in tan and the expanded half width due to uncertainty caused by estimation of the input parameters in blue. As we can see from this graph in this example we have significantly more uncertainty attributable to estimation of the input parameters (blue) than uncertainty attributed to the limited number or experiment replications (tan). In fact, the Half Width Expansion Factor tells us that the input related uncertainty is over 7 times the experimentation related uncertainty.

Sample Size Error Analysis Illustrating High Input Related Uncertainty



The Sample Size Error analysis also provides a second bar chart (selected using the lower tabs) that shows the relative benefit of collecting additional data for each input parameter. In many cases the bar chart will appear similar to the relative contribution bar chart. However this is not always the case since it is possible the relative value of additional samples will differ from the relative contributions. This is the case when the number of real world samples differs for each input parameter.

The SimBit project named [InputAnalysis](#) includes two models, [ResponseSensitivity](#) and [SampleSizeError](#), that show an example of using input parameters with the analysis techniques described above.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

OptQuest Add-In

What is 'OptQuest for Simio'?

OptQuest is offered for Simio as an add-on optimizer to the standard Simio products. OptQuest enhances the optimization capabilities of Simio by searching for optimal solutions to your models.

Simulation models are usually used within the context of a decision making process, evaluating different possibilities for the models controls and how these possibilities affect the system. Simio has the ability to create experiments allowing users to enter input values and run multiple replications to return estimated value of the system performance. This can be done by varying several different input value combinations eventually leading to an optimal solution. This method works well for simple models, but it is easy to see that it could get quite tedious as the complexity expands.

OptQuest helps remove some of the complexity by automatically searching for the optimal solution for you. You describe your simulation problem to OptQuest and it searches for input controls to maximize or minimize your objective (i.e. maximizing profit or minimizing cost).

For additional information on the OptQuest engine, see the [The OptTek Documentation on Optimization](#).

What does OptQuest do to my Simio Model?

OptQuest takes on the duties of setting property values, starting replications, and retrieving results. When the Add-In is launched, OptQuest adds its own parameters in addition to the ones provided by the standard Experiment.

When the optimization runs, OptQuest starts the simulation by first resetting the run and changing the values of the input properties to those identified by the Add-In. Then, OptQuest starts to run replications of the first scenario. After the scenario runs, OptQuest retrieves the response values needed for the objective function and uses its set of algorithms to determine the value of the next set of inputs to create. It then uses these inputs to run the next scenario and repeats this process until it reaches a specified stopping condition or you stop the session.

An OptQuest Example

The AirportTerminal model that is part of the Simio example models available with the Simio installation, contains an experiment that is set up to use the OptQuest Add-in. For detailed information on this example, see [Experiment Example](#). The Emergency Department example model also utilizes the OptQuest Add-In. This experiment uses the Multi-Weighted Objective, where one response is being maximized and another is being minimized, as part of the objective function.

How OptQuest Works

OptQuest makes use of intelligent search methods and incorporates its special optimization algorithms alongside Simio's modeling power. Instead of using the algorithms to optimize a set of mathematical equations, it uses them to optimize a set of stochastic process interactions.

Once the problem is described by defining the controls, objective, and constraints, OptQuest has enough information to begin running scenarios. After the first scenario, OptQuest evaluates the responses of the scenario, analyzes, and then returns values to be considered in the next scenario. After the second scenario runs, it once again analyzes the response given to it by Simio, compares it to the previous response from the prior scenario, and once again returns new values to be evaluated by Simio. This process of obtaining results and comparing to previous objective values repeats over and over again until OptQuest meets one of its terminating criteria – either after reaching the maximum number of iterations or if OptQuest has determined the objective value has stopped improving.

OptQuest Licensing

OptQuest is an optional add-on to Simio. It may be purchased when you purchase Simio or it may be added on later. OptQuest is included with our non-commercial academic products but in such case is subject to the same limitations of non-commercial use only. For commercial users, OptQuest requires activation to exploits its full capabilities.

Commercial customers who have not purchased OptQuest will find that it works in evaluation mode. Evaluation mode allows a maximum of two controls and two constraints, and will produce a maximum of 20 scenarios.

What does an unlicensed version of OptQuest allow me to do?

The unlicensed version allows a maximum of 2 controls, 2 constraints, and will produce a maximum of 20 scenarios.

The Experiment Design Window

Because OptQuest is provided by Simio as an Add-In, OptQuest operates and appears very similar to a Standard Simio Experiment. When a standard Simio experiment first opens there are four default columns:

- Scenario Name – Defaults to a standard scenario number (i.e. 001, 002, etc.) but you can rename them any name that you want.
- Scenario Status – Reflects the state of the scenario. Status has five possible values:
 - Idle – Experiment has yet to run
 - Pending – Scenario is scheduled to run
 - Running – Scenario has started but has not completed required number of replications
 - Canceled – Scenario has been canceled during a run.
 - Completed – All replications complete
- Replications Required - Reflects the number of Replications that Simio is scheduled to run. This defaults to five but you can manually enter the number of replications required before running the experiment.
- Replications Completed – The number of replications that are complete at the current point in time.

Experiment Properties

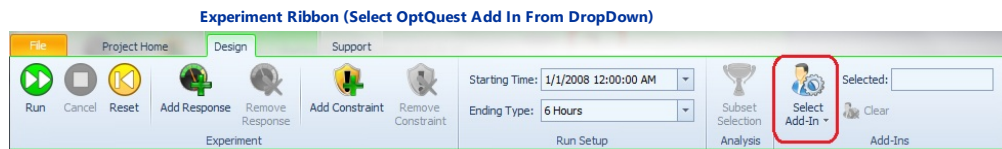
Also, Standard Simio Experiment properties/parameters can be found in the Properties window of the Experiment (select Experiment in the Navigation window and right click to select Experiment properties):

- Warm-up Period – The amount of time the user specifies to get the model to steady-state or to the level of interest
- Default Replications – The number of replications to be run by each scenario.
- Confidence Level – The level of accuracy you want Simio to use when calculating confidence intervals for statistics
- Upper Percentile - The percentile of data which is the upper bound of the box in the SMORE plot. For example, if it is set to 75%, that means that 75% of the data is lower than the value created by the plot.
- Lower Percentile - The percentile of data which is the lower bound of the box in the SMORE plot.
- Primary Response – The Objective Function (discussed further in Objective Function Section)

Simio will automatically import all of the model's user defined properties and referenced properties (initially populated with their initial values) as [Controls](#) for the experiment as columns. These are all grouped under "Controls" and the columns are colored blue.

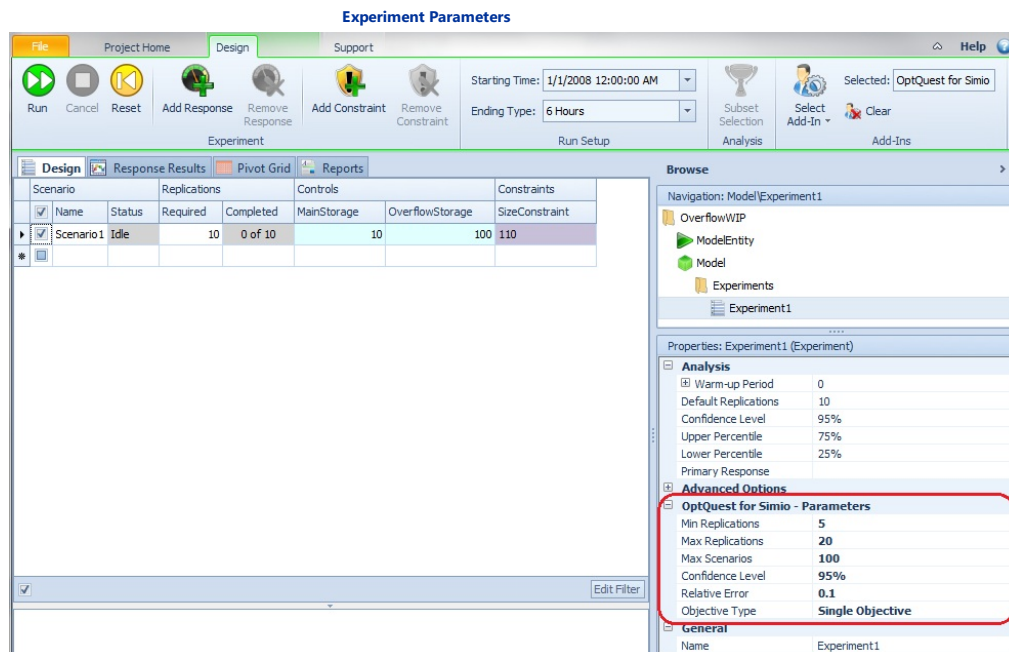
Using the OptQuest Add-In

Go to the Add-Ins Section of the Design Ribbon and select "OptQuest for Simio" from the Select Add-In drop down list.



Right-click on the experiment name and select Experiment Properties. The OptQuest Parameters will now be displayed in the Properties window. These new OptQuest parameters are:

- **Min Replications** – The minimum number of replications you would like Simio to run.
- **Max Replications** – The maximum number of replications you would like Simio to run.
- **Max Scenarios** – Determines the maximum number of scenarios you want OptQuest to create. It is possible that OptQuest will come to an optimum solution before it creates this amount of scenarios.
- **Confidence Level** – The level of accuracy you want OptQuest to use when statistically comparing one response value to another.
- **Relative Error** – Percentage of mean in which the half-width of the confidence interval is determined. For example, if the mean is 50 and you set the Error Percent to 0.1 (or 10%), the confidence interval will be 50 ± 5 . After running the default number of Replications, OptQuest will determine if this condition has been satisfied. If not, Simio will run more replications to reduce the variability.
- **Objective Type** - Single Objective, Multi-Objective Weighted or Pattern Frontier. In Single Objective, OptQuest determines the set of control values that optimizes the experiment's Primary Response. In Multi-Objective Weighted, OptQuest optimizes across all responses with Objective set to Minimize or Maximize, taking into account each response's Weight value, to determine a single "optimal" solution. In Pattern Frontier, OptQuest optimizes across all responses with Objective set to Minimize or Maximize, and finds the set of scenarios that are optimal, rather than a single "optimal" solution based on weights.



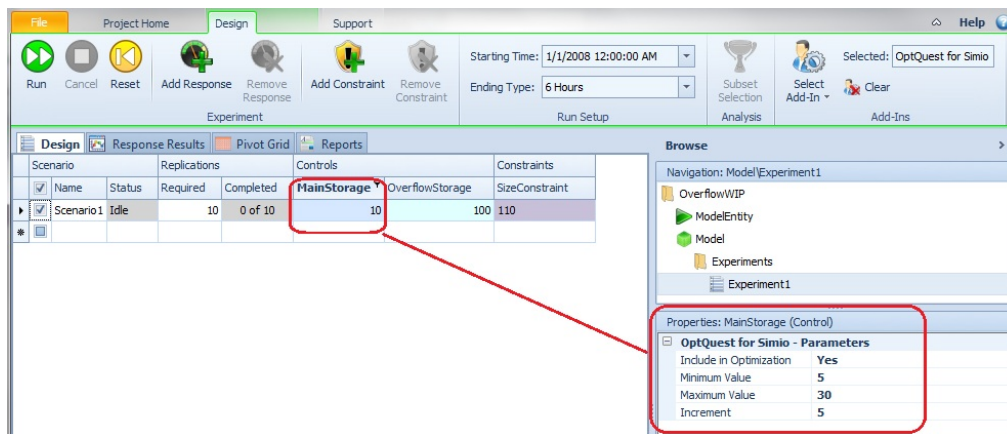
Controls

The OptQuest Add-In adds additional properties to the input Controls. Without OptQuest, the user has to manually enter the combinations of Control values that are to be tested. With OptQuest, each Input Control has a minimum value, maximum value, and an increment size that dictates the step size for which OptQuest will move between the minimum and maximum values to create scenario combinations.

Any Enum properties, List or Boolean properties that are defined as controls may also be used within the OptQuest optimization. For example, a 'ListProperty1' is a string list, consisting of East, West, North and South and an 'EnumerationProperty1' is a selection type enum. OptQuest will then simply choose different combinations of these values as part of the optimization. By default, these are not automatically included in the optimization (see next paragraph for details).

Sometimes not all Controls are used in an experiment. Each Control has a property called *Include in Optimization* with values of 'Yes' or 'No'. By default this is set to 'No' for Enum and List controls and 'Yes' for all other controls. To exclude the control from the experiment, simply set this parameter to 'No' and every scenario will use the default value for that property.

Control Properties



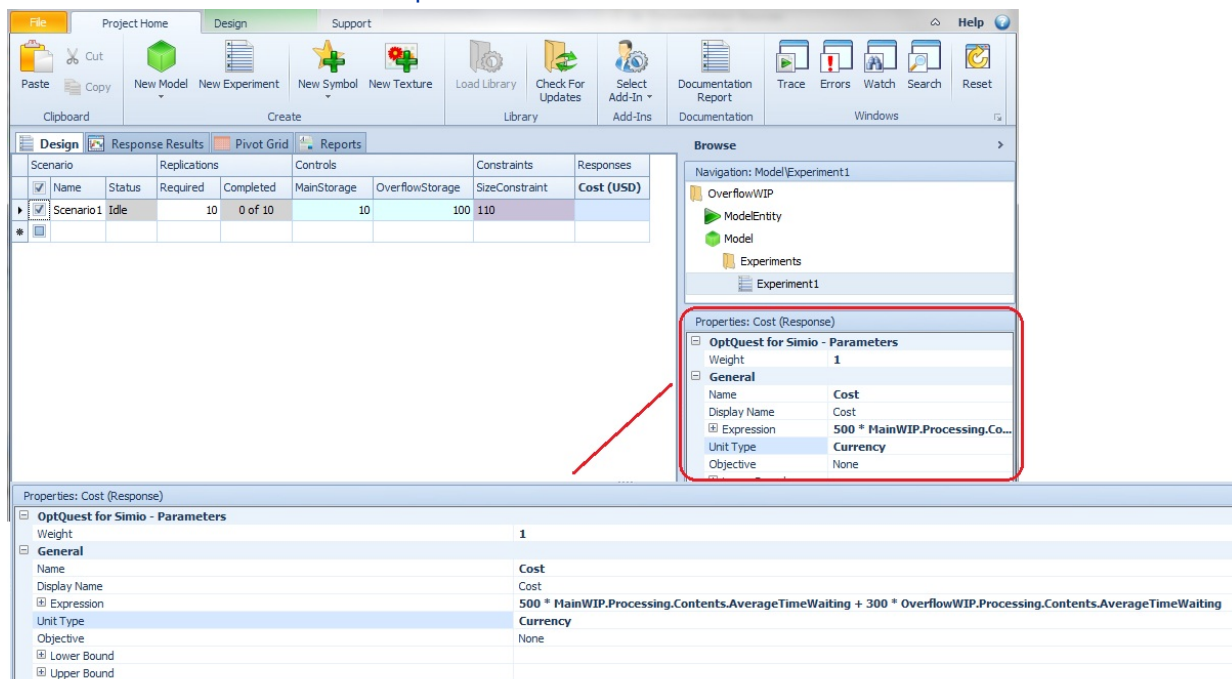
Responses

To add a **Response**, simply click the **Add Response** button in the Experiment section of the Design Ribbon. The resulting Response column can either act as a purely informative display of a certain statistic of interest or an objective function that you are trying to optimize. Responses are designed to display the output from a model. In addition, you can add limits to responses to assist in identifying any scenarios that violate the limits. If a Response is found to be infeasible for a particular Scenario, the value will still be displayed, but will have its cell highlighted in red and will not be considered in optimization. If the value is greater than the upper bound the cell will be shaded with a gradient that has more red on the right side of the cell. Conversely, if the value of the Scenario is less than the lower bound, the cell will be shaded towards the left side of the cell.

You can customize a Response column type with the information entered in the following properties:

- **Name** – The name of this Response, which is reflected in the column header.
- **Expression** – Where you enter the left-hand side of your expression or objective function.
- **Objective** – Determines how OptQuest is to use the response information when determining the optimal solutions. Objective can either be:
 - **None** – Just an informative response and not used to make decisions
 - **Maximize Objective** – OptQuest drives the Experiment to achieve the greatest values for the objective function
 - **Minimize Objective** – OptQuest drives the Experiment to achieve the lowest value for objective function
- **Upper Bound** – Any values for this response above this value are considered infeasible.
- **Lower Bound** – Any values for this response that are below this value are considered infeasible.

Response



Constraints

If you want to put a limitation on your experiment that involves input Controls “interacting” with each other – such as the sum of the lengths of a set of conveyors has to be less than a certain value – then you can represent this using a Constraint column. Constraints are added by clicking the **Add Constraint** button in the Experiment section of the Design Ribbon. Constraints are designed to identify any out of limit controls that have been manually defined as well as to limit automatic creation (by OptQuest) of scenarios with controls that would be out of limits. The values displayed in a constraint column will be the value of the expression defined in the **Expression** property. When using OptQuest, this column will only show values that are within the constraint bounds – this only works for linear constraints (no multiplication or division). All scenarios that violate input constraint bounds are discarded before running in Simio thus will never show up on the screen. If the constraint is used to validate a set of manually defined Controls, a violated Constraint will be shaded red in the same manner as a violated Response. List and Enum type controls cannot be part of a Constraint expression, as they are not linear. Boolean controls evaluate to ‘1’ (True) or ‘0’ (False), so may be included within a Constraint expression. For additional information on using Responses and Constraints in an Experiment, see the [Responses and Constraints](#) page.

In Summary: If all members of your system limitation expression can be represented by an input Control – use a **Constraint**.

If any member of your limiting expression needs to run a Scenario in order to be evaluated – use a **Response**.

Constraint

File Project Home Design Support Help

Run Cancel Reset Add Response Remove Response Add Constraint Remove Constraint

Starting Time: 1/1/2008 12:00:00 AM Ending Type: 6 Hours

Selected: OptQuest for Simio

Your maintenance will expire on 12/31/2012 (36 days remain).

Design Response Results Pivot Grid Reports

Scenario	Name	Status	Replications Required	Replications Completed	Controls MainStorage	Controls OverflowStorage	Constraints SizeConstraint
Scenario1	Idle	10	0 of 10	10	100	110	

Browse

Navigation: Model\Experiment1

- OverflowWIP
- ModelEntity
- Model
- Experiments
- Experiment1

Properties: SizeConstraint (Constraint)

General

Name	SizeConstraint
Expression	MainStorage + OverflowStorage
Lower Bound	
Upper Bound	110

Objective Function Options

An Objective Function is a Response that defines the key area of interest, or goal of the Experiment. Because this Response is used to summarize system performance, you must identify what constitutes optimal system performance and then let OptQuest know how to interpret the values for this Response. OptQuest will use this information to steer the experiment towards optimal value for this specific Response obeying the constraints given.

You may create a Response, which consists of an expression that is the Objective Function to be optimized. To optimize a **single Response**, follow these 3 steps:

1. Create a Response expression that represents the performance of your system - This expression can consist of one or multiple components (Controls and/or other Expressions - to use a Response in the component, you must use the Response expression value not the Response name). Usually, the expression is a sum of components multiplied by Weights, with the Weight representing the amount of benefit or penalty of that Control or Expression.
2. Set the Objective Goal to Maximize objective or Minimize objective - A general rule of thumb would be that you want to maximize your benefits and minimize your costs. If an expression has more benefits than costs, maximize the entire expression. If there are more costs, minimize.
3. Tell OptQuest to use this Response to optimize via the *Primary Response* property - OptQuest can only optimize based on one objective function at a time, although it is possible to have multiple Responses with Maximize Objective or Minimize Objective selected. To set the Primary Response, select the name of the response column containing the objective function to be used in the optimization from the drop down list located in the Experiment Properties window.

In order for OptQuest to be able to run you must designate a Primary Response complete with an Objective Goal and a valid Expression.

As Scenarios pass Lower or Upper Bounds on Responses, OptQuest will uncheck these Scenarios. An unchecked Scenario indicates it is an infeasible solution.

The Emergency Department example model contains an experiment that uses 'Single Objective' *Objective Type*. It seeks to minimize the Average Length of Stay response with upper bounds on both responses.

Using a Primary Response

File Project Home Design Support Help

Run Cancel Reset Add Response Remove Response Add Constraint Remove Constraint

Starting Time: 1/1/2008 12:00:00 AM Ending Type: 6 Hours

Selected: OptQuest for Simio

Experiment Run Setup Analysis Add-Ins

Design Response Results Pivot Grid Reports

Scenario	Replications	Controls	Constraints	Responses			
Name	Status	Required	Completed	MainStorage	OverflowStorage	SizeConstraint	Cost
Scenario1	Idle	10	0 of 10	10	100	110	

Navigation: Model\Experiment1

- OverflowWIP
 - ModelEntity
 - Model
 - Experiments
 - Experiment1

Properties: Experiment1 (Experiment)

- Analysis
 - Warm-up Period: 0
 - Default Replications: 10
 - Confidence Level: 95%
 - Upper Percentile: 75%
 - Lower Percentile: 25%
 - Primary Response: **Cost**
- Advanced Options
- OptQuest for Simio - Parameters

Properties: Cost (Response)

- OptQuest for Simio - Parameters
 - Weight: 1
 - General
 - Name: Cost
 - Display Name: Cost
 - Expression: $500 * \text{MainWIP.Processing.Contents.AverageTimeWaiting} + 300 * \text{OverflowWIP.Processing.Contents.AverageTimeWaiting}$
 - Unit Type: Currency
 - Objective: None
 - Lower Bound:
 - Upper Bound:

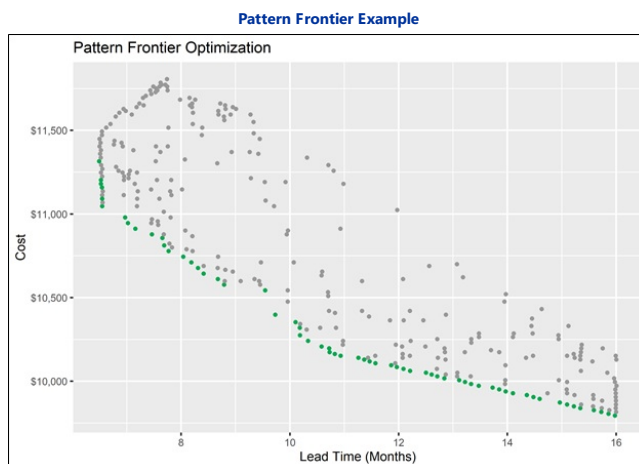
You may want to find an optimal solution based on more than one Response by using the **Multi-Objective Weighted** optimization. Follow these steps:

1. Decide which Responses should be part of the objective function.
2. Set the Objective Goal to Maximize Objective or Minimize Objective for each Responses that will be included in the optimization. Set the Weight property for each Response that will be used. (OptQuest multiplies each response value by the weight, negates it if it is set to Minimize, and then adds them together to get the value of the overall objective.)
3. Tell OptQuest to consider more than one Response via the *Objective Type* property. Set this to Multi-Objective Weighted.

The Multi-Objective Weighted type will only leave one Scenario checked at the end of the run. Even if Response results are identical for more than one Scenario, only one will be considered as best. When comparing Scenarios with identical Response values, one is not better than the other, so the Scenario that finished first remains checked and the second Scenario is unchecked.

Instead of just one optimal solution, to find an optimal set of solutions based on more than one Response, set the *Object Type* property to **Pattern Frontier**. In this case, the Weight property on each Response is not used, but each Response to be included in the optimization must still have its *Objective* property set to either 'Minimize' or 'Maximize'. Because the Weight property is not used in this optimization, more than one scenario may be checked at the end of the run, indicating its position at the "frontier" of the optimization space and therefore a potential optimal solution.

Below is a plot of all scenarios from an OptQuest Experiment with the *Objective Goal* set to 'Pattern Frontier'. The two responses shown are 'Cost' and 'Lead Time', each of which had their *Objective* property set to 'Minimize'. Scenarios left checked at the end of an Experiment run are shown in green. These scenarios lie along the optimal frontier of the solution space, and any of the green scenarios chosen represent an optimal scenario striking a different balance between opposing objectives.



Results and the Optimal Solution

The Objective Function will be graphed using **Simio Measure of Risk & Error (SMORE) Plots**. Although the Mean Value is the primary area of concern, the SMORE plot allows the user to see the variability behind the values and to make decisions accordingly. If two values have similar Means, the variability of the system can act as another level of information as to which Scenario would be the best choice.

To easily spot the Optimal Solution, the Objective function can be sorted before running the Experiment so that as OptQuest creates Scenarios it will plot Response values, in real time, ranked either ascending or descending. The Optimal Solution would simply be the Scenario that is listed at the top of the Primary Response Column in the Design Tab or the

furthest to the left in the Response Chart.

Note: Any Scenarios that result in NaN will be unchecked.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Scenario Subset Selection

Subset Selection

In general, the purpose of creating a simulation model and, subsequently, an experiment for that model is to create multiple scenarios for that system and pick the best one to implement in the future. Therefore, it is extremely important to accurately choose the best alternative for the system being studied.

A common comparison technique is to display a mean, and often, a confidence around the mean and allow for sorting based on comparison of the means. A user is inclined to pick the scenario with the highest/lowest mean with little to no information regarding how the scenario compares to other alternatives.

There has been considerable effort to research and develop statistically valid Ranking and Selection procedures over the past several decades – a significant amount of which was done by Barry L. Nelson from Northwestern University. He has come up with a series of algorithms that group Response values into the subgroups “Possible Best” and “Rejects” within each Response. The Possible Best group will consist of scenarios that cannot be proven statistically different from each other, but can be proven to be statistically better than all of the scenarios in the Rejects group. For more information, you can read Boesel, J., B.L. Nelson, and S.Kim, “Using Ranking and Selection to ‘Clean Up’ After Simulation Optimization,” *Operations Research*, Vol. 51, No. 5, September-October 2003, pp. 814-825.

To indicate what constitutes “Best” the user will define a Response’s Objective in the Response Properties window, to be either Maximize or Minimize. Maximize Objective will group the values that are statistically greater than the group of rejects, and conversely the Minimize Objective will group together the smallest response values.

In the Experiment Window’s Design window, the cells in a Response column will be highlighted goldenrod yellow if they are considered to be “Rejects” leaving the “Possible Best” scenarios with their intended coloring. A scenario can have a cell in one column considered “Possible Best” and at the same time “Rejects” for another response.

With this visual feedback, a user can easily see what scenarios are worth further examining using the [‘Select Best Scenario Using KN’ Add-In](#).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Select Best Scenario Using KN Add-In

Select the Best Scenario using KN

What is 'Select the Best Scenario using KN'?

This free Add-In implements the procedure developed by Seong-Hee Kim and Barry L. Nelson (Kim and Nelson, 2001) for selecting the best scenario. Their procedure works with or without common random numbers. This Add-in incorporates their ranking and selection procedure as published, with a few modifications to improve performance and make use of Simio's multi-processor capability.

What does this Add-In do?

It takes an initial set of scenarios and runs additional replications until it is confident that a particular scenario is the best, or within a specified range of the best (called the indifference zone). It will uncheck all scenarios except for the best. As it is executing it will "kick out" scenarios that are not a candidate for the best, and no more replications will be performed on those scenarios.

When to use this Add-In

This Add-In is to be used with a set of "interesting" scenarios. An interesting scenario is one that after running an initial number of replications still may possibly be the best. To achieve this, simply run your entire set of scenarios and make use of [Simio's Subset Selection](#) feature. This feature will present to you a set of Possible Bests; sometimes this set will be just one scenario while other times it may be several. This Add-In is designed to take it one step further to attempt to narrow it down to just one best as opposed to a set of possible bests.

Note that the KN Add-In will only narrow down the scenarios based on a single response, designated by the *Primary Response Experiment* property.

How does it work?

The first thing that the algorithm does is force the scenarios with fewer replications to run until every scenario has the same number of replications completed. The Add-In will then run a fixed number of additional replications for each scenario and analyze the results to determine if any of the candidate scenarios can be deemed as rejects. If a scenario is rejected, it gets unchecked and no longer included in the selection process. These scenarios will be removed from the [Response Chart](#). It repeats this process, running a fixed number of replications for each remaining candidate until one candidate can be selected as the best.

How to use the Add-In

As stated earlier, you must first run an initial number of replications in order to determine a set of interesting scenarios. Then you must select the *Select the Best Scenario using KN* Add-In in the Add-Ins section of the Design Ribbon.

After the Add-In is engaged, the Experiment Properties will now show a "Select the Best Scenario using KN – Parameters" section with three new Parameters:

1. Confidence Level – The statistical confidence in which the algorithm is to compare scenarios
2. Indifference Zone – This is number is the smallest meaningful distance between the best response so far and the other candidate responses, and is required for the Add-In to operate. If two response values are separated by less than this number (i.e. they are in "The Indifference Zone") they are considered to be the same and either can be considered to be the best.
 - For example, you are buying a car and have an indifference zone of \$300 with an objective to minimize cost. If one car costs \$25,600 and the same model costs \$25,800, you would consider them to be the same price. You would keep looking at other versions of the same model until you found one listed at \$25,400. Now, the \$25,800 is rejected and you continue to look for cars until you have exhausted your search and are down to one for \$24,850 and \$24,900. You have two cars that are considered to have the same price and now you make your decision based on color instead of price.
3. Replication Limit – The maximum number of replications you are willing to wait for the algorithm to find the definite best. If it reaches this limit, and no definite best can be determined, it will just leave the possible bests left checked while un-checking the rejects.

Note: A small Indifference Zone might need more replications to determine the best or subset of bests. If you see a large subset at the end of the run, experiment with a larger Indifference Zone or a larger Replication Limit. Keep in mind, with a

smaller Indifference Zone, the algorithm needs to be more precise in picking a subset that contains the best. Therefore, there is more bias about unchecking a Scenario. If a larger subset of Scenarios remains, there is a better likelihood that there exists a Scenario within the small zone of the best.

The Add-In can only analyze one response at a time, so you must define a Primary Response in the Analysis window. This is the Response column that you are going to base your decision upon and represents the quality of your system, similar to an Objective Function. You must also define an Objective (Maximize or Minimize) so that the Add-In can determine what is considered "Best" – lower values or higher values.

Once you have all your parameters defined correctly, press the Run button and the Add-In will do all of the work. After, the add-in is done running, if a best is found, there will be only one scenario left checked in the Design Tab and only one scenario in the [Response Chart](#).

S. Kim and B. L. Nelson, "A Fully Sequential Procedure for Indifference-Zone Selection in Simulation," *ACM Transactions on Modeling and Computer Simulation* 11 (2001), 251-273.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Select Best Scenario Using GSP Add-In

Select the Best Scenario using GSP

The GSP add-in is a scenario selection algorithm with good performance for large scale models. This add-in is similar to the add-in described in [Select Best Scenario using KN Add-In](#), but may provide improved performance when doing larger scale models that require evaluation of many scenarios in a parallel processing environment.

This add-in was created by Sijia Ma as part of his PhD work at Cornell University. You can read his thesis here: <https://ecommons.cornell.edu/handle/1813/64955>

For more information on this algorithm, please read [Eric C. Ni, Dragos F. Ciocan, Shane G. Henderson, Susan R. Hunter \(2017\) Efficient Ranking and Selection in Parallel Computing Environments. Operations Research 65\(3\):821-836.](#)

Experiment Properties

Properties/parameters can be found in the Properties window of the Experiment (select Experiment in the Navigation window and right click to select Experiment properties) when specifically using this option:

- Confidence Level – Confidence level for selecting the best scenario.
- Indifference Zone – The smallest meaningful difference that is used for defining the best. All scenarios that fall within this smallest meaningful difference are close enough that any of them may be selected as the best.
- Replication Limit – The maximum number of replications for each scenario that will be made before automatically terminating the procedure.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

bi-PASS Scenario Elimination

bi-PASS Scenario Elimination

What is 'bi-PASS Scenario Elimination'?

The bi-PASS Scenario Elimination experimentation add-in implements the bisection Parallel Adaptive Survivor Selection (bi-PASS) algorithm for eliminating non-competitive systems developed by [Linda Pei, Barry L. Nelson, and Susan R. Hunter \(Pei et al., 2022\)](#).

What does this add-in do?

First, initial replications are completed for all scenarios. After, the add-in iteratively runs one replication of each active scenario, then attempts to eliminate scenarios based on their response values. The cycle of collecting one additional replication of each active scenario and checking each scenario for elimination continues until either one scenario remains or the user-specified maximum replications is reached.

How to use the add-in

To use the bi-PASS Scenario Elimination experimentation add-in, ensure the target experiment already includes a set of scenarios. Select the bi-PASS Scenario Elimination add-in from the "Select Add-In" dropdown in the Add-Ins section of the Design ribbon. Once the add-in is selected, the experiment property grid will show a "bi-PASS Scenario Elimination" category with four new parameters.

- **Confidence Level** - The expected proportion of retained competitive scenarios relative to the total number of competitive scenarios. A higher Confidence Level reduces the expected proportion of competitive scenarios that are falsely eliminated, but might require more replications to eliminate scenarios.
- **Initial Replications** - The number of replications of each scenario to run before any scenario can be eliminated.
- **Maximum Replications** - The maximum number of replications of each scenario to run.
- **Lower Bound** - The lower bound of the overall sample mean when checking for elimination. Higher values will cause scenarios to be eliminated more aggressively, while lower values may have no effect. If a Lower Bound would cause all scenarios to be eliminated, the bound will revert to negative infinity for the duration of the current experiment run.

In addition to the experiment parameters, two response parameters are used by the add-in.

- **Weight** - The weight given to this response when calculating scenario statistics. The response value for each replication is multiplied by this weight when calculating the composite response value.
- **Objective** - Indicates if larger or smaller values of this response are preferred. If 'None', the response will not be included in the composite response value. At least one response must have an Objective other than 'None'. If a response has an Objective Type of 'Minimize', its value will be multiplied by '-1' when calculating the composite response value.

The add-in works even if some or all scenarios have existing replication data. Therefore, if the experiment run completes due to reaching the maximum number of replications, the *Maximum Replications* parameter could be increased and the experiment resumed. Or, new scenarios could be added after completing an experiment run.

The specialization of bi-PASS is quickly eliminating non-competitive scenarios in experiments with many scenarios. However, once an experiment has been reduced to a quantity of competitive scenarios less than the number of concurrent replications that can be run, other methods to find the best scenario such as Select Best Scenario Using KN might take less time by fully utilizing the number of available concurrent replications. The [bi-PASS with Rinott Experimentation Add-In](#) can also be used, which automatically switches to Rinott to reduce the required number of replications to find the best scenario.

How does it work?

The add-in first runs initial replications of each scenario as specified by the 'Initial Replications' experiment parameter. Using the replication data, the add-in calculates the mean and variance of the scenario responses across all replications. To combine multiple responses for each scenario, the add-in uses a composite response value as show below.

The composite response value r is calculated as

$$r = \sum_{k \in M}^K o_k w_k v_k$$

Where

- M is the set of responses.
- o_k is the objective type multiplier of response k . o_k is '1' if the *Objective* of response k is 'Maximize', '-1' if the *Objective* of response k is 'Minimize', and '0' if the *Objective* of response k is 'None'.
- w_k is the weight of response k .
- v_k is the value of response k .

Across the experiment, the overall sample mean of the composite response values $\bar{\mu}$ is calculated as

$$\bar{\mu} = \frac{\sum_{i \in Q} R_i}{\sum_{i \in Q} n_i}$$

Where

- Q is the set of surviving scenarios.
- $R_i = \sum_{j=1}^{n_i} r_{ij}$ is the sum of each replication's composite response values of scenario i .
- n_i is the number of replications completed by scenario i .

The pooled sample variance is similarly calculated as

$$\hat{\sigma}_{pooled}^2 = \frac{\sum_{i \in Q} S_i - R_i^2/n_i}{\sum_{i \in Q} n_i - |Q|}$$

Where

- Q is the set of surviving scenarios.
- $S_i = \sum_{j=1}^{n_i} r_{ij}^2$ is the sum of squares of each replication's composite response values of scenario i .
- $R_i = \sum_{j=1}^{n_i} r_{ij}$ is the sum of each replication's composite response values of scenario i .
- n_i is the number of replications completed by scenario i .

Each scenario is then checked for elimination using the following test.

$$\frac{n_i}{\hat{\sigma}^2} \left(\frac{R_i}{n_i} - \mu^* \right) < -g\left(\frac{n_i}{\hat{\sigma}^2}\right)$$

Where

- $\hat{\sigma}^2$ is the sample variance $\hat{\sigma}_i^2$ if $n_i \geq 10$ or the pooled variance $\hat{\sigma}_{pooled}^2$ otherwise.
- $g(t) = \sqrt{[c + \log(t+1)](t+1)}$ where $c = 3.6, 5.0, 8.5$ for confidence intervals 0.9, 0.95, 0.99.
- $\mu^* = \max(LB, \bar{\mu})$ where LB is the user-specified lower bound of the sample mean $\bar{\mu}$. If the lower bound would cause all scenarios to be eliminated, the bound is set to $-\infty$, the set of remaining scenarios is restored, the statistics are updated, and the elimination is checked again.

The add-in iteratively collects one additional replication of each remaining scenario, updates statistics, and checks each scenario for elimination as described. This cycle continues until either only one scenario remains or the user-specified maximum number of replications is reached.

bi-PASS with Rinott Experimentation Add-In

bi-PASS with Rinott Experimentation Add-In

What is 'bi-PASS with Rinott'?

The bi-PASS with Rinott add-in executes the bisection Parallel Adaptive Survivor Selection (bi-PASS) algorithm ([Pei et al., 2022](#)), followed by the Rinott selection procedure ([Rinott 1978](#)).

What does this add-in do?

This add-in runs replications of existing scenarios, eliminates non-competitive scenarios using bi-PASS, then automatically switches to Rinott to run additional replications to find the best scenario. The switch from bi-PASS to Rinott is timed to reduce the number of additional replications and maximize the number of replications Simio runs concurrently.

How to use the add-in

To use the bi-PASS with Rinott add-in, ensure the target experiment already includes a set of scenarios. Select the bi-PASS with Rinott add-in from the "Select Add-In" dropdown in the Add-Ins section of the Design ribbon. Once the add-in is selected, the experiment property grid will show a "bi-PASS with Rinott" category with four new parameters.

- **Confidence Level** - While executing bi-PASS, the confidence level corresponds to the expected proportion of retained competitive scenarios relative to the total number of competitive scenarios. While executing the Rinott selection procedure, the confidence level is the expected probability of selecting the best scenario.
- **Initial Replications** - The number of replications of each scenario to run before any scenario can be eliminated.
- **Maximum Replications** - The maximum number of replications of each scenario to run.
- **Lower Bound** - The lower bound of the overall sample mean when checking for elimination. Higher values will cause scenarios to be eliminated more aggressively, while lower values may have no effect. If a Lower Bound would cause all scenarios to be eliminated, the bound will revert to negative infinity for the duration of the current experiment run.
- **Indifference Zone** - The smallest meaningful difference used by the Rinott selection procedure for defining the best scenario. All scenarios that fall within this smallest meaningful difference are considered equally competitive. Smaller indifference zones could require many more replications to find the best scenario.

In addition to the experiment parameters, two response parameters are used by the add-in.

- **Weight** - The weight given to this response when calculating scenario statistics. The response value for each replication is multiplied by this weight when combining multiple response values into a composite response value.
- **Objective** - Indicates if larger or smaller values of this response are preferred. If 'None', the response will not be included in the composite response value. At least one response must have an Objective other than 'None'. If a response has an Objective Type of 'Minimize', its value will be multiplied by '-1' when calculating the composite response value.

The add-in works even if some or all scenarios have existing replication data. Therefore, if execution of the add-in stopped due to reaching the maximum number of replications, the *Maximum Replications* parameter could be increased and the experiment resumed.

How does it work?

The add-in begins the experiment run by executing the bi-PASS algorithm. For more information about bi-PASS, see [bi-PASS Scenario Elimination Experimentation Add-In](#). After each bi-PASS iteration, the add-in conducts a series of checks to determine if the switch from bi-PASS to Rinott should occur immediately.

If any of the following conditions are met, the add-in switches from bi-PASS to the Rinott selection procedure.

- The number of remaining scenarios is less than the maximum number of concurrent replications.
- '15' bi-PASS iterations have occurred without any scenario eliminations.
- The number of additional replications required by the Rinott selection procedure is '0'.

- The estimated number of additional replications required by bi-PASS to eliminate all but one scenario is greater than the number of replications required by the Rinott selection procedure. This check only occurs if at least 50% of scenarios have been eliminated to avoid switching prematurely.

After switching to the Rinott selection procedure, the add-in uses the existing replication data to calculate the number of additional replications required to find the best scenario as shown by the equation below.

$$N_i = \max\left\{\left\lceil \frac{h^2 \hat{\sigma}_i^2}{\delta^2} \right\rceil - n_i, 0\right\}$$

Where

- i is the index of the scenario.
- N_i is the number of required additional replications for scenario i .
- h^2 is an estimation of Rinott's constant. A binary search is used to find an estimation corresponding to the user-specified confidence level, which is the target probability of correct selection. The probability of correct selection for any given value of h is estimated using a Gauss-Laguerre quadrature.
- $\hat{\sigma}_i^2$ is the variance of the composite response value. For more information about the composite response value, see [bi-PASS Scenario Elimination Experimentation Add-In](#).
- δ^2 is the user-specified indifference zone.

- n_i is the number of replications completed by scenario i .

Once the required number of additional replications is calculated for each scenario, the add-in runs the additional replications. When the replications are completed, all remaining scenarios except the scenario with the largest composite response mean are deactivated, leaving only one scenario remaining.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Experiment Example

An Example of Creating and Running an Experiment (Using the AirportTerminal Example Model)

The Simio example model, titled AirportTerminal.spfx, contains an Experiment that utilizes the [OptQuest for Simio Add-In](#). It is recommended that you familiarize yourself with this example in order to learn about the features and capabilities that are available when running experiments in Simio. The following is a step by step approach on how this example experiment was created. To create a new experiment, click on the New Experiment icon in the Project Home ribbon. You can also create a new experiment by right clicking on the name of the model from within the [Navigation window](#).

Experiment Controls

A [Control](#) is automatically added to your experiment for any user defined properties that are at the model level. In this example, there are four model level Properties that have their *Visible* property set to 'True', which indicates that they will appear as a control in an experiment. The properties are: CheckIn Kiosks, CheckIn Clerks, ID Check, Scanner Stations. In an experiment that does not use the OptQuest Add In, these controls do not have properties. But when the OptQuest Add-In is used, you need to specify the following properties, which tell OptQuest whether or not to include this in its optimization, the minimum and maximum values to set this control to, and how to increment the value. Notice that in this example, only the first two controls are included in the optimization because their *Include In Optimization* property is set to 'yes'.

Properties of the "Checkin Kiosks" Control

Properties: CheckinKiosks (Control)	
OptQuest for Simio - Parameters	
Include in Optimization	Yes
Minimum Value	1
Maximum Value	8
Increment	1

Experiment Responses

Click on *Add a Response* in the Experiment Design ribbon to add a Response to your experiment. This example contains three responses: Revenue, Profit and Cost. Click on the column header for Profit to see the properties of this Response. The properties indicate that the expression being evaluated is the value of the "Profit" OutputStatistic (look in the Elements panel within the model to see details on this statistic), and the *Objective* property indicates that the profit value should be maximized. The other two responses have an expression defined, but not an objective.

Properties of the Response "Profit"

Properties: Profit (Response)	
General	
Name	Profit
Display Name	Profit
Expression	Profit.Value
Unit Type	Unspecified
Objective	Maximize
Lower Bound	0
Upper Bound	

Experiment Properties

The Experiment properties for this example are shown below. Note: To get the Experiment Properties window to display at any time, right click on the name of the experiment in the navigation window and select Experiment Properties.

Experiment Properties window (when OptQuest Add-In is used)

Properties: Experiment1 (Experiment)	
Analysis	
Warm-up Period	0
Default Replications	10
Confidence Level	95%
Upper Percentile	75%
Lower Percentile	25%
Primary Response	
Advanced Options	
Concurrent Replicati...	0
Replication Runner	InProcess
Distribute Runs	False
Generate Statistics	True
General	
Name	Experiment1

The *Upper Percentile* and *Lower Percentile* properties are used in the [SMORE plots](#). The *Primary Response* property tells Simio which Response to use for determining the optimal solution. In this example, we want our optimal solution to be based on the Profit Response and the objective that is defined on that Response.

The OptQuest for Simio-Parameters section of the Experiment properties is where the user defines the information for the OptQuest Add In. In this example, Simio will run a minimum of 6 replications and a maximum of 10 replications for each scenario. OptQuest decides how many replications are required for each scenario. Before the user hits the run button to begin the experiment, it appears to contain only one scenario. However, the OptQuest Add In runs multiple scenarios, until it has found the optimal solution, based on your defined Controls, Constraints and Objective on the Primary Response. The Experiment property, *Max Scenarios*, indicates that a maximum of 10 scenarios will be run. In this example, the full 10 scenarios are run.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Replication Runner

What is Simio Replication Runner?

Simio Replication Runner is a program that allows an Experiment in Simio to distribute its replications to other computers on the local network. When other network computers are running Simio Replication Runner, the Simio experiment will use the resources on that computer for the experiment run, thus distributing the processing requirements across multiple computers on the network.

An experiment will distribute its runs across other machines running Replication Runner when it has its *Distribute Runs* property set to 'True'.

If the model contains a process that writes to an external file, the file and file path must exist or be accessible to any machines where runs have been distributed. Consider instead writing to a database or using an MQTT step.

NOTE: SIMIO REPLICATION RUNNER IS ONLY AVAILABLE FOR USE WITH RPS AND PROFESSIONAL LICENSES. Simio RPS and Simio Professional allow up to 16 concurrent replications. Additional replication runner licenses may be purchased.

Installation and Starting Simio Replication Runner

Simio Replication Runner is installed separately from the Simio software installation. The installation file is provided alongside the Simio software installation file. To install Simio Replication Runner, simply double click on the SimioReplicationRunnerInstall-XXXXX.msi file, where XXXX is the version of the program. This version should match the software version of Simio that is currently installed. Note: When a Simio client connects to a Replication Runner it will verify that the Replication Runner is a version that is equal to or newer than it.

Once the Simio Replication Runner is installed, it needs to be started before it can be used by Simio experiments. To start the Simio Replication Runner, navigate to the folder location where it was installed (for example C:\Program Files\Simio LLC\Simio\Simio Replication Runner), and double click on the SimioReplicationRunner.exe file. A new window will appear and a message should appear in that window that says, "Experiment replication runner service is ready." Do not hit Enter or else the service will stop.

Once started, Simio Replication Runner will attempt to figure out the computer's IP address and it will start opening available ports for communication.

Firewalls and TCP Ports

Depending on your firewall setup, you may be automatically prompted to allow Simio Replication Runner to go through the firewall, or you may need to add it as an exception manually.

If you wish to configure Simio Replication Runner to use a specific name or IP for the machine, one that is known on the network so other machines can reach it, or wish to configure it to use a single specific port (for example because you want to open a specific port on your firewall for it), then you can do that by editing the SimioReplicationRunner.exe.config file. To change the machine name that will be used by clients to connect, edit the "DNSMachineName" value. To change the TCP Port that will be used for connections, edit the "TCPPort" value.

Replication Runner Broadcast and Subnets

When a Simio client wishes to distribute its runs it will do a multicast UDP broadcast over its local subnet. Simio Replication Runners that are running will listen for this broadcast and respond with their address, so the client can then start using them. Furthermore, when a Simio Replication Runner is first started up, it will send out a UDP broadcast to let any running clients know a new Replication Runner is available.

Therefore, in order to use the Simio Replication Runner, both the Simio client machine and the Replication Runner machine must be on the same subnet and must be able to send UDP messages to each other.

Accessing Replication Runners via a VPN Connection

If you are distributing experiment runs to a replication runner service and you are connected to the network via VPN, you will need to specify the address and port of the machine you are trying to reach. This is done in the [Application Settings](#) window of Simio, which is found under the File menu.

CMD Command for Setting Concurrent Replications on the Replication Runner

To control the number of current replications per Replication Runner, you can use the command-line interface. You may want to use this method so that you get better utilization of the computer's or computers' cores. Simio allows up to 16

concurrent replications, with more available for purchase. The following instructions are for the 64-bit version of Simio, appropriate syntax substitutions can be made for 32-bit.

- First, using Task Manager, ensure that the SimioReplicationRunner service is not running.
- Then, using command-line prompts, set the directory to the installation folder of Simio. Something like `<cd\Program Files\Simio LLC\Simio>`.
- Next, type `<SimioReplicationRunner.exe -maxreps:1>`.
- Repeat this process to open 16 (or $n \leq 16$, where n is the number of concurrent replications you want to run and $n \leq$ cores) separate instances of Simio Replication Runner.

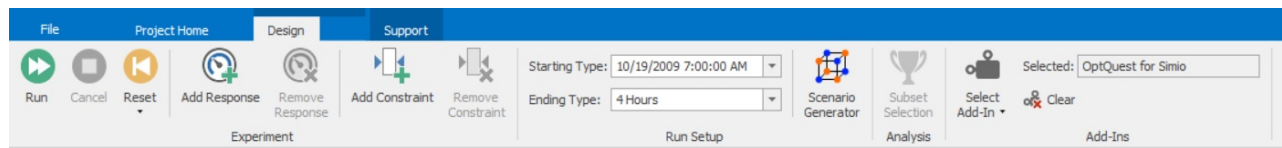
Alternatively, you can use `<start SimioReplicationRunner.exe -maxreps:1>` to open a new window with the Simio Replication Runner. With this technique, you can then use the up-arrow key to repeat the last entry and press enter to open another instance. Or, you can write your own *.bat or *.cmd file to call where there are n lines of `<start SimioReplicationRunner.exe -maxreps:1>`.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Scenario Generator

The Scenario Generator allows for the quick, customizable, full factorial creation of Scenarios within an Experiment. The Scenario Generator allows you to toggle on/off any Controls and set their *Minimum*, *Maximum*, and *Level*. When generating Scenarios, the Scenario Generator will respect any Constraints in the Experiment.



Clicking the Scenario Generator button opens up a window where you can expand each eligible Control and specify its *Minimum*, *Maximum*, and *Level*. Turning the toggle to green includes the Control in the generation, leaving it grey tells the Generator to use that Control's *Default Value*. Toggling the "Use Control Values to Name Scenarios" on will have each Scenario's name be based on the values being used and toggling it off will have each Scenario's name be based off of the "Scenario1, Scenario2, ScenarioN" pattern.

Scenario Generator

Name	Include	Summary
CheckinKosks	<input checked="" type="checkbox"/>	Minimum Value:0 Maximum Value:1 Number of Levels:2
Min	Max	Levels
0	1	2
CheckinClerks	<input type="checkbox"/>	Not included. Default value of '2' will be used.
Min	Max	Levels
0	1	2
IDCheck	<input type="checkbox"/>	Not included. Default value of '1' will be used.
ScannerStations	<input type="checkbox"/>	Not included. Default value of '2' will be used.

☐ Use control values to name scenarios

Current settings may generate up to 2 scenarios (2)

Generate Scenarios

Note: The Scenario Generator will not create duplicate Scenarios or if a Scenario already exists it won't be recreated if using the Generator again.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

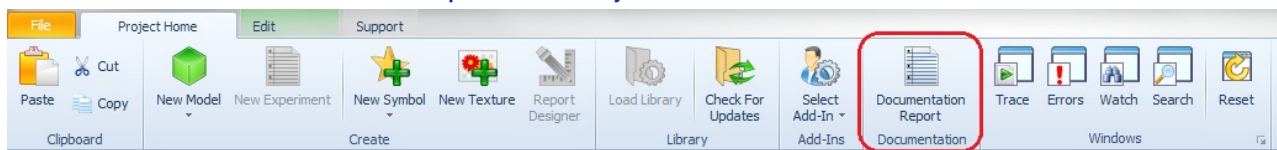
Model Summary Report

Producing a Report Containing the Contents of your Model

By clicking on the Documentation Report icon found in the Project Home ribbon, an HTML report is produced, that contains the contents of the models located in the current project. The report will contain information such as:

- All of the models in the projects and all of the objects contained within each model
- The Nodes that are associated with each object and the Links that arrive to the object and leave from the object
- Properties that have been overridden from their default values
- Custom Properties
- Custom States
- Custom Elements
- Custom Processes and the Steps contained in each
- Custom Tokens
- Tables and all of the data within the tables

The Documentation Report icon in the Project Home Ribbon



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Custom Simio Extensions

Introduction

Simio's architecture provides several extension points where users can integrate their own custom functionality written in a .Net language such as Visual C# or Visual Basic .Net.

The types of user extensions supported include:

- User Defined Steps
- User Defined Elements
- User Defined Selection Rules
- Application Add-Ins

Simio's extension points have been exposed as a set of *interfaces* that describe the methods and calling conventions to be implemented by any user extended components. This set of interfaces is referred to as the Simio *application programming interface* (API). For detailed information on the Simio API, see the file **C:\Program Files\Simio LLC\Simio\Simio API Reference Guide.chm**.

Simio Visual Studio Templates and Examples:

To help get started creating Simio user extensions, a number of predefined Simio project and project item templates for Microsoft Visual Studio 2012, 2013 and 2015 are available. These templates provide reusable and customizable project and item stubs that may be used to accelerate the development process, removing the need to create new projects and items from scratch.

For more information, see [Simio Visual Studio Templates](#).

Additionally, several [User Extension Examples](#) are included with the Simio installation.

General Steps to Create and Deploy a User Extension:

The general recommended steps to create and deploy a user extension are as follows:

- Create a new .Net project in Visual Studio, or add an item to an existing project, using one of the [Simio Visual Studio templates](#). Note that, in addition to the commercial versions of Visual Studio, Microsoft also offers "Express" editions which are available as free downloads from www.microsoft.com/express/Windows.
- Complete the implementation of the user extension and then build the .Net assembly (.dll) file.
- To deploy the extension, copy the .dll file into the **[My Documents]\SimioUserExtensions** directory. If the folder does not already exist you must create it. As an alternative, you may also copy it to [Simio Installation Directory]\UserExtensions, but ensure that you have proper permission to copy files here.

Using a Deployed User Extension in Simio:

In a model's [Processes Window](#), all deployed user defined steps will be available from the left hand steps panel under User Defined.

In a model's [Definitions Window](#), when defining elements, all deployed user defined elements will be available via the User Defined button in the Elements tab of the ribbon interface.

All deployed application add-ins will be available for use in a project via the Actions (Add-Ins) button in the Project Home ribbon. RPS licensed software will also include a Modeling Helpers button in the Add-Ins area of the Project Home ribbon.

All deployed [user defined selection rules](#) will be available for use in a model as dynamic selection rules.

▲ See Also

Simio Visual Studio Templates

User Extensions: Getting Started Using Simio Visual Studio Templates

To help get started creating Simio user extensions, a number of predefined project and project item templates for Microsoft Visual Studio 2012, 2013, 2015 and 2017 are available to be installed. These templates provide reusable and customizable project and item stubs with example code snippets that may be used to accelerate the development process, removing the need to create new projects and items from scratch.

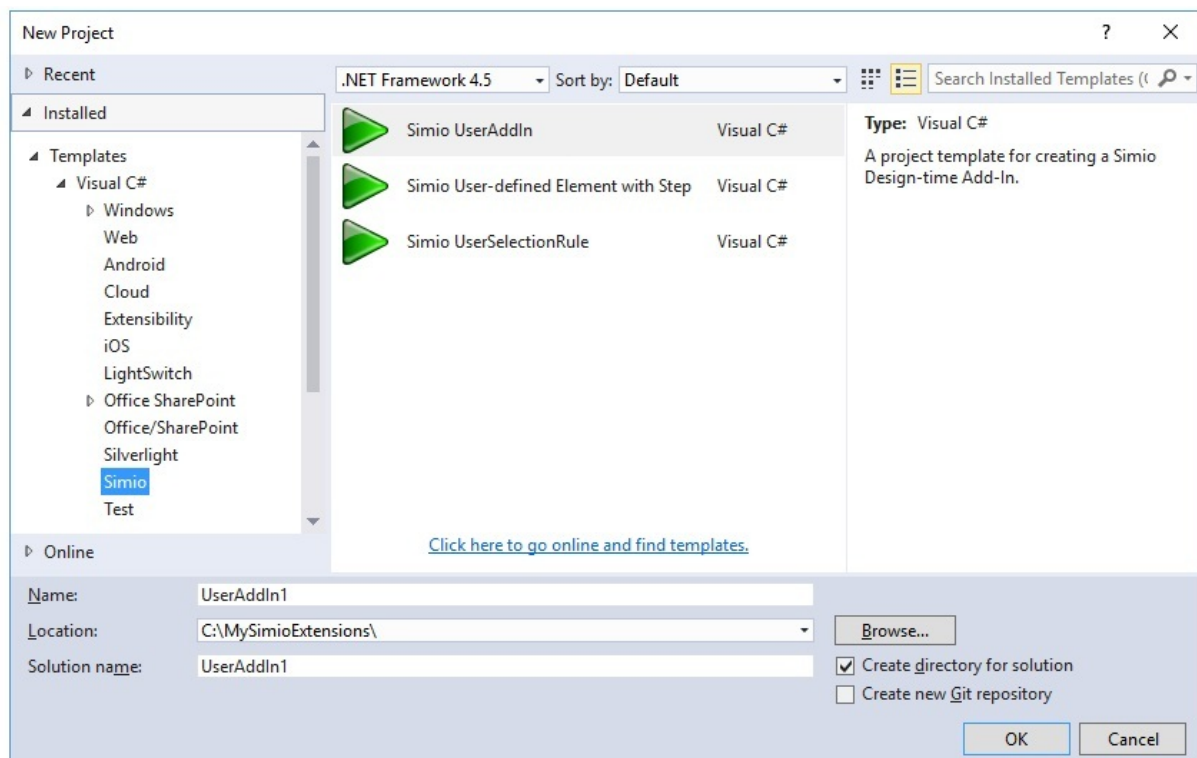
Installing the Simio Visual Studio Templates:

Simio Visual Studio templates are installed separately from the Simio application. To install the templates, go to the Windows Start menu, and select the **Simio>Install Simio Visual Studio Templates** menu item.

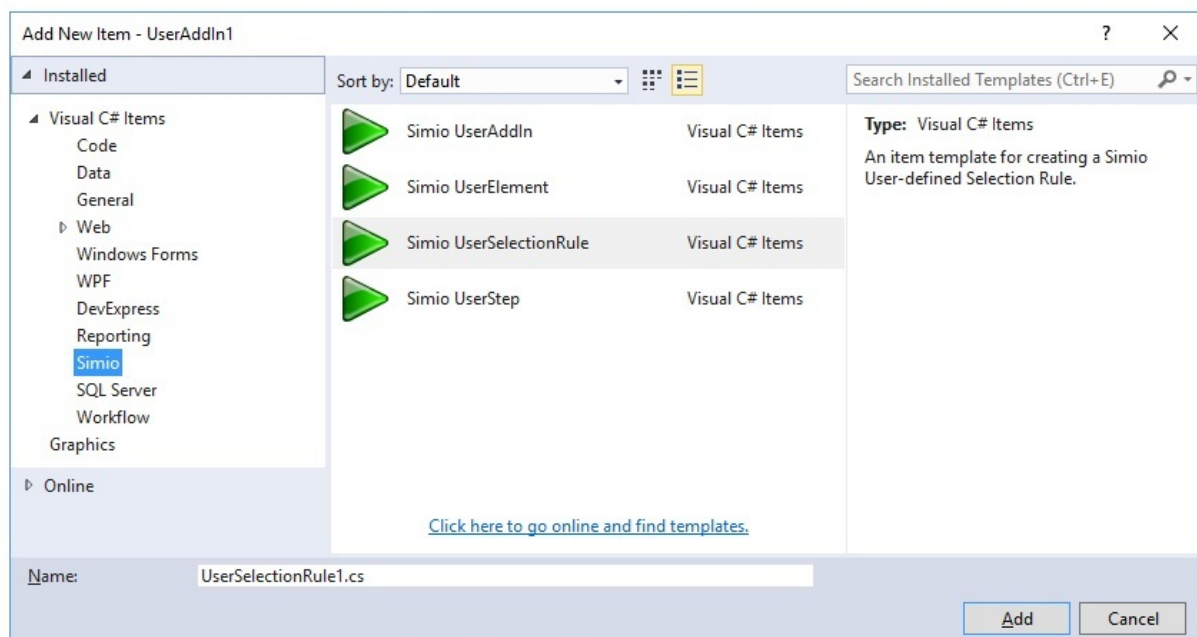
Using the Simio Visual Studio Templates:

Once the templates have been installed, they will be available for selection in the **New Project** and **Add New Item** dialog boxes in Visual Studio. Refer to screen shots of those dialogs below. First select the 'Simio User Extensions' category and then select the desired template.

[Adding Simio Templates into Visual Studio](#)



Visual Studio 'New Project' Dialog



Visual Studio 'Add New Item' Dialog

After creating your custom project in Visual Studio and building the .dll file, move the .dll file into the [My Documents]\SimioUserExtensions directory. If the folder does not already exist you must create it. Restart Simio and it will automatically find that .dll and recognize your custom user extension.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Simio User Extension Examples

Simio User Extension Examples

Several user extension examples are included with the Simio installation. By default, these examples are installed in the directory **C:\Program Files\Simio LLC\Simio\Examples\UserExtensions**.

User Defined Step and Element Examples

- *TextFileReadWrite* – This folder contains an example project implementing a custom [File](#) element and custom [Read](#) and [Write](#) steps. The File element may be added to a model to specify an external file name. The Read and Write steps may then be used in process logic to perform runtime read or write to a specified File element.
- *BinaryGate* – This folder contains an example project implementing a custom [BinaryGate](#) element and custom [OpenGate](#), [CloseGate](#), and [PassThruGate](#) steps. A BinaryGate element may be added to a model to specify an 'Open/Closed' constraint. The OpenGate and CloseGate steps may then be used in process logic to open or close a specified BinaryGate element. The PassThruGate step may be used in process logic to only allow an executing token to continue (i.e., 'pass thru') to the next step in the process if the specified BinaryGate element is 'Open'.

User Defined Selection Rule Examples

- *SimioSelectionRules* – This folder contains the implementation of several custom selection rules, including commonly used 'Largest Value First' and 'Smallest Value First' rules.

Application Add-In Examples

- *SourceServerSink* – This folder contains an example project implementing a custom add-in that places Source, Server, and Sink objects from the Simio Standard Library into the current model, and connects those objects using Path objects from the Standard Library.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Dynamic Selection Rules

Dynamic selection rules allow various places in **Simio** to select from a set of things based on some dynamic criteria.

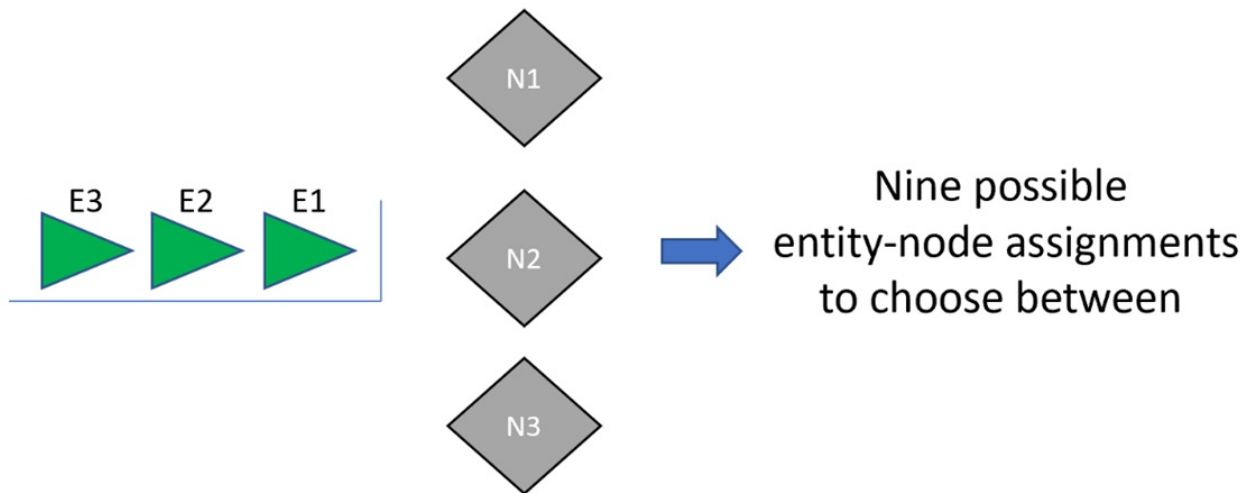
Users can write their own rules in any .NET language, by implementing the interfaces **ISelectionRuleDefinition** and **ISelectionRule**. For more information about creating your own Dynamic Selection Rules, see [Custom Simio Extensions](#) and the file **C:\Program Files\Simio LLC\Simio\Simio API Reference Guide.chm**.

These rules are used in Resource allocation selection (used within many of the Standard Library objects, such as [Server](#), [Resource](#), [Combiner](#), [Separator](#), [Worker](#), [Vehicle](#)) and [Workstation \(Deprecated\)](#). Also used within the [Routing Group](#) route request selection, [TransferNode](#) route request selection and [Station](#) entry selection.

Note

Most of the time you will want to use the keyword [Candidate](#) when entering an expression.

Consider an example where a routing group with three possible destination nodes is checking a route request queue containing three entities. Therefore, the routing group has nine possible entity-node assignments to choose between.



Assume for the purposes of this example that the routing group is using a dynamic selection rule. And that the selection priority values for each of the nine possible entity-node assignments are as shown in Table 1.

Table 1 - Selection Priority Values for Each Possible Entity-Node Assignment

Node	Entity	Dynamic Selection Priority	Route Request Queue Rank	Node Selection Priority
N1	E1	2	1	3
N1	E2	3	2	3
N1	E3	3	3	3
N2	E1	2	1	1
N2	E2	1	2	1
N2	E3	1	3	1
N3	E1	2	1	2
N3	E2	1	2	2
N3	E3	2	3	2

Where:

Dynamic Selection Priority – Is a priority level of the entity-node assignment given the dynamic selection rule being used. For example, if the Least Setup Time dispatching rule were used, then the entity-node assignment with the smallest expected setup time would have the highest priority. In this example, assume a priority value of 1 is the highest level of priority.

Route Request Queue Rank – Is the rank of the entity in the routing group's route request queue. The entity at rank 1 has the highest level of priority.

Node Selection Priority – This is a priority level of the entity-node assignment given the Selection Goal property on the Route step or TransferNode object. For example, if the Preferred Order selection goal were used, then the node selection priority would be according to the routing group's node list order. In this example, assume a priority value of 1 is the highest level of priority.

Dynamic Selection Rule Algorithm

The dynamic selection rule algorithm implemented by a routing group will execute as follows:

- Choose the best entity-node assignment from all possible entity-node pairs using the dynamic selection rule priority.
- The tie-breaker between entity-node assignments with equal dynamic selection rule priority is the route request queue rank.

- The secondary tie-breaker between entity-node assignments with equal dynamic selection rule priority that involve the same entity is the node selection priority as determined by the *Selection Goal* property on the Route step or TransferNode object.

The above algorithm will choose the expected entity-node assignment of entity 'E2' to destination node 'N2' as shown in Table 1.

Built In Rules

There are a number of built in selection rules. The C# sources to these rules are provided at C:\Program Files\Simio LLC\Simio\Examples\UserExtensions\SimioSelectionRules.

These rules were built to be as generic as possible. Several domain specific rules can be used by simply using the right expression in one of the rules.

Smallest Value First

Selects the entity which would have the smallest value for the given expression.

Largest Value First

Selects the entity which would have the largest value for the given expression.

For both the Smallest Value First and Largest Value First rules above, the *Value Expression* is used as the expression used with the rule. The keyword 'Candidate' may be used to reference an object in the collection of candidates (e.g., Candidate.Entity.Priority). The *Filter Expression* may be used to filter the list before the specified rule is applied. Again, the keyword 'Candidate' may be used to reference an object in the collection of candidates.

Standard Dispatching Rule

Allows a user to easily specify a commonly used dispatching rule such as Earliest Due Date (EDD) or Least Setup Time as the dynamic selection rule for selecting the next entity to process at a location such as a Server or Workstation (Deprecated).

A *Filter Expression* can be specified which is an optional condition for each candidate entity that must be true for the entity to be considered for selection. In the expression, use the syntax 'Candidate.[EntityClass].[Attribute]' to reference an entity attribute (e.g., Candidate.Entity.Priority).

A *Look Ahead Window* can be specified whereby only candidate entities whose due dates fall within a specific time window will be considered for selection. If there are no candidates whose due dates fall within the look ahead window, then the window will be automatically extended to include the candidate(s) with the earliest due date.

The below dispatching rules are available as drop-down list choices for the *Dispatching Rule* and *Tie Breaker Rule* sub-properties (when *Repeat Groups* for Standard Dispatching Rule is 'False') or within the multiple *Dispatching Rule* properties (when *Repeat Groups* for Standard Dispatching Rule is 'True').

Listed below are the **Standard Dispatching Rules** that are currently supported:

Dispatching Rule	Description
FirstInQueue	The entity ranked nearest the front of the queue is selected.
LargestPriorityValue	The entity with the largest priority state value is selected.
SmallestPriorityValue	The entity with the smallest priority state value is selected.
EarliestDueDate	The entity with the earliest due date is selected.
CriticalRatio	The entity with the smallest critical ratio is selected. Critical ratio is the amount of time remaining to complete the entity's assigned sequence in order to meet its due date divided by the total operation time remaining to complete the entity's sequence. Assumes that each candidate entity has been assigned a destination sequence.
LeastSetupTime	The entity with the least setup time is selected.
LongestProcessingTime	The entity with the longest operation time is selected.
ShortestProcessingTime	The entity with the shortest operation time is selected.
LeastSlackTime	The entity with the least slack time is selected. Slack time is the amount of time remaining until the entity's due date minus the total operation time remaining to complete the entity's sequence. Assumes that each candidate entity has been assigned a destination sequence.
LeastSlackTimePerOperation	The entity with the least average slack time per its remaining operations is selected. Assumes that each candidate entity has been assigned a destination sequence.
LeastWorkRemaining	The entity with the least total operation time remaining to complete its assigned sequence is selected. Assumes that each candidate entity has been assigned a destination sequence.
FewestOperationsRemaining	The entity with the fewest number of operations remaining to complete its assigned sequence is selected. Assumes that each candidate entity has been assigned a destination sequence.
LongestTimeWaiting	The entity that has been waiting the longest time in the queue is selected.
ShortestTimeWaiting	The entity that has been waiting the least time in the queue is selected.

LargestAttributeValue	The entity with the largest value of the specified expression is selected.
SmallestAttributeValue	The entity with the smallest value of the specified expression is selected.
CampaignSequenceUp	The entity that has a campaign value equal to or next largest compared to the value of the last processed entity is selected. If none of the waiting entities have the same or larger value, then the rule starts a new campaign by selecting the entity with the smallest value.
CampaignSequenceDown	The entity that has a campaign value equal to or next smallest compared to the value of the last processed entity is selected. If none of the waiting entities have the same or smaller value, then the rule starts a new campaign by selecting the entity with the largest value.
CampaignSequenceCycle	Alternates back and forth between a campaign sequence up and a campaign sequence down. If the campaign is increasing, it will continue to increase until no entities remain with the same or larger campaign value. When this occurs, the rule then switches to a decreasing campaign and begins selecting entities that have the same or smaller campaign value. When all such entities are exhausted, the rule returns to an increasing campaign strategy and the cycle repeats.

Dispatching Rule Usage Notes:

The use of a particular dispatching rule may require some user-specified information about the candidate entities, such as due dates, expected setup or operation times, destination sequences assigned to the entities, etc.

- Dispatching rules such as Critical Ratio, Least Slack Time, Least Slack Time Per Operation, Least Work Remaining, and Fewest Operations Remaining require the entities to be assigned destination sequences in a [Sequence Table](#).
- For a dispatching rule that looks at due dates (e.g., Earliest Due Date, Critical Ratio, Least Slack Time, Least Slack Time Per Operation), refer to the *Due Date Expression* property in the Advanced Options of an entity type to specify the expression used to return an entity's due date value.
- For a dispatching rule that looks at an entity's total work remaining to complete an assigned sequence (e.g., Critical Ratio, Least Slack Time, Least Slack Time Per Operation, Least Work Remaining), refer to the *Sequence Expected Operation Time* property in the Advanced Options of the nodes in the entity's assigned destination sequence to specify the expressions used to estimate the sequence step operation times.
- For the Least Setup Time dispatching rule, refer to the *Expected Setup Time Expression* property in the Advanced Options of the processor object (e.g., the Server or Workstation) to specify the expression used to estimate an expected setup time for an entity at the processing location.
- For the Longest Processing Time or Shortest Processing Time dispatching rules, refer to the *Expected Operation Time Expression* property in the Advanced Options of the processor object (e.g., the Server or Workstation) to specify the expression used to estimate an expected operation time for an entity at the processing location.
- Properties with *Expected* in the name are deterministic, so random distributions evaluate to the average/expected value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Protection

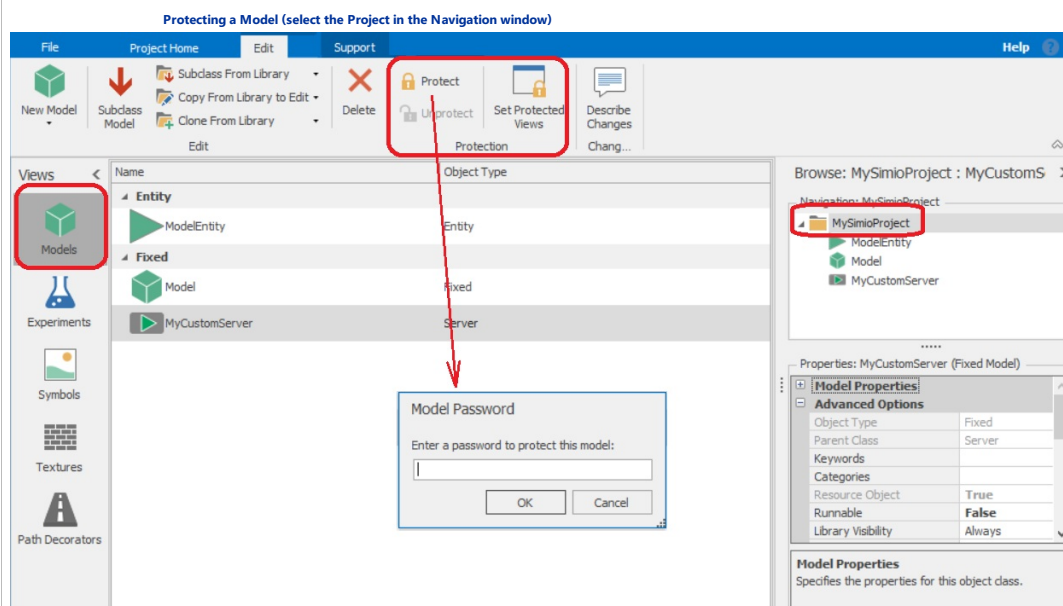
Protection

Protection features within Simio are located within the Edit ribbon of the main Simio project. First, navigate to the Project window from within the Navigation window. The Models view will display the various models within the project.

Protect / Unprotect Buttons

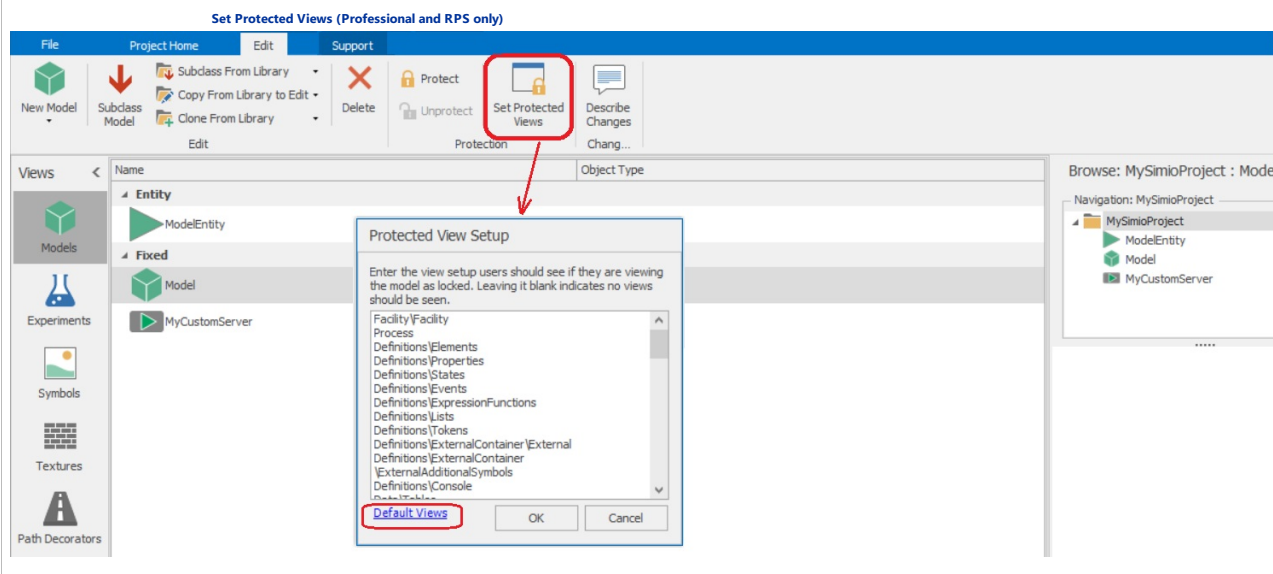
The **Protect/Unprotect** feature allows an object builder to prevent an unauthorized user from viewing, subclassing, or editing the object definition. This can be useful to protect your intellectual property or simply to prevent an object from being accidentally changed by a novice or unauthorized user.

With the Models view open, click on the model to protect and select the Protect button from the Edit ribbon. You will be asked to enter a password. When a project containing a protected object is loaded, the user will be prevented from viewing or accessing the definition of any protected objects unless they have the password. The object can still be instantiated normally in another model. To unprotect an object, select the object from within the Models view of the Project window and select Unprotect from the ribbon. Note that protection is at the object/model level. A project might contain multiple objects, some unprotected and others protected individually.



Set Protected Views - Professional and RPS Editions Only

For users with Professional Edition, RPS Edition, or Academic RPS licensing, Simio provides an additional feature to **Set Protected Views** for a model before it is password protected. By selecting the Set Protected Views button, the Protected View Setup dialog will open, as shown below. Initially, the view setup will appear blank, meaning that no views should be seen when the model is password protected. By selecting Default Views in the bottom left, all possible views within the selected model will be displayed. The user may then delete the views that should not be seen. Any views shown in the dialog will still be shown when a model is password protected.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Scheduling in Simio RPS

What is Risk-Based Planning and Scheduling (RPS)?

Risk-based Planning and Scheduling (RPS) is the next generation of Advanced Planning and Scheduling (APS) that is specifically designed to account for risk and uncertainty. It is the dual use of a simulation model to generate both a detailed resource-constrained deterministic schedule as well as a probability-based risk analysis of that schedule to account for variation in the system. RPS is used to generate schedules that minimize risks and reduce costs in the presence of uncertainty.

Traditional Advanced Planning and Scheduling (APS) systems generate schedules by assuming there is no variation or uncertainty in the system. A deterministic APS schedule quickly becomes obsolete as machines break, processes vary in time, material arrives late, etc. In addition, workers may be absent or perform poorly because they are sick or distracted. Although these variations are not included when the simulation model generates a plan, they directly impact the ability of your real system to meet your plan. APS schedules which appear initially feasible become infeasible over time as variation degrades performance. By ignoring variation APS schedules are optimistic by nature - they promise more than can be delivered. The user of traditional APS has no way to assess or mitigate the underlying risk inherent in the schedule. Risk-based Planning and Scheduling augments the deterministic schedule with risk measures that allow the decision maker to properly account for the underlying variation and uncertainty in the system.

RPS uses a purpose-built simulation model of the system to fully capture both the detailed constraints and variations in the system. It then uses this model in two ways. The first is to generate a detailed schedule/plan. In this case the model is executed in a purely deterministic mode; machines do not break, process times are always constant, materials arrive on time, etc. This is the optimistic view assumed by all APS systems and produces a deterministic plan/schedule. Once the schedule has been generated, RPS then replicates this same simulation model with variation added back into the system and performs a probabilistic analysis to estimate the underlying risks associated with the schedule.

In planning and scheduling applications, you often have targets that you wish to meet for individual transactions being processed by your system. In a production system, for example, you might have targets related to delivery dates for each individual order, as well as activity-based costing assigned to each order. A feasible plan or schedule is defined as one where all targets are met by the plan/schedule. In RPS, when you run the model, you generate your operational plan and this plan may or may not be feasible relative to the targets you have set. By then applying risk analysis, you can know the underlying risk associated with hitting each target that has been defined for each individual transaction that you are planning (Order-02, 47% probability of being on-time). This provides you with the ability to plan critical operations while fully accounting for the underlying risk imposed by variations in the system.

By providing up-front visibility into the inherent risk associated with a specific plan/schedule, RPS provides the necessary information to take early action in the operational plan to mitigate risks and reduce costs. RPS provides a realistic view of expected schedule performance. Specific alternatives such as overtime or expediting external material/components from suppliers can be compared in terms of their impact on both risks of meeting schedule targets, and costs of mitigating those risks, thereby providing a customer-satisfying operational strategy at a minimum cost. The interface provides a simple approach for changing such parameters, allowing these alternatives to be compared easily by the user.

RPS uses the validated Simio model of the system to simulate the flow of a specific set of jobs. The orders to be scheduled are downloaded to the RPS model from the ERP/MRP system. The same Simio simulation model that is used by RPS can also be used to analyze and improve the design of the underlying production system by randomly generating orders over a long planning horizon. For example, the model could be used to evaluate the long term impact on performance of capital equipment purchases, changes in process flows, or new product introductions. Hence the same simulation model can be used to both improve the facility design as well as provide the detailed model logic for planning/scheduling of day-to-day operations.

Simulation tools of the past are not designed or equipped to work well with the application of Risk-based Planning and Scheduling. Simio supports rapid modeling and has an easy and flexible interface to a wide range of enterprise data that is typically held in spreadsheets, data bases, or MRP/ERP/SCM systems. Simio also makes it easy to define and properly evaluate alternatives without requiring sophisticated modeling skills or knowledge of statistics. And finally, it allows a user to go beyond the traditional use of simulation for comparing alternative designs, to providing schedule risk analysis which will deliver information that could contribute to the overall success of an assembly plant.

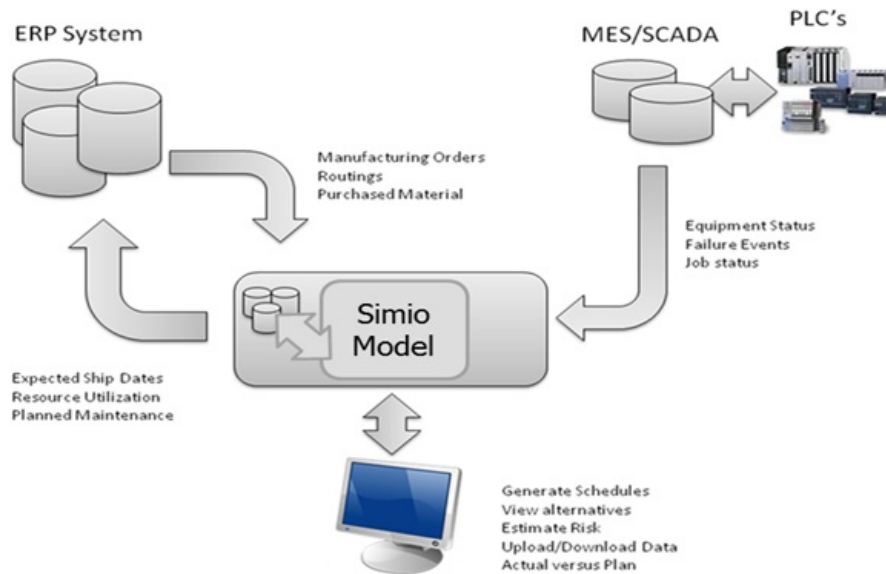
Simio Products

The Simio model that drives the RPS solution can be built using any of the Simio simulation products. However the Simio RPS Edition contains a number of unique features that are useful in building planning and scheduling solutions. The Simio RPS Edition encompasses all the functionality of the Simio Design/Professional Edition plus additional scheduling functionality; it is the ideal product for the internal/external consultant that is building a scheduling solution. The Simio Scheduling Edition is a runtime deployment platform for planning and scheduling. This product has no model building

functionality of its own, but uses a model built by the Simio RPS Edition. This is a streamlined product that is specifically designed for day to day use of the Simio model for planning and scheduling.

ERP Integration

The Simio RPS model is typically integrated with the existing ERP and MES/SCADA system to provide the data on the current status of the system and the actual jobs to be processed through the system. This data is provided to Simio in the form of relational data sets that are imported and held in memory by Simio for fast execution. These data sets typically contain data such as a list of jobs to be processed, a bill of material for each job, job routings including setup and processing times, purchased material, etc, along with the status of all jobs in the system at the start of the planning period. Simio does not have a fixed data schema – the data sets used by Simio can be configured as needed to match the form of the external data. The flow of jobs as defined by the data sets is then simulated using the Simio model to generate a detailed schedule. A schematic of a typical implementation is shown below:

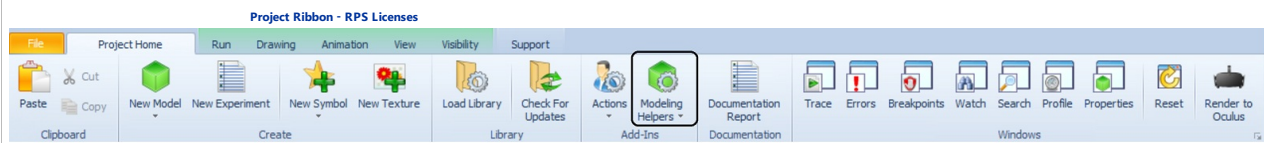


Once the data is imported from the ERP system, a schedule is generated by simulating the flow of jobs using the Simio model with no variation. The simulation is then replicated multiple times with variation incorporated to generate probabilistic risk measures for the associated schedule. The risk measures include the probability of meeting user-defined targets, as well as expected, pessimistic, and optimistic schedule performance. Various built-in analytical tools like Gantt charts, resource and entity activity tracking, root cause analysis, optimization, and even 3D animation can be used to analyze and reduce the risk and improve the performance. The improved schedule and associated risk analysis may then be exported back to the ERP system or deployed directly from Simio, if desired.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RPS Modeling Helpers



RPS users can author design-time "Modeling Helper" Add-Ins. These are pieces of code that are "bound" to a particular model.

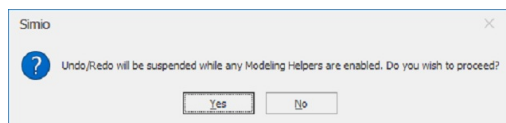
User starts implementation with the `SimioAPI.Extensions.IModelHelperAddIn` interface.

- Add-Ins are loaded with the model, and unloaded when the model is closed.
- Add-Ins have access to the full IModel design time API.
- Add-Ins can subscribe to events on the model (right now only save).
- Add-Ins can 'augment' the model's property display and put their own properties in there as well.
- Add-Ins specify an 'environment' under which they are valid to run (currently ONLY selection is 'Desktop').
- Add-Ins are displayed in the Project Home ribbon under the Modeling Helper pull-down.

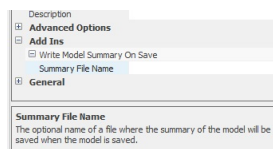
A normal Simio install will have no model helpers installed along with it.

By selecting an icon in the drop list, the user enables (or disables) that Add-In for the active project.

Once ANY Add-Ins are enabled, that will disable Undo/Redo. Simio will make sure users are aware of this:



Enabled Add-Ins can optionally add their own 'properties' alongside the model's properties:



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

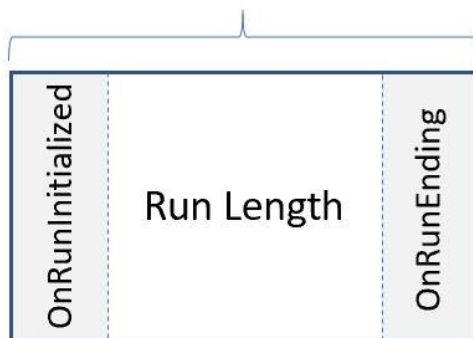
Single-Pass vs Multi-Pass Simulation Approach

Single-Pass vs Multi-Pass Simulation Approach - RPS Licenses Only

Typically, creating a resource schedule or generating the results for a scenario replication instance (experiment mode) requires only one run of the simulation model. A model run is initialized, is executed for some run length and then ended, completing the Create Plan or Run a Replication action.

Single-Pass Simulation Approach

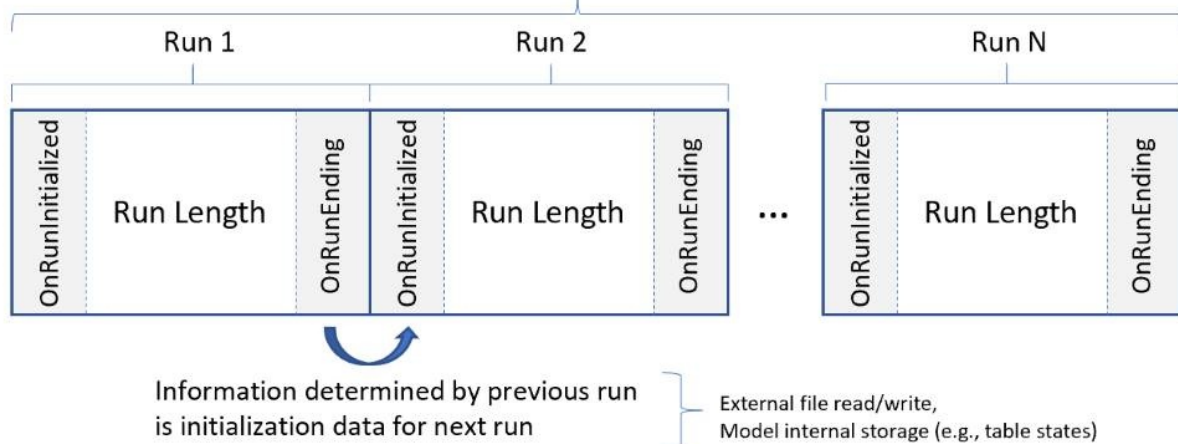
Create Plan/Run a Replication



There are cases however where it is insufficient to run the model only once to create a schedule or generate results for a scenario replication instance. Instead, the model must be run multiple times in succession, with information determined by previous runs used as input data for the next run. Simulation runs are continued in succession until the desired objective is achieved. This type of simulation approach is often referred to as a multi-pass approach.

Multi-Pass Simulation Approach

Create Plan/Run a Replication



Simio provides the following features to support a multi-pass simulation approach:

RestartRun Step - The RestartRun step may be used to set the ending time of the simulation run to the current time. This will cause the run to end once all simulation events scheduled for the current time have been processed. Interactive mode will require the next run to be started manually. Otherwise, a new run will be automatically started, treated as a restarted run for the same scenario replication instance if running in experiment mode.

Run.RunNumber Function - Returns a one-based run count, incremented if a run restart occurs using the RestartRun step. For example, if executing a simulation with run restarts, then for the first run you'd see Run.RunNumber return 1. For the next run if restarted using the RestartRun step, you'd see Run.RunNumber return 2. And so forth. An example of using the Run.RunNumber function in process logic might be in a condition expression to decide whether to execute a RestartRun step. For example, if implementing a two-pass algorithm, then you'd only want to execute the RestartRun step if 'Run.RunNumber=1'.

Table States – *Keep Values Between Runs* Option - For any [State Column](#) in a Data Table that is not an object or element reference state type, or any [Output Table](#), a *Keep Values Between Runs* option will be provided indicating whether to keep the state column's values from the previous run if a new run is started using the RestartRun step.

Note: Automatic data imports only import on the first run of a multi-pass simulation.

Note: Automatic data exports only export on the final run of a multi-pass simulation.

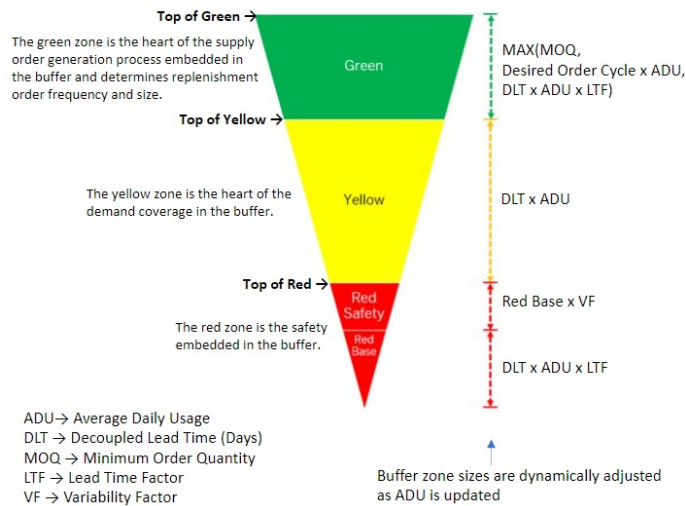
[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Demand Driven Material Requirements Planning

DDMRP Overview

Demand driven material requirements planning, also known as "DDMRP", revolves around strategically placed stock positions called buffers that decouple demand from supply. These inventory buffers are comprised of three color-coded zones; Green, Yellow, and Red. Each zone serves a specific purpose and is sized as illustrated below.



Average Daily Usage (ADU) is the calculated rate of use for each specific part. Length of Period, Frequency of Update, Past, Forward, or Blended, and other exceptions need to be considered when calculating ADU.

Decoupled Lead Time (DLT) is a qualified cumulative lead time concept in DDMRP. It is an intermediate lead time that assumes stock at decoupling point buffers is available and is defined by Ptak and Smith as "the longest cumulative coupled lead time chain in a manufactured item's product structure. It is a form of cumulative lead time but is limited and defined by the placement of decoupling points within a product structure."

Minimum Order Quantity (MOQ) is the fewest number of units required to be ordered at one time.

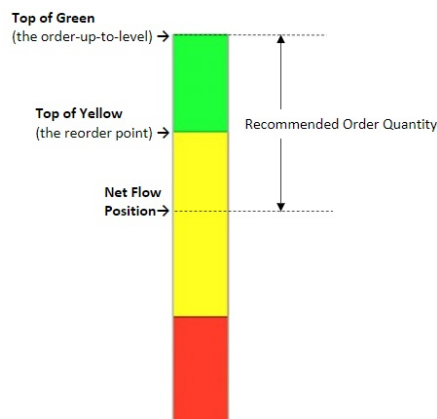
Lead Time Factor (LTF) is broken up into three recommended lead time factor ranges assigned to different lead time categories:

- For a long lead time, a LTF of 20%-40% ADU x DLT is recommended
- For a medium lead time, a LTF of 41%-60% ADU x DLT is recommended
- For a short lead time, a LTF of 61%-100% ADU x DLT is recommended

Variability Factor (VF) is also broken up into three recommended variability factor ranges assigned to different variability rates:

- For high variability, a VF of 61%-100%+ of the safety base is recommended
- For medium variability, a VF of 41%-60% of the safety base is recommended
- For low variability, a VF of 0%-40% of the safety base is recommended

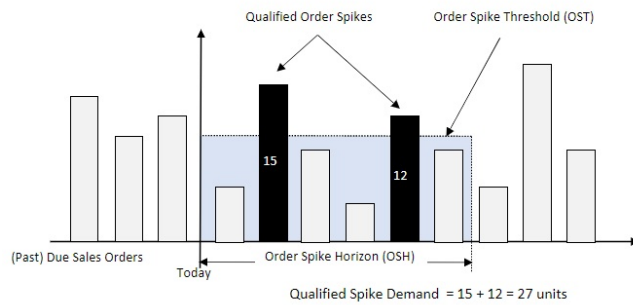
Replenishment order planning in DDMRP uses an inventory buffer's *Net Flow Position* and green zone to determine when to reorder and how much. If the position is at or below the top of the yellow zone, then a reorder quantity is recommended to return the net flow position to the top of the green zone (see figure below).



Net flow position = On-hand + On-order - Qualified sales order demand

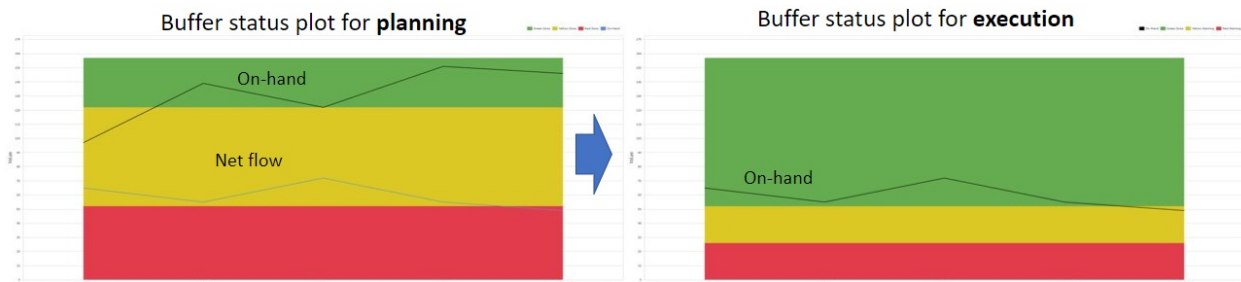
Qualified sales order demand = Orders past due + Orders due today + Qualified order spikes

Where a qualified order spike is a quantity of known cumulative daily demand that is within a qualifying time window (order spike horizon) and over a specified threshold (order spike threshold).



DDMRP Buffer Status Alerts

In DDMRP, a careful distinction is made between planning (supply order generation) and execution (supply order priority management). Execution focuses on current and projected on-hand position instead of net flow position to evaluate buffer status.



DDMRP *buffer status alerts* clearly communicate priorities to suppliers as opposed to conventional MRP's due date driven approaches which might misrepresent the proper order sequencing.

Order Priority by Due Date (MRP)

Order #	Due Date	Order Type	Customer
MO 831145	05/12	MTS	Internal
MO 821158	05/12	MTS	Internal
MO 831162	05/12	MTS	Internal
MO 845172	05/13	MTS	Internal
MO 645181	05/14	MTS	Internal

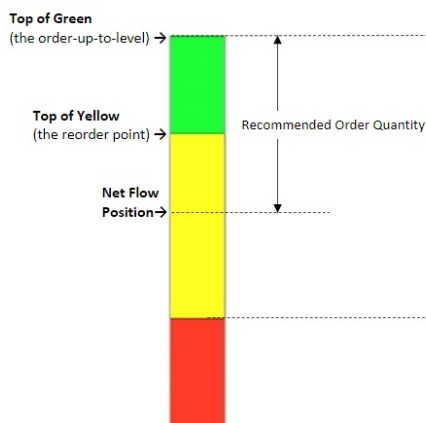
VS

Order Priority by Buffer Status (DDMRP)

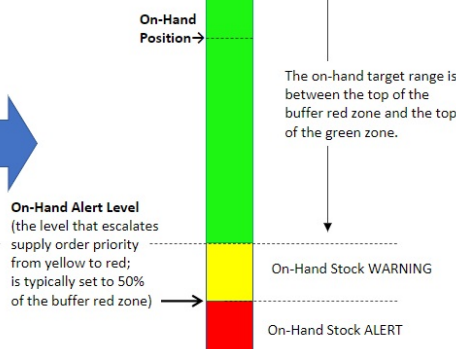
Order #	Due Date	Buffer Status	Order Type	Customer
MO 645181	05/14	RED - 13.2%	MTS	Internal
MO 845172	05/13	RED - 36.2%	MTS	Internal
MO 831162	05/12	YELLOW - 62.1%	MTS	Internal
MO 831145	05/12	YELLOW - 72.1%	MTS	Internal
MO 821158	05/12	YELLOW - 87.2%	MTS	Internal

Status information indicating whether the on-hand position is in the buffer's target range (green), in the warning range (yellow), or below the alert level (red). Penetration of the buffer's embedded safety is quantified based on the on-hand level as a percentage of the red zone [% = (On-Hand / Red Zone Size) * 100]. The lower the percentage, the less safety remains and the higher the supply order priority.

Buffer Zones (Planning)



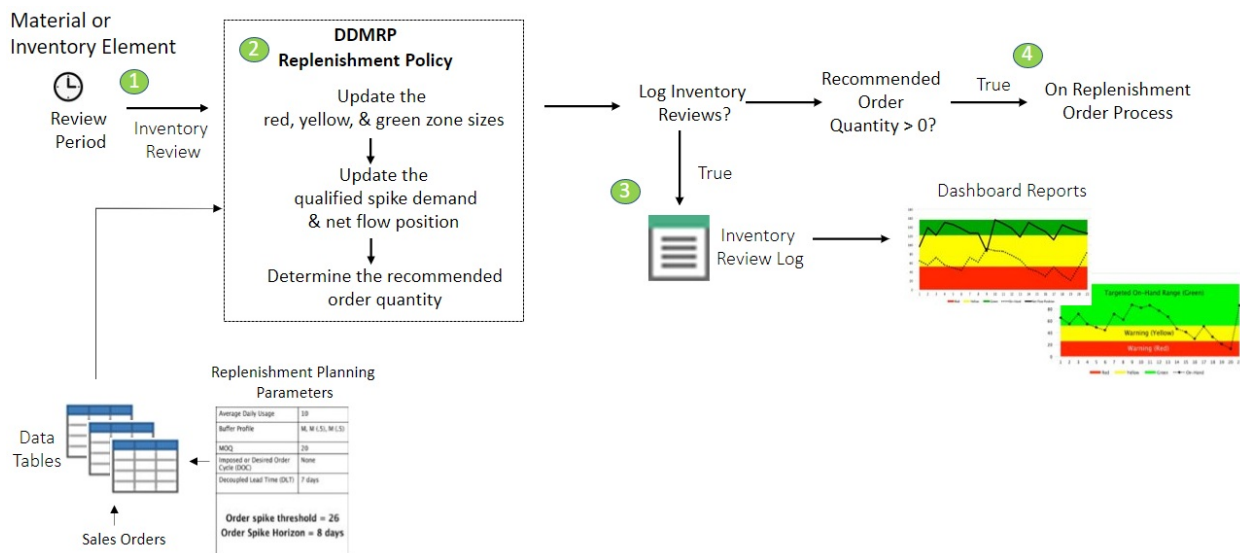
Buffer Status (Execution)



Buffer Status

Green – 144%

DDMRP in Simio



1. The *Review Period* of a Material or Inventory element determines when inventory reviews take place (e.g., daily or weekly). Each inventory review is scheduled as a latest priority event on the current events calendar which allows for custom calculations beforehand.
2. Replenishment planning is performed using the 'DDMRP' *Replenishment Policy*. This policy recalculates and updates the inventory's buffer zone sizes and net flow position using specified *Red Zone Size*, *Yellow Zone Size*, *Green Zone Size*, and *Qualified Spike Demand* expressions. Pertinent input for evaluating those expressions such as replenishment planning parameters or sales orders is typically defined in data tables. If the net flow position is at or below the top of the yellow zone, then a reorder quantity is recommended to return the net flow position to the top of the green zone.
3. The Inventory Review Log (RPS Edition Only) may optionally be used as a data source for visualizing inventory buffer levels and performance in dashboard reports.
4. The *On Replenishment Order Process* specified for the Material or Inventory element is executed to handle a replenishment order. The created process token has a reference to the material order detail.

Both the Inventory and Material elements contain the same properties when selecting a *Replenishment Policy* of 'Demand-Driven MRP'.

Listed below are the DDMRP properties of **Inventory and Material**:

Property	Description
Red Zone Size	The expression used to calculate and update the buffer red zone size. The red zone is the safety embedded in the buffer.
Yellow Zone Size	The expression used to calculate and update the buffer yellow zone size. The top of the yellow zone is the minimum threshold for the net flow position that signals the need to place a replenishment order.
Green Zone Size	The expression used to calculate and update the buffer green zone size. The top of the green zone is the target level to return the net flow position to if at or below the top of the yellow zone.
Qualified Spike Demand	The expression used to calculate and update the qualified spike demand value for calculating the net flow position.

Note: If a Material element is location-based inventory then the above functions return sums across all site locations.

DDMRP Calculators

Simio offers the following four calculators for demand-driven material requirements planning (DDMRP):

Average Daily Usage Calculator	Calculates the average daily usage values for inventory items.
Decoupled Lead Times Calculator	Calculates the decoupled lead times for inventory items.
Buffer Zone Sizes Calculator	Calculates the buffer red, yellow, and green zone sizes for inventory items identified as decoupling points.
Qualified Spike Demand Calculator	Calculates the qualified spike demand values for inventory items identified as decoupling points.

The DDMRP calculators are typically run in sequence as shown in Figure 1 and are designed to work with a specific set of data table schemas for inputs and outputs.

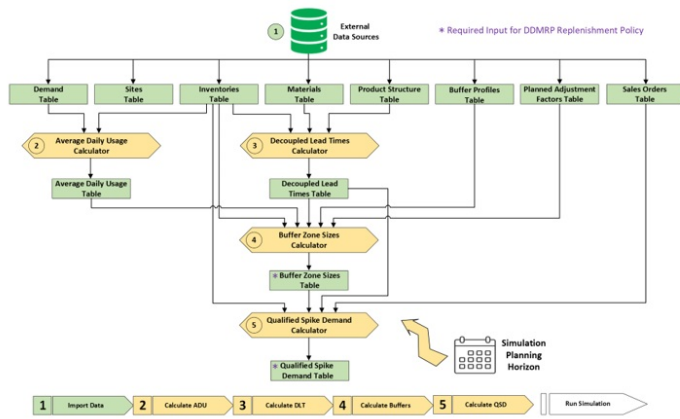
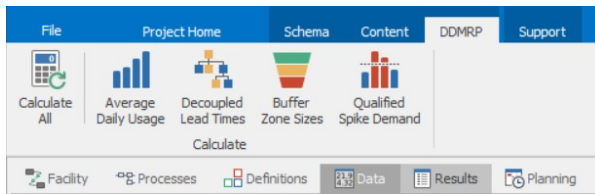


Figure 1: Simulation Input Data Preparation Using DDMRP Calculators



The DDMRP buttons live in the Data tab, specifically the Tables view. The DDMRP ribbon will only appear once a DDMRP Data Connector has been bound to a Data Table and the Data Table has been named to match the name of the calculator. Each respective button will run the specific calculator for that aspect of DDMRP. You can run all with the Calculate All button and they will run in the following order:

1. Average Daily Usage
2. Decoupled Lead Times
3. Buffer Zone Sizes
4. Qualified Spike Demand

To enable a calculator, you must first set up a Data Connector of that type and set the active import binding.

Average Daily Usage Calculator

The Average Daily Usage Calculator may be used to calculate the average daily usage values for inventory items.

Inputs and Outputs

Figure 2 shows the inputs and outputs of the Average Daily Usage Calculator.

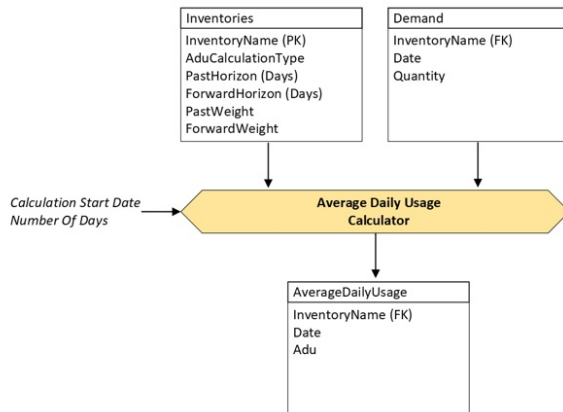


Figure 2: Average Daily Usage Calculator Inputs and Outputs

Properties

The Average Daily Usage Calculator provides the following properties as calculation parameters:

Property Name	Description
Calculation Start Date	The start date of the calculation horizon. Enter the string representation of a date using current locale formatting conventions. Or, alternatively, use the syntax \$(property:DateTimePropertyName) to interpolate the date string from a specified DateTime property defined on the model. If left unspecified then defaults to the simulation model's start date.
Number Of Days	The number of days forward to calculate the average daily usage values. Enter an integer value. Or, alternatively, use the syntax \$(property:IntegerPropertyName) to interpolate the number of days from a specified Integer property defined on the model. If left unspecified then defaults to the simulation model's run length rounded up to the nearest integer number of days.

Data Table Schemas

The Average Daily Usage Calculator uses Inventories, Demand, and Average Daily Usage data table schemas for inputs

and outputs.

Inventories Table

The Inventories table contains information describing the inventory items.

The table properties used for inputs to the Average Daily Usage Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Inventory Element Reference Property (Primary Key)	The inventory identifier.
AduCalculationType	String or List Property returning literal string values 'None', 'Past', 'Forward', or 'Blended'	The method used for ADU calculation. None – ADU not calculated. Past – ADU calculated based on historical usage. Forward – ADU calculated based on forecasted usage. Blended – ADU calculated based on a combination of historical usage and forecasted usage.
PastHorizon	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	The integer number of days backward from the calculation start date to consider historical demand.
ForwardHorizon	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	The integer number of days forward from the calculation start date (including that date) to consider forecasted demand.
PastWeight	Real Property	The weight to apply to the past ADU when a blended ADU is calculated.
ForwardWeight	Real Property	The weight to apply to the forward ADU when a blended ADU is calculated.

Demand Table

The Demand table contains the historical and forecasted demand quantities used to calculate average daily usage values for inventory items.

The table properties used for inputs to the Average Daily Usage Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
Date	DateTime Property	The date to which the historical or forecasted demand quantity applies.
Quantity	Real Property	The historical or forecasted demand quantity.

Average Daily Usage Table

The Average Daily Usage table contains the calculated average daily usage values for inventory items.

The table properties used to store outputs from the Average Daily Usage Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
Date	DateTime Property	The date to which the calculated ADU applies.
Adu	Real Property	The calculated past, forward, or blended ADU.

Remarks and Examples

Missing dates in the Demand table are treated as zeros.

Duplicate dates for an inventory item in the Demand table are treated as the sum of the quantity values.

A past ADU is calculated using the following equation:

$$\text{Past ADU} = \frac{\text{Sum of Past Horizon Demand}}{\text{Past Horizon}}$$

A forward ADU is calculated using the following equation:

$$\text{Forward ADU} = \frac{\text{Sum of Forward Horizon Demand}}{\text{Forward Horizon}}$$

A blended ADU is calculated using the following weighted average equation:

$$\text{Blended ADU} = \frac{(\text{Past ADU} * \text{Past Weight}) + (\text{Forward ADU} * \text{Forward Weight})}{\text{Past Weight} + \text{Forward Weight}}$$

Figure 3 shows an example of calculating blended ADU.

Calculation Start Date: 2/10/2023
 Number Of Days: 2
 Past Horizon (Days): 3
 Forward Horizon (Days): 4
 Past Weight: 0.6
 Forward Weight: 0.4

Demand Date	Quantity
2/5/2023	10
2/6/2023	11
2/7/2023	13
2/8/2023	20
2/9/2023	16
2/10/2023	12
2/11/2023	10
2/12/2023	15
2/13/2023	10
2/14/2023	15

2/10/2023

$$\text{Past ADU} = (13 + 20 + 16)/3 = 16.33$$

$$\text{Forward ADU} = (12 + 10 + 15 + 10)/4 = 11.75$$

$$\text{Blended ADU} = \frac{(16.33 \times 0.6) + (11.75 \times 0.4)}{0.6 + 0.4} = 14.5$$

2/11/2023

$$\text{Past ADU} = (20 + 16 + 12)/3 = 16$$

$$\text{Forward ADU} = (10 + 15 + 10 + 15)/4 = 12.5$$

$$\text{Blended ADU} = \frac{(16 \times 0.6) + (12.5 \times 0.4)}{0.6 + 0.4} = 14.6$$

Figure 3: Blended ADU Calculation Example

Decoupled Lead Times Calculator

The Decoupled Lead Times Calculator may be used to calculate the decoupled lead times for inventory items.

Inputs and Outputs

Figure 4 shows the inputs and outputs of the Decoupled Lead Times Calculator.

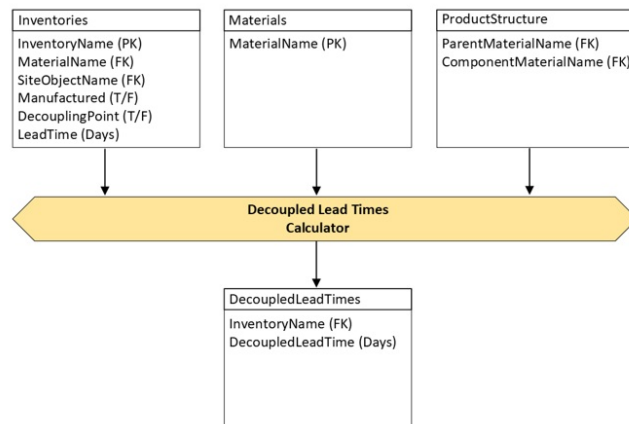


Figure 4: Decoupled Lead Times Calculator Inputs and Outputs

Properties

The Decoupled Lead Times Calculator has no properties (calculation parameters).

Data Table Schemas

The Decoupled Lead Times Calculator uses Inventories, Materials, Product Structure, and Decoupled Lead Times data table schemas for inputs and outputs:

Inventories Table

The Inventories table contains information describing the inventory items.

The table properties used for inputs and to store outputs from the Decoupled Lead Times Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Inventory Element Reference Property (Primary Key)	The inventory identifier.
MaterialName	Foreign Key Property to Materials.MaterialName	The material identifier.
SiteObjectName	Foreign Key Property to Sites.SiteObjectName	The site object identifier.
Manufactured	Boolean Property	Indicates whether the inventory is manufactured (is in-house production).
DecouplingPoint	Boolean Property	Indicates whether the inventory is a decoupling point (is buffered).
LeadTime	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	The manufacturing, purchasing, or transfer lead time in integer days.

Materials Table

The Materials table contains information describing the raw materials used in production as well as the goods produced that are available for sale.

The table properties used for inputs to the Decoupled Lead Times Calculator are as follows:

Property Name	Property Type	Description
MaterialName	Material Element Reference Property (Primary Key)	The material identifier.

Product Structure Table

The Product Structure table contains information describing the hierarchical decomposition (parent-component

relationships) of manufactured materials.

The table properties used for inputs to the Decoupled Lead Times Calculator are as follows:

Property Name	Property Type	Description
ParentMaterialName	Foreign Key Property to Materials.MaterialName	The parent material identifier.
ComponentMaterialName	Foreign Key Property to Materials.MaterialName	The component material identifier.

Decoupled Lead Times Table

The Decoupled Lead Times table contains the calculated decoupled lead time values for inventory items.

The table properties used to store outputs from the Decoupled Lead Times Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
DecoupledLeadTime	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	The calculated decoupled lead time in integer days, defined as the longest cumulative coupled lead time chain in the product structure if the inventory is manufactured.

Remarks and Examples

Decoupled lead time (DLT) is a qualified cumulative lead time concept in DDMRP. For manufactured items, it can be defined as:

The longest cumulative coupled lead time chain in the product structure, limited and defined by the placement of decoupling point buffers within that structure.

Any manufactured item with at least one coupled component will always have a longer decoupled lead time than its manufacturing lead time. If an inventory item is not manufactured (i.e., is replenished by purchase or stock transfer orders) then the decoupled lead time is simply the purchasing or transfer lead time.

The relationship between the Inventories table and the Decoupled Lead Times table is a one-to-one relationship.

Figure 5 shows an example of calculating decoupled lead times.

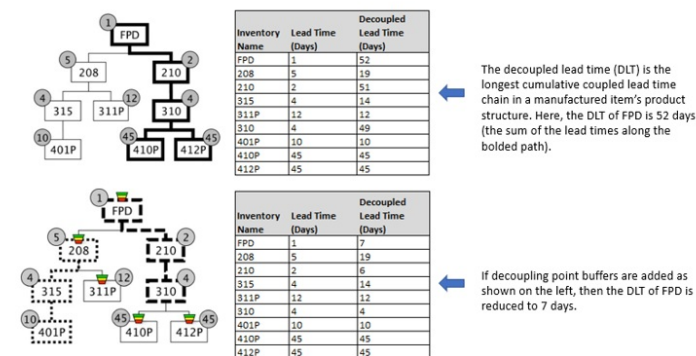


Figure 5: Decoupled Lead Times Calculation Example

Buffer Zone Sizes Calculator

The Buffer Zone Sizes Calculator may be used to calculate the buffer red, yellow, and green zone sizes for inventory items identified as decoupling points.

Inputs and Outputs

Figure 6 shows the inputs and outputs of the Buffer Zone Sizes Calculator.

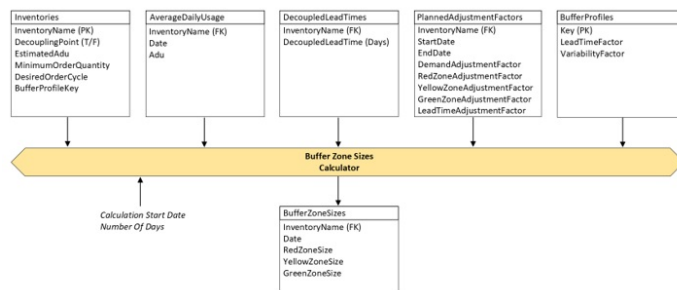


Figure 6: Buffer Zone Sizes Calculator Inputs and Outputs

Properties

The Buffer Zone Sizes Calculator provides the following properties as calculation parameters:

Property Name	Description
Calculation Start Date	The start date of the calculation horizon. Enter the string representation of a date using current locale formatting conventions. Or, alternatively, use the syntax \$(property:DateTimePropertyName) to interpolate the date string from a specified DateTime property defined on the model. If left unspecified then defaults to the simulation model's start date.
Number	The number of days forward to calculate the buffer zone sizes. Enter an integer value. Or,

Of Days alternatively, use the syntax `$(property:IntegerPropertyName)` to interpolate the number of days from a specified Integer property defined on the model. If left unspecified then defaults to the simulation model's run length rounded up to the nearest integer number of days.

Data Table Schemas

The Buffer Zone Sizes Calculator uses Inventories, Average Daily Usage, Decoupled Lead Times, Buffer Profiles, Planned Adjustment Factors, and Buffer Zone Sizes data table schemas for inputs and outputs.

Inventories Table

The Inventories table contains information describing the inventory items.

The table properties used for inputs to the Buffer Zone Sizes Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Inventory Element Reference Property (Primary Key)	The inventory identifier.
DecouplingPoint	Boolean Property	Indicates whether the inventory is a decoupling point (is buffered).
EstimatedAdu	Real Property	The default ADU used for a buffer zone size calculation if an ADU is not otherwise defined.
MinimumOrderQuantity	Real Property	A minimum order quantity that if significant determines the buffer green zone size.
DesiredOrderCycle	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	A desired integer number of days between replenishment orders that if significant determines the buffer green zone size.
BufferProfileKey	Foreign Key Property to BufferProfiles.Key	The buffer profile identifier.

Average Daily Usage Table

The Average Daily Usage table contains the calculated average daily usage values for inventory items.

The table properties used for inputs to the Buffer Zone Sizes Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
Date	DateTime Property	The date to which the calculated ADU applies.
Adu	Real Property	The calculated past, forward, or blended ADU.

Decoupled Lead Times Table

The Decoupled Lead Times table contains the calculated decoupled lead time values for inventory items.

The table properties used to store inputs to the Buffer Zone Sizes Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
DecoupledLeadTime	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	The calculated decoupled lead time in integer days, defined as the longest cumulative coupled lead time chain in the product structure if the inventory is manufactured.

Planned Adjustment Factors Table

The Planned Adjustment Factors table contains demand, zonal, or lead time adjustment factors for manipulating buffer zone sizes at specific points in time.

The table properties used for inputs to the Buffer Zone Sizes Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
StartDate	DateTime Property	The start of the date range to apply the set of demand, zonal, and lead time adjustment factors.
EndDate	DateTime Property	The end of the date range to apply the set of demand, zonal, and lead time adjustment factors.
DemandAdjustmentFactor	Real Property	Multiplicative factor that adjusts the ADU input.
RedZoneAdjustmentFactor	Real Property	Multiplicative factor that adjusts the buffer red zone size.
YellowZoneAdjustmentFactor	Real Property	Multiplicative factor that adjusts the buffer yellow zone size.
GreenZoneAdjustmentFactor	Real Property	Multiplicative factor that adjusts the buffer green zone size.
LeadTimeAdjustmentFactor	Real Property	Multiplicative factor that adjusts the decoupled lead time input.

Buffer Profiles Table

The Buffer Profiles table contains information describing the buffer profile settings for inventory items with similar lead time, variability, and order management characteristics.

The table properties used for inputs to the Buffer Zone Sizes Calculator are as follows:

Property Name	Property Type	Description
Key	String Property (Primary Key)	The buffer profile identifier.
LeadTimeFactor	Real Property	A multiplier based on lead time categorization (e.g., long, medium, or short) that is used for buffer zone size calculations.
VariabilityFactor	Real Property	A multiplier based on variability categorization (e.g., high, medium, or low) that is used for buffer zone size calculations.

Buffer Zone Sizes Table

The Buffer Zone Sizes table contains the calculated buffer red, yellow, and green zone sizes for inventory items identified as decoupling points.

The table properties used to store outputs from the Buffer Zone Sizes Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
Date	DateTime Property	The date to which the calculated buffer zone sizes apply.
RedZoneSize	Real Property	The calculated buffer red zone size.
YellowZoneSize	Real Property	The calculated buffer yellow zone size.
GreenZoneSize	Real Property	The calculated buffer green zone size.

Remarks and Examples

Missing dates in the Average Daily Usage table are treated as the inventory item's estimated ADU.

Duplicate dates for an inventory item in the Average Daily Usage table are treated as the sum of the ADU values.

Missing values in the Decoupled Lead Times table are treated as decoupled lead times of zero.

Multiple values for an inventory item in the Decoupled Lead Times table are summed.

If there are multiple rows for an inventory item in the Planned Adjustment Factors table that apply to a date then the last applicable row is used for the buffer zone size calculations.

The equations used to calculate the buffer zone sizes are as follows:

$ADU = ADU * Demand\ Adjustment\ Factor$

$DLT = DLT * Lead\ Time\ Adjustment\ Factor$

$Red\ Zone\ Base = ADU * DLT * Lead\ Time\ Factor$

$Red\ Zone\ Safety = Red\ Zone\ Base * Variability\ Factor$

$Red\ Zone\ Size = (Red\ Zone\ Base + Red\ Zone\ Safety) * Red\ Zone\ Adjustment\ Factor$

$Yellow\ Zone\ Size = ADU * DLT * Yellow\ Zone\ Adjustment\ Factor$

$Green\ Zone\ Size = MAX(Minimum\ Order\ Quantity, ADU * Desired\ Order\ Cycle, ADU * DLT * Lead\ Time\ Factor) * Green\ Zone\ Adjustment\ Factor$

The calculated buffer red, yellow, and green zone sizes are rounded to the nearest integer value.

Figure 7 shows an example of calculating buffer zone sizes.

Calculation Start Date: 1/17/2023 Number Of Days: 2 Estimated ADU: 250 Decoupled Lead Time (Days): 20 Minimum Order Quantity: 250 Desired Order Cycle (Days): 3 Lead Time Factor: 0.25 Variability Factor: 0.5	1/17/2023 $ADU = 250$ $DLT = 20$ $Red\ Zone\ Base = ADU * DLT * LTF: 0.25 = 1250$ $Red\ Zone\ Safety = Red\ Zone\ Base * VF: 0.5 = 625$ $Red\ Zone\ Size = 1250 + 625 = 1875$ $Yellow\ Zone\ Size = ADU * DLT * 20 = 5000$ $Green\ Zone\ Size = MAX(MOQ: 250, DOC: 3 * ADU: 250, ADU * DLT * LTF: 0.25) = 1250$
Planned Adjustment Factors For 1/18/2023 <ul style="list-style-type: none"> • Demand Adjustment Factor: 1.2 • Red Zone Adjustment Factor: 0.75 • Yellow Zone Adjustment Factor: 0.5 • Green Zone Adjustment Factor: 0.8 • Lead Time Adjustment Factor: 1.5 	1/18/2023 $ADU = 250 * Demand\ Adjustment\ Factor: 1.2 = 300$ $DLT = 20 * Lead\ Time\ Adjustment\ Factor: 1.5 = 30$ $Red\ Zone\ Base = ADU * DLT * LTF: 0.25 = 2250$ $Red\ Zone\ Safety = Red\ Zone\ Base * VF: 0.5 = 1125$ $Red\ Zone\ Size = (2250 + 1125) * Red\ Zone\ Adjustment\ Factor: 0.75 = 2531$ $Yellow\ Zone\ Size = ADU * DLT * 30 * Yellow\ Zone\ Adjustment\ Factor: 0.5 = 4500$ $Green\ Zone\ Size = MAX(MOQ: 250, DOC: 3 * ADU: 300, ADU * DLT * LTF: 0.25) = 1800$

Figure 7: Buffer Zone Sizes Calculation Example

Qualified Spike Demand Calculator

The Qualified Spike Demand Calculator may be used to calculate the qualified spike demand values for inventory items identified as decoupling points.

Inputs and Outputs

Figure 8 shows the inputs and outputs of the Qualified Spike Demand Calculator.

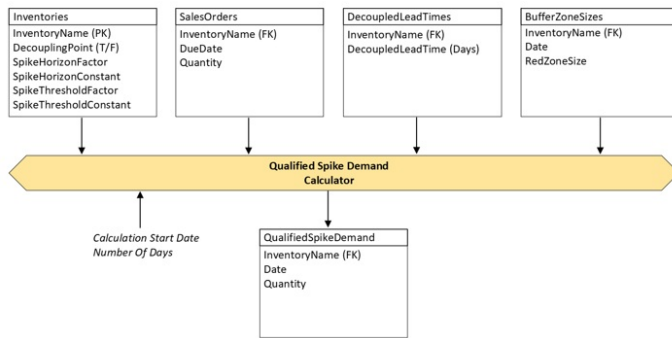


Figure 8: Qualified Spike Demand Calculator Inputs and Outputs

Properties

The Qualified Spike Demand Calculator provides the following properties as calculation parameters:

Property Name	Description
Calculation Start Date	The start date of the calculation horizon. Enter the string representation of a date using current locale formatting conventions. Or, alternatively, use the syntax <code>\$(property:DateTimePropertyName)</code> to interpolate the date string from a specified <code>DateTime</code> property defined on the model. If left unspecified then defaults to the simulation model's start date.
Number Of Days	The number of days forward to calculate the qualified spike demand values. Enter an integer value. Or, alternatively, use the syntax <code>\$(property:IntegerPropertyName)</code> to interpolate the number of days from a specified <code>Integer</code> property defined on the model. If left unspecified then defaults to the simulation model's run length rounded up to the nearest integer number of days.

Data Table Schemas

The Qualified Spike Demand Calculator uses Inventories, Sales Orders, Decoupled Lead Times, Buffer Zone Sizes, and Qualified Spike Demand data table schemas for inputs and outputs:

Inventories Table

The Inventories table contains information describing the inventory items.

The table properties used for inputs to the Qualified Spike Demand Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Inventory Element Reference Property (Primary Key)	The inventory identifier.
DecouplingPoint	Boolean Property	Indicates whether the inventory is a decoupling point (is buffered).
DecoupledLeadTime	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	The calculated decoupled lead time in integer days, defined as the longest cumulative coupled lead time chain in the product structure if the inventory is manufactured.
SpikeHorizonFactor	Real Property	Multiplier applied to the inventory's decoupled lead time to determine the order spike horizon. Order Spike Horizon = (DLT x Spike Horizon Factor) + Spike Horizon Constant
SpikeHorizonConstant	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	A constant integer number of days that is summed with the product of the spike horizon factor and decoupled lead time to determine the order spike horizon. Order Spike Horizon = (DLT x Spike Horizon Factor) + Spike Horizon Constant
SpikeThresholdFactor	Real Property	Multiplier applied to the inventory's buffer red zone size to determine the order spike threshold. Order Spike Threshold = (Red Zone Size x Spike Threshold Factor) + Spike Threshold Constant
SpikeThresholdConstant	Real Property	A constant number of units that is summed with the product of the spike threshold factor and buffer red zone size to determine the order spike threshold. Order Spike Threshold = (Red Zone Size x Spike Threshold Factor) + Spike Threshold Constant

Sales Orders Table

The Sales Orders table contains the qualified sales orders for inventory items.

The table properties used for inputs to the Qualified Spike Demand Calculator are as follows:

Column Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
DueDate	DateTime Property	The sales order due date.
Quantity	Real Property	The sales order quantity.

Decoupled Lead Times Table

The Decoupled Lead Times table contains the calculated decoupled lead time values for inventory items.

The table properties used to store inputs to the Qualified Spike Demand Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.

DecoupledLeadTime	Real Property with Unit Type set to 'Time' and Default Units set to 'Days'	The calculated decoupled lead time in integer days, defined as the longest cumulative coupled lead time chain in the product structure if the inventory is manufactured.
-------------------	--	--

Buffer Zone Sizes Table

The Buffer Zone Sizes table contains the calculated buffer red, yellow, and green zone sizes for inventory items identified as decoupling points.

The table properties used for inputs to the Qualified Spike Demand Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
Date	DateTime Property	The date to which the calculated buffer zone sizes apply.
RedZoneSize	Real Property	The calculated buffer red zone size.

Qualified Spike Demand Table

The Qualified Spike Demand table contains the calculated qualified spike demand values for inventory items identified as decoupling points.

The table properties used to store outputs from the Qualified Spike Demand Calculator are as follows:

Property Name	Property Type	Description
InventoryName	Foreign Key Property to Inventories.InventoryName	The inventory identifier.
Date	DateTime Property	The date to which the calculated qualified spike demand quantity applies.
Quantity	Real Property	The calculated qualified spike demand quantity.

Remarks and Examples

Missing values in the Decoupled Lead Times table are treated as decoupled lead times of zero.

Multiple values for an inventory item in the Decoupled Lead Times table are summed.

Missing dates in the Buffer Zones Sizes table are treated as red zone sizes of zero.

Duplicate dates for an inventory item in the Buffer Zones Sizes table are treated as the sum of the red zone sizes.

Qualified spike demand is a quantity of known cumulative daily demand within a qualifying future time window (*an order spike horizon*) and over a qualifying level (*an order spike threshold*) that is included in the DDMRP net flow equation used to provide a replenishment order recommendation signal.

The order spike horizon (a future time window starting 'tomorrow') is calculated using the following equation:

Order Spike Horizon (OSH)=(DLT*Spike Horizon Factor)+Spike Horizon Constant

The OSH is rounded to the nearest integer value.

The order spike threshold is calculated using the following equation:

Order Spike Threshold (OST)=(Red Zone Size*Spike Threshold Factor)+Spike Threshold Constant

The calculated qualified spike demand values are rounded to the nearest integer value.

Figure 9 shows an example of calculating qualified spike demand.

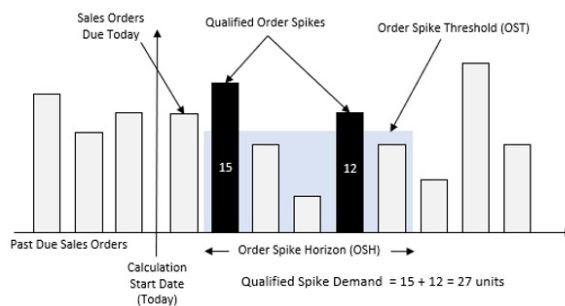


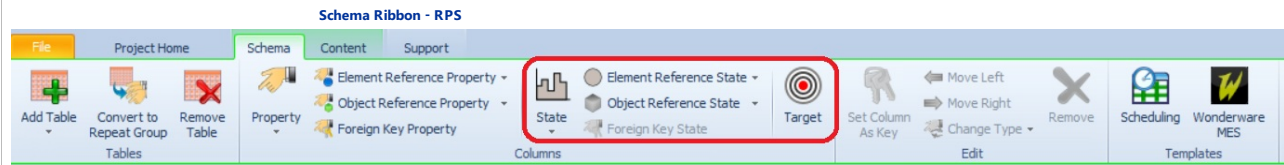
Figure 9: Qualified Spike Demand Calculation Example

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RPS Data Window

With the RPS product, there are two additional sections of the Schema ribbon that are available within the Data window, Tables panel. These include [States](#) and [Targets](#). Additionally, RPS users have the ability to add one or more [Output Tables](#) to the model.



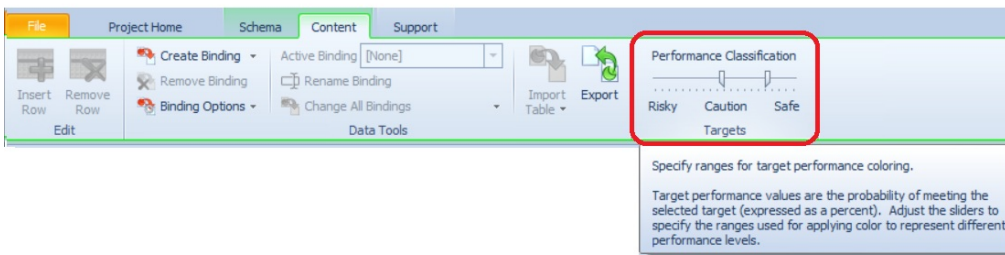
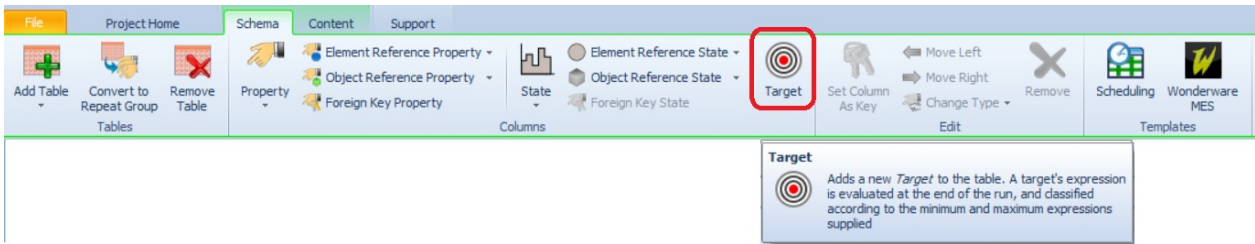
When defining a data table, many different type of columns are available from the Schema ribbon, including standard properties, element reference properties and object reference properties, as seen above. These allow the user to define a column as an expression, real, entity, transporter, tally statistic and so on. The data that is entered within each row of that column must be of that certain type, as defined. Typically, each row in the table then has a value for each given column. Additionally, state columns can be used to write data to specific rows either at initialization or during the simulation run.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Targets - RPS

A target is an expression that will be evaluated at the end of the simulation run to determine if a particular goal is met. Within the Targets ribbon, there are two buttons, including Add Target and Remove Target, as well as a performance slider.

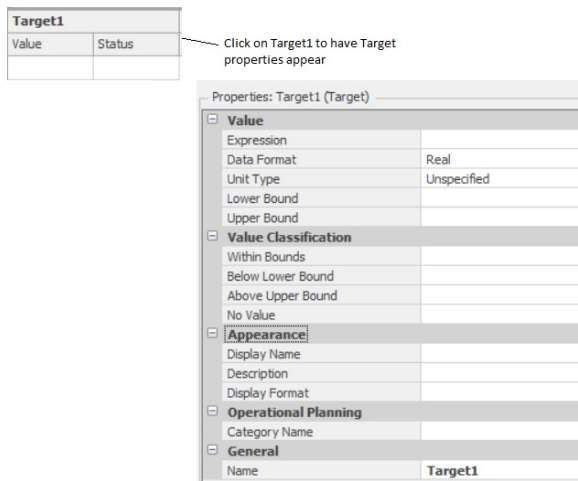


The Target button on the Schema ribbon adds a new target to the table. A target's *Expression* property is evaluated at the end of a simulation run and classified according to the upper and lower bounds specified. When a target is added to a table, two columns named *Value* and *Status* are automatically added. The Target name is shown above both of the columns. To edit the target, click on the target name and the properties of the target will be visible.

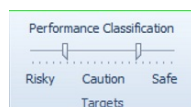
The Value Classifications section of properties allow the user to specify a string value that will be written to the Status column of the target after the simulation run is complete. For example, if the target is a time based value, these may include On Time, Early, and Late.

The Appearance section contains a Display Name, which is shown on the top of the columns for the target. The Display Format allows you to customize how the target value is displayed. For example a format of N1 indicates to display a numeric with 1 digit to the right of the decimal point.

The Remove Target button removes a selected target from the table.



The performance classification slider on the Content ribbon allows the user to specify the ranges for target performance coloring (red/yellow). Target performance values are the probability of meeting the specified target (expressed as a percent). Adjust the sliders to specify the ranges used for applying color to represent different performance levels. When a slider is moved, an indication of the percentages is shown above the slider.



Listed below are the properties of a **Target**:

Property	Description
Expression	The expression to be evaluated as the target's value at the end of the simulation run. This may include values from other columns in the table, as well as table states.
Data Format	The format of the result of the expression (Real, Integer, Boolean, DateTime).

Unit Type	The classification of units for the expression (Unspecified, Time, TravelRate, Length, Currency, CurrencyPerTimeUnit).
Lower Bound	Specifies the minimum value that should be considered acceptable for the target. Values below this bound will be highlighted in the table.
Upper Bound	Specifies the maximum value that should be considered acceptable for the target. Values above this bound will be highlighted in the table.
Units	Units for the expression. These may be found under the Lower Bound and Upper Bound properties and will depend on the Unit Type specified.
Within Bounds	Optional string used to classify the status of the target if its value is within the target's specified lower and upper bounds.
Below Lower Bounds	Optional string used to classify the status of the target if its value is below the target's specified lower bound.
Above Upper Bounds	Optional string used to classify the status of the target if its value is above the target's specified upper bound.
No Value	Optional string used to classify the status of the target if the order has not completed and thus there is no value for that particular target. The Value column will read NaN, while the Status will be reported as No Value (or optional string specified).
Category Name	Optional string used as the Category for displaying the target to the Operational Planning user. If defaulted, then the word "Targets" is used.
Enabled Export	Visible when the table is bound to a data connector for export. If true, the column will be exported. If false, it will not. There are four properties that are shown with the particular data exporter, including Value, Status, Expected and WithinBoundsProbability. Note that when in Interactive view, exporting the table with targets will export Value and Status. When running in Planning mode, the targets will export Value, Status, Expected and Within Bounds Probability. These <i>Enabled Export</i> values are also reflected within the Data Connectors Exporter.

Below is an example of two tables that each includes a date time column (Due Date), a table state that is defined as a date time (Ship Date), and a target (Slack / Target Date). Notice that there are two columns added for the target, including *Value* and *Status*. In the table on the top left, the target's *Expression* property is calculated by subtracting the Ship Date from the Due Date column. The target is then a 'Real' value and is defined as the Slack time (in days). In the table on the bottom right, the target's *Expression* property is the same as the Ship Date and is a 'DateTime' value.

The target's *Expression* property is evaluated at the end of the simulation run and that expression is put into the appropriate row under the Value column. If the entity/order is incomplete, the Value will be reported as NaN. The upper and lower bounds are then evaluated and any string value classifications are used to determine the status of the same row.

Example of Table States and Targets at the End of a Simulation Run

Due Date	Ship Date	Slack	
		Value (Days)	Status
10/12/2011 11:00:00 PM	10/12/2011 8:57:58 AM	0.5847	On Time
10/14/2011 9:00:00 PM	10/14/2011 4:47:48 PM	0.1751	On Time
10/13/2011 4:00:00 PM	10/11/2011 11:48:36 AM	2.1746	On Time
10/14/2011 2:00:00 PM	10/13/2011 1:27:29 PM	1.0226	On Time
10/19/2011 9:00:00 PM	10/20/2011 8:40:27 AM	-0.4864	Late

In the table to the left, the Target is 'Slack' which is a calculation of the 'ManufacturingOrders.DueDate - ManufacturingOrders.ShipDate' as the expression property for the target (in a 'Real' format).

In the table to the right, the Target is 'Target Ship Date' which is an expression based on the table column 'ManufacturingOrders.ShipDate'. The data format is DateTime, which allows additional information to be shown within the Gantt.

Due Date	Ship Date	Target Ship Date	
		Value	Status
10/12/2011 11:00:00 PM	10/12/2011 8:57:58 AM	10/12/2011 8:57:58 AM	OnTime
10/14/2011 9:00:00 PM	10/14/2011 4:47:48 PM	10/14/2011 4:47:48 PM	OnTime
10/13/2011 4:00:00 PM	10/11/2011 11:48:36 AM	10/11/2011 11:48:36 AM	OnTime
10/14/2011 2:00:00 PM	10/13/2011 1:27:29 PM	10/13/2011 1:27:29 PM	OnTime
10/19/2011 9:00:00 PM	10/20/2011 8:40:27 AM	10/20/2011 8:40:27 AM	Late

Targets Displayed in the Entity Workflow Gantt

It is important to note that some targets may also be shown graphically within the Entity Workflow Gantt. In order for a target to appear in the gantt, it must meet the following criteria:

- Must be an entity-related Target
- Must be of *DataFormat* 'DateTime'
- Must have *Lower Bound* and/or *Upper Bound* that are also DateTimes *See note below

Note: Bounds should be in format of 'ManufacturingOrders.DueDate' for example, or may be 'Table.ForecastDateTime + 1'. It is important to note that unless units are specified (which there are no units on Bounds), if you add 1 to a DateTime you are adding 1 hour.

In the above screen shots, the Slack target would NOT be shown in the Entity Workflow Gantt, as the slack value calculated is a 'Real'; however, the Target Ship Date target has a 'DateTime' format and thus would be shown in the gantt.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

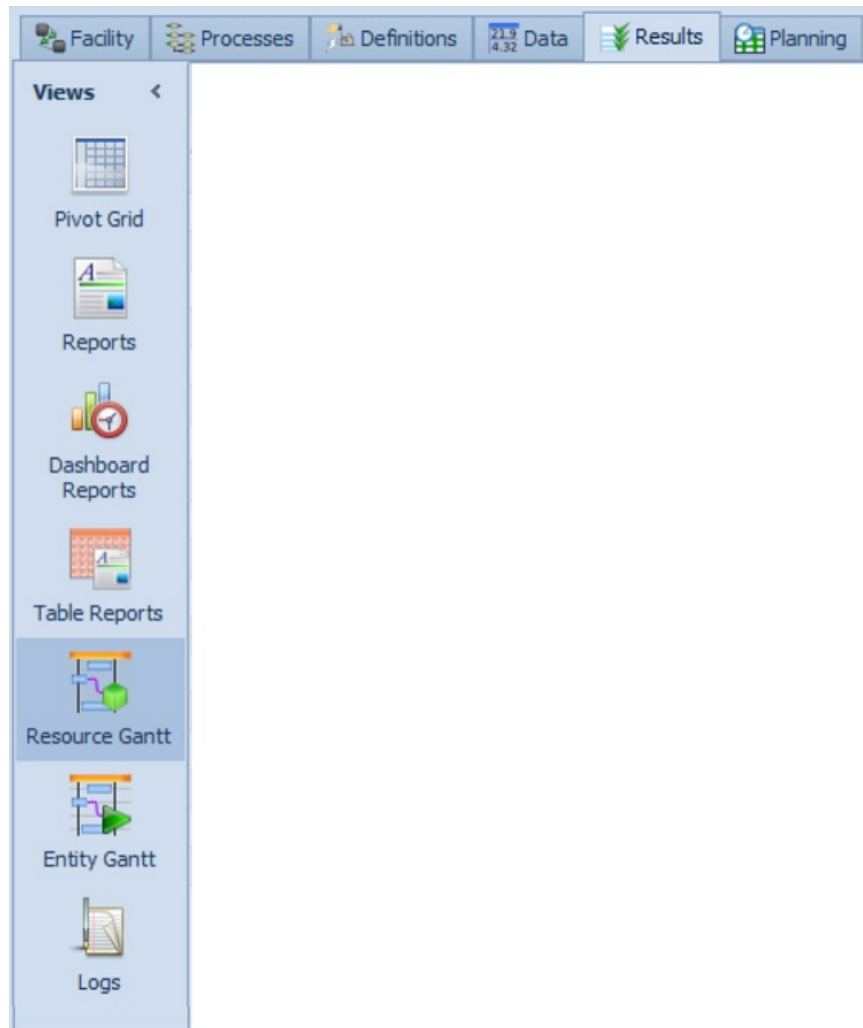
Send comments on this topic to [Support](#)

RPS Results Window

RPS Results Window

The RPS Results window allows the user to display the results in the form of a [Pivot Grid](#), [Reports](#), [Dashboard Reports](#), [Table Reports](#), Gantt (see below) or [Logs](#), depending upon which is selected in the panel. The data from the Pivot Grid and Logs can be [exported](#) into a .csv file.

For information on how to add manual statistics to the project, see [OutputStatistic](#), [StateStatistic](#) and [TallyStatistic](#).



Resource and Entity Gantt

***Important Note*:** There are several differences between the Interactive Gantt within this Results window and those shown in the Planning window.

First, the two Gantt views shown in this Results window operate on the interactive run, so the ribbons have the same Run control buttons that other interactive views have. These Gantt views also get their data from the interactive logs, rather than the planning logs. Right clicking and selecting "Filter to xxx" applies the filtering to the various interactive logs. Similarly, double-clicking on an item moves to the corresponding item in the other interactive Gantt view.

Second, the Zoom and Visibility groups on the Gantt ribbon are per view, allowing configuration of the interactive Gantt and planning Gantt independently. However, both the interactive and the planning versions use the same View options on the Gantt ribbon. For example, setting *Task Row* or *Task Group* in one also sets it for the other.

One of the main differences between interactive and planning modes is that in interactive mode, all randomness, including distributions, failures, and selection weight paths, is included. In contrast, within planning, the model is run in a

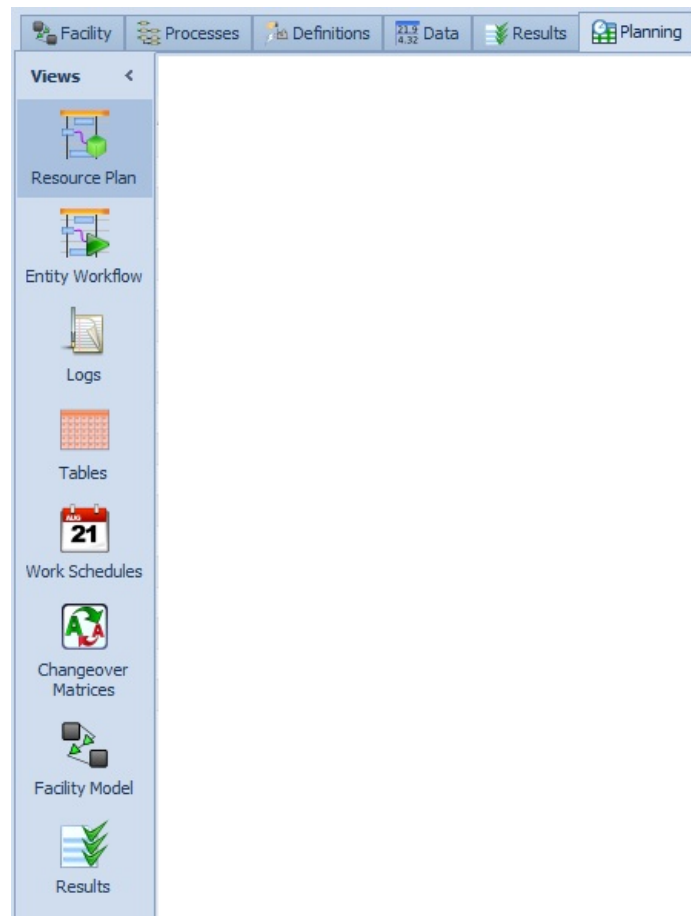
deterministic mode with no uncertainty to generate a resource plan. All distributions used in the model return their expected values and all failures in the model are disabled.

See the [Resource Plan Gantt](#) and [Entity Workflow Gantt](#) in the Planning section for more details on each of these similar Gantt charts.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RPS Planning Window



The Planning Window provides access to various windows used for scheduling purposes. These include the following:

[Resource Plan](#) displays information about how the resources are being used in a Gantt chart format.

[Entity Workflow](#) provides information about specific entities and their flow in a Gantt chart format.

[Logs](#) provides all information about how the resources are being used and the order of entity processing in various log formats – including the a Resource Usage Log, Resource State Log and Constraint Log.

[Tables](#) provides a view of all of the data and sequence tables that have been defined through the Data window.

[Work Schedules](#) allow users to model resource capabilities that change over time.

[Changeover Matrices](#) are used to model time durations that are dependent upon changes from one type of operation to another (e.g., sequence-dependent setup times).

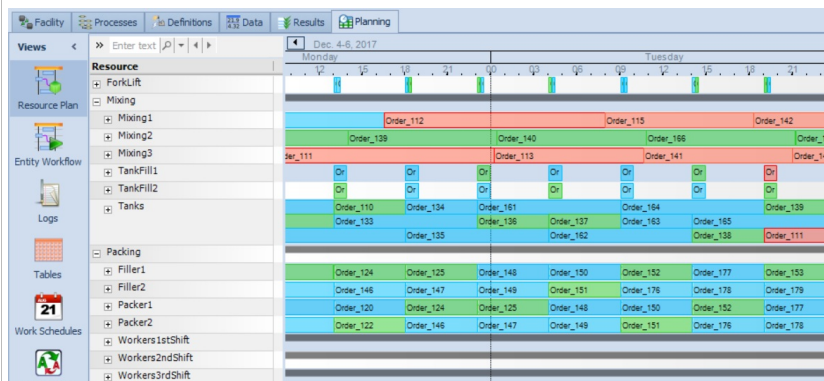
[Facility Model](#) allows the user to view the Facility window animation without editing it.

[Results](#) displays simulation and scheduling results in various formats, including Target Summary, Risk Plots and Detailed Results.

Resource Plan - RPS

The Resource Plan window provides a Gantt chart that displays each resource along with time bars that show each entity (e.g. a job) that utilizes the resource. The changing resource state is shown on each Gantt row behind the time bars. You can expand each row to graphically edit the resource capacity and to also show each entity that is waiting for the resource. Both the Gantt and Operational Planning ribbons are available.

Note that in the screenshots for this topic, the SchedulingBatchBeverageProduction example has been used. There is one example with State Statistics in which a standalone example is used for simplicity.



The format of this Gantt window shows all of the resources in the simulation model in a Gantt chart format and includes the Gantt ribbon, as discussed below.

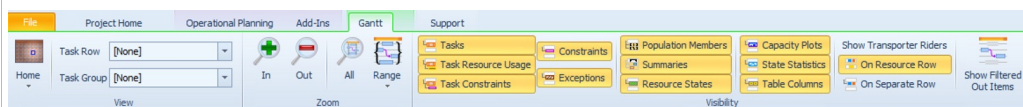
In order for the Gantt chart to display a resource, the *Log Resource Usage* property under the Advanced Options category of the particular object (within the Facility window) must be set to 'True'. The Standard Library objects Server, Combiner, Separator, Resource, Worker, Vehicle and Workstation (Deprecated) have resource logging capability. The Display Category property also allows users to specify an optional text for hierarchically arranging resources within the Resource Plan window. Backslashes may be used for multiple levels (e.g., One\Two\Three).

Within the Operational Planning ribbon, the Create Plan button allows the user to run the model to generate a resource plan. Once Create Plan is selected, the bottom left of the Gantt will display an initializing message before showing Running. This model will be run in a deterministic mode with no uncertainty to generate a resource plan. All distributions used in the model will return their expected values and all failures in the model will be disabled.

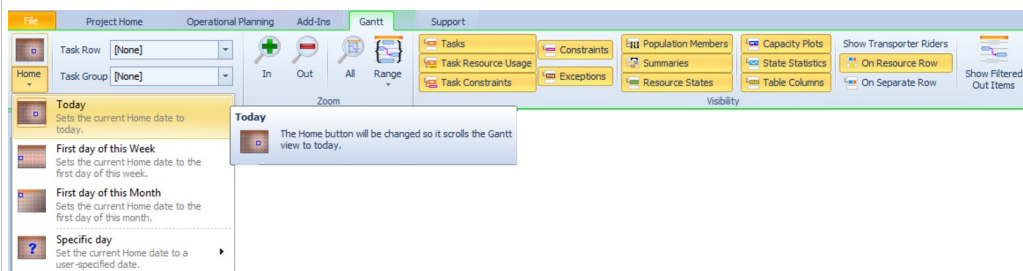
Entities can be moved around on the Resource Gantt by dragging and dropping to a new location within the Gantt. See the Plan Resource Constraints section below for more details.

Gantt Ribbon

Within the Gantt ribbon, there are a number of options for navigating within the Gantt chart (View/Zoom sections), as well as turning on/off specific rows under the various resources (Visibility section).

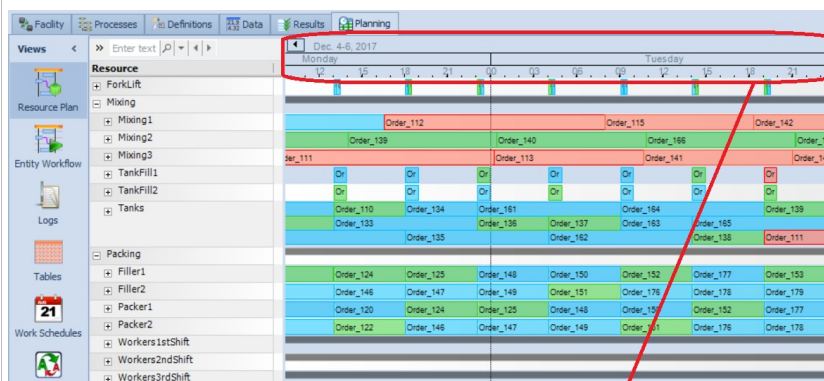


The Home button allows you to select the current home date for the Gantt chart. Options include Today (default value), First day of this Week, First day of this Month and a Specific day. If Specific day is selected, a dialog will appear for the user to select the particular starting day from the calendar.



The Task Row and Task Group allows the various Tasks under each resource to be grouped by a custom expression selected from within the Task Log. If nothing is specified, then the tasks go into the "Tasks" row under the corresponding resource. If a Task Row is specified, then the tasks go in a row of that name "under" the existing "Tasks" row (expanded with the +). See the Entity Workflow page, under Categorizing the Tasks, for an example.

Buttons are available on the Gantt ribbon for zooming In and Out of the Gantt chart view. Additionally, by placing the cursor within the time scaler area (see below), the scroll wheel can be used to zoom in/out. Buttons are also available to see All of the Gantt chart, as well as to zoom to a particular Range, such as an hour, day, week, month or year.



Place cursor in the time scaler area and use scroll wheel to zoom in and out

The Visibility buttons allow you to toggle on/off the various rows beneath the resources shown within the Gantt. These

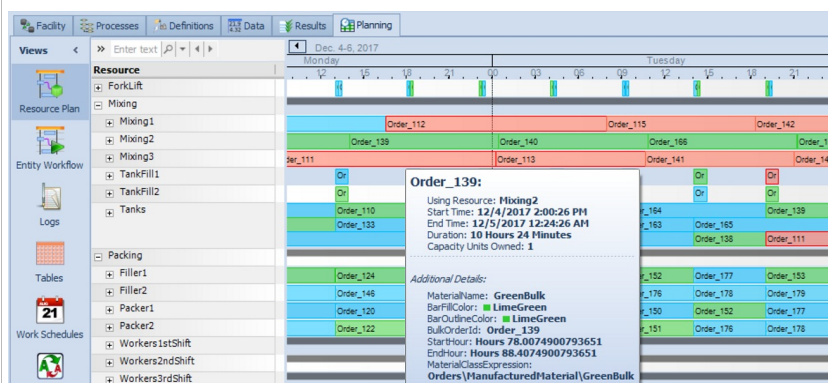
include:

- **Tasks** - Show or hide the task rows.
- **Task Resource Usage** - Show or hide the resource usage rows under task rows.
- **Task Constraints** - Show or hide the constraint rows under task rows.
- **Constraints** - Show or hide the constraint rows. These display the various constrained entities.
- **Exceptions** - Show or hide the exception rows. These relate to the resource schedules and includes day exceptions and downtimes and other exceptions.
- **Population Members** - Show or hide the population member rows. This would include specific member information within vehicles and/or workers.
- **Summaries** - Show or hide summary rows.
- **Resource States** - Show or hide the resource state rows.
- **Capacity Plots** - Show or hide the capacity plots rows. These are shown for multiple capacity resources only.
- **State Statistics** - Show or hide the rows that display State Statistics that have their Log Observations property set to 'True'.
- **Table Columns** - Show or hide the rows that display time-indexed column values.
- **Show Transporter Riders (On Resource Row / On Separate Row)** - Show the Transporter's riders on either the same row as the Transporter or on a separate row. Hide the riders all together by deselecting both options.
- **Show Filtered Out Items** - When the Gantt is filtered to a specific Resource or Entity, the user can show or hide the non-filtered items in the Gantt. By default, the items that are filtered out are not displayed. If the user clicks onto Show Filtered Out Items, they will be shown on the Gantt in a muted, lighter color.

Gantt Chart

Within the Gantt chart, the resource names in the model are shown on the left side of the chart. To the right of each resource is a calendar formatted view of the entities or orders that have utilized the resource. The entity orders are color coded so it is easier to visualize the resources that a particular order has been processed through.

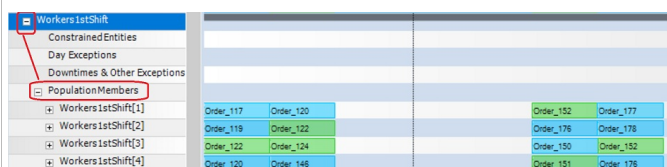
When hovering the mouse over the colored portion of particular entity, details such as the entity identifier, resource name, start time, end time, duration and capacity units owned, are provided in an 'info tip' box. In the below example, the mouse is hovered over Order_139 at the Mixing2 resource. The Additional Details section of the info tip will show entity specific information specified within added columns in the Resource Usage Log.



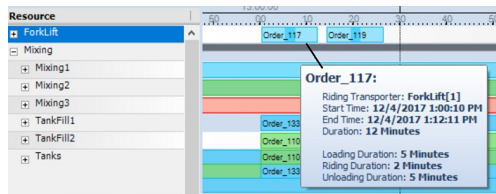
In addition to the entity information shown above for a given resource, if the entity has an associated table row, all table information will also be shown in the property window to the right when a particular entity at a given resource is selected. This property window allows the user to edit any specific table data that is defined with the Editable property for the column as 'True'. Grayed data (not Editable) is also displayed for reference purposes.

Clicking on a given order/entity within the Gantt will display any associated table properties with the properties window - note that the Priority value of '1' is editable, while other properties, such as ProductionCost, are simply for display only.

If the resource is a Fixed object, the Gantt chart is shown as above, with the entities shown directly to the right of the resource name. However, if the Resource is a member of a population (such as a Worker or Vehicle), the Resource name should be expanded (using the '+'). This will display the option to show the Population Members (again using the '+'). Once the population members are displayed (such as Workers1stShift[1], WorkersFirstShift[2], etc.), the entities that were processed with that resource are displayed to the right, as shown below.

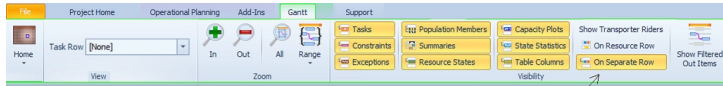


If the Resource is a Worker or Vehicle that includes Load and Unload times for transporting, the graphical representation on the Resource Plan Gantt shows a lighter color on the left (Load Time) and/or lighter color on the right (Unload Time) while the actual Transport time is displayed in the middle with a darker shade of the color, as shown below. When hovering over the particular entity that seized the Worker or Vehicle, the Loading, Riding, and Unloading durations are displayed.

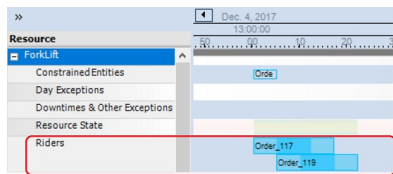


By default, when a Resource is a Vehicle or Worker, any entity that is riding on the Vehicle or Worker will be displayed in the Resource Gantt on the same row as the Vehicle/Worker. Whether or not these entities are displayed at all and where they are displayed can be controlled by the user.

The user can select On Separate Row from the Gantt Ribbon, under Show Transporter Riders in the Visibility section of the ribbon to change how the riding entities are displayed.



This will change the Gantt to display the riding entities on a separate row within the Gantt, as displayed below. In the example below, we have changed the Initial Ride Capacity of the ForkLift to '2' and thus there are two entities riding on a vehicle.



If a Resource has been seized, but not released, when the run finishes, the Resource will be displayed on the Gantt with three dots following at the end. This indicates that the End Time of the seize is unknown and has not happened as of the time of the run completion.



User-Defined Gantt Colors

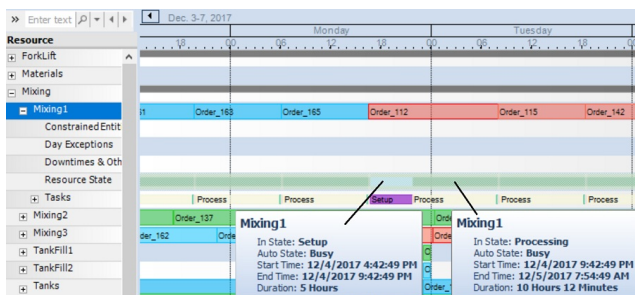
By default, the colors that are used for various entities / orders within the Gantt chart are randomly assigned. However, there are multiple ways in which the colors assigned to the various entities may be through the Properties panel of the Definitions window for the Entity object. Additionally, within a data table, a standard property of type 'Color' can be defined. Finally, a color type column can be added to the Usage Log (see below) which references a table column.

If there are multiple places in which a color property is defined, Simio will first look at the Usage Log column, then any associated data table columns, and finally to the entity color property. If no color property value is found in any of the above 3 ways, a random color will be assigned in the Gantt chart for the entity/order.

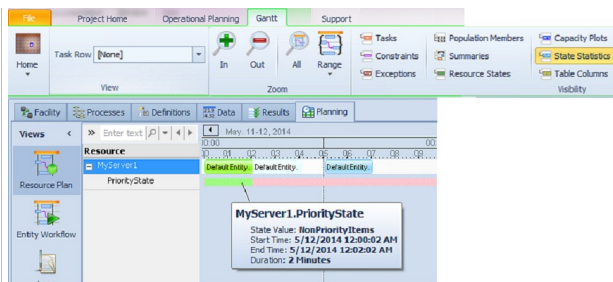
Expanding the Resource

To the left of each resource is a '+' sign that, when clicked, will expand into additional information for that particular resource. This includes the Resource State information, any Constrained Entities, as well as Tasks that may be specified for a given resource object.

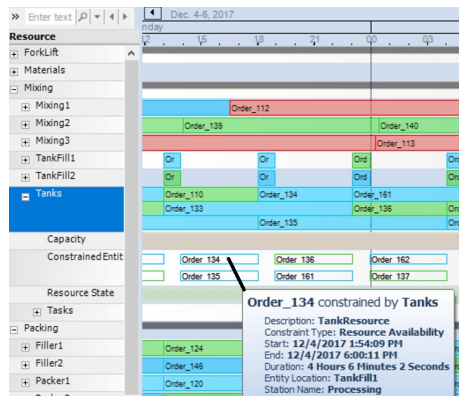
The Resource State row for a given resource will graphically display the state(s) of the resource. When hovered over a given area, information about the state is displayed, as shown below (in the two Mixing1 dialogs). Here, the light blue area shows the 'In State: Setup' and 'Auto State: Busy' and provides information about what state the resource is in while processing the specified entity. The green area shows the 'In State: Processing' and 'Auto State: Busy'. Note that the Setup activity occurs when the orders go from blue to red in this example.



If there are State Statistics for a user defined List State in the model, whose Log Observation property is set to 'True', the State Statistic will be displayed in the Resource Gantt. If the State Statistics button is selected in the Gantt ribbon, hovering over the row will show the value of the State Statistic, the start time, end time and the duration that the state has had this value. The example below shows a custom Server, MyServer1, which has a state statistic for a user defined List State, named PriorityState. The value of this List State changes throughout the model between NonPriorityItems to PriorityItems.

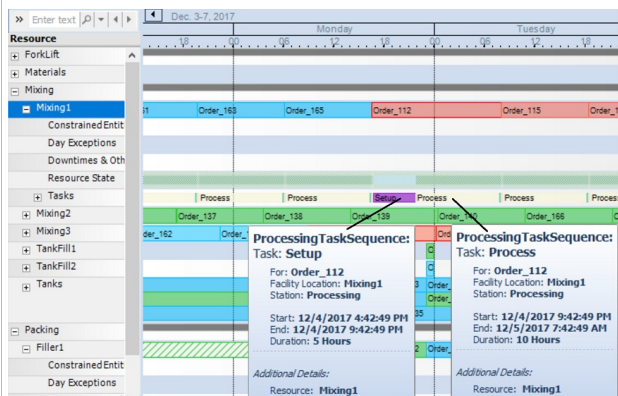


The Constrained Entities section will show any entities that have been waiting at the resource for processing. An example of the Tanks resource's Constrained Entities is shown below. The Constrained Entities can be hovered over individually to find the starting time (in the input buffer), and ending time (in the input buffer). The constrained entities' colors correspond to the entity color as it is shown processing. The example below shows the Tanks resource, which has a capacity of 3 (as you can see 3 entities processing at a time). There can also be multiple constrained entities, as shown below (2) for a given resource.

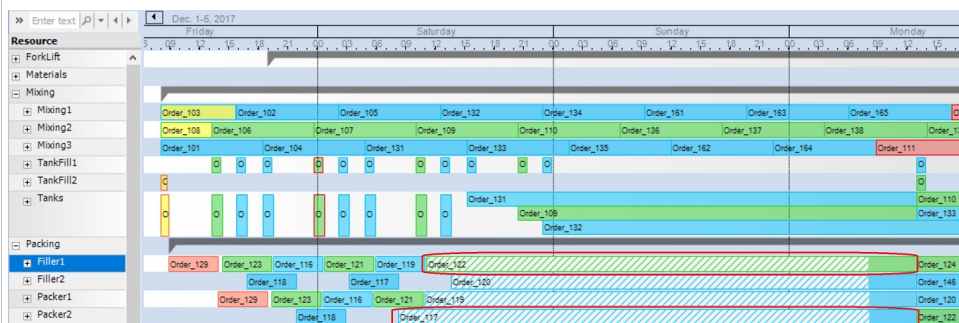


Tasks and the Resource Plan Gantt

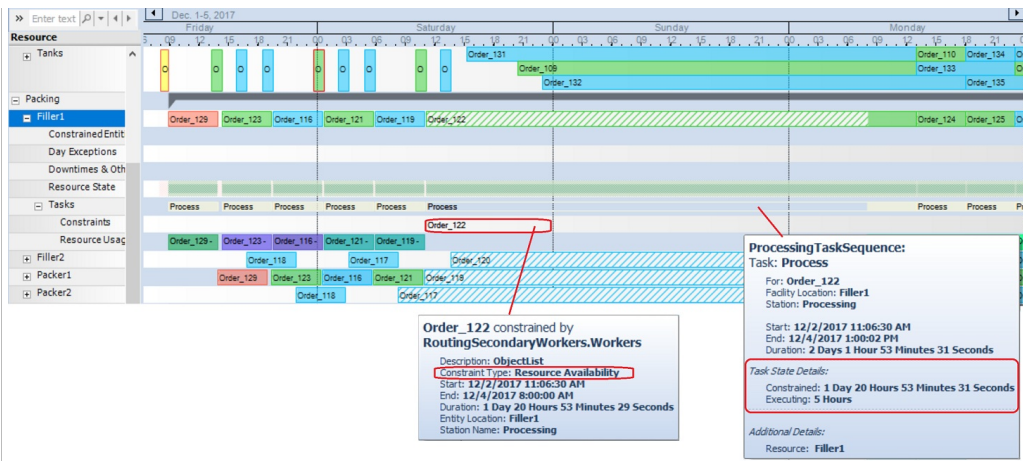
Expanding a resource using '+' will also display the Tasks section, showing the Task Sequence defined tasks associated with each resource. Many of the Standard Library objects, such as Server, Combiner, and Separator include the Processing Type of 'Task Sequence' in which individual Processing Tasks are specified in a repeating property. Those Tasks for each resource can then be shown in the Gantt, as shown below. The '+' next to the Tasks can also be expanded to show any Constraints and Resource Usage associated with the task.



When using Task Sequences, each task may have required secondary resources and materials before processing may begin. These additional resources and materials, if not available, can cause the processing at a Server, Combiner or Separator to be constrained. If resource processing is constrained by a task's constraint, the Resource Plan Gantt bar will be hatched, as shown below. This should indicate to the user to open the resource using '+' to investigate further any constraints with the task.



When the main resource(s) on the Gantt has a hatched area, the '+' should be used to open the resource to review the tasks. The tasks can then also be opened using '+' to display the constraint(s) for the task. Typical constraints may include a shortage of material, availability of the secondary resource (given busy at other locations) or availability of the secondary resource due to offshift status. In the below example, the Filler1 resource shows a hatching area. When expanded, the Process task associated with that resource is shown to be constrained as seen in the hover info box. Additionally, the constrained entity (Order_122) box shows that it is constrained due to worker resource availability. Note that the hatched areas in this example occur on Saturday and Sunday, which is when all 3 shift workers are off-shift.



Searching the Gantt for a Specific Item

Gantt views have a small search box at the top that does basic searching. If a user types or pastes a desired string in the search, Simio will go to the first instance that contains that text, which could be a Resource / Entity along the left side or within the Gantt timeline. The search does partial matches and is not case sensitive. The user can start the search by typing in the desired string and pressing either Enter (keyboard), the search magnifying glass or the right arrow. Selecting the magnifying glass or right arrow multiple times will highlight found instances on a row by row basis throughout the Gantt. The sequence of found items is by row and not by timeline. The left arrow can be used to go backwards in the search to the top of the Gantt. Simio will expand an entry to show instances found within the Tasks (such as a task name) or Constraints of an entry. Search does not look for items within the hover tooltip of a Gantt entry.

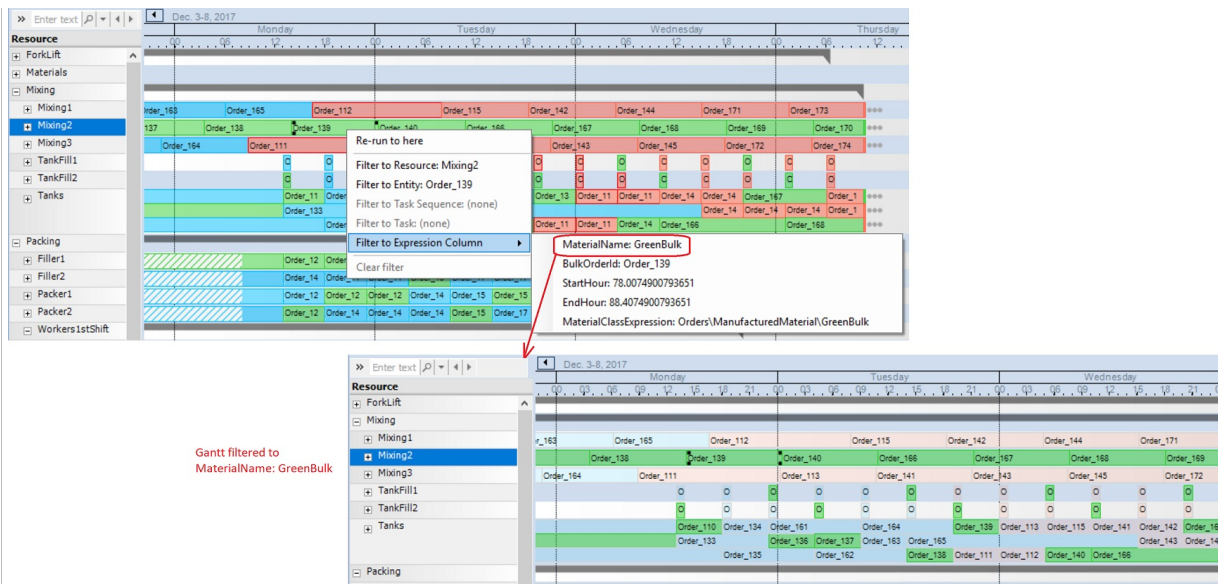
In the example below, searching for Mixing1 matched a row containing that resource, while searching for 102 finds the first instance of Order_102. Selecting the magnifying glass or right arrow will take the user to the next instance of 102 and so on.



Filtering the Gantt to a Specific Resource or Entity

The user can select an Entity in the Gantt and right click. The menu gives the user the option to "Re-run to here, Filter to Resource: X, Filter to Entity: X, Filter to Task Sequence: X, Filter to Task X, Filter to Expression Column (with a list of expression columns) and Clear Filter", where X is the current Resource, Entity, Task Sequence or Task that is selected.

Selecting "Re-run to here" will bring the user to the Operational Planning Facility window and run the model and the animation until that particular entity is processed at that particular Resource. Selecting "Filter to Resource: X" will filter the Gantt so that only that Resource is displayed in both Gantt charts. The user also has the option to show the non-filtered items in the Gantt, in a muted, lighter color, making it easy to focus only on that particular Resource. For this option, the user should click onto Show Filtered Out Items in the Gantt ribbon. Selecting "Filter to Entity: X" will filter the Gantt so that only instances of that Entity in the Gantt will be displayed in both Gantt charts. The user also has the option to show the non-filtered items in the Gantt in a muted, lighter color, making it easy to focus only on that particular Entity. For this option, the user should click onto Show Filtered Out Items in the Gantt ribbon. When the Resource Plan Gantt is filtered to a particular Resource or Entity, the Entity Workflow Gantt is also filtered to that same Resource or Entity. And when the Entity Workflow Gantt is filtered to a particular Resource or Entity, the Resource Plan Gantt is also filtered to that same Resource or Entity. In the example below, the Gantt is filtered to the expression column MaterialName such that only GreenBulk materials are displayed.



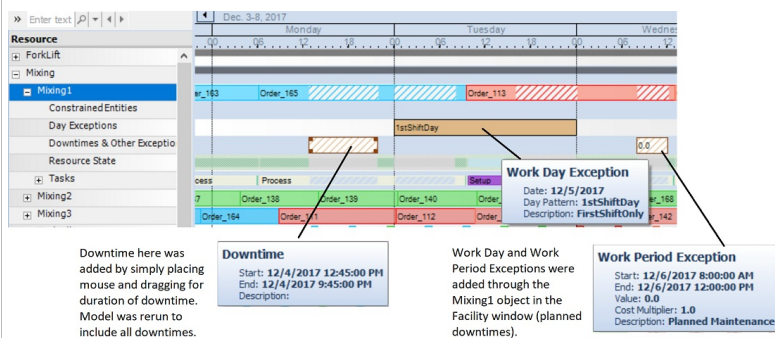
Expanding the Resource State for Exceptions

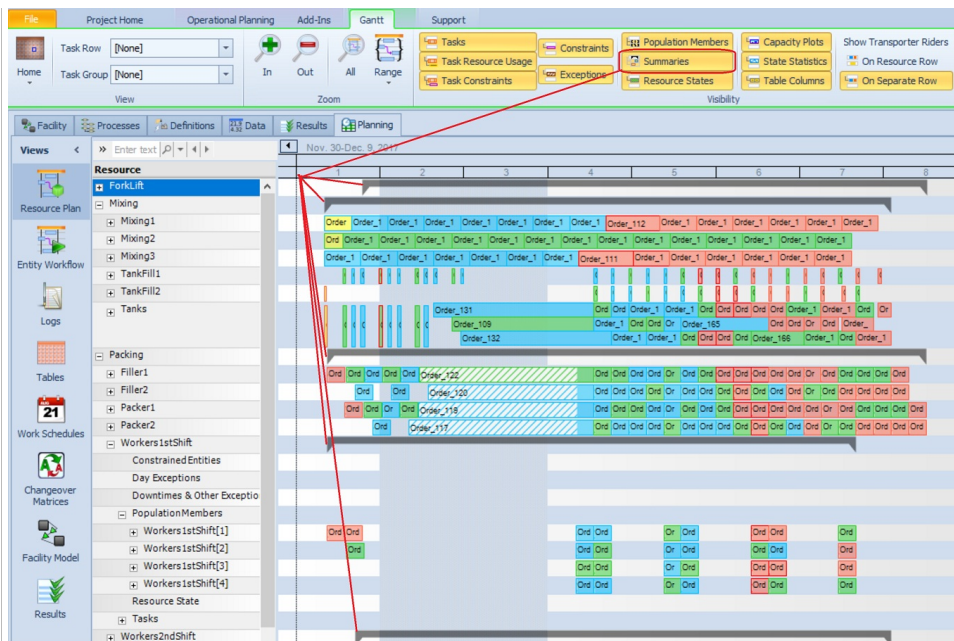
For any resources that have a Capacity Type of 'WorkSchedule', the Resource State will display a '+' sign to the left that can be expanded into work schedule exceptions. This includes Day Exceptions, as well as Downtimes and Other Exceptions.

Day Exceptions are an entire day with a given Day Pattern that may be different than the work day associated with the resource's schedule. For example, perhaps a work schedule consists of five standard work days. A user may wish to have an extended day type schedule on any particular day, which will then override the work schedule. To add a Day Exception using the Gantt, click on the beginning of a given day and drag the mouse to the right. You will see a brown rectangle added to the Gantt. The property window on the right is used to enter the desired Day Pattern name. One day at a time can be added to the day exceptions.

Downtimes and Other Exceptions are periods of time that can be specified as a 'Downtime' (where the resource capacity Value is 0) automatically or a 'GeneralException' (where the user can specify the resource capacity Value as well as Cost Multiplier). To add a Downtime or Other Exception using the Gantt, click on the beginning of a given time period and drag the mouse to the right. You will see a brown hatched rectangle added to the Gantt. The property window on the right is used to enter or change the desired Start Time, End Time, Exception Type and associated properties.

With all exceptions, you can view the exception details by hovering the mouse over the brown rectangle, as seen below. Clicking and highlighting any exception will allow you to edit the details within the property window on the right.





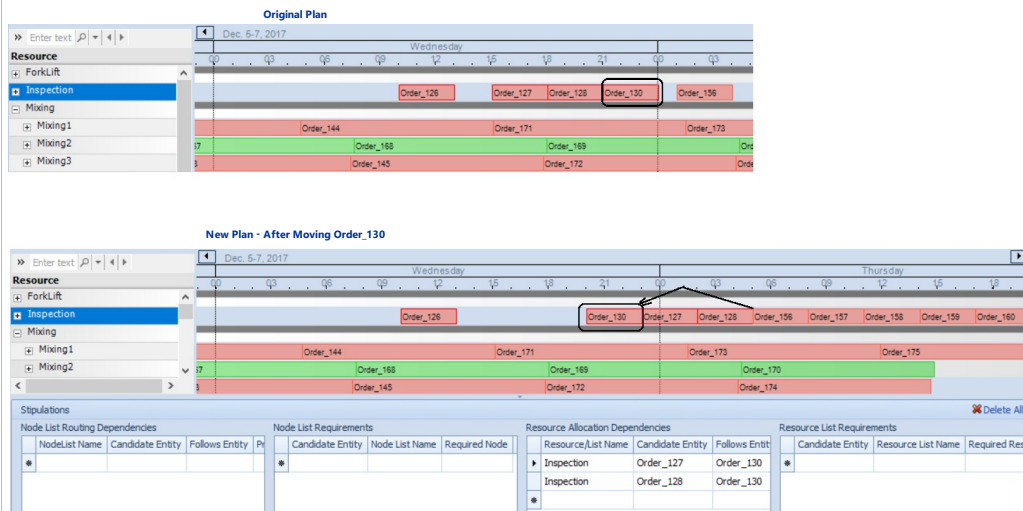
Plan Resource Constraints (Drag and Drop within the Gantt Chart)

If the `IncludeInPlanResourceConstraints` property on the object instance is set to 'True', entities that have seized that object can be reordered in the Resource Plan Gantt by dragging and dropping the entities within the Gantt.

Resource Allocation Dependencies (Reordering Entities Within the Same Resource Row)

If the `IncludeInPlanResourceConstraints` property on the object instance is set to 'True', entities that have seized that object can be reordered in the Resource Plan Gantt by dragging and dropping the entities within the Object's row in the Gantt. If the user drags and drops an entity to a new location in the row, a new Allocation Rule is created and an entry is placed in the Resource Allocation Dependencies table. When the Plan is re-generated, the rules listed in this table are used to determine the order in which resource allocations occur. In other words, when some **Candidate Entity Name** wants to seize from some **Resource or List Name**, it will not be allowed to do so until some other **Dependent Entity Name** (in Stipulations table, i.e. Follows Entity) has seized that same **Resource or List Name** for a specified number of seizures (in Stipulations table, i.e. Prior Usage Count). The user can manually delete rows from the Resource Allocation Dependencies table. The plan should then be re-generated so that the deleted rules will not be applied to the plan.

In the example below, we have slightly modified the same example we have been using to include a single Inspection server for RedPack orders before shipping. Order_130 is scheduled on the Inspection resource after Order_127 and Order_128. The user drags Order_130 before Order_127 and Order_128 within the Inspection resource row in the Gantt Chart, to create two new allocation dependencies: Order_130 must be scheduled before Order_127 and Order_128. The plan is re-run and the new Gantt shows that Order_130 is scheduled before these two orders on the Inspection resource.

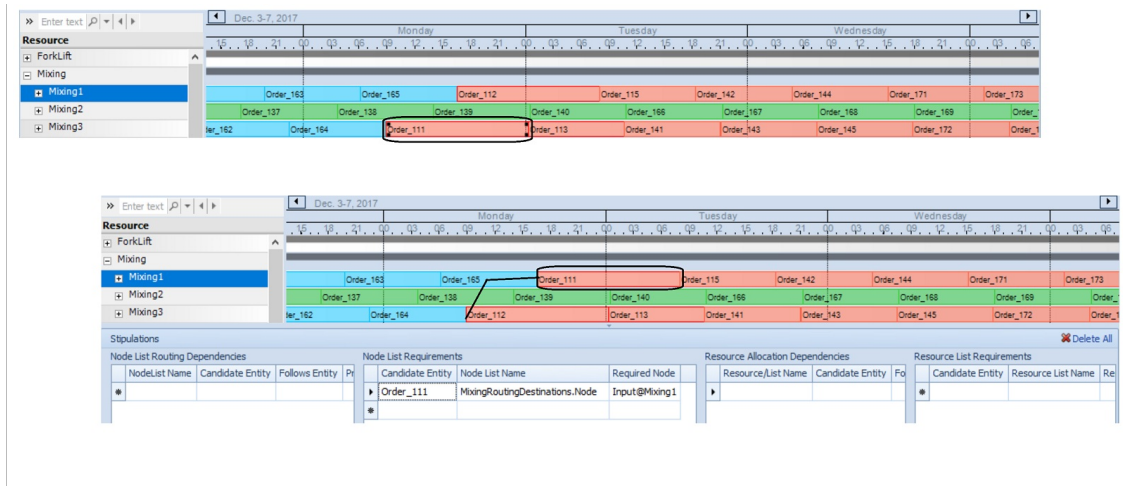


Resource List and Node List Requirements (Moving Entities to a New Resource Row in the Gantt)

If an entity is seizing from an object list, then it is possible for that entity to be forced to seize a specific object from that object list, if the `IncludeInPlanResourceConstraints` property on the object instance is set to 'True'. This is done by dragging and dropping the entity into the row of the object that it should seize. If the user drags and drops an entity to a new resource row, a new Resource List Requirement is created in the Resource List Requirement table. When the plan is re-generated, the rules listed in this table are used to determine which object should be seized by the entities. In other words, the **Candidate Entity Name** is going to seize from the list **Resource List Name**, and it will not be able to seize any resource from the list, but instead it will only be able to seize (and have to wait for) the **Required Resource**. The user can manually delete rows from the Resource List Requirements table. The plan should then be re-generated so that the deleted rules will not be applied to the plan.

This same concept is true when selecting a destination from a Node List. Depending upon the model structure, you may be selecting a Server (or other object) destination from a list of nodes based on particular rules. If an entity destination is being selected from a node list, then it is possible for that entity to be forced to move to a specific node from that node list, if the `IncludeInPlanResourceConstraints` property on the object instance is set to 'True'. This is done by dragging and dropping the entity into the row of the object that it should seize (and thus the associated node destination that it will select). If the user drags and drops an entity to a new resource row, a new Node List Requirement is created in the Node List Requirement table. When the plan is re-generated, the rules listed in this table are used to determine which node (and associated object resource) should be selected by the entities. In other words, the **Candidate Entity Name** is going to select from the list **Node List Name**, and it will not be able to move to any node (and associated object resource) from the list, but instead it will only be able to move to the **Required Node**. The user can manually delete rows from the Node List Requirements table. The plan should then be re-generated so that the deleted rules will not be applied to the plan.

In the example below, the entities are transferred to one of three Mixing locations (with associated Server resources): Mixing1, Mixing2 and Mixing3. Originally, Order_111 moved to Mixing3 for processing. The Gantt below shows the new Plan after the scheduler dragged Order_111 from Mixing3, up to Mixing1, creating an entry in the Node List Requirement table. This entry caused the new plan to force Order_111 to transfer to and be processed by Mixing1, thereby also moving Order_112 to Mixing1.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

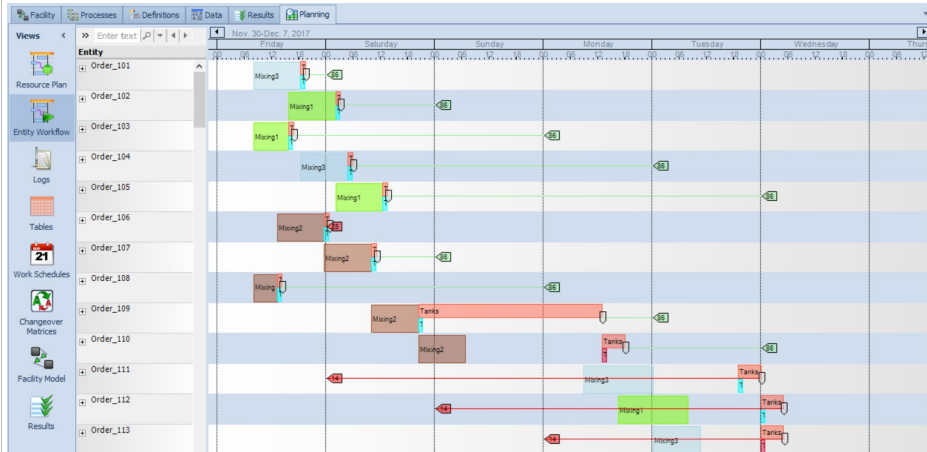
Entity Workflow - RPS

The Entity Workflow window provides a Gantt chart that displays each entity along with time bars that shows each resource the entity utilizes. You can expand each row to show each constraint that impedes the progress of the entity, as well as any tasks within a task sequence that the entity has completed. The Gantt also displays date-time based milestones and targets for each entity, along with associated risk measures.

The color of a resource displayed in the Gantt is automatically generated. This color may be changed by specifying a Display Color property within a particular resource object's Advanced Options group of properties when Log Resource Usage is True.

If an Entity has seized a resource, but not yet released it when the run finishes, the Entity will be displayed on the Gantt with three dots following at the end. This indicates that the End Time of the seize is unknown and has not happened as of the time of the run completion.

Note that in the first section of screenshots for this topic, the SchedulingBatchAverageProduction example has been used. Later in the topic when the Owner Row and Task Group/Task Row are discussed, the SchedulingBicycleAssembly example is used due to complexity.



The Gantt ribbon is available from the Entity Workflow window. Within the Gantt ribbon, there are a number of options for navigating within the Gantt chart (View/Zoom sections, see Resource Plan page for details), as well as turning on/off specific rows under the various entities (Visibility section). Within the View section of the Gantt ribbon, there are multiple fields to allow the user to customize the Entity Workflow Gantt presentation. First, the Owner Group allows the various entities (items) within the Gantt to be grouped based on a column within the Resource Usage Log (see Logs panel). See example 3 on the Resource Usage Log help page for more details. Additionally, the Owner Row can be specified as a Resource Usage Log custom expression to split out the "main" resource time items into sub-rows under the main row. For example, if the entity has parallel activities with multiple resources or multiple entities within an "order", instead of having them all shown together under the same category, they can be separated by work area or row grouping as specified within a Resource Usage Log column. See the example below under Using Owner Row.

The View section of the Gantt ribbon also contains several task related properties. For those models with Task Sequences and Tasks, the Task Row and Task Group options allow the various Tasks under each entity to be grouped by a custom expression selected from within the Task Log. If nothing is specified, then the tasks go into the "Tasks" row under the corresponding entity. If a task row is specified, then the tasks go in a row of that name "under" the existing "Tasks" row (expanded with the +). See below for examples of Categorizing the Tasks.

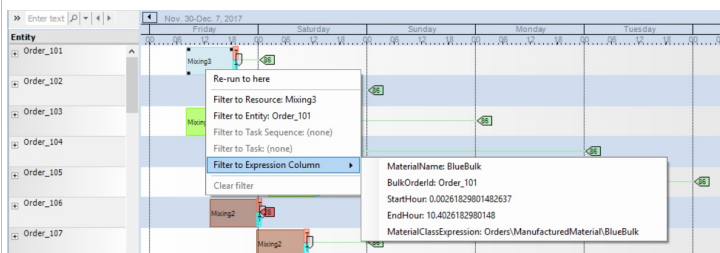


By hovering over a given resource within the Gantt for an entity, the entity (order) name is displayed, along with the start time, end time, duration and capacity units owned. If the entity has an associated table row, that information is also displayed.

Right-Clicking in the Gantt

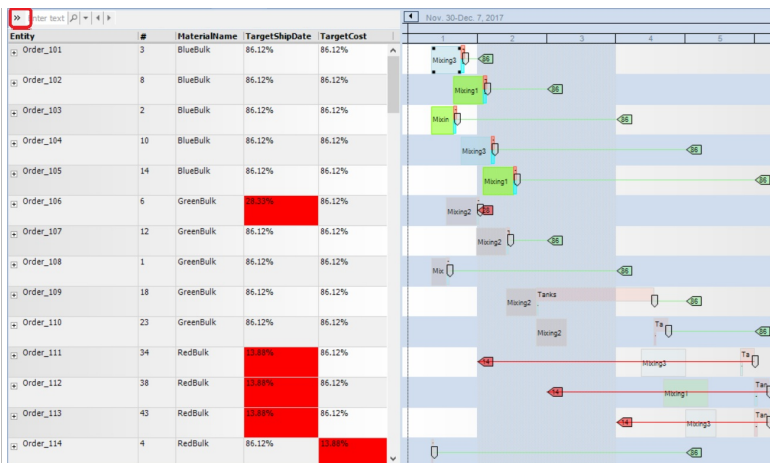
The user can select a Resource in the Gantt and right click. The menu gives the user the option to "Re-Run to here, Filter to Resource: X, Filter to Entity: X, Filter to Expression Column and Clear Filter", where X is the current Resource that is selected and the current entity that is selected.

Selecting "Re-Run to here" will bring the user to the Operational Planning Facility window and run the model and the animation until that particular entity is processed at that particular Resource. Selecting "Filter to Resource: X" will filter the Gantt so that all of the instances of that Resource in the Gantt will be brightly colored, but everything else in the Gantt will be muted a lighter color, making it easy to focus only on that particular Resource. Selecting "Filter to Entity: X" will filter the Gantt so that all of the instances of that Entity in the Gantt will be brightly colored, but everything else in the Gantt will be muted a lighter color, making it easy to focus only on that particular Entity. When the Entity Workflow Gantt is filtered to a particular Resource or Entity, the Resource Plan Gantt is also filtered to that same Resource or Entity and vice versa.



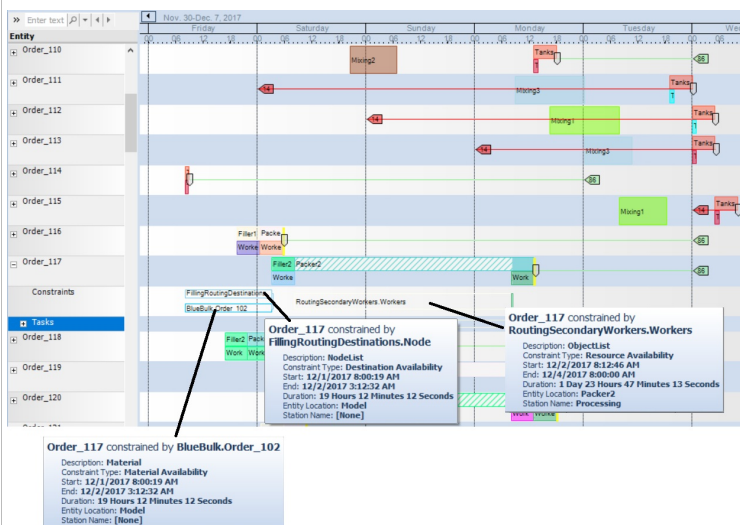
Expanding the Gantt Headings

Expanding the Entity Workflow window by selecting the >> at the top of the word Entity will display additional information about the Target results when a Risk Analysis is performed. This includes any target values and color coding based on target limits and performance classification that are set. The Entity Workflow Gantt results may be sorted by clicking to the right of the target name of a particular column.



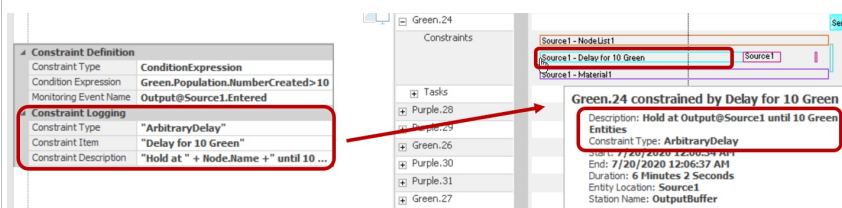
Viewing the Entity Constraints

Similar to the Resource Plan Gantt, the Entity information may be expanded by clicking on the '+' to the left of the entity name. This will provide additional information on the Constraints (if the Constraints button on the Gantt ribbon is selected) for each entity during the simulation run. As you can see below, the entity Order_117 is first waiting for Material Availability (BlueBulk) as well Destination Availability (Filler2). The order then has a period of time for Resource Availability while the 3rd shift workers are off shift over the weekend.



When the mouse hovers over a constraint for the entity, a display is shown that includes the type of object (Server, Worker, Resource, etc.), as well as the type of constraint. Types of constraints include Destination Availability (waiting on Node List allocation to a destination), Resource Availability (waiting for resource capacity to become available), Resource Arrival (waiting for a worker or vehicle to arrive to the entity location) and Material Availability (waiting for material(s) specified to be available for reservation).

You can set additional information values shown by hovering over a constraint in the Constraint Logging properties of the Constraint. Constraint Item appears in the Constraint Bar on the Gantt, Constraint Description appears in the Tooltip under Description, and Constraint Type appears in the Tooltip.

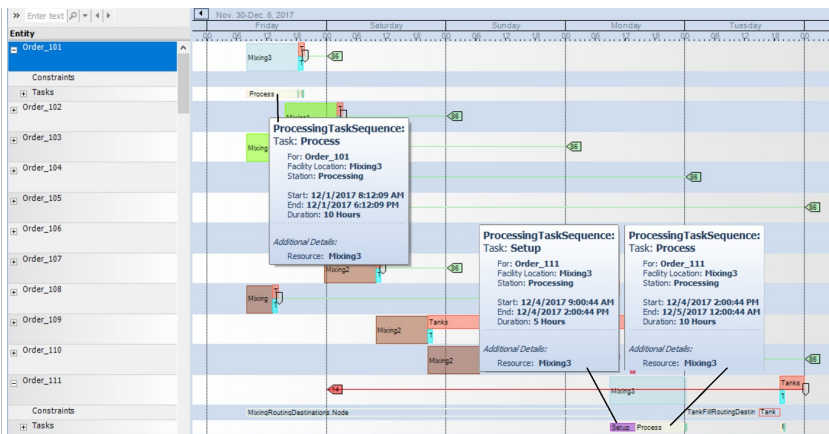


Viewing the Targets

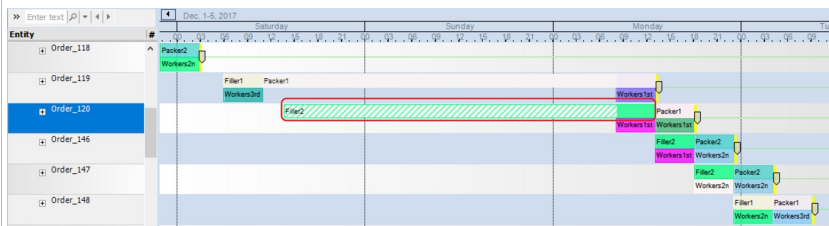
Targets are shown on the Entity Workflow Gantt only when they meet certain criteria, including being an entity-related target, being of Data Format 'DateTime' and having Bounds that are also DateTimes. See the [Targets - RPS](#) page for more information.

Viewing the Tasks

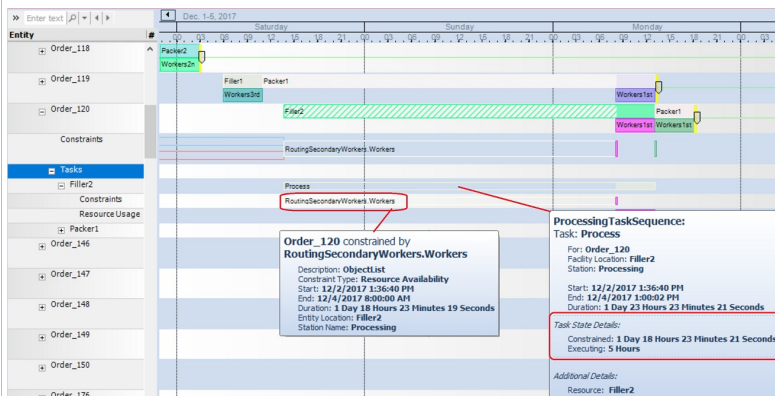
Expanding the entity information using the '+' will also provide detailed information on any Tasks (if the Tasks button on the Gantt ribbon is selected) for each entity. Tasks are specified within many Standard Library objects by using the Processing Tasks method of processing to specify the 'Task Sequence' for the object. Below the entity Order_101 includes a Process task within the Mung3 object, while Order_111 show both Setup and Process tasks. Hovering over a specific task provides additional details about the task, including starting and ending times and duration. The Task Log shows detailed information for all entities as well.



The "+" next to the Tasks can also be expanded to show any Constraints and Resource Usage associated with the task. When using Task Sequences, each task may have required secondary resources and materials before processing may begin. These additional resources and materials, if not available, can cause the processing at a Server, Combiner or Separator to be constrained. If an entity is constrained by a task's constraint, the Entity Workflow Gantt bar will be hatched, as shown below. This should indicate to the user to open the entity/order using "+" to investigate further any constraints with the task.

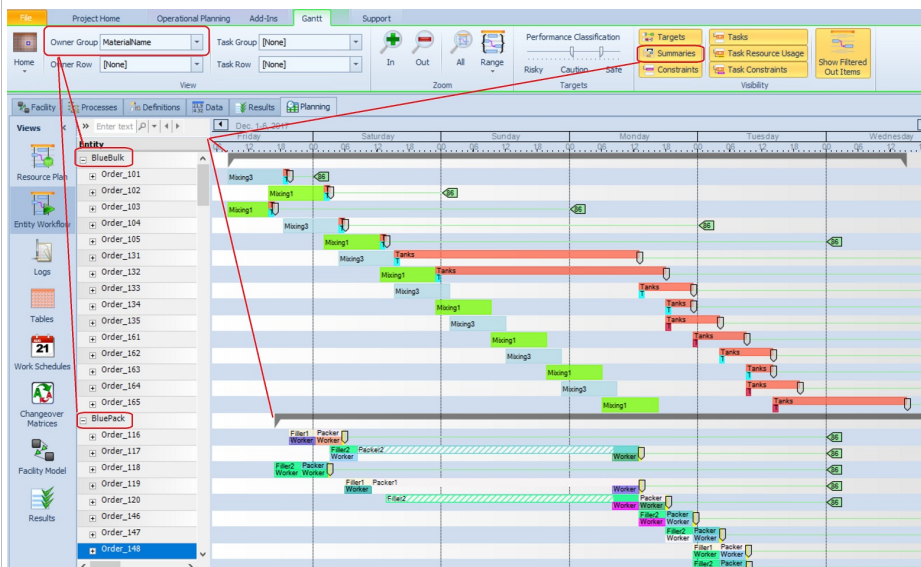


When a resource (on the Entity Workflow Gantt) has a hatched area, the "+" should be used to open the entity to review the tasks performed within that resource. The tasks can then also be opened using "+" to display the constraint(s) for the task. Typical constraints may include a shortage of material, availability of the secondary resource (given busy at other locations) or availability of the secondary resource due to offshift status. In the below example, the Filler1 resource shows a hatched area. When expanded, the Process task associated with that resource is shown to be constrained as seen in the lower info box. Additionally, the constrained entity (Order_120) box shows that it is constrained due to worker resource availability. Note that the hatched area in this example occurs on Saturday and Sunday, which is when all 3 shift workers are off-shift.



Using the Owner Group and Viewing the Summaries

Within the Entity Workflow Gantt, the Owner Group field on the Gantt ribbon allows users to group the entities (orders) into categories. In the example below, the "Material Name" is used as the Owner Group. Within the Resource Usage Log, any custom columns added to the log can be used for the Owner Group or Owner Row categories. For any owner groups, the Summaries are shown, which are simply gray bars across the top of each category or section to show the overall length of time the entities (orders) in group are the system.

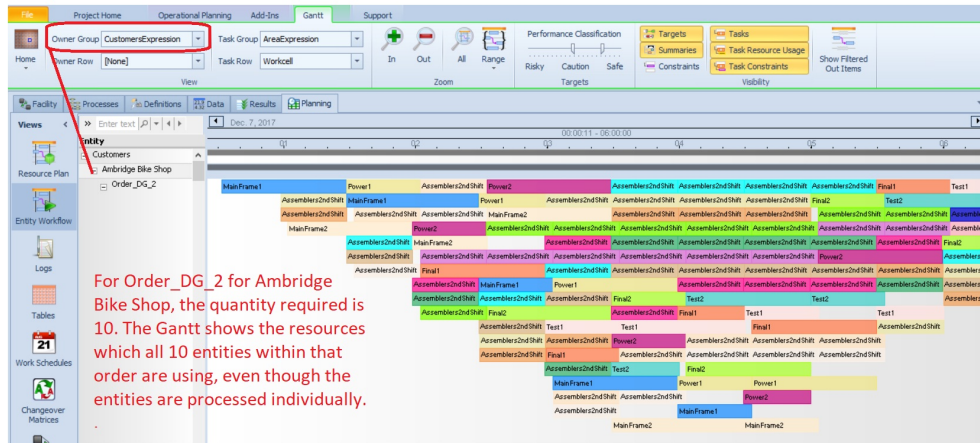


Using the Owner Row

The Owner Row can be specified to split out or group the "main" resource time items into sub-rows under the main row for an entity or order. This is typically done if multiple entities are being processed simultaneously for a given order or assembly, for example. By default, the Owner Row within the Gantt ribbon is blank.

In the next few screenshots, we switch to the SchedulingBicycleAssembly example, as each of those order's quantity is manufactured separately. All 10 quantity items for the given order are processed separately, but shown on the Gantt under the order Order_DG_2. Note that the Owner Group is specified as CustomerExpression to group entity flow into different types of customers - see Example 3 in Resource Usage Log for example.

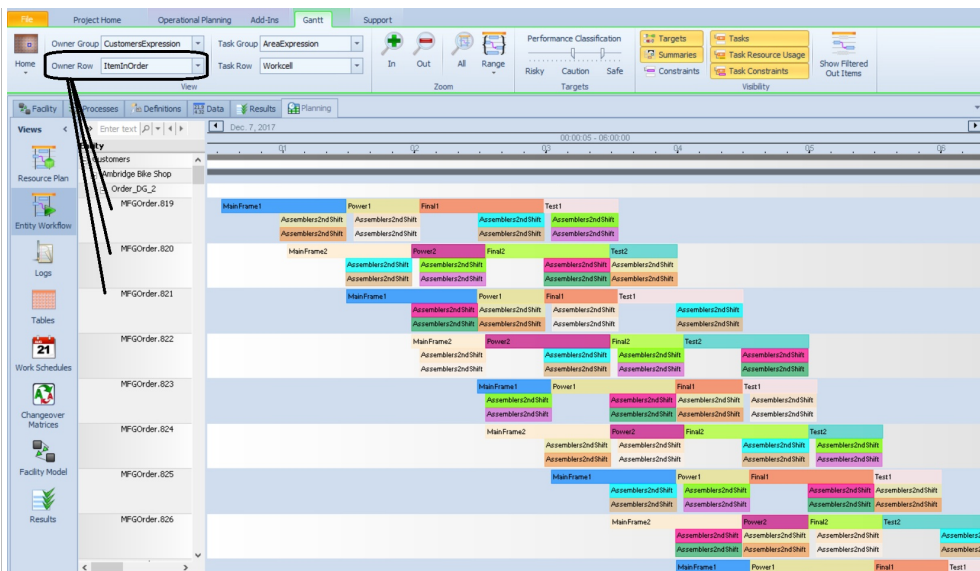
Resources	Materials	Routing	Material Lots	Manufacturing Orders	Purchase Orders	Bill Of Materials	Work In Process	Manufacturing Ord
Order Id	Customer	Material Name	Release Date	Due Date	Order Status	Priority	Quantity	
1	Order_SG_1	Walmart	EcoBikeSTDGreen	12/4/2017 8:00:00 AM	12/6/2017 4:00:00 PM	New	1	12
2	Order_SW_1	Walmart	EcoBikeSTDWhite	12/4/2017 8:00:00 AM	12/6/2017 4:00:00 PM	New	1	10
3	Order_DG_1	Walmart	EcoBikeEXTGreen	12/4/2017 8:00:00 AM	12/6/2017 4:00:00 PM	New	1	8
4	Order_SW_1	Walmart	EcoBikeEXTWhite	12/4/2017 8:00:00 AM	12/6/2017 4:00:00 PM	New	1	9
5	Order_SG_2	Ambridge Bike Shop	EcoBikeSTDGreen	12/4/2017 8:00:00 AM	12/7/2017 4:00:00 PM	New	3	15
6	Order_SW_2	Ambridge Bike Shop	EcoBikeSTDWhite	12/4/2017 8:00:00 AM	12/7/2017 4:00:00 PM	New	3	9
7	Order_DG_2	Ambridge Bike Shop	EcoBikeEXTGreen	12/4/2017 8:00:00 AM	12/7/2017 7:00:00 PM	New	3	10



Similar to using the Owner Group field, an additional column(s) in the Resource Usage Log can be added to further group the items into their individual entities, in this case, for each of the 10 items in the order. Note that an expression is used to determine what entity number (Owner Object Name) is used. This information can then be used to put each entity in the order into a separate owner row.

</

Below, the Owner Row is now the new resource log column 'ItemInOrder', so each of the 10 items in the order can be seen in its own row under the order itself.



To better distinguish which Task is associated with an Owner, have a Log Expression Column in the Task Log that corresponds to a Log Expression Column in the Resource Usage Log and then set the Owner Row of the Gantt to this column. This will cause the Task to appear underneath the Owner.

To better distinguish which Transporter is associated with an Owner, have a Log Expression Column in the Transporter Usage Log that corresponds to a Log Expression Column in the Resource Usage Log and then set the Owner Row of the Gantt to this column. This will cause the Transporter to appear in the Owner's Row.

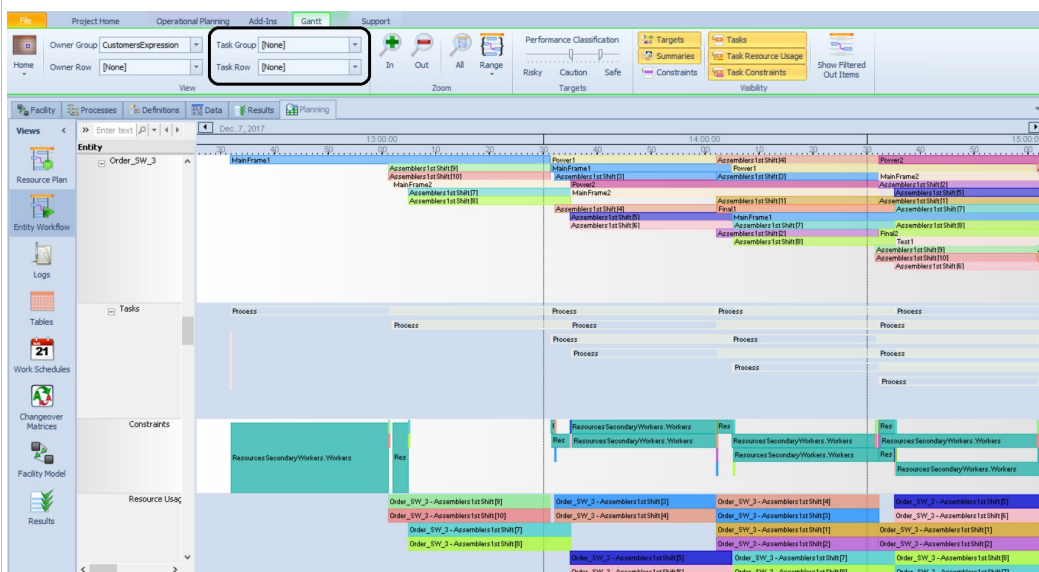
Categorizing the Tasks

Within the Task Log (Log), the Log ribbon allows for users to add additional columns. This is similar to the Resource Usage Log where additional columns may be added for various reasons. Adding a custom expression column(s) to the Task Log allows additional categorizing of tasks with the Tasks section for each entity. Tasks are specified within many Standard Library objects by using the Processing Tasks method of processing to specify the 'Task Sequence' for the object. The below will go through several examples of using the Task Row and Task Group properties on the Gantt ribbon.

All of the examples shown below are taken from the *SchedulingBicycleAssembly* example.

Task Row and Task Group Not Specified

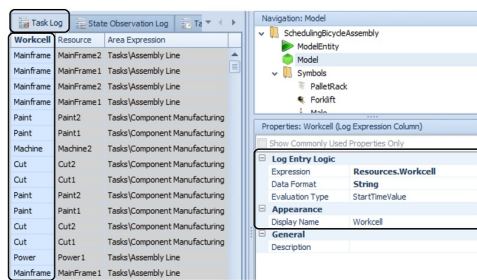
In the below example, note that the Task Row and Task Group properties on the Gantt ribbon are specified as 'None' meaning there is not categorizing of the tasks at this point. This is simply a much larger example than that shown above of the tasks shown under the entity. In this example, there are multiple resources shown next to Order_SW_3, which is an assembled entity with many tasks at multiple workcenters. Under each entity, the Tasks are displayed. In this example, there are multiple tasks occurring for the assembled part at the same time. Note that the Task section is also expanded using the '+' to show both Constraints and Resource Usage. These categories may be turned off by using the corresponding Task Constraints and Task Resource Usage buttons on the Visibility section of the Gantt ribbon. ***IMPORTANT NOTE:** The Constraints and Resource Usage sections under Tasks will only display material/resource constraints that are associated with the task. This includes only those specified within the Processing Tasks repeating property editor. The Resource Requirements and Material Requirements sections for a given task associate the resource/material with the task. The task states (shown as Constrained and Executing in this example) indicate if the entity is waiting for either secondary resource or material specified within the task. If a resource is specified as a secondary resource within the main property window of a Server, it is not associated with a particular task.



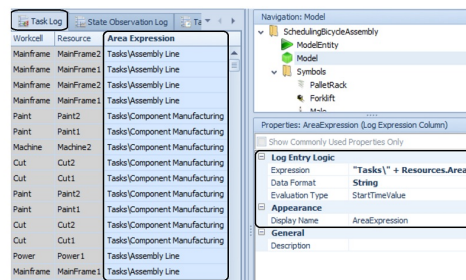
Adding Task Log Columns

Within the Task Log, additional columns may be added for use in categorizing and displaying the tasks within the Entity Workflow and Resource Plan Gantt. Below, the Workcell and Area Expression columns have been added to the Task Log. Note that within the Log Entry Logic properties section for the column, the Expression property can reference any table information associated with the task. The Workcell column references the 'Resources.Workcell', while the Area Expression column uses a concatenated string to reference 'Resources.Area' combined with the string 'Tasks', resulting in 'Tasks' + 'Resources.Area'. These additional columns are then used to categorize how tasks are displayed within the Gantt charts.

Note: These additional columns on the Gantt will have a minimum width of the width of the first column on the Gantt. A resize of the first column may need to be performed in order to reveal more of the Gantt.



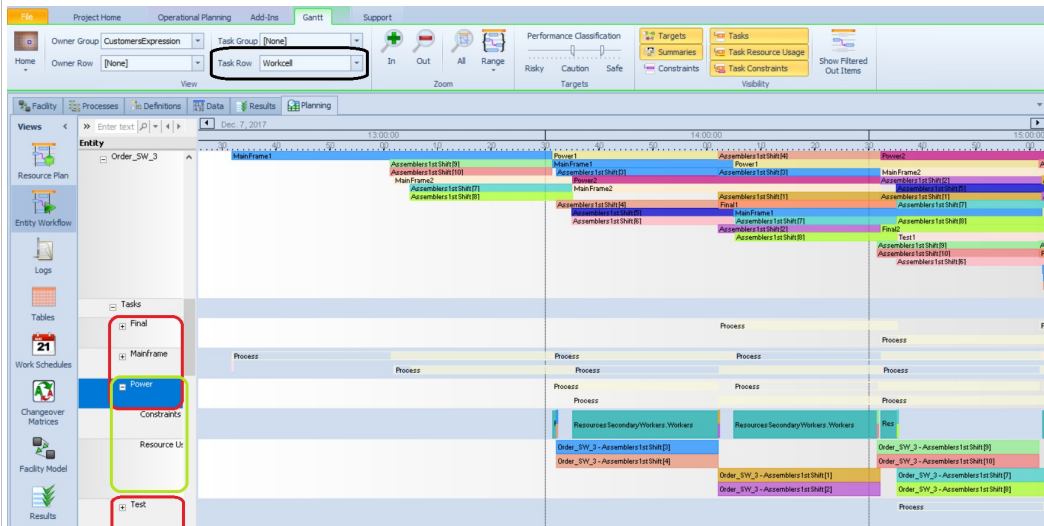
Used for the Task Row property



Used for the Task Group property

Using Task Row

The Task Row property on the Gantt ribbon indicates the expression column from the Task Log that holds the name of the row that each task item should go into. If the Task Row is not specified, the items go into a 'Tasks' row under their associated owner. If specified, the tasks go into a row of that name under the specified grouping. In the example below, the Task Row is specified as the 'Workcell' column from Task Log. Each Task Row category may then be expanded to view the Constraints and Resource Usage for that section of tasks.

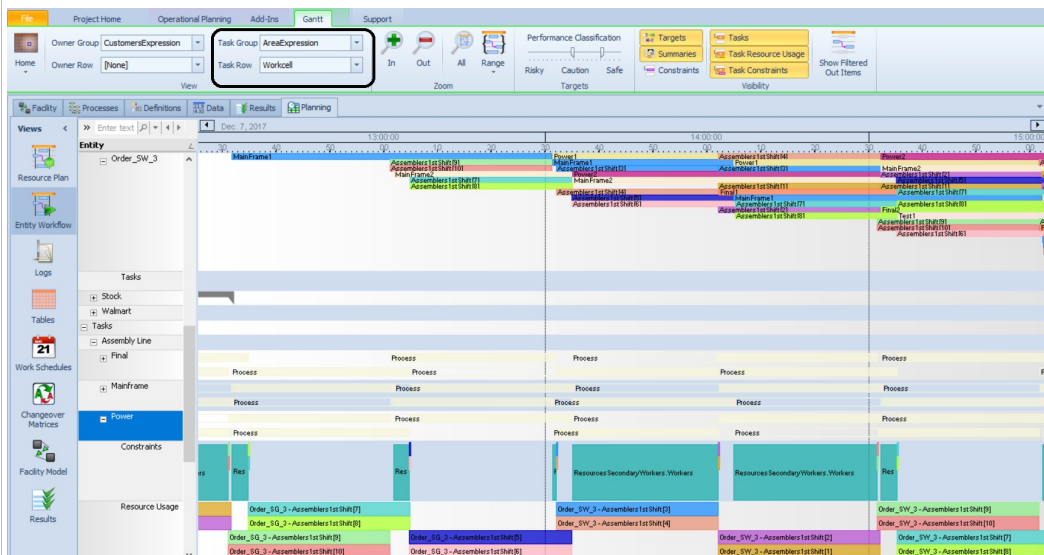


Note that the Task Row in the Gantt ribbon is 'Workcell' taken from the Task Log column. When a Task Row is specified, the tasks *and* their associated constraints and resource usage *are* categorized.

Red shows the Task Row by Workcell (Final, Mainframe, Power, Test) where Green shows the Power tasks expanded using the +.

Using Task Group

The Task Group property on the Gantt ribbon indicates the expression column from the Task Log to use to identify the grouping/category for a row a task item goes into. If not specified, the task row will appear below the associated owner. In the example below, both the Task Row and Task Group are specified. Thus, the Tasks are categorized first using the Task Group, which is 'Area Expression', and then within each 'Area Expression', the tasks are broken into the Task Row categories. Using the Task Group actually moves the tasks from originally "under" the owner (entity) to separate categories and will display all owners (entities) under the each task category.



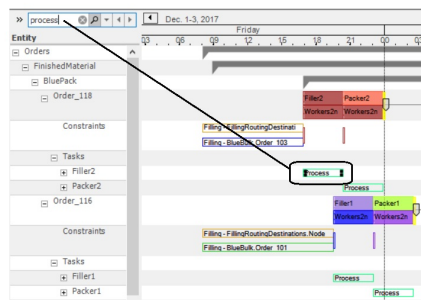
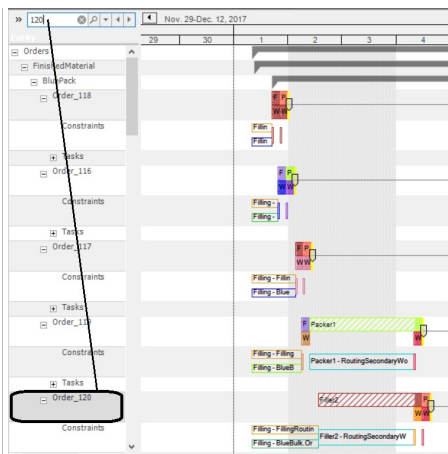
Using the Task Group option for categorizing tasks actually takes the Tasks out from under the particular order number (or entity) and puts them in a Tasks section. This allows the tasks for all orders (entities) to be seen in the same area.

If the Task Row is designated, the tasks will be categorized for each Task Group.

Searching the Gantt for a Specific Item

Gantt views have a small search box at the top that does basic searching. If a user types or pastes a desired string in the search, Simio will go to the first instance that contains that text, which could be a Resource / Entity along the left side or within the Gantt timeline. The search does partial matches and is not case sensitive. The user can start the search by typing in the desired string and pressing either Enter (keyboard), the search magnifying glass or the right arrow. Selecting the magnifying glass or right arrow multiple times will highlight found instances on a row by row basis throughout the Gantt. The sequence of found items is by row and not by timeline. The left arrow can be used to go backwards in the search to the top of the Gantt. Simio will expand an entry to show instances found within the Tasks (such as a task name) or Constraints of an entry. Search does not look for items within the hover tooltip of a Gantt entry.

In this example, searching for 120 matched a row containing that string, while searching for process expanded the tasks for Order_118 to show the first row containing 'process' which is the name of a task.



Copyright 2006-2025, Simio LLC. All Rights Reserved.
 Send comments on this topic to [Support](#)

Logs - RPS (Enterprise)

The Logs panel provides access to various resource and entity specific logs. These include the following:

[Resource Usage Log](#) provides The information detailing each resource and the entity that utilized it.

[Resource State Log](#) includes details of the resource states over time.

[Resource Capacity Log](#) records changes in each resource's capacity over time.

[Resource Info Log](#) provides information for each of the resources in the system.

[Constraint Log](#) provides details of the constraints to the entity over time.

[Transporter Usage Log](#) includes details of the start and end times of the transporter's usage.

[Material Usage Log](#) includes details of the material usage including time of quantity changes and current stock level.

[Inventory Review Log](#) includes details of inventory review events

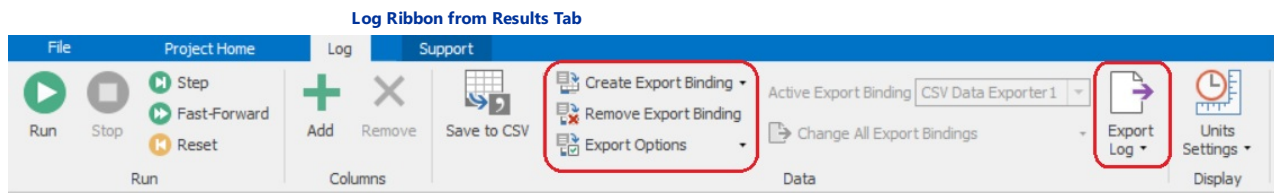
[Task Log](#) includes details of the task sequence type tasks executed over time. It includes the task name, duration and location.

[Task State Log](#) simply records all task state changes.

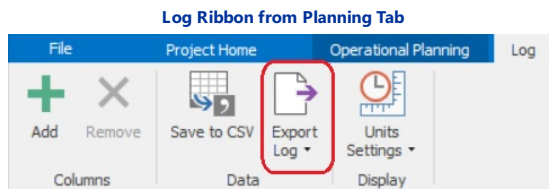
[State Observation Log](#) displays the log of values over time of State Statistic elements which had *Log Observations* set to 'True'. This log is included in all license versions of Simio.

[Tally Observation Log](#) displays the log of values over time of Tally Statistic elements which had *Log Observations* set to 'True'. This log is included in all license versions of Simio.

Within RPS only, the Log ribbon allows users to bind a log to an Export [Data Connectors](#), similar to binding to tables. Data bindings are edited within the Data Connectors view in the Data tab.



The Log view under the Planning tab only contains the Export Log, thus all export bindings should initially be configured within the Results tab Log view shown above.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Resource Usage Log - RPS

The Resource Usage Log provides information detailing each resource and the entity / order that utilized it. The columns provided with the log include:

- **Resource Id** - The name of the resource that is utilized.
- **Resource** - The name of the resource that is utilized. This defaults to the Resource Id unless a *Display Name* is specified with the Resource.
- **Resource List** - The name of the resource list, if any, from which the resource was selected.
- **Node List** - The name of the node list, if any, from which the resource was selected, such as a destination node list for Servers.
- **Owner Id** - The specific entity or order number utilizing the resource.
- **Owner** - The specific entity or order number utilizing the resource. This defaults to the Owner Id unless a *Display Name* is specified with the owner.
- **Task Id** - The specific task, if any, associated with the allocation of the resource.
- **Start Time** - The time that the entity seized the resource.
- **End Time** - The time that the entity released the resource.
- **Duration** - The total time that the entity was allocated the resource, as calculated by 'End Time - Start Time' in default time units.
- **Avg** - The average number of units of the resource that has been seized.
- **Min** - The minimum number of units of the resource that has been seized.
- **Max** - The maximum number of units of the resource that has been seized.

NOTE: The Avg, Min and Max values represent the number of units of a given resource that has been seized by this particular owner. Therefore, if 1 unit of Resource1 is seized, a delay occurs, then another unit of Resource1 is seized, a delay occurs, then both are released, the Min value would be '1', the Max value would be '2' and the Avg value would be the time weighted average over the Duration.

Resource Usage Log		Resource State Log		Resource Capacity Log		Constraint Log		Transporter Usage Log		Material Usage Log		Task Log		State Observation	
Resource Id	Resource	Resource List	Node List	Owner Id	Owner	Task Id	Start Time	End Time	Duration (Hours)	Avg	Min	Max			
[-] Cut2	Cut2	CutList	DefaultEntity_39	Order05	0	12/1/2016 8:00:05 AM	12/1/2016 4:18:05 PM	8.3	1	1	1				
	Cut1	CutList	DefaultEntity_38	Order04	0	12/1/2016 8:00:06 AM	12/1/2016 11:00:06 AM	3	1	1	1				
[-] Weld2	Weld2	WeldList	DefaultEntity_37	Order03	0	12/1/2016 8:00:15 AM	12/1/2016 10:18:15 AM	2.3	1	1	1				
[-] Resource2	Resource2		DefaultEntity_37	Order03	0	12/1/2016 8:00:15 AM	12/1/2016 10:18:15 AM	2.3	1	1	1				
[-] Weld1	Weld1	WeldList	DefaultEntity_36	Order01	0	12/1/2016 8:00:17 AM	12/1/2016 11:42:17 AM	3.7	1	1	1				
[-] Resource1	Resource1		DefaultEntity_36	Order01	0	12/1/2016 8:00:17 AM	12/1/2016 11:42:17 AM	3.7	1	1	1				
[-] Finish1	Finish1	FinishList	DefaultEntity_37	Order03	0	12/1/2016 10:18:22 AM	12/1/2016 4:18:22 PM	6	1	1	1				

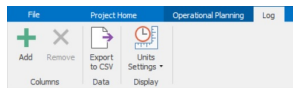
By selecting the + button within a particular Resource row, the Resource is expanded to include the detailed information about that resource and its state, as shown below.

[-] Finish		Order0007	10/3/2011 9:12:00 AM	10/3/2011 3:00:00 PM	348	1	1	1
Resource State Details								
[-] State		Auto State	Start Time	End Time	Duration (Minutes)			
[-] Processing		Busy	10/3/2011 9:12:00 AM	10/3/2011 12:00:00 PM	108			
[-] OffShift		OffShift	10/3/2011 12:00:00 PM	10/3/2011 1:00:00 PM	60			
[-] Processing		Busy	10/3/2011 1:00:00 PM	10/3/2011 3:00:00 PM	120			
✓								

Note: The small checkbox that can be seen at the bottom left of the Resource State Details, as well as on the Resource Usage Log displays any Filters that may have been set for the data.

Adding a Column to the Usage Log

When the Resource Usage Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Resource Usage Log.

For example, the Add button is available to add a column to the Resource Usage Log table. Additional columns can be shown within the 'Additional Details' hover tooltip for a Resource on the Gantt views, as well as used for categorizing the Entity Gantt. When a column is added to the Resource Usage Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Resource Usage Log Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource usage row in the log. Note that the expression may contain references to either the resource object or the owner object; if referencing the resource object, the keyword 'Object' must be used (e.g., Object.Capacity.ScheduledUtilization); if referencing the owner object, the keyword 'Owner' must be used (e.g., Owner.Entity.Priority).
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each resource usage row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Gantt Display Type	Indicates if this column should appear in one of the Gantt views. Options include None, ResourceGantt, and EntityGantt. It is important to note that the values shown in the ResourceGantt and EntityGantt will be the MOST RECENT values <u>in the log</u> for that resource or entity.
Use as Gantt Color	Specifies how this color expression should be used when drawing a Gantt chart.
Show in Gantt Tooltips	Specifies if this log expression should appear in tooltips in the Gantt views.
Show in Gantt Dropdowns	Specifies if this log expression should appear in the dropdowns in the Gantt views. This includes the "Group" and "Row" dropdowns on the ribbon, as well as the "Filter to Expression Column" sub-menu in the window's context menu.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user; others will be filtered out.

Note on Trait Filter property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

Note: Backslashes may be used for multiple levels (e.g., One\Two\Three).

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Resource Usage Log may also be exported to a *.csv file by using the Export to CSV button.

Example 1 - Adding a Column to the Resource Usage Log for Sorting the Resources

This example shows the addition of a column to the Resource Usage Log chart. This column has the Gantt Display Type property as 'ResourceGantt' so that the values are displayed within the Gantt chart as well. The Expression property is generated from the Resources table.

New column added to the `SchedulingDiscretePartsProduction.spx` example to specify a desired sorting column.

This table column can then be referenced within a new column in the Resource Usage Log as 'Resources.GanttOrdering' and the Resource Plan Gantt can be sorted based on these values.

Reference to the Resources table, GanttOrdering column

New column will appear in the Resource Plan Gantt

Click on Gantt Column to sort alphabetically (or numerically if numbers)

Click and move to right to expand

Example 2 - Adding a Column to the Resource Usage Log to Show Resource Utilization

This example shows the addition of a column to the Resource Usage Log chart that will be displayed in the Resource Plan Gantt. The Expression property references the resource 'Object'.

The Resource 'Object's scheduled utilization is calculated and shown in the column. The MOST RECENT values in the log will be shown for that resource.

Log Entry Logic:
A column defining a custom expression value to be recorded for each resource usage row in the log.

The Utilization column values are shown based on the MOST RECENT in the Usage Log for a given resource.

Note that the Gantt does NOT currently support this information for dynamic operators (operator).

Example 3 - Adding a Column to the Resource Usage Log to Group within the Entity Workflow Gantt

This example is similar to Examples 1 and 2 in that it shows the addition of a column to the Resource Usage Log chart. This column, however, will be used within the Entity Workflow Gantt. The GanttDisplayType property is 'EntityGantt'. Items within the Entity Workflow Gantt can now be grouped using the Owner Group property on the ribbon, which is set to the 'MaterialName' column.

Expression: `Math.If(Owner.Object.IsDefaultEntity, Owner.ManufacturingOrders.MaterialName, String.Empty)`

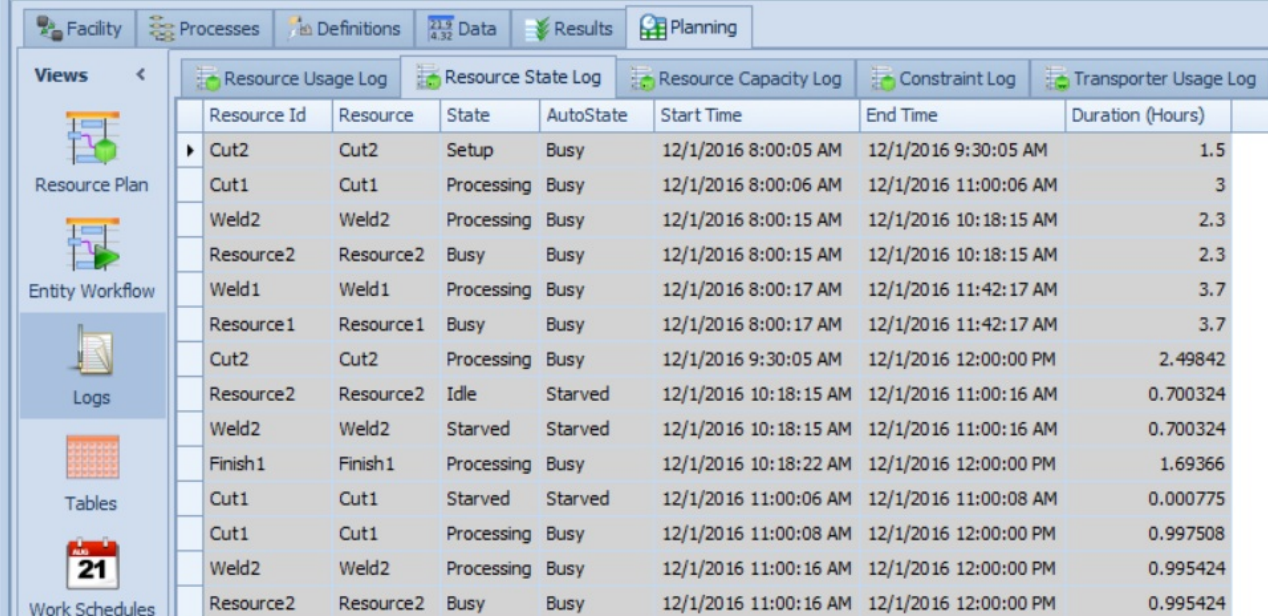
Log Entry Logic:
A column defining a custom expression value to be recorded for each resource usage row in the log.

Resource State Log - RPS

When the Resource State Log tab is selected, a smaller version of the Log ribbon and the standard Operational Planning ribbon are available. Additional columns may not be added to the Resource State Log; therefore, only the Unit Settings properties and the Export to CSV options are available from Resource State Log's Log ribbon.

The Resource State Log includes details of the resource states over time. The columns that are shown in the Resource State Log include:

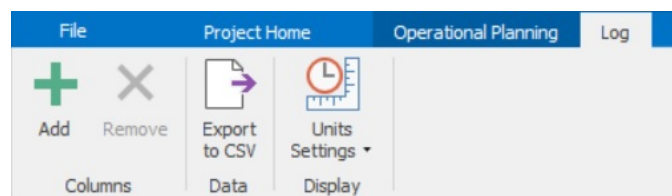
- **Resource Id** - The name of the resource that is utilized.
- **Resource** - The name of the resource that is utilized. This defaults to the Resource Id unless a *Display Name* is specified with the Resource.
- **State** - The specific state that the resource is in (e.g., Processing, Setup).
- **Auto State** - The auto state associated with the state (e.g., Busy).
- **Start Time** - The simulation time that the resource was seized.
- **End Time** - The simulation time that resource was released.
- **Duration** - The total time that the resource was utilized in that given state, as calculated by 'End Time – Start Time'.



Resource Id	Resource	State	AutoState	Start Time	End Time	Duration (Hours)
Cut2	Cut2	Setup	Busy	12/1/2016 8:00:05 AM	12/1/2016 9:30:05 AM	1.5
Cut1	Cut1	Processing	Busy	12/1/2016 8:00:06 AM	12/1/2016 11:00:06 AM	3
Weld2	Weld2	Processing	Busy	12/1/2016 8:00:15 AM	12/1/2016 10:18:15 AM	2.3
Resource2	Resource2	Busy	Busy	12/1/2016 8:00:15 AM	12/1/2016 10:18:15 AM	2.3
Weld1	Weld1	Processing	Busy	12/1/2016 8:00:17 AM	12/1/2016 11:42:17 AM	3.7
Resource1	Resource1	Busy	Busy	12/1/2016 8:00:17 AM	12/1/2016 11:42:17 AM	3.7
Cut2	Cut2	Processing	Busy	12/1/2016 9:30:05 AM	12/1/2016 12:00:00 PM	2.49842
Resource2	Resource2	Idle	Starved	12/1/2016 10:18:15 AM	12/1/2016 11:00:16 AM	0.700324
Weld2	Weld2	Starved	Starved	12/1/2016 10:18:15 AM	12/1/2016 11:00:16 AM	0.700324
Finish1	Finish1	Processing	Busy	12/1/2016 10:18:22 AM	12/1/2016 12:00:00 PM	1.69366
Cut1	Cut1	Starved	Starved	12/1/2016 11:00:06 AM	12/1/2016 11:00:08 AM	0.000775
Cut1	Cut1	Processing	Busy	12/1/2016 11:00:08 AM	12/1/2016 12:00:00 PM	0.997508
Weld2	Weld2	Processing	Busy	12/1/2016 11:00:16 AM	12/1/2016 12:00:00 PM	0.995424
Resource2	Resource2	Busy	Busy	12/1/2016 11:00:16 AM	12/1/2016 12:00:00 PM	0.995424

Adding a Column to the Resource State Log

When the Resource State Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Resource State Log.

For example, the Add button is available to add a column to the Resource State Log table. Additional columns can be used within Dashboard and Table reports for filtering. When a column is added to the Resource State Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Resource State Log Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource state row in the log. Note that the expression may contain references to the resource object, model variable or other expression; if

referencing the resource object, the keyword 'Object' must be used (e.g., Object.Capacity.ScheduledUtilization).

Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each resource state row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user; others will be filtered out.

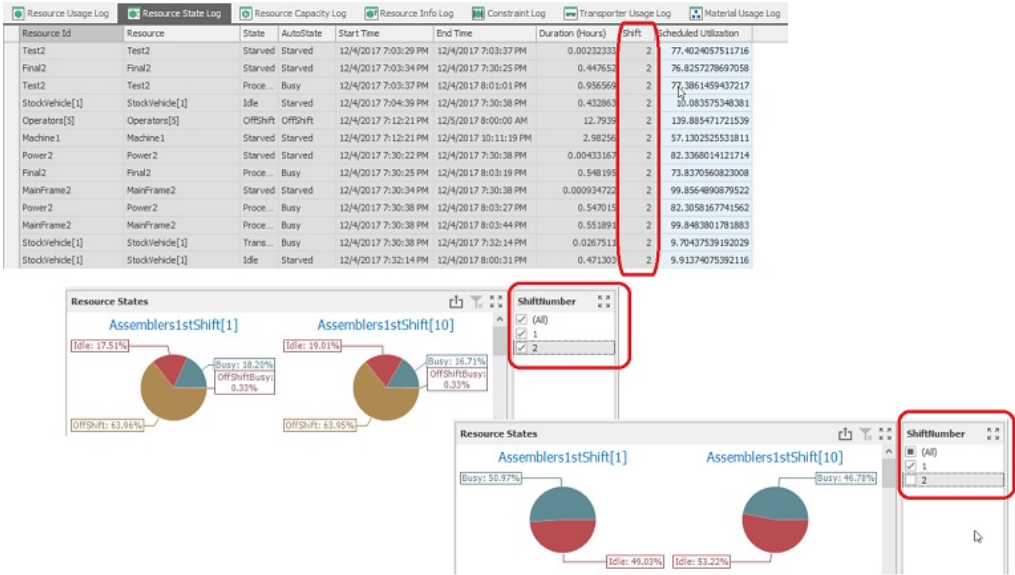
Note on *Trait Filter* property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Resource State Log may also be exported to a *.csv file by using the Export to CSV button.

Example 1 - Adding a Column to the Resource State Log for Filtering the Resource States by Shift

This example shows the addition of a column to the Resource State Log. The 'Shift' column uses a model variable defined/changed in process logic for the shift number. This value can then used in a Dashboard Report to graphically filter the utilization pie charts for the resource states.



Resource Capacity Log - RPS

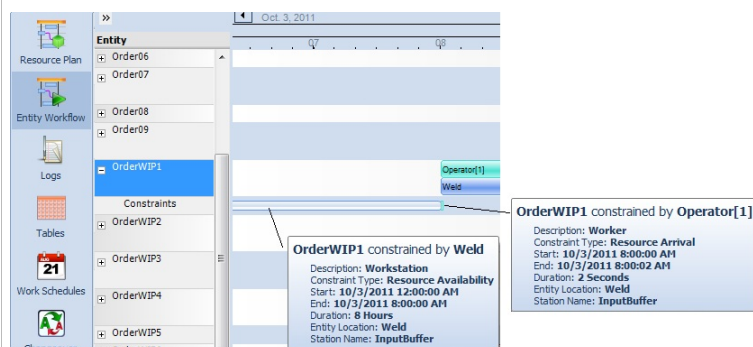
The Resource Capacity Log provides information detailing the resource capacity changes during the simulation run, based on various resource schedules. The columns within the log include:

- **Resource Id** - The name of the resource that is scheduled.
- **Resource** - The name of the resource that is scheduled. This defaults to the Resource Id unless a *Display Name* is specified with the Resource.
- **Scheduled** - The maximum capacity of the specified resource at the given time.
- **Allocated** - The number of resources that have been allocated or 'seized'.
- **Utilized** - The number of resources that are in a 'utilized' state. This may be less than the 'allocated' value when a failure occurs or when a part is in a fixed object and has seized the resource but is waiting for material or a secondary resource, for example.
- **Start Time** - The time that the resource capacity/allocation/utilization change started.
- **End Time** - The time that the resource capacity/allocation/utilization change ended.
- **Duration (Time Units)** - The total time that the capacity/allocation/utilization combination occurred, as calculated by 'End Time - Start Time' in default time units.

Resource Usage Log		Resource State Log		Resource Capacity Log		Constraint Log		Transporter Usage Log	
	Resource Id	Resource	Scheduled	Allocated	Utilized	Start Time	End Time	Duration (Hours)	
	Shape	Shape	0	0	0	10/3/2011 12:00:00 AM	10/3/2011 8:00:00 AM	480	
	Weld	Weld	0	0	0	10/3/2011 12:00:00 AM	10/3/2011 8:00:00 AM	480	
	Cut	Cut	0	0	0	10/3/2011 12:00:00 AM	10/3/2011 8:00:00 AM	480	
	Finish	Finish	0	0	0	10/3/2011 12:00:00 AM	10/3/2011 8:00:00 AM	480	
	Shape	Shape	1	0	0	10/3/2011 8:00:00 AM	10/3/2011 12:00:00 PM	240	
	Weld	Weld	1	1	0	10/3/2011 8:00:00 AM	10/3/2011 8:00:02 AM	0.0406667	
	Cut	Cut	1	1	1	10/3/2011 8:00:00 AM	10/3/2011 12:00:00 PM	240	
	Finish	Finish	1	1	1	10/3/2011 8:00:00 AM	10/3/2011 12:00:00 PM	240	
	Operator[1]	Operator[1]	1	1	1	10/3/2011 8:00:00 AM	10/19/2011 9:38:29 AM	23138.5	
	Weld	Weld	1	1	1	10/3/2011 8:00:02 AM	10/3/2011 12:00:00 PM	239.959	

In the above example, the Shape, Weld, Cut and Finish resources have no scheduled capacity, based on their work schedules. Thus, the allocated and utilized values are also zero. Then, at 8:00 am, the resources all have scheduled capacity of 1. Note the Weld resource that has a scheduled capacity of 1, an allocated value of 1, yet a utilized value of 0 for a very short period of time. This corresponds to a wait for material or secondary resource. Notice that the Weld resource has allocated and utilized values of 1 when the material/resource arrive.

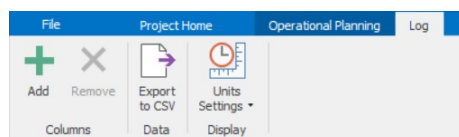
In the corresponding Entity Workflow gantt below, note that the OrderWIP1 entity that is processing at Weld is first constrained by the availability of the Weld resource until 8 am. Then, there is a very short period of time where the entity is waiting for the worker to arrive for processing. Once the worker arrives, the Weld resource goes into a 'utilized' state, as seen in the Resource Capacity Log above.



Note: The small checkbox that can be seen at the bottom left of the Resource Capacity Log displays any Filters that may have been set for the data.

Adding a Column to the Resource Capacity Log

When the Resource Capacity Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Resource Capacity Log.

For example, the Add button is available to add a column to the Resource Capacity Log table. Additional columns can be used within Dashboard and Table reports for filtering. When a column is added to the Resource Capacity Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Resource Capacity Log Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource state row in the log. Note that

	the expression may contain references to the resource object, model variable or other expression; if referencing the resource object, the keyword 'Object' must be used (e.g., Object.Capacity.ScheduledUtilization).
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each resource capacity row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user, others will be filtered out.

Note on *Trait Filter* property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Resource Capacity Log may also be exported to a *.csv file by using the Export to CSV button.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

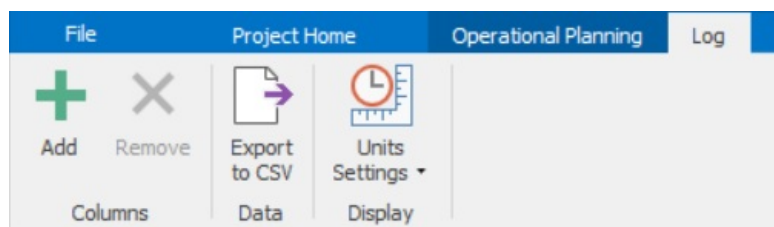
Resource Info Log - RPS

The Resource Info Log provides information for each of the resources in the system. The columns within the log include:

- **Resource Id** - The name of the resource in the system.
- **Resource** - The name of the resource system. This defaults to the Resource Id unless a *Display Name* is specified with the Resource.
- **Display Category** – The display category as defined within the resource object's Advanced Options properties.

Adding a Column to the Resource Info Log

When the Resource Info Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Resource Info Log.

For example, the Add button is available to add a column to the Resource Info Log table. When a column is added to the Resource Info Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Resource Info Log Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource info row in the log. Note that the expression may contain references to the resource object but there is no 'owner'; if referencing the resource object, the keyword 'Object' must be used (e.g., Object.Capacity).
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user; others will be filtered out.

Note on Trait Filter property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Resource Info Log may also be exported to a *.csv file by using the Export to CSV button.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Constraint Log - RPS

When the Constraint Log tab is selected, a smaller version of the Log ribbon and the standard Operational Planning ribbon are available. Additional columns may not be added to the Constraint Log; therefore, only the Unit Settings properties and the Export to CSV options are available from Constraint Log's Log ribbon.

The Constraint Log includes details of the constraints to the entity over time. The columns that are shown in the Constraint Log include:

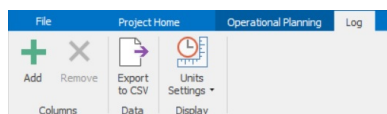
- **Entity Id** - The name of the entity that is being tracked.
- **Entity** - The name of the entity that is being tracked. This defaults to the Entity Id unless a *Display Name* is specified with the Entity.
- **Task Id** - The specific task, if any, associated with the constraint.
- **Facility Location** - The name of the Server / Object from the facility window where the entity is being constrained.
- **Station** - The area within the Facility Location where the entity is being constrained (for example, inputbuffer).
- **Constraint Type** - The type of constraint that is causing the delay. This includes Resource Availability, Resource Arrival, Material Availability, TransporterPickup, Destination Availability, StationAvailability, Link Availability and Node Availability. Resource Availability means the entity is waiting for capacity to become available of the given constraint item. Resource Arrival indicates that the entity is waiting for a vehicle or worker to arrive to its location for processing. Material Availability means that a material required by the entity is not available for consumption or reservation. Transporter Pickup means that the entity has been allocated the transporter (vehicle or worker) and is waiting for the transporter to arrive to the entity location for transport to its destination. Destination Availability means that the entity is waiting for a destination (typically from a list of destinations) to become available. Station availability indicates that the entity is waiting for a specific station location to become available. Link Availability indicates that the entity is waiting to move onto a link, such as a conveyor. Node availability means that the entity is waiting for entry into a node location that has a fixed traveler capacity.
- **Constraint Item Id** - The name of the constraint at the specified facility location. The constraint item may be the same as the facility location if the entity is waiting for the resource within that facility location. For example, the entity may be located at the Weld facility location and waiting for the Weld resource. The constraint item will be a different name than the facility location if the entity is waiting for transportation from that location. For example, the entity may be located at the Weld facility location and waiting for the Operator. If the constraint is a material, the *Lot ID* (if applicable) will also be shown in addition to the material name. See additional notes below for more information on constraint items.
- **Constraint Item** - The name of the constraint at the specified facility location. This defaults to the Constraint Item Id unless a *Display Name* is specified with the Constraint Item.
- **Constraint Description** - The type of constraint at the specified facility location. This specifies the type of constraint item for which the entity is waiting. It is a descriptor for the object type, and includes Server, Worker, Vehicle, Material, etc.
- **Start Time** - The simulation time that the entity began to wait for the constraint item (either for resource capacity or resource arrival).
- **End Time** - The simulation time that entity was allocated the resource (resource capacity) or that the resource arrived to the entity location (resource arrival).
- **Duration** - The total time that the entity was waiting, as calculated by 'Wait End Time - Wait Start Time'.

As with all tables, the Constraint Log can be sorted on any column and/or filtered by column.

Entity Id	Entity	Task Id	Facility Location	Station	Constraint Type	Constraint Item Id	Constraint Item	Constraint Description	Start Time	End Time	Duration (Minutes)
SickPatient.28	SickPatient.28	2	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:00:02 AM	6/8/2015 12:00:07 AM	0.0695833
SickPatient.28	SickPatient.28	3	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:01:07 AM	6/8/2015 12:01:11 AM	0.0756667
SickPatient.29	SickPatient.29	4	WaitingArea	Processing	Resource Availability	Room1	Resource	Resource	6/8/2015 12:02:32 AM	6/8/2015 12:03:16 AM	0.72125
SickPatient.29	SickPatient.29	5	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:03:16 AM	6/8/2015 12:03:20 AM	0.0695833
SickPatient.29	SickPatient.29	6	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:04:20 AM	6/8/2015 12:04:24 AM	0.0756667
SickPatient.30	SickPatient.30	7	WaitingArea	Processing	Resource Availability	Room1	Resource	Resource	6/8/2015 12:05:02 AM	6/8/2015 12:06:29 AM	1.4425
SickPatient.30	SickPatient.30	8	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:06:29 AM	6/8/2015 12:06:33 AM	0.0695833
SickPatient.31	SickPatient.31	10	WaitingArea	Processing	Resource Availability	Room1	Resource	Resource	6/8/2015 12:07:32 AM	6/8/2015 12:09:42 AM	2.16375
SickPatient.30	SickPatient.30	9	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:07:33 AM	6/8/2015 12:07:38 AM	0.0756667
SickPatient.31	SickPatient.31	11	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:09:42 AM	6/8/2015 12:09:46 AM	0.0695833
SickPatient.32	SickPatient.32	13	WaitingArea	Processing	Resource Availability	Room1	Resource	Resource	6/8/2015 12:10:02 AM	6/8/2015 12:12:56 AM	2.885
SickPatient.31	SickPatient.31	12	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:10:46 AM	6/8/2015 12:10:51 AM	0.0756667
SickPatient.33	SickPatient.33	16	WaitingArea	Processing	Resource Availability	Room1	Resource	Resource	6/8/2015 12:12:32 AM	6/8/2015 12:16:09 AM	3.60625
SickPatient.32	SickPatient.32	14	WaitingArea	Processing	Resource Arrival	Nurse[1]	Worker	Worker	6/8/2015 12:12:56 AM	6/8/2015 12:13:00 AM	0.0695833

Adding a Column to the Constraint Log

When the Constraint Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Constraint Log.

For example, the Add button is available to add a column to the Constraint Log. Additional columns can be used within Dashboard and Table reports for filtering. When a column is added to the Constraint Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Constraint Log Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource state row in the log. Note that the expression may contain references to the resource object, model variable or other expression; if referencing the resource object, the keyword 'Object' must be used (e.g., Object.Capacity.ScheduledUtilization).
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each Constraint row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user; others will be filtered out.

Note on *Trait Filter* property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their

own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Constraint Log may also be exported to a *.csv file by using the Export to CSV button.

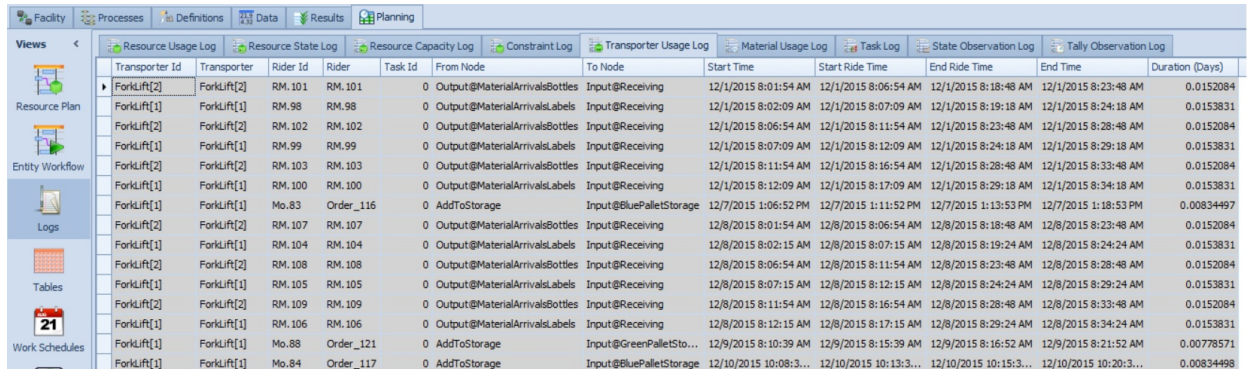
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Transporter Usage Log - RPS

The Transporter Usage Log provides information detailing each transporter and the entity / order that utilized it. There are five automatic columns that are provided with the log. These include:

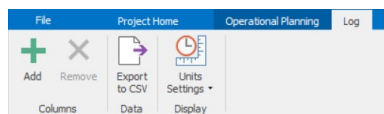
- **Transporter Id** - The name of the transporter that is utilized.
- **Transporter** - The name of the transporter that is utilized. This defaults to the Transporter Id unless a *Display Name* is specified with the Transporter.
- **Rider Id** - The specific entity or order number transporting (riding) on the transporter.
- **Rider** - The specific entity or order number transporting (riding) on the transporter. This defaults to the Rider Id unless a *Display Name* is specified with the entity or object.
- **Task Id** - The specific task, if any, associated with the transporter move.
- **From Node** - The transfer node where the transporter picked up the rider.
- **To Node** - The node where the transporter dropped off the rider.
- **Start Time** - The time that the entity was physically on the transporter. This does not include transport to the entity or load time.
- **Start Ride Time** - The time that the entity starts riding on the transporter once loading has been completed.
- **End Ride Time** - The time that the entity arrives at the destination prior to unloading.
- **End Time** - The time that the entity released the transporter.
- **Duration** - The total time that the entity was on the transporter, as calculated by 'End Time - Start Time' in default time units.



Transporter Id	Transporter	Rider Id	Rider	Task Id	From Node	To Node	Start Time	Start Ride Time	End Ride Time	End Time	Duration (Days)
ForkLift[2]	ForkLift[2]	RM. 101	RM. 101	0	Output@MaterialArrivalsBottles	Input@Receiving	12/1/2015 8:01:54 AM	12/1/2015 8:06:54 AM	12/1/2015 8:18:48 AM	12/1/2015 8:23:48 AM	0.0152084
ForkLift[1]	ForkLift[1]	RM. 98	RM. 98	0	Output@MaterialArrivalsLabels	Input@Receiving	12/1/2015 8:02:09 AM	12/1/2015 8:07:09 AM	12/1/2015 8:19:18 AM	12/1/2015 8:24:18 AM	0.0153831
ForkLift[2]	ForkLift[2]	RM. 102	RM. 102	0	Output@MaterialArrivalsBottles	Input@Receiving	12/1/2015 8:06:54 AM	12/1/2015 8:11:54 AM	12/1/2015 8:23:48 AM	12/1/2015 8:28:48 AM	0.0152084
ForkLift[1]	ForkLift[1]	RM. 99	RM. 99	0	Output@MaterialArrivalsLabels	Input@Receiving	12/1/2015 8:07:09 AM	12/1/2015 8:12:09 AM	12/1/2015 8:24:18 AM	12/1/2015 8:29:18 AM	0.0153831
ForkLift[2]	ForkLift[2]	RM. 103	RM. 103	0	Output@MaterialArrivalsBottles	Input@Receiving	12/1/2015 8:11:54 AM	12/1/2015 8:16:54 AM	12/1/2015 8:28:48 AM	12/1/2015 8:33:48 AM	0.0152084
ForkLift[1]	ForkLift[1]	RM. 100	RM. 100	0	Output@MaterialArrivalsLabels	Input@Receiving	12/1/2015 8:12:09 AM	12/1/2015 8:17:09 AM	12/1/2015 8:29:18 AM	12/1/2015 8:34:18 AM	0.0153831
ForkLift[1]	ForkLift[1]	Mo.83	Order_116	0	AddToStorage	Input@BluePalletStorage	12/7/2015 1:06:52 PM	12/7/2015 1:11:52 PM	12/7/2015 1:13:53 PM	12/7/2015 1:18:53 PM	0.00834497
ForkLift[2]	ForkLift[2]	RM. 107	RM. 107	0	Output@MaterialArrivalsBottles	Input@Receiving	12/8/2015 8:01:54 AM	12/8/2015 8:06:54 AM	12/8/2015 8:18:48 AM	12/8/2015 8:23:48 AM	0.0152084
ForkLift[1]	ForkLift[1]	RM. 104	RM. 104	0	Output@MaterialArrivalsLabels	Input@Receiving	12/8/2015 8:02:15 AM	12/8/2015 8:07:15 AM	12/8/2015 8:19:24 AM	12/8/2015 8:24:24 AM	0.0153831
ForkLift[2]	ForkLift[2]	RM. 108	RM. 108	0	Output@MaterialArrivalsBottles	Input@Receiving	12/8/2015 8:06:54 AM	12/8/2015 8:11:54 AM	12/8/2015 8:23:48 AM	12/8/2015 8:28:48 AM	0.0152084
ForkLift[1]	ForkLift[1]	RM. 105	RM. 105	0	Output@MaterialArrivalsLabels	Input@Receiving	12/8/2015 8:07:15 AM	12/8/2015 8:12:15 AM	12/8/2015 8:24:24 AM	12/8/2015 8:29:24 AM	0.0153831
ForkLift[2]	ForkLift[2]	RM. 109	RM. 109	0	Output@MaterialArrivalsBottles	Input@Receiving	12/8/2015 8:11:54 AM	12/8/2015 8:16:54 AM	12/8/2015 8:28:48 AM	12/8/2015 8:33:48 AM	0.0152084
ForkLift[1]	ForkLift[1]	RM. 106	RM. 106	0	Output@MaterialArrivalsLabels	Input@Receiving	12/8/2015 8:12:15 AM	12/8/2015 8:17:15 AM	12/8/2015 8:29:24 AM	12/8/2015 8:34:24 AM	0.0153831
ForkLift[1]	ForkLift[1]	Mo.88	Order_121	0	AddToStorage	Input@GreenPalletSto...	12/9/2015 8:10:39 AM	12/9/2015 8:15:39 AM	12/9/2015 8:16:52 AM	12/9/2015 8:21:52 AM	0.00778571
ForkLift[1]	ForkLift[1]	Mo.84	Order_117	0	AddToStorage	Input@BluePalletStorage	12/10/2015 10:08:3...	12/10/2015 10:13:3...	12/10/2015 10:15:3...	12/10/2015 10:20:3...	0.00834498

Adding a Column to the Transporter Usage Log

When the Transporter Usage Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Transporter Usage Log.

For example, the Add button is available to add a column to the Transporter Usage Log. Additional columns can be used within Dashboard and Table reports for filtering. When a column is added to the Constraint Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Transporter Usage Log Column**.

Property	Description
Expression	The expression value to be recorded in this column for each resource state row in the log. NNote that the expression may contain references to either the resource object or the owner object; if referencing the resource object, the keyword 'Object' must be used (e.g., Object.Capacity.ScheduledUtilization); if referencing the owner object, the keyword 'Owner' must be used (e.g., Owner.Entity.Priority).
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each Transporter Usage row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user; others will be filtered out.

Note on *Trait Filter* property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Transporter Usage Log may also be exported to a *.csv file by using the Export to CSV button.

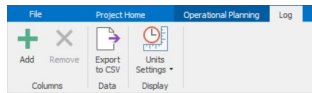
Material Usage Log - RPS

The Material Usage Log displays the log of material production and consumption over time of Material elements which had Log Material Usage set to "True". There are eight automatic columns that are provided with the log. These include:

- **Time** - The simulation date and time that a material quantity changed.
- **Material** - The name of the material that has been either produced or consumed.
- **Lot Id** - The name of the optional string *Lot ID* that has been either produced or consumed.
- **Entity Id** - The specific entity that produced or consumed the material quantity.
- **Entity** - The specific entity that produced or consumed the material quantity. This defaults to the Entity Id unless a Display Name is specified with the Entity.
- **Site Id** - The specific inventory site location, if any, associated with the material production or consumption.
- **Task Id** - The specific task, if any, associated with the material production or consumption.
- **BOM Group** - The name of the BOM Group that has been either produced or consumed.
- **BOM** - The name of the BOM that has been either produced or consumed.
- **Component Id** - The name of the Component Id that has been either produced or consumed.
- **Quantity** - The quantity of material that was utilized. A positive quantity is a production of the material, while a negative value is a consumption of the material.
- **Stock Level** - The amount of material left after the quantity specified was added or subtracted from the total.
- **Lot Stock Level** - The amount of material of a particular Lot ID that is left after the quantity specified was added or subtracted from the total.

Adding a Column to the Material Usage Log

When the Material Usage Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Material Usage Log.

For example, the Add button is available to add a column to the Material Usage Log table. Additional columns can be used within Dashboard and Table reports for filtering. When a column is added to the Resource Capacity Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Material Usage Log Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource state row in the log.
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each Material Usage row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user; others will be filtered out.

Note on *Trait Filter* property - Within Simio Portal, user would have a trait defined called "XXTrait". Each user then has their own value for comma separated values assigned to "XXTrait" for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Material Usage Log may also be exported to a ".csv" file by using the Export to CSV button.

Note that the Material Usage Log tracks "actual" consumption, not "requested" consumption. So if you ask for 10 units of material, but only 4 are available, you'll get an entry saying quantity = 4. Then if 10 more units get produced at some time later, you'll get "another" entry with = 6, fulfilling the consumption request.

In the example shown below, note that the first two entries at the start of the simulation run indicate that the *Initial Quantity* for each of these materials had been specified. For several minutes after that, the quantities are reduced of each material. Note also that at time 12:25 AM, an additional quantity of 80 Nails is produced, indicating a replenishment type of logic. This example log was taken directly from the SimBit ServerWithMaterialConsumptionAndReplenish.spx. The SimBit was modified slightly by changing the *Log Material Usage* property to "True" for both Wood and Nails Materials elements. In this example, there are no *Lot ID* values specified for any materials.

Example 1 - Basic Materials

Time	Material	Lot Id	Entity Id	Entity	Site Id	Site	Task Id	Quantity	Stock Level	Lot Stock Level
1/1/2008 12:00:00 AM	Wood							0	30	30
1/1/2008 12:00:00 AM	Nails							0	60	60
1/1/2008 12:00:04 AM	Wood		DefaultEntity.11	DefaultEntity.11				0	-2	28
1/1/2008 12:00:04 AM	Nails		DefaultEntity.11	DefaultEntity.11				0	-8	52
1/1/2008 12:01:04 AM	Wood		DefaultEntity.12	DefaultEntity.12				0	-2	26
1/1/2008 12:01:04 AM	Nails		DefaultEntity.12	DefaultEntity.12				0	-8	44
1/1/2008 12:02:04 AM	Wood		DefaultEntity.13	DefaultEntity.13				0	-2	24
1/1/2008 12:02:04 AM	Nails		DefaultEntity.13	DefaultEntity.13				0	-8	36
1/1/2008 12:03:04 AM	Wood		DefaultEntity.14	DefaultEntity.14				0	-2	22
1/1/2008 12:03:04 AM	Nails		DefaultEntity.14	DefaultEntity.14				0	-8	28
1/1/2008 12:04:04 AM	Wood		DefaultEntity.15	DefaultEntity.15				0	-2	20
1/1/2008 12:04:04 AM	Nails		DefaultEntity.15	DefaultEntity.15				0	-8	20
1/1/2008 12:05:04 AM	Wood		DefaultEntity.16	DefaultEntity.16				0	-2	18
1/1/2008 12:05:04 AM	Nails		DefaultEntity.16	DefaultEntity.16				0	-8	12
1/1/2008 12:05:34 AM	Nails							0	80	92

The example shown below is also a modified version of the SimBit ServerWithMaterialConsumptionAndReplenish.spx. In this example, the original DefaultEntity object has been replaced a ModelEntity object named "Order" that has a Display Name of "Order". Additionally, a *Lot ID* property (string) has been specified for the materials. In this case, with the Wood and Nails materials, the *Initial Quantities* values are broken up into "W11" and "W12" *Lot ID* values. The top level bill of material, BookShelfMaterials, when specified within the Server object, also indicates the appropriate *Lot ID* string property value (such as through an table or math function) that is required for that particular order. Note that the log on the left below shows all materials and their respective *Lot ID* values. The log on the right was filtered on *Lot ID* to show only the "W12" material. The Stock Level column indicates the total amount of inventory of the particular material name, whereas the Lot Stock Level indicates the specific inventory for the *Lot ID*.

Example 2 - Materials with Pegging (Lot ID)

Time	Material	Lot Id	Entity Id	Entity	Site Id	Site	Task Id	Quantity	Stock Level	Lot Stock Level
1/1/2008 12:00:00 AM	Wood							0	0	0
1/1/2008 12:00:00 AM	Wood	W11						0	15	15
1/1/2008 12:00:00 AM	Wood	W12						0	15	30
1/1/2008 12:00:00 AM	Nails							0	0	0
1/1/2008 12:00:00 AM	Nails	W11						0	30	30
1/1/2008 12:00:00 AM	Nails	W12						0	30	60
1/1/2008 12:00:04 AM	Wood	W12	Order.11	Order				0	-2	28
1/1/2008 12:00:04 AM	Nails	W12	Order.11	Order				0	-8	52
1/1/2008 12:01:38 AM	Wood	W12	Order.12	Order				0	-2	26
1/1/2008 12:01:38 AM	Nails	W12	Order.12	Order				0	-8	44
1/1/2008 12:02:26 AM	Wood	W12	Order.13	Order				0	-2	24
1/1/2008 12:02:26 AM	Nails	W12	Order.13	Order				0	-8	36
1/1/2008 12:03:11 AM	Wood	W11	Order.14	Order				0	-2	22
1/1/2008 12:03:11 AM	Nails	W11	Order.14	Order				0	-8	28
1/1/2008 12:05:21 AM	Wood	W12	Order.15	Order				0	-2	20
1/1/2008 12:05:21 AM	Nails	W12	Order.15	Order				0	-6	22

Time	Material	Lot Id	Entity Id	Entity	Site Id	Site	Task Id	Quantity	Stock Level	Lot Stock Level
1/1/2008 12:00:00 AM	Wood	W12						0	15	30
1/1/2008 12:00:00 AM	Nails	W12						0	30	60
1/1/2008 12:00:04 AM	Wood	W12	Order.11	Order				0	-2	28
1/1/2008 12:00:04 AM	Nails	W12	Order.11	Order				0	-8	52
1/1/2008 12:01:38 AM	Wood	W12	Order.12	Order				0	-2	26
1/1/2008 12:01:38 AM	Nails	W12	Order.12	Order				0	-8	44
1/1/2008 12:02:26 AM	Wood	W12	Order.13	Order				0	-2	24
1/1/2008 12:02:26 AM	Nails	W12	Order.13	Order				0	-8	36
1/1/2008 12:05:21 AM	Wood	W12	Order.15	Order				0	-2	20
1/1/2008 12:05:21 AM	Nails	W12	Order.15	Order				0	-6	22

Filtered on Lot ID "W12"

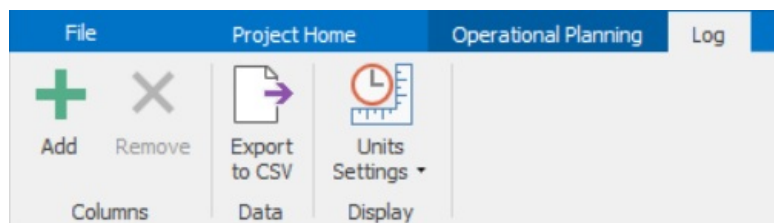
Inventory Review Log - RPS

The Inventory Review Log provides information on the inventory review events. This can be enabled on either the Inventory or Material element. The columns provided with the log include:

- Time - The simulation date and time that a inventory review event occurred.
- Material - The name of the material that has been reviewed.
- SiteId - The specific inventory site location, if any, associated with the inventory review.
- Site - The reference to the inventory review event at the specified site.
- QuantityInStock - The current quantity of material present in the inventory.
- QuantityOnOrder - The current quantity of material that has been ordered to replenish the inventory but has not yet been received.
- QuantityBackordered - The current quantity of material required to satisfy all material consumption requests waiting in the inventory's allocation queue.
- QuantityReserved - The current quantity of material from the inventory that is reserved.
- InventoryPosition - The inventory's quantity in stock plus quantity on order minus quantity backordered minus quantity reserved (excluding reserved units already backordered).
- QualifiedSpikeDemand - The current qualified spike demand value for calculating the net flow position if using a DDMRP replenishment policy.
- NetFlowPosition - Is equal to the quantity in stock plus quantity on order minus quantity backordered minus the qualified spike demand. Used to determine the available stock situation based on color-coded buffer zones (red, yellow, and green) if using a DDMRP replenishment policy.
- RedZoneSize - The current size of the buffer red zone if using a DDMRP replenishment policy.
- YellowZoneSize - The current size of the buffer yellow zone if using a DDMRP replenishment policy.
- GreenZoneSize - The current size of the buffer green zone if using a DDMRP replenishment policy.
- RecommendedOrderQuantity - The recommended quantity for the next order calculated by the selected replenishment policy.

Adding a Column to the Log

When the Inventory Review Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Inventory Review Log.

For example, the Add button is available to add a column to the Inventory Review Log table. When a column is added to the Resource Usage Log, there are several properties that may be specified for the column, including:

Note: Any custom expression columns referencing table data should use the material or inventory element's table row references.

Listed below are the properties of a **Log Expression Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource usage row in the log. Note that the expression may contain references to either the resource object or the owner object; if referencing the resource object, the keyword 'Object' must be used (e.g., Object.Capacity.ScheduledUtilization); if referencing the owner object, the keyword 'Owner' must be used (e.g., Owner.Entity.Priority).
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each resource usage row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user, others will be filtered out.

Note on *Trait Filter* property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Task Log - RPS

The Task Log displays the list of Task Sequence related Tasks completed over time for entities processed through objects using Task Sequence type processing. There are nine columns that are provided with the Task Log, including:

- **Task Id** - The unique identifier for each task performed during the simulation. This Task Id is also used within a number of other logs, including Resource Usage Log, Transporter Usage Log, Material Usage Log, Constraint Log and Task State Log. Note that the Task Id has a '+' to expand the task information. This provides additional information on the task states, which includes executing, constrained and suspended.
- **Entity Id** - The specific entity that completed the task.
- **Entity** - The entity type's *Display Name* that completed the task. If no *Display Name* is specified on the instance of the ModelEntity object, this will be the same as the Entity Id field.
- **Facility Location** - The name of the object location at which the task is performed.
- **Station** - The station location at which the task is performed.
- **Task Sequence** - The name of the Task Sequence element within which the task is specified. For all Standard Library object processing tasks, this will be 'ProcessingTaskSequence'.
- **Task** - The name of the task that was completed.
- **Start Time** - The start time of the task that was completed.
- **End Time** - The end time of the task that was completed.
- **Duration** - The total time that the entity was processed through the specified task.

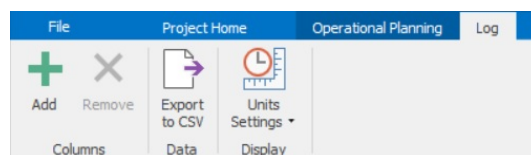
Task Id	Entity Id	Entity	Facility Location	Station	Task Sequence	Task	Start Time	End Time	Duration (Hours)
6716	MFGOrder.849	Order_SW_3	MainFrame1	Processing	ProcessingTaskSequence	Consume	12/7/2017 12:31:42...	12/7/2017 12:31:54...	0.00333333
6718	MFGOrder.849	Order_SW_3	MainFrame1	Processing	ProcessingTaskSequence	Process	12/7/2017 12:31:54...	12/7/2017 1:31:09 PM	0.987491
6735	MFGOrder.849	Order_SW_3	Power1	Processing	ProcessingTaskSequence	Process	12/7/2017 1:31:13 PM	12/7/2017 2:02:04 PM	0.514128
6753	MFGOrder.849	Order_SW_3	Final1	Processing	ProcessingTaskSequence	Process	12/7/2017 2:02:07 PM	12/7/2017 2:35:15 PM	0.552374
6769	MFGOrder.849	Order_SW_3	Test1	Processing	ProcessingTaskSequence	Process	12/7/2017 2:35:19 PM	12/7/2017 3:32:24 PM	0.95119
6771	MFGOrder.849	Order_SW_3	Test1	Processing	ProcessingTaskSequence	Produce	12/7/2017 3:32:24 PM	12/7/2017 3:32:36 PM	0.00333333

The Task Id can be expanded using the '+' to view the states of each particular task. This information is also available within the Task State Log. A task is started when the entity enters the object and begins the processing tasks for that object. The task may, however, be **Constrained** if a resource (including worker/vehicle) or material specified for the task is not currently available. Once all resources and materials have been allocated, the task will start the **Executing** state. If the main or secondary resource goes off-shift and the *Off Shift Rule* is 'Suspend Processing', the task state becomes **Suspended**. These task states can also be seen graphically on the Gantt charts. See the example below that shows the MainFrame1 task 'Process' expanded to show that the task was constrained for an amount of time prior to executing.

Resource Usage Log										Resource State Log										Resource Capacity Log										Constraint Log										Transporter Usage Log										Material Usage Log										Task Log																													
Task Id	Entity Id	Entity	Facility Location	Station	Task Sequence	Task	Start Time	End Time	Duration (Hours)																																																																																
6716	MFGOrder.849	Order_SW_3	MainFrame1	Processing	ProcessingTaskSequence	Consume	12/7/2017 12:31:42...	12/7/2017 12:31:54...	0.00333333																																																																																
6718	MFGOrder.849	Order_SW_3	MainFrame1	Processing	ProcessingTaskSequence	Process	12/7/2017 12:31:54...	12/7/2017 1:31:09 PM	0.987491																																																																																
Task State Details																																																																																									
State		Start Time		End Time		Duration (Hours)																																																																																			
Constrained		12/7/2017 12:31:54 PM		12/7/2017 1:01:09 PM		0.487491																																																																																			
Executing		12/7/2017 1:01:09 PM		12/7/2017 1:31:09 PM		0.5																																																																																			
<input checked="" type="checkbox"/>																																																																																Edit Filter									
6735	MFGOrder.849	Order_SW_3	Power1	Processing	ProcessingTaskSequence	Process	12/7/2017 1:31:13 PM	12/7/2017 2:02:04 PM	0.514128																																																																																
6753	MFGOrder.849	Order_SW_3	Final1	Processing	ProcessingTaskSequence	Process	12/7/2017 2:02:07 PM	12/7/2017 2:35:15 PM	0.552374																																																																																
6769	MFGOrder.849	Order_SW_3	Test1	Processing	ProcessingTaskSequence	Process	12/7/2017 2:35:19 PM	12/7/2017 3:32:24 PM	0.95119																																																																																
6771	MFGOrder.849	Order_SW_3	Test1	Processing	ProcessingTaskSequence	Produce	12/7/2017 3:32:24 PM	12/7/2017 3:32:36 PM	0.00333333																																																																																

Adding a Column to the Task Log

When the Task Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Task Log. For example, the Add button is available to add a column to the Task Log table. Additional columns can be shown within the 'Additional Details' hover tooltip for a Task on the Gantt views, as well as used for categorizing the Entity Gantt. When a column is added, there are several properties that may be specified for the column, as seen below. Any custom expression columns may be then used within the Resource Plan Gantt or Entity Workflow Gantt to sort the tasks specified by Task Row and/or Task Group (see Gantt ribbon on those Gantt charts). An example of using these task categories is shown on the [Entity Workflow](#) page. Additionally, the [SchedulingBicycleAssembly](#) and [SchedulingBatchBeverageProduction](#) examples have additional Task Log columns for use within the Task Row and Task Group categories in the Gantt charts.

Listed below are the properties of a **Task Log Column**:

Property	Description
----------	-------------

Expression	The expression value to be recorded in this column for each task row in the log. Note that the expression may contain references to either the entity that started the task, the token performing the task or a table reference associated with task.
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each task row in the log. If 'StartTimeValue', then the expression's value at the start time of the task is logged. If 'EndTimeValue', then the expression's value at the end time of the task is logged. If 'ValueChange', then the expression's change in value from the start of the task to the end of the task is logged.
Show in Gantt Tooltips	Specifies if this log expression should appear in tooltips in the Gantt views.
Show in Gantt Dropdowns	Specifies if this log expression should appear in the dropdowns in the Gantt views. This includes the "Group" and "Row" dropdowns on the ribbon, as well as the "Filter to Expression Column" sub-menu in the window's context menu.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible tot the user, others will be filtered out.

Note on *Trait Filter* property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

Backslashes may be used for multiple levels (e.g., One\Two\Three).

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Task State Log - RPS

The Task State Log displays the list of Task States over time for entities processed through objects using Task Sequence type processing. There are seven columns that are provided with the Task State Log, including:

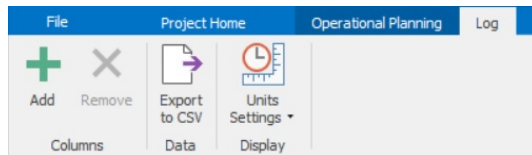
- **Task Id** - The unique identifier for each task performed during the simulation. This Task Id is also used within a number of other logs, including Resource Usage Log, Transporter Usage Log, Material Usage Log, Constraint Log and Task Log.
- **State** - The state of the task, which may include executing (in the task delay), suspended (due to resource state change) or constrained (cannot process yet due to resource or material constraint).
- **Entity Id** - The specific entity that was in the task state.
- **Entity** - The entity type's *Display Name* that was in the task state. If no *Display Name* is specified on the instance of the ModelEntity object, this will be the same as the Entity Id field.
- **Start Time** - The start time of the task state.
- **End Time** - The end time of the task state.
- **Duration** - The total time that the entity was in the specified state.

Resource Usage Log	Resource State Log	Resource Capacity Log	Constraint Log	Transporter Usage Log	Material Usage Log	Task Log	Task State Log
Task Id	State	Entity Id	Entity	Start Time	End Time	Duration (Hours)	
582	Executing	MFGOrder.757	Order_SG_1	12/4/2017 8:00:12 AM	12/4/2017 8:00:24 AM	0.00333333	
583	Executing	MFGOrder.756	Order_SG_1	12/4/2017 8:00:12 AM	12/4/2017 8:00:24 AM	0.00333333	
584	Constrained	MFGOrder.757	Order_SG_1	12/4/2017 8:00:24 AM	12/4/2017 8:00:30 AM	0.00179889	
585	Constrained	MFGOrder.756	Order_SG_1	12/4/2017 8:00:24 AM	12/4/2017 8:00:30 AM	0.00152778	
586	Executing	MFGOrder.748	Order_GFHB_1	12/4/2017 8:00:25 AM	12/4/2017 8:12:25 AM	0.2	
587	Executing	MFGOrder.744	Order_GFFRM_1	12/4/2017 8:00:26 AM	12/4/2017 8:12:26 AM	0.2	
589	Executing	MFGOrder.727	Order_FRM_2	12/4/2017 8:00:29 AM	12/4/2017 1:20:29 PM	5.33333	
585	Executing	MFGOrder.756	Order_SG_1	12/4/2017 8:00:30 AM	12/4/2017 8:30:30 AM	0.5	
584	Executing	MFGOrder.757	Order_SG_1	12/4/2017 8:00:30 AM	12/4/2017 8:30:30 AM	0.5	
590	Executing	MFGOrder.742	Order_F_1	12/4/2017 8:00:48 AM	12/4/2017 8:12:48 AM	0.2	

A task is started when the entity enters the object and begins the processing tasks for that object. The task may, however, be **Constrained** if a resource (including worker/vehicle) or material specified for the task is not currently available. Once all resources and materials have been allocated, the task will start the **Executing** state. If the main or secondary resource goes off-shift and the *Off Shift Rule* is 'Suspend Processing', the task state becomes **Suspended**. These task states can also be seen graphically on the Gantt charts. The [Task Log](#) includes all the task states for a particular task and can be expanded using the '+' to view the Task State Log for a particular task.

Adding a Column to the Task State Log

When the Task State Log tab is selected, the Operational Planning and Log ribbons are available.



The Log ribbon provides the ability to change the Task State Log.

For example, the Add button is available to add a column to the Task State Log table. Additional columns can be used within Dashboard and Table reports for filtering. When a column is added to the Resource Capacity Log, there are several properties that may be specified for the column, including:

Listed below are the properties of a **Task State Log Column**:

Property	Description
Expression	The expression value to be recorded in this column for each resource state row in the log.
Data Format	The data type format for the value of this column's specified Expression (e.g., Real, Integer, Boolean, DateTime, String or Color).
Unit Type	Classifies the units of the value returned by this column's specified Expression (available for Data Format types Real and Integer).
Evaluation Type	The method used to evaluate and record this column's expression value for each Task State row in the log. If 'StartTimeValue', then the expression's value at the start time of the usage occurrence is logged. If 'EndTimeValue', then the expression's value at the end time of the usage occurrence is logged. If 'ValueChange', then the expression's change in value from the start time to the end time of the usage occurrence is logged.
Trait Filter	Specifies the name of a Trait defined on a Simio Portal instance. That Trait's per-user value is used to filter what the user can see from this log. Rows with values that match the value of the Trait for the user will be visible to the user, others will be filtered out.

Note on Trait Filter property - Within Simio Portal, user would have a trait defined called 'XXTrait'. Each user then has their own value (or comma separated values) assigned to 'XXTrait' for themselves. If the value of all expression columns with

Trait Filters assigned in a particular row of the log matches the values for the user's trait(s), then they can see that row, otherwise they can't. Row filtering works for viewing the logs, using them as datasources for the Dashboards, and the Gantt data as well. The filtering is "filtering in", meaning not assigning a trait value for a user, or not even having a referenced trait of that same name defined in the Simio Portal, will filter away all rows for that log.

In addition to the Add button, the Log ribbon includes options to change the Unit Settings. These options are similar to the Unit Settings section of the Run ribbon within the Facility window and will change the units for all relevant windows.

All data within the Task State Log may also be exported to a *.csv file by using the Export to CSV button.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Tables - RPS

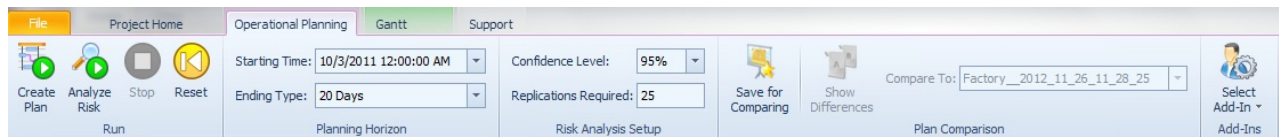
The Tables window provides a view of all of the data and sequence tables that have been defined through the Data window. These tables contain a combination of input data, output states, and target values for the model. Once the Risk Analysis is run the target values are color coded for High, Medium, and Low risk. You may edit the input data columns that are not "grayed". This view also provides import/export functionality for the table data.

Within each of the tables in the Data window, each column has an Operational Planning area of properties. Both the Tables section and Gantt Chart section have three properties, *Visible*, *Editable* and *Category Name*. *Visible* specifies whether or not the column will be visible within the Planning data table or Gantt chart. *Editable* determines whether or not the column will be editable within the Planning data table or Gantt chart. *Category Name* is an optional string for displaying the column in the planning table or within the Gantt chart. If both Table type properties, *Editable* and *Visible*, are set to 'True' for all columns, then the data tables within the Planning tab Tables window are exact copies of the corresponding tables within the Data window. A Gantt chart column that is *Visible* will be displayed in the Resource Plan Gantt alphabetically, based on the column name. This column may be turned off within the Gantt by toggling the Table Columns button in the Gantt ribbon under Visibility.

States and Targets that have been defined for a table will appear slightly different than the original table. Each of the targets defined will appear with two groupings of columns. First is the '*TargetName* - Plan', which will have the columns 'Value (Units)' and 'Status' as in the original data table. Additionally, there is the '*TargetName* - Risk Analysis' grouping, which will include an 'Expected (Units)' column and a '*Within Bounds Probability*' column. The table states and targets columns shown in the Tables window will initially be empty.

Within the Tables window, the Operational Planning ribbon is available.

Operational Planning Ribbon



The Create Plan button will run the simulation model in a deterministic mode with no uncertainty to generate a resource plan. All distributions used in the model will return their expected values and all failures in the model will be disabled. When the Create Plan button is selected, Simio will run the simulation and display the results of the table states columns, as well as the '*TargetName* Plan' columns in the table. Creating the plan will not display any values within the '*TargetName* Risk' columns of the table.

The Analyze Risk button will run the simulation model in a stochastic mode with uncertainty to analyze the target risks specified within the model.

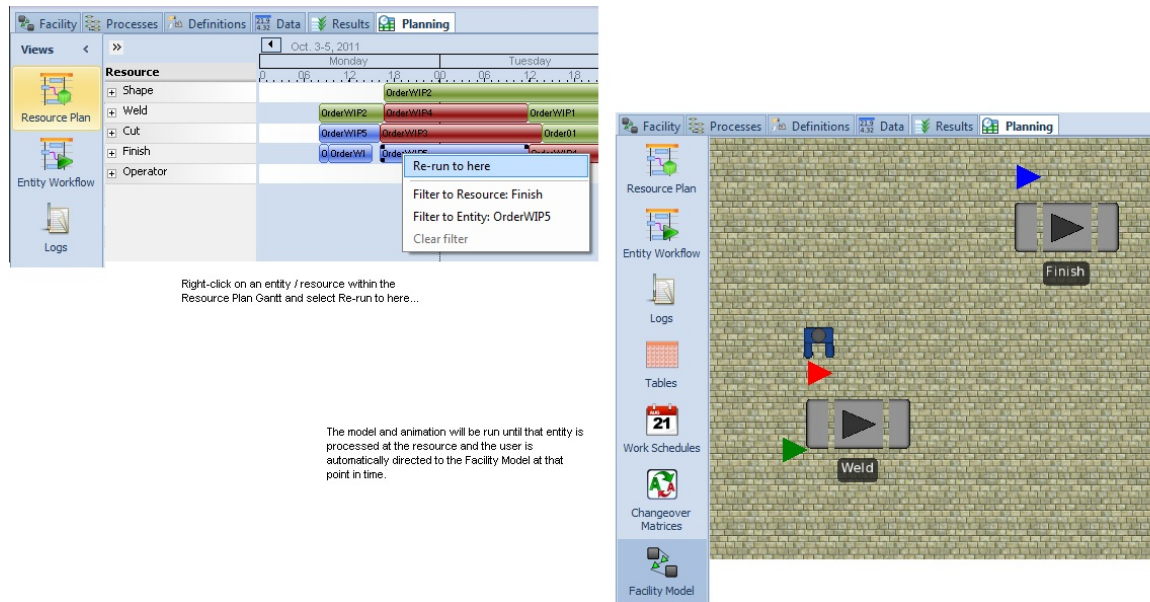
The Planning Horizon *Starting Time* and *Ending Type* specify the starting and ending times of the simulation model and are used for both creating the plan and analyzing the risk. These values can also be seen in the Facility window's Run ribbon.

The *Confidence Level* and the *Replications Required* properties are utilized only when analyzing risk. The *Confidence Level* is used to calculate the confidence interval half-width statistics for the target averages and feasibility probabilities estimated by the risk assessment. The default confidence level is '95%', however the user may also select from the list to include 90%, 95%, 98% and 99%. The *Replications Required* property specifies the number of simulation replications to run for the risk assessment. The default value is '10'.

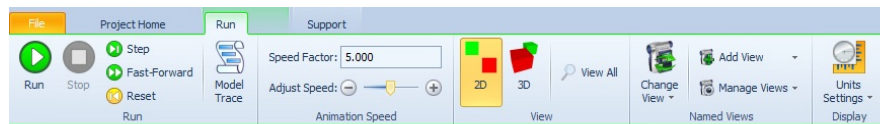
When performing the risk analysis, the number of replications specified is run and then the values within the '*TargetName* Risk' columns will be generated. The 'Average (Time Units)' column is calculated by using the target expression and taking the average value of that expression for the number of replications. The '*Within Bounds Name Probability*' column values are then determined from the number of simulation replications whose average values fall within the upper and lower bounds specified by the user.

Facility Model - RPS

The Facility Model is a 3D animated facility view that is useful for viewing an animation of the plan. You can jump to a specific resource and time in the animation by right clicking on a resource bar in Gantt within the Resource Plan view and selecting "Re-run to here". Right-clicking on a particular resource within an Entity row in the Entity Workflow Gantt will also move to the Facility Model with "Re-run to here".



The Facility Model button can also be used to view the animated model running in planning mode. The Run ribbon within this Facility Model view provides for functionality available on multiple ribbons in the standard Facility window. It includes Run options, such as Run / Step / FastForward, as well as animation speed options. The Model Trace button allows users to view the trace of the model while running in Planning mode. The ribbon also provides several of the View ribbon options, such as 2D / 3D and named views.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Results - RPS

The Results window provides information about the risk based planning and scheduling results, including a Target Summary, Risk Plots, Detailed Results and Reports. The Risk Analysis page includes many frequently asked questions regarding the risk based analysis.

The Target Summary tab provides a high level summary of target performance including the number and percent of entities that fall within target bounds. The Risk Plots provide a graphical summary of target performance including confidence intervals and percentiles for each target. The Detailed Results tab provides detailed results (queue lengths, waiting times, etc.) for the plan in a pivot grid that you can sort, filter, and pivot. The Reports tab provides detailed reports in both chart and graphical formats. These reports include Resource Dispatch Lists, Workflow Constraint Analyses and Resource Utilization Summaries.

[Risk Analysis](#)

[Target Summary](#)

[Risk Plots](#)

[Detailed Results](#)

[Reports](#)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Risk Analysis - RPS

What does the risk percentage mean?

The risk percentage approximates the on-time probability for an order with appropriate consideration of the number of replications or "experiments." It tells the user how confident they can be in meeting the due date given how many trials they have conducted.

How does Simio calculate the on-time probability?

Simio adjusts from a base rate of 50% with each risk replication. If an order is on time in an individual replication, Simio updates the probability, increasing it closer to 100%. If the order is late, Simio decreases the probability closer to 0%. Each replication is an experiment that provides new information about the likelihood of success or failure. More experiments mean more confidence in the answer.

Why is the base rate 50%?

Before any plan is generated or any activity is simulated, there is no information about the order other than the possible outcomes. Because there are only two outcomes that matter (on time or not), the base rate is set to 50%.

I have an overdue order in my system. Why is it not always 0%?

Because the calculation is an adjustment of a base rate of 50%, Simio needs a lot of evidence before it will guarantee that an order will be late (or on time for that matter). If the user runs 1000 replications, and the result is late in all of them, Simio will reflect a 0% on time probability.

Why not just report the outcome of the replications as the probability (e.g., if 9 of 10 are on time, report 90% on time probability)?

This was the original implementation. However, it gives a false sense of confidence and can be misleading. A single replication would always yield either 100% on time or 0% on time. We wanted the answer to also give decision makers a sense of how confident they could be in the answer. Using the Wilson Score, a single replication will yield a result of 60% at best and 40% at worst (using 95% confidence level). This helps the decision maker identify that they have a very small sample of data and would encourage them to run additional replications.

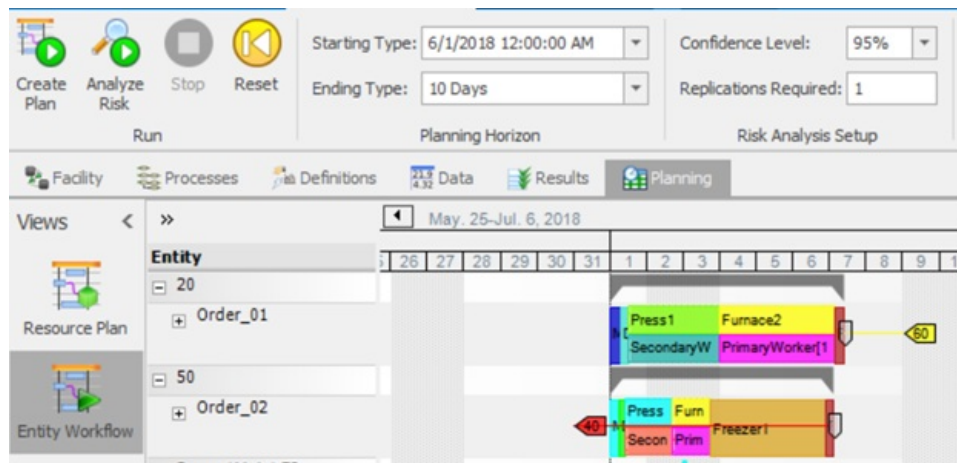
What formula does Simio use to calculate the probability?

For the statistics experts, Simio uses a binomial proportion confidence interval formula known as the Wilson Score. We report the midpoint of the confidence interval as the risk measure.

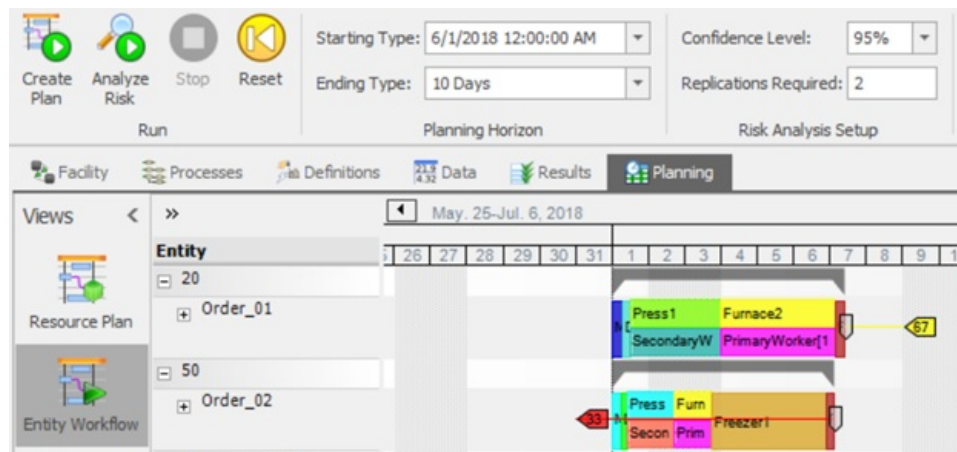
Can you give me an example of how this works?

Risk analysis can be demonstrated using any scheduling example. It is best viewed in the Entity Gantt. In the screenshots below, we've included 2 orders from a generic Candy Manufacturing Scheduling example. One of the orders is overdue (will be late always), and the other has plenty of time (will be on time always).

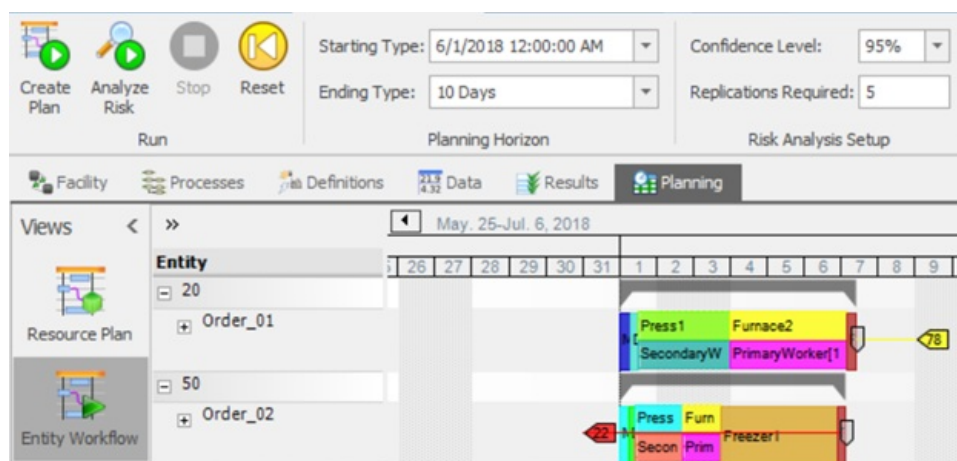
The base rate is 50%. After 1 replication, Simio updates the probabilities. Order 1 now has a 60% on time probability. Order 2 has a 40% on time probability.



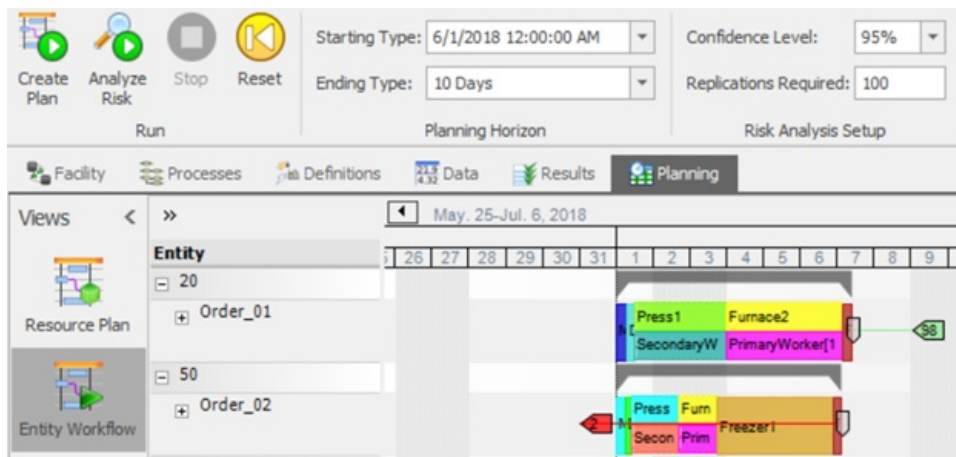
After 2 replications, 67% and 33%:



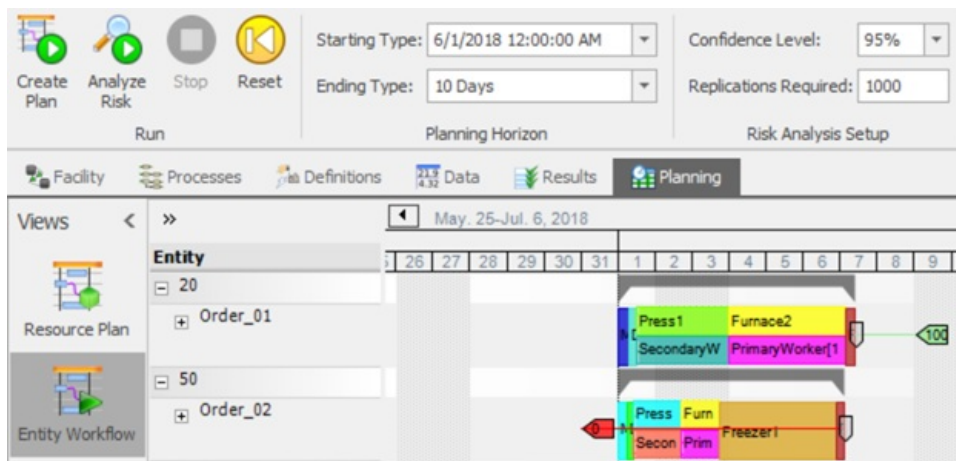
After 5 replications, 78% and 22%:



After 100 replications, 98% and 2%:



Finally, after 1000 replications, 100% and 0%:



How many replications should I run?

By default, we suggest 10 replications (and 95% confidence level). With these settings, a risk measure of 86% is a good sign, while 14% is a bad one. Beyond the default settings, there are several additional factors which are dependent on the situation and use case. One of these factors is slack time (the time between estimated completion and due date). On the Gantt, slack time is the distance between the grey marker and the green marker. If the slack time is large, a single replication may suffice. If the slack time is small, additional replications will help identify if the order is in trouble or not.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Target Summary - RPS

The Target Summary window provides information in the form of a pivot grid for the entities / orders that are completed through the system. For each 'Target' that is specified in any given data table, data is provided on the *Above Upper Bounds*, *Within Bounds*, *Below Lower Bounds* and *No Value* property values. This includes both the percent of entities / orders that fall within the category, as well as the total number completed within that category.

The Target Summary is only generated on the Plan (with no variation within the model). Only the Operational Planning ribbon is available for this window.

Target Summary				
<div> Target Summary Target Detail Risk Plots Detailed Results Reports </div>				
Drop Filter Fields Here				
<div> <div>Value</div> <div>Drop Column Fields Here</div> </div>				
Table Name ▲	Target Name ▲	Data Item ▲	Statistic ▲	Value Total
ManufacturingOrders	TargetCost	NumberBelowBounds	Percent	0
			Total	0
		NumberIncomplete	Percent	0
			Total	0
		NumberOnBudget	Percent	81.25
			Total	13
		NumberOverrun	Percent	18.75
			Total	3
	TargetShipDate	NumberBelowBounds	Percent	0
			Total	0
		NumberIncomplete	Percent	0
			Total	0
		NumberLate	Percent	0
			Total	0
		NumberOnTime	Percent	100
			Total	16

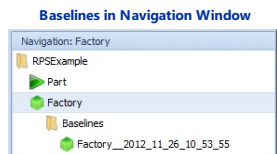
Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Operational Plan Comparison - RPS

The user can save a copy of the model in its current state for use as a baseline for comparison. The user might create an Operational Plan and then want to make changes to the model or the data being read into the model. They would like to compare the results of the new Operational Plan with the results of another Operational Plan. The user can save a copy of the model in its current state. The user can save multiple copies of the model in different states and compare each Operational Plan against the others. The user should take note that the size of the Simio project file increases each time a baseline model is created and saved.

To save a model to compare Operational Plans, the user must first create a plan by clicking **Create Plan** in the Operational Planning ribbon. Users may also run a risk analysis by clicking **Analyze Risk** in the Operational Planning ribbon, although that is not necessary to save the plan for comparison. The Save For Comparing button in the Operational Planning ribbon is available after either Create Plan or Analyze Risk has been selected. When the user clicks this button, a copy of the model is created and this baseline model now appears in the Navigation window under a new folder called Baselines.



At this point, the user can make changes to the model and then re-run the Create Plan and/or re-run the Analyze Risk. The new model can then be compared to the saved baseline model to see what differences the new changes made to the Targets. In order to show the differences between the current model and the baseline, the user must go to the Target Detail tab in the Results Panel. From within the **Target Details** tab, if there is a baseline model, the user will see the Show Differences button available in the ribbon. If there is only one baseline, it will automatically be populated in the Compare To selection box in the ribbon. If there is more than one baseline, the user can select which baseline model to compare against.

Comparing Two Operational Plans

File Project Home Operational Planning Support

Create Plan Analyze Risk Stop Reset Run

Starting Time: 10/3/2011 12:00:00 AM Ending Type: 20 Days Replications Required: 25

Confidence Level: 95% Save For Comparing Show Differences Compare To: Factory__2012_11_26_10_53_55 Select Add-In

Target Summary		Target Ship Date - Plan			Target Ship Date - Risk Analysis			Target Cost - Plan			Target Cost - Risk Analysis		
Order ID	Value	Change	Status Change	Expected	Change	Probability Change	Value (USD)	Change	Status Change	Expected (USD)	Change	Probability Change	
Order01	10/12/2011 3:11:49 PM	0d 4h 36m	Still OnTime	10/12/2011 5:20:17 PM	0d 12h 19m	89.87% → 72.54%	128,344.1332	3,799.9593	Still OnBudget	126,970.1508	2,807.7162	89.87% → 79.47%	
Order02	10/14/2011 4:24:20 PM	No change	Still OnTime	10/15/2011 4:20:09 PM	-0d 0h 32m	Still 58.67%	135,245.9138	0.0407	Still Overrun	139,791.2602	-108.6740	6.66% → 10.13%	
Order03	10/11/2011 1:35:49 PM	0d 4h 36m	Still OnTime	10/11/2011 11:43:15 PM	0d 16h 42m	Still 93.34%	112,529.8258	-2,800.0000	Still OnBudget	115,490.6069	2,414.4357	Still 93.34%	
Order04	10/6/2011 4:07:44 PM	-6d 0h 50m	OnTime → Late	10/7/2011 6:45:56 AM	-6d 0h 14m	93.34% → 10.13%	89,876.2214	-28,045.6451	Still OnBudget	92,610.1810	-28,410.2117	Still 93.34%	
Order05	10/19/2011 4:16:30 PM	No change	Still OnTime	10/20/2011 12:08:34 AM	0d 2h 19m	58.67% → 44.80%	149,619.7894	0.0000	Still Overrun	151,228.4003	327.1938	Still 6.66%	
Order06	10/10/2011 3:43:47 PM	0d 6h 22m	Still OnTime	10/10/2011 5:09:14 AM	0d 18h 32m	Still 93.34%	97,889.9569	1,272.0000	Still OnBudget	95,764.3967	3,696.6384	Still 93.34%	
Order07	10/18/2011 9:33:56 AM	No change	Still OnTime	10/18/2011 8:38:21 AM	-0d 0h 45m	89.87% → 93.34%	136,791.2065	0.0000	Still Overrun	134,489.9893	-430.0738	Still 13.59%	
Order08	10/12/2011 9:23:49 AM	0d 18h 36m	Still OnTime	10/12/2011 6:18:32 PM	0d 7h 29m	93.34% → 89.87%	96,623.4258	3,720.0000	Still OnBudget	98,399.1791	1,488.9984	Still 93.34%	
Order09	10/17/2011 1:45:56 PM	No change	Still OnTime	10/17/2011 6:47:55 PM	0d 6h 16m	Still 93.34%	117,363.6065	0.0000	Still OnBudget	120,380.4011	1,564.5632	89.87% → 93.34%	
OrderWIP1	10/7/2011 1:34:11 PM	No change	Still OnTime	10/9/2011 4:33:37 PM	0d 13h 59m	89.87% → 55.20%	120,490.7049	0.0000	Still OnBudget	132,768.6574	3,279.5293	48.27% → 41.33%	
OrderWIP2	10/5/2011 3:30:06 PM	No change	Still OnTime	10/6/2011 12:53:41 AM	0d 0h 40m	34.40% → 30.93%	117,758.4338	0.0000	Still OnBudget	119,929.7736	134.5084	Still 93.34%	
OrderWIP3	10/10/2011 9:55:47 AM	3d 17h 48m	OnTime → Late	10/8/2011 7:33:17 PM	1d 21h 7m	93.34% → 41.33%	142,556.3976	17,880.1762	OnBudget → Overrun	134,169.2978	8,722.3104	93.34% → 48.27%	
OrderWIP4	10/5/2011 9:12:04 AM	No change	Still OnTime	10/5/2011 7:33:15 AM	No change	Still 89.87%	126,210.6711	0.0000	Still OnBudget	125,379.4748	0.0000	Still 93.34%	
OrderWIP5	10/4/2011 11:50:28 AM	No change	Still OnTime	10/4/2011 1:59:07 PM	No change	Still 93.34%	114,712.2711	0.0000	Still OnBudget	115,142.8589	0.0000	Still 93.34%	
OrderWIP6	10/3/2011 9:12:01 AM	No change	Still OnTime	10/3/2011 9:08:20 AM	No change	Still 93.34%	124,888.0814	0.0000	Still OnBudget	124,873.3312	0.0000	Still 93.34%	
OrderWIP7	10/3/2011 3:00:01 PM	No change	Still OnTime	10/3/2011 5:18:21 PM	No change	Still 65.60%	119,192.0814	0.0000	Still OnBudget	119,665.1525	0.0000	Still 93.34%	

When the **Show Differences** button is selected, the user will see new columns appear in the Target Details table. The **Show Differences** button can be toggled on or off by the user to display and hide the new differences columns. When the button is selected, there will be a new column named Change under each Target's plan results and risk analysis results. This will show the value change between the current model and the baseline model that is currently selected in the **Compare To** selection box. In the Target's plan results, there will also be a new column named Status Change that will show any change in the Target's status. In the Target's risk analysis results, there will be a new column named Probability Change that will show changes in the probability that this Target will be within the defined bounds. For more information on defining a Target and its lower bound and upper bound, see the [Target](#) help page.

In the example above, the user saved a baseline operational plan and then made changes to the schedule within the current model. After making the changes, they wanted to compare the target results of the current model with the new schedule to the target results in the baseline. You'll notice that some orders went from OnTime to Late. You can see how much later the order is expected to be in the Change column under Target Ship Date - Risk Analysis. You'll also notice that some orders went from OnBudget to Overrun. And the Change column under Target Cost - Risk Analysis shows the new expected value for the Cost Target.

Risk Plots - RPS

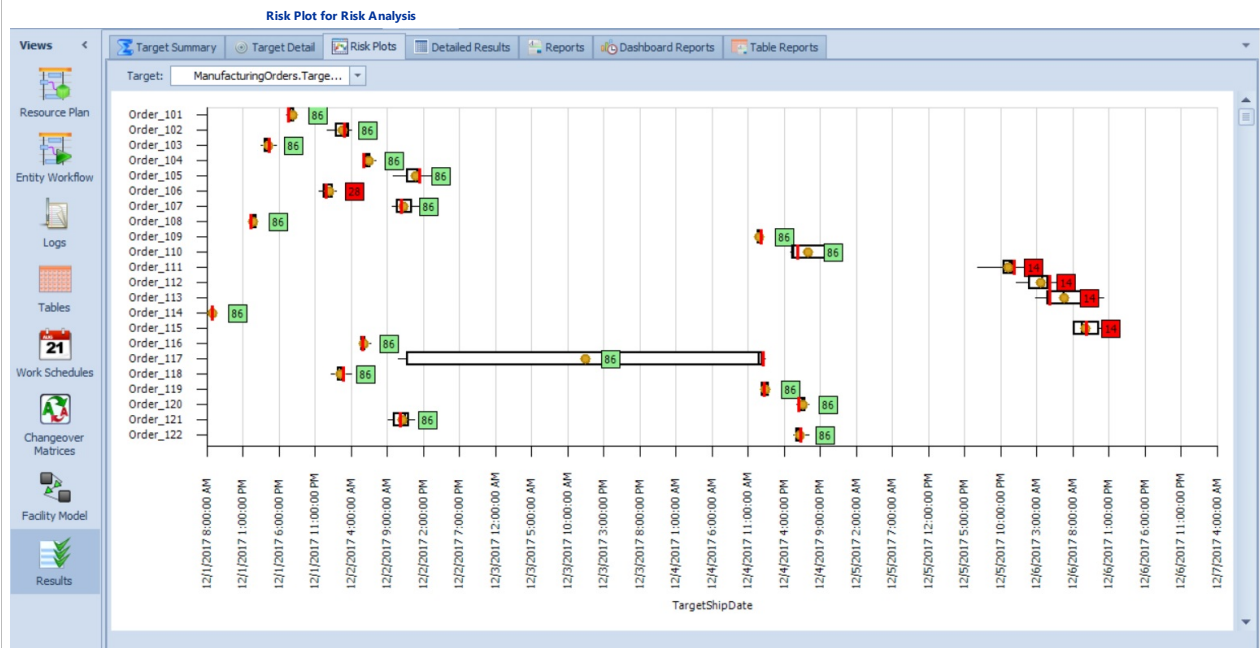
The Risk Plots window provides additional information on the target(s) specified within a Data table. Risk Plots charts can be generated with both the Plan (deterministic run without variation), as well as with the [Risk Analysis](#) (stochastic run with variation).

By default, the format of the results are originally shown in a Box and Whisker type format, with the Target name shown at the top of the chart, and each row within the *TableName*.Target shown along the y-axis. The x-axis shows the Target values.

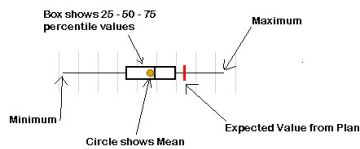
If the user has only generated the plan, the results shown in the Risk Plots will include only the Expected Value of the Target (shown as a red vertical line). The Limits button on the Target Chart ribbon will show any lower and/or upper limits on the Target based on the *Below Lower Bound* and *Above Upper Bound* properties as defined by the Target within the Data Table. These limits can be turned on/off and are shown 'on' as the pink shaded area. In the example below, most of the orders are expected to be time, as the pink shaded area is located to the right of the expected value, with the exception of orders Order_106, Order_111, Order_112, Order_113 and Order_115. The expected value of the target ship date for these orders is later than the *Above Upper Bound* value specified for the target, thus the red expected value line is within the pink shaded area.



If the user has performed the risk analysis, the results shown in the Risk Plots will include the mean, min/max values, range and percentile values, as shown below.

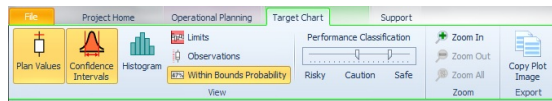


Key to Target Chart



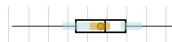
When the mouse is placed on the outer lines, the minimum and maximum values are displayed as the range of the observations. When the mouse is hovered over the tan circle, the mean of the observations is displayed. When the mouse is placed over the upper portion of the box, it will display the top range (50% - 75%) including the median and upper percentile value. Finally, when the mouse is hovered over the lower portion of the box, the lower range (25% - 50%) including the lower percentile value and the median are shown.

The Target Chart and Operational Planning ribbons are available within the Risk Plots window. The Target Chart ribbon includes buttons for the target data to be displayed in various ways.

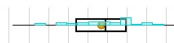


The Plan Values button will display a red line for each observation on the box and whisker plot for the target. This displays the Expected Value of the target as derived from the Plan. This expected value is also automatically calculated when performing risk analysis, as the plan is run first.

The Confidence Intervals button will display three confidence intervals on the chart. The uppermost light blue rectangle shows the confidence interval of the upper percentile and includes the start and end value of the interval. The lower light blue rectangle shows the corresponding data for the lower percentile. The light brown rectangle shows the confidence interval of the mean, and includes the start and end values, as well as the half-width.



The Histogram button will display a histogram of the replication data points on the SMORE plot. No additional data is provided when hovering the mouse over the histogram.

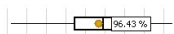


The Limits button will display the upper and lower bound from the target when run in plan mode. The SMORE itself scales to the available result data, so even if you have limits, they will not display if they are off the 'sides' of the SMORE. However, if you have result data in the range of 0 - 10 for example, and the limits are at 2 and 6, the Limits button will display them in light red over the SMORE plot area.

The Observations button simply displays each replication value for the target on the plot in light blue dots.



The Within Bounds Probability button will display a value within a box to indicate the probability of the target falling within the bounds that were specified (including both the upper and lower bounds).



Detailed Results - RPS

The Detailed Results window includes summary statistics for the simulation run, as generated with the resource Plan. It is important to remember that the results are calculated in a deterministic mode with no uncertainty. Therefore, the results displayed will be different than those displayed within the Results tab window when running the general simulation model. Below is an example of the Detailed Results window.

Facility Processes Definitions Data Results Planning						
Views Target Summary Target Detail Risk Plots Detailed Results Reports						
Drop Filter Fields Here						
Average Drop Column Fields Here						
Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average Total
Workstation	Cut	[Resource]	Capacity	ScheduledUtilization	Percent	52.2667
				UnitsAllocated	Total	11.0000
				UnitsScheduled	Maximum	1.0000
					Average	0.2500
				UnitsUtilized	Maximum	1.0000
					Average	0.1307
			Costs	IdleCost	Total (USD)	1,145.6000
				UsageCostCharged	Total (USD)	3,136.0000
			ResourceState	OffShiftTime	Total (Minutes)	21,600.0000
					Percent	75.0000
					Occurrences	31.0000
					Average (Minutes)	696.7742
				ProcessingTime	Total (Minutes)	3,763.2000
					Percent	13.0667
					Occurrences	16.0000
					Average (Minutes)	235.2000
			StarvedTime	Total (Minutes)		3,436.8000
					Percent	11.9333
					Occurrences	15.0000
					Average (Minutes)	229.1200
		InputBuffer	Content	NumberInStation	Maximum	4.0000
					Average	1.3348
			Costs	CostPerItem	Minimum (USD)	1,600.0000
					Maximum (USD)	35,583.8789
					Average (USD)	11,649.3555

See the [Pivot Grid](#) page for more information regarding specifics of sorting, filtering and named views.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Reports - RPS

The Reports window provides several different chart-based and graphical summary reports for the plan. Again, these reports are calculated in a deterministic mode with no uncertainty. The reports include Entity Activity List, Resource Dispatch List, Workflow Constraints Analysis and Resource Utilization Summary. Each of these is described in more detail below. The reports can be submitted with the Report Designer, if the standard report does not meet your need.

File Project Home Operational Planning

Current Report: Resource Dispatch List

- Entity Activity List
- Resource Dispatch List
- Workflow Constraints Analysis
- Resource Utilization Summary

Entity Activity List

The Entity Activity List reports include chart and graphical representations for resources/activities for a particular entity/order for a given time period. Below you will see an example of the Order01 entity. On the left side of the report, the user is required to input several parameters, including the Entity Name, StartDate, EndDate and ShowGraphics. Once those are complete, clicking on the Submit button will generate the report. Changing the ShowGraphics from 'No' (default) to 'Yes' will turn off all graphical reports and show charts only.

Data provided within the chart includes the Resource, Start Time, End Time and Duration. These are shown in chronological order. The graph below the chart displays the resources in a stacked fashion. Note that any resources that are active at the start time of the time interval will be shown in the chart. If the start time and end time of the resource allocation falls outside of the times specified, the time within the box will be grayed.

Facility Processes Definitions Data Results Planning

Views Target Summary Target Detail Risk Plots Detailed Results Reports Dashboard Reports Table Reports

Parameters

Entity Name: Order01

Start Date: 12/1/2016 8:00

End Date: 12/22/2016 8:00

Show Graphics: Yes

Reset Submit

Entity Activity List

Generated On: 3/21/2017 8:12:24 AM

Date Range: 12/1/2016 8:00:00 AM - 12/22/2016 8:00:00 AM [Show Detail](#)

Owner: Order01

Resource	Start Time	EndTime	Duration (Hours)
Weld1	12/1/2016 8:00:17 AM	12/1/2016 11:42:17 AM	3.7000
Resource1	12/1/2016 8:00:17 AM	12/1/2016 11:42:17 AM	3.7000
Shape1	12/1/2016 11:42:30 AM	12/2/2016 8:30:30 AM	20.8000
Finish1	12/2/2016 8:30:41 AM	12/2/2016 1:54:41 PM	5.4000

Where the 2nd page is shown in graphical form (when Show Graphics is set to 'Yes')

Date Range: 12/1/2016 8:00:00 AM - 12/22/2016 8:00:00 AM [Show Detail](#)

Owner: Order01

Clicking on 'Show Detail' in the top right corner of the report will display more information about each specific resource, as seen below. For each resource that has been allocated to the given entity, information is displayed regarding the amount and percentage of time the resource spent in various states. For example, states of the resource may include OffShift, Processing, Setup, Standby or Waiting/SecondaryResource.

Facility Processes Definitions Data Results Planning

Views Target Summary Target Detail Risk Plots Detailed Results Reports Dashboard Reports Table Reports

Parameters

Entity Name: Order01

Start Date: 12/1/2016 8:00

End Date: 12/22/2016 8:00

Show Graphics: Yes

Reset Submit

Date Range: 12/1/2016 8:00:00 AM - 12/22/2016 8:00:00 AM [Hide Detail](#)

Owner: Order01

Resource	Start Time	EndTime	Duration (Hours)
Shape1	12/1/2016 11:42:30 AM	12/2/2016 8:30:30 AM	20.8000

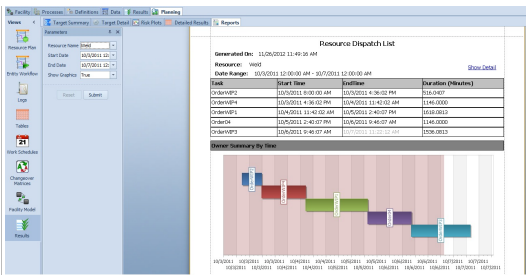
Resource State Details

State	Start Time	End Time	Duration (Hours)
Processing	12/1/2016 11:42:30 AM	12/1/2016 12:00:00 PM	0.2914
OffShift	12/1/2016 12:00:00 PM	12/1/2016 1:00:00 PM	1.0000
Processing	12/1/2016 1:00:00 PM	12/1/2016 5:00:00 PM	4.0000
OffShift	12/1/2016 5:00:00 PM	12/2/2016 8:00:00 AM	15.0000
Processing	12/2/2016 8:00:00 AM	12/2/2016 8:30:30 AM	0.5086

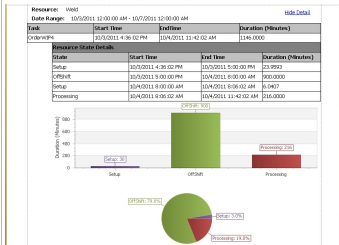
Resource Dispatch List

The Resource Dispatch List reports include chart and graphical representations for tasks on a particular resource for a given time period. Below you will see an example of the Weld1 resource. On the left side of the report, the user is required to input several parameters, including the ResourceName, StartDate, EndDate and ShowGraphics. Once those are complete, clicking on the Submit button will generate the report. Changing the ShowGraphics from 'No' (default) to 'Yes' will turn off all graphical reports and show charts only.

Data provided within the chart includes the Task, Start Time, End Time and Duration. These are shown in chronological order. The graph below the chart displays the tasks in a stacked fashion. Note that any tasks that have a Start Time or End Time that corresponds to the StartDate or EndDate parameter selected will be shown in a 'grayed' box in the chart. Additionally, the graph will display a shading corresponding to the StartDate and EndDate specified.



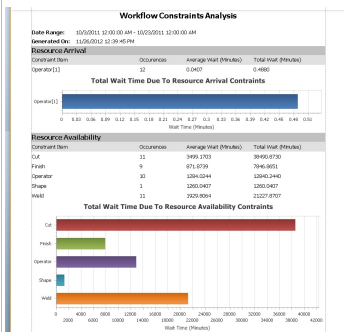
Clicking on "Show Detail" in the top right corner of the report will display more information about each specific task, as seen below. For each task that has been processed at the given resource, information is displayed regarding the amount and percentage of time the resource spent in various states. For example, states of the resource may include OFFSH, Processing, Setup, Standdown or WaitingForSecondaryResource.



Workflow Constraint Analysis

The Workflow Constraint Analysis reports provide additional information to the Constraint Log within the Entity Workflow window. The parameters to be specified for this type of report include the the StartDate, EndDate and ShowGraphics. Once these are complete, clicking on the Submit button will generate the report. Changing the ShowGraphics from "Yes" (default) to "No" will turn off all graphical reports and show charts only.

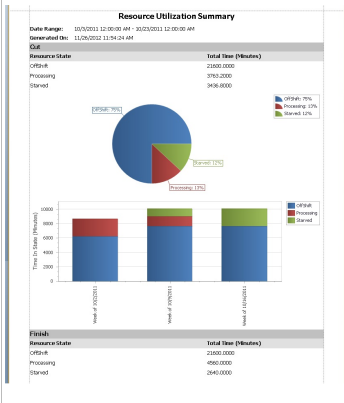
For each constraint type, the report includes information on the Constraint Item, number of Occurrences, Average Wait and Total Wait times, both in chart and graph formats. Types of constraints include Resource Availability and Resource Actual. An example of the report sections is shown below.



Resource Utilization Summary

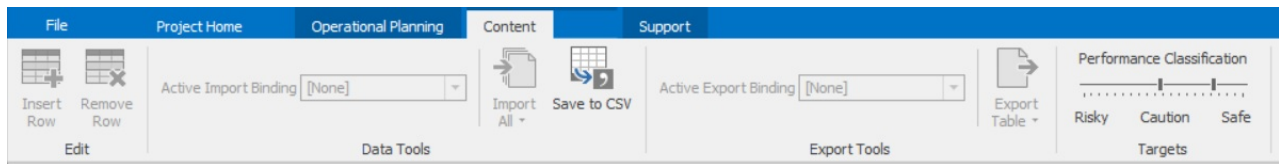
The Resource Utilization Summary reports provide utilization summary information for all resources in the system. The parameters to be specified for this type of report include the the StartDate, EndDate and ShowGraphics. Once these are complete, clicking on the Submit button will generate the report. Changing the ShowGraphics from "Yes" (default) to "No" will turn off all graphical reports and show charts only.

For each resource, the report includes information on the Resource State and Total Time that the resource is in that state, both in chart and graph formats. This utilization report provides summary information for the State Log under the Resource Plan panel. Resource States include OFFSH, Setup, Processing, Standdown, Standby, and WaitingForSecondaryResource. An example of the Resource Utilization Summary for a single resource is shown below.



Content Ribbon - RPS

The Content ribbon when the Planning tab is active provides several specific add-in buttons for use in particular scheduling applications.



See the the [Project Templates](#) page for additional table generating add-ins.

Once the various tables have been created within the Data tab, the user may utilize the different table data binding options such as AVEVA MES, CSV, Database, and Web API.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Selected Topics from Shared Items User Forum

*** This section is currently under construction ***

It will contain help on some of the Shared Items within the Simio Users Forum.

[Extras Library](#)

[Visio Interface](#)

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Extras Library

Extras Library

Simio has developed an Extras Library that currently includes 7 pre-built object definitions that can be used to model material handling systems.

This library will automatically be loaded when you open Simio. To change whether the library is automatically loaded, you can change the Load Extras Library application setting.

Name	Class	Description
Bay	Fixed	A Bay defines a rectangular region over which cranes may move.
Crane	Transporter	A crane is moved by a Cab and may pickup and drop-off entities at nodes that fall within the specified Bay.
Robot	Fixed, Entities and Transporter	A composite object that represents a robot. The robot hand is a transporter that can pickup/drop-off entities. Select the hand to edit its properties.
Rack	Fixed	A rack is used to hold entities for a specified storage time. A rack can be used with a Lift Truck for placing items on different shelves within the Rack.
LiftTruck	Transporter	A LiftTruck object may be used to transport entity objects between node locations. When used in conjunction with a Rack, the Lift on the LiftTruck will move vertically to the appropriate shelf height before executing a pickup or drop-off. Additionally, an 'On Demand' routing type LiftTruck may be used as a moveable resource that is seized and released for non-transport related tasks by model process logic.
Elevator	Transporter	An elevator is used to carry entities in a vertical direction between a collection of ElevatorNodes.
ElevatorNode	Node	An ElevatorNode is either used to define entry and exit nodes for entities that ride on an elevator, or to route entities to elevators within a bank of elevators.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Bay

The Simio Extras Library contains two objects designed for modeling multiple cranes operating simultaneously in a bay - the Crane object and the Bay object. These objects may be used in conjunction with the Simio Standard Library.

A Bay is a fixed object that defines a rectangular region over which one or more Bridges may move. A Bay can be introduced into the model by dragging it from the Extras Library, then setting its properties. The Bay's *Number of Zones* property specifies the number of zones used to control Bridge movements to prevent collisions. Each Bridge in the Bay occupies its current zone, and only one Bridge may be in a zone at one time. Before a Bridge can move to a new location, it must seize its destination zone. It is not permitted to seize its destination zone until all intermediate zones are free. During a Bridge move, the Bridge holds both its starting zone and destination zone and only releases its starting zone once it has reached its destination.

The Bay also has two properties related to deadlock prevention. The first is the *Blocking Action* which can be specified as 'None' or 'Push Idle Bridge'. If 'Push Idle Bridge' is selected, then a busy Crane will automatically push one or more idle Cranes out of the way to complete a move if necessary. The second deadlock prevention property is a Boolean named *Pre-Check Zone Availability*. If 'True', a move will not be started unless all zones are available for both the pickup and drop-off operation. This prevents a crane from beginning an operation it cannot complete due to deadlocking.

The Bay is a composite object made up of Zones which do not rotate with the Bay if you were to manually rotate a Bay instance. Instead, the Traversal Direction property on the Bay will change how the Zones are created and specifies the Crane's movement direction. If you change the Traversal Direction, from the default 'Left and Right', to 'Forward and Back' the Bay will act as if it was rotated 90 degrees. The Zones of the Bay will now be created parallel to the horizon.

The Traversal Direction property should live in the Bay's Travel Logic category. If you do not see this property listed, you may have an older version of the Bay object. To remedy this, you will need to go into the Bay definition and change the Traversal Direction property's Visible property to 'True'. This will make the Traversal Direction property visible on instances of this custom Bay.

If the Bay needs to be wider than it is long or vice versa, you can change its size physical characteristics by specifying the Length or Width properties, found under the General > Physical Characteristics category, so that the Bay has the correct dimensions.

Listed below are the properties of the **Bay**:

Property	Valid Entry	Description
Traversal Direction	Left and Right, Forward and Back	The direction that the bridges in the Bay move, specified as either 'Left and Right' or 'Forward and Back'.
Number of Zones	Real	The number of control zones used in the Bay to prevent collisions. Only one bridge may occupy a zone at a time. The destination zone must be seized and all intermediate zones must be idle before bridge movement can be initiated.
Blocking Crane Action	None, Push Idle Bridge	Specifies the action to take when a bridge is blocked by another bridge in the Bay. The 'Push Idle Bridge' option will allow a busy Crane to push an idle bridge out of the way.
Bay Visibility	True, False	If 'True', the Bay is visible during the simulation run.
Zone Visibility	True, False	If 'True', the pickup and dropoff zones are highlighted.

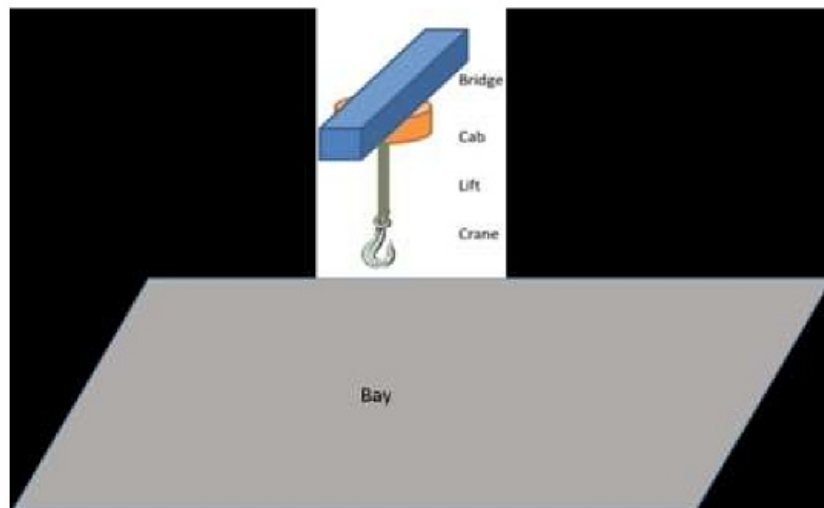
Send comments on this topic to [Support](#)

Crane

The Simio Extras Library contains two objects designed for modeling multiple cranes operating simultaneously in a bay - including the Crane object and the Bay object. These objects may be used in conjunction with the Simio Standard Library.

Crane pickups are done using the *Ride on Transporter* feature on the standard TransferNode (e.g. the output side of a Server). The Crane will pick up an entity at the requesting TransferNode, and then drop the entity off at its destination node. Crane drop-offs can be done at either a BasicNode or TransferNode. The Bay and Crane objects can also be used with custom libraries as long as they support rider pickups using the standard transporter ride features.

The Bay and Crane objects are used together to model multiple cranes moving in a single bay. A Crane movement occurs by first lifting the load from the pickup node to a specified travel height, traveling laterally at that height, and then lowering the load to the specified drop-off node. All travel is done through free space without the need to explicitly draw a network. The Crane object fully supports independent acceleration/deceleration for lateral movements. The Crane object also supports multiple interacting cranes in a single Bay and provides sophisticated logic for preventing deadlocks.



The Crane is a composite object made up of a Crane hook (transporter), Lift (entity), Bridge (entity), and Cab (entity). To edit the Crane properties, click on the Crane label or the Crane's transporter symbol represented by the hook. Users can also click independently on the Bridge, Cab, and Lift objects to edit their properties. This is typically only done to change the traveling height of the Bridge and or Cab. The key object is the Crane which represents the carrier and is specified as a transporter. The Crane in turn causes its associated Lift, Cab, and Bridge to move across the Bay. The Crane is derived from the Vehicle object in the Standard Library, inheriting many of the same properties.

More Details

A Crane is a transporter that represents the physical tool or device (e.g., a hook or clamp) on the end of the Lift that is used to pick up the load. Locally, a Crane represents the entire device (Bridge, Cab, Lift, and Crane) and is what is specified on the TransferNode to make any move. A Crane is a dynamic Transporter that is created at runtime and may pick up, move, and drop-off entities at Nodes. When using the Standard Library, the Crane is specified in the same way as any other Transporter when traveling between Nodes.

The Crane object is created from the Vehicle and has most of the same properties. However, the Crane has been "locked down" to run in free space only and has a fixed dynamic population of 1. The Crane location is initialized to its *Initial (Home) Node*. Note that the Initial Node for the crane, as well as any pickup or drop-off nodes for the Crane must all fall within the boundary of the Bay. The Crane has additional required properties that specify its *Desired Lateral Speed* and *Lateral Acceleration* and *Lateral Deceleration*.

The *Idle Action* property specifies the behavior of the Crane when it returns to the idle state. The options are 'None' (remain at last location in the down position), 'Go To Home' (move up to the travel height and then make a lateral move to the Initial Node location), and 'Go To Up Position' (move up to the travel height but remain at the last lateral location). Note that once a Crane initiates an idle action, it rejects any new pickup request until the idle action is completed.

Listed below are the properties of the **Crane**:

Property	Valid Entry	Description
Task Selection Strategy	Smallest Distance, Largest Distance, Smallest Priority, Largest Priority, First In Queue	The strategy used by the Crane object to select its next transport pickup or dropoff task.
Load Time	Expression	The time required for this Crane object to load an entity (i.e., the load time per entity).
Unload Time	Expression	The time required for this Crane object to unload an entity (i.e., the unload time per entity).
Minimum Dwell Time Type	Dwell Until Event, Dwell Until Full, No Requirement, Specific Time	Specifies the minimum dwell time requirement for this Crane object when loading and unloading entities at a node. A 'minimum dwell time' is a minimum amount of time the Crane is required to wait, or 'dwell', at a node to load and unload entities.
Event Name	Event Name	The name of the event that will indicate the expiration of the minimum wait time requirement for this Crane object when loading and unloading entities at a node.
Minimum Dwell Time	Expression	The specific minimum amount of time that this Crane object is required to wait, or 'dwell', at a node to load and unload entities.
Dwell Only If	Expression	If specified, this condition will be used to indicate whether the Crane must respect the minimum dwell time at a node if that requirement has not yet expired, thereby forcing the Crane to wait, or 'dwell', at the node for possible additional loads. Example uses of this property might include allowing the Crane to exit a node before the minimum dwell time has expired if the crane has reached full ride capacity, or if there is little to no chance of additional loads. Another example condition might be to only respect the minimum dwell time requirement if the crane is at a node meeting some specific criteria.
Initial Desired Lateral Speed	Expression	The initial desired lateral speed value for this Crane. NOTE: Refer to the user-assignable 'DesiredSpeed' state variable of a Crane to dynamically change its desired lateral speed during a simulation run.
Lateral Acceleration	Real	The rate of acceleration of the Crane when starting a lateral movement across the Bay.
Lateral Deceleration	Real	The rate of deceleration of the Crane when stopping a lateral movement across the Bay.
Lift Speed	Expression	The vertical speed value the Lift moves the Crane in the up and down direction.
Associated Bay	Valid Bay Name	The associated Bay to which this Crane is assigned.
Travel Height	Expression	The vertical distance above the Bay where the Crane travels in the lateral direction.
Initial Priority	Expression	The initial priority value of this Crane object. NOTE: Refer to the user-assignable 'Priority' state of the Crane to dynamically change its priority value during the simulation run.

Initial Node (Home)	Node Instance Name	The initial node location of this Crane object at the beginning of the simulation run and the home location where the Crane returns.
Idle Action	Go To Home, Go To Up Position, None	Specifies the action of the Crane object when it becomes idle. The options include None, Go To Up Position or Go To Home. No pickup requests are accepted until the idle action is completed.
Capacity Type	Fixed, Work Schedule	The availability of this Crane type to perform tasks. 'Fixed' indicates that this type of Crane is always available. 'WorkSchedule' indicates that each Crane of this type follows a work schedule defining its 'On-shift', 'Off-shift' availability over time.
Work Schedule	Schedule Property	The name of the schedule that defines the 'On-shift', 'Off-shift' availability of this type of vehicle over time.
Failure Type	No Failures, Calendar Time Based, Event Count Based	Specifies whether this Crane object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The event count between failure occurrences.
Uptime Between Failures	Expression	The uptime between failure occurrences.
Time To Repair	Expression	The time required to repair a failure when one occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Parent Cost Center	Cost Center Name	The parent cost center that costs charged, or accrued to the Crane are rolled up into. If a parent cost center is not explicitly specified, then costs accrued to a Crane will be automatically rolled up into the parent object that contains

		the Crane's current location.
Capital Cost Per Crane	Expression	The initial one-time setup cost to add a Crane of this type to the system.
Cost Per Rider (Transport Costs)	Expression	The cost to load and transport an entity using a Crane, irrespective of the transportation time. This cost is allocated to an entity when it is loaded onto the Crane.
Transport Cost Rate (Transport Costs)	Expression	The cost per unit time to transport an entity using a Crane.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of a Crane when it is idle.
Cost Per Use(Resource Costs)	Expression	The one-time cost that is accrued each time a Crane is used for a non-transport task, regardless of the usage duration. This cost will be charged to the cost of the owner object(i.e., task) using the Crane.
Usage Cost Rate(Resource Costs)	Expression	The cost per unit time to use a Crane for a non-transport task. This cost will be charged to the cost of the owner object (i.e., task) using the Crane.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is ending.
Allocated Add-On Process Triggers	Process Instance Name	Occurs when this Crane resource has been allocated to perform some transport tasks, or when another object has seized this Crane resource for some non-transport related task.
Released Add-On Process Triggers	Process Instance Name	Occurs when this Crane resource has been released after completing a set of transport tasks, or when another object has released this Crane resource from some non-transport related task.
Failed Add-On Process Triggers	Process Instance Name	Occurs when the Crane has failed.
Repaired Add-On Process Triggers	Process Instance Name	Occurs when a failure is repaired at the Crane.
Entered Node Add-On Process Triggers	Process Instance Name	Occurs when this Crane has entered a node's crossing area, including occurrences when the Crane object has entered by unparking from a node's parking station.

Unloaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is unloaded from this Crane.
Loaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is loaded onto the Crane.
Exiting Node Add-On Process Triggers	Process Instance Name	Occurs when this Crane object is about to begin exiting a node's crossing area, including occurrences when the Crane is exiting by parking in a node's parking station.
Evaluating Transport Request Add-On Process Triggers	Process Instance Name	Occurs when this Crane is evaluating whether to accept or reject a transport request from a potential rider. In the executed decision process, assigning the value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the transport request is rejected.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when a Crane is evaluating whether to accept or reject a request from another object to seize the Crane resource. In the executed decision process, assigning a value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the seize request is rejected.
On Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'off shift' because of a specified work schedule.
Log Resource Usage	True or False	Indicates whether usage related events for this Crane are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Name	String Expression	String expression returning the name displayed for this object in log entries and trace. If not specified, then this property defaults to the object's unique name.
Gantt Visibility	True or False	An expression evaluated at run time to determine if the agent/entity should appear in the Entity Gantt window.
Experiment Animation Events	True or False	If True, incremental animation events are executed when running in experiment mode. This will generate the exact same sequence of events as in animation mode, but will execute much slower.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.
Bridge - Travel Height	Expression	The travel height of the center of the Bridge above the floor.

Cab - Travel Height	Expression	The travel height of the center of the Cab above the floor. The Cab may be placed above or below its associated Bridge.
---------------------	------------	---

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Robot

The Simio Extras Library contains a Robot object designed for modeling entity movement between locations via a revolute robot. The Robot may be used in conjunction with the Simio Standard Library.

The Simio Robot is a composite object that is used to model a revolute/jointed robot capable of picking up entities at one node and dropping them at a second node. This is the most common robot configuration found in industrial applications. The robot composite object is comprised of five objects that include the Robot (fixed base object), the Robot Rotator (entity), Robot Upper Arm (entity), Robot Lower Arm (entity), and the Robot Hand (transporter). Only the Robot fixed base object is visible in the Extras library; the other four components are automatically created by the Robot Base when it is placed in the Facility window.

The Robot object may be used in conjunction with the Standard Library or any custom library that supports standard pickup and drop-off logic using transporters. Robot pickups are done using the *Ride on Transporter* feature on the TransferNode. The Hand will pick up the entity at the requesting TransferNode, then drop the entity off at its destination node. The Hand is specified as the transporter when initiating a transfer using the Robot.

The Rotator turns about its vertical axis that is attached to the dark gray base to align the Robot through 360 degrees of rotation. The Upper Arm rotates in pitch relative to the extended arm of the Rotator, and the Lower Arm rotates in pitch relative to the Upper Arm. These three axes of rotation can position the tip of the Lower Arm at any location within its reach. The Hand is attached to the end of the Lower Arm and can pitch up and down.

The properties for the Robot are edited by selecting the Base or the individual components of the Robot. Basic editing is done by selecting the Base. This is used for editing the *Initial Node (Home)* for the Hand, which is required to both initialize the Hand position, and provide a home node when the *Idle Action* on the Hand is specified as *Go to Home*. The *Travel Mode* is also specified on the Base as either 'Fixed Height' or 'Direct'. In the case of the 'Fixed Height' mode, the Robot first moves up/down as necessary to a specified height before the Rotator is permitted to rotate to its next location. Once the rotation is completed, it then moves up/down as necessary to its destination. In the case of the 'Direct' mode the Rotator, Upper Arm, and Lower Arm all move simultaneously as needed to reach its new destination. In general, 'Direct' moves execute faster than 'Fixed Height' moves.

These components are typically only edited to change the rotation speeds. The Hand is the actual transporter device that is used to pick up and drop off entities; it is selected to edit the transporter related properties of the Robot.

The Rotator has a property named *Rotation Rate* that is specified in degrees per second. When needed to execute a move, the Rotator can rotate in either direction at the specified rate. The Rotator also has a set of properties for specifying the geometry of the joints associated with the Rotator. The geometry properties of the Robot are typically not changed unless the Robot components are resized or the graphics for the components are modified/changed. The Rotator has an additional animation property to define the degrees of Rotator turn per animation update. Smaller values provide a smoother animation. The Upper Arm specifies its *Pitch Change Rate* in degrees per second. This is the rate at which the Upper Arm changes pitch relative to the rotator. When needed to execute a move, the Upper Arm can rotate in either direction of pitch at this specified rate. The Upper Arm offset values should only be changed from their default when updating the sizes or replacing the graphical symbols for the components.

The Lower Arm specifies its *Pitch Change Rate* in degrees per second. This is the rate at which the Lower Arm changes pitch relative to the Upper Arm. When needed to execute a move, the Lower Arm can rotate in either direction of pitch at this specified rate. The geometry properties for the Lower Arm includes joints offsets like the Upper Arm, as well as a vertical offset for the Hand and a lateral offset for moving the Lower Arm sideways from the centerline.

The Hand is a transporter that represents the physical tool or device that is attached to the end of the Lower Arm and picks up and drops off the entity. The Hand is specified on the TransferNode as the transporter that is used to pick up an entity and move it to its destination. The Hand object is derived from the Vehicle and has many of the same properties. However, the Hand has been "locked down" to run in free space only.

The Hand may change its pitch relative to the Lower Arm to which it is attached. The *Pitch Change Rate* is specified as a property of the Hand, along with the *Loading Pitch Angle*, *Travel Pitch Angle*, and *Unloading Pitch Angle*. These last three properties are used to specify the desired pitch angle of the Hand relative to the floor when loading, traveling, or unloading.

The size of the Rotator, Upper Arm, Lower Arm, and Hand can be modified by changing the *Length*, *Width*, and *Height* properties in the Physical Characteristics section of the General property group on each of the objects. Note that this might change the reach of the Robot. When changing the size of the Rotator, Upper Arm, Lower Arm, and Hand, the Joint Geometry will need to be modified accordingly. The Joint Geometry properties can be found in the Joint Geometry property groups on each of these objects. Refer to the descriptions below or on the Robot properties to determine which properties must be updated to adjust for the change in size.

Listed below are the properties of the **Robot**:

Property	Valid Entry	Description
Initial Node (Home)	Node Name	The initial node location of the Robot Hand at the beginning of the simulation run.
Travel Mode	Fixed Height, Direct	The Travel Mode for this Robot can be either 'Direct' or 'Fixed Height'. If 'Direct', the Robot moves in a direct path to its destination. .If 'Fixed Height', the Robot first moves to a specified height, and then travels at that height to its destination.
Travel Height	Expression	The travel height for the Robot when using a 'Fixed Height' travel mode.
Experiment Animation Events	True, False	If True, incremental animation events are executed when running in experiment mode. This will generate the exact same sequence of events as in animation mode, but execute much slower.
RobotRotator - Rotation Rate	Expression	The rotation rate, in degrees per second, of the Rotator which is attached to the base.
RobotRotator - Run Initialized (Add-On Process Triggers)	Process Name	Occurs when the simulation run is initialized.
RobotRotator - Run Ending (Add-On Process Triggers)	Process Name	Occurs when the simulation run is ending.
RobotRotator - Radial Offset (Joint Geometry)	Real	The radial offset of the lower arm attachment joint from the center of the base in the direction of the robot hand.
RobotRotator - Vertical Offset (Joint Geometry)	Real	The vertical offset of the attachment joint measured from the top of the Rotator.
RobotRotator - Rotational Offset X (Joint Geometry)	Real	The offset of the Rotator relative to the Base in the X direction.
RobotRotator - Rotational Offset Y (Joint Geometry)	Real	The offset of the Rotator relative to the Base in the Y direction.
RobotRotator - Rotational Offset Z (Joint Geometry)	Real	The offset of the Rotator relative to the Base in the Z direction.
RobotRotator - Degrees Rotation Per Update (Animation)	Real	The maximum degrees of rotation with each update of the base, arms and hand of the robot. The smaller this value, the smoother the animation.

RobotLowerArm - Pitch Change Rate	Expression	The pitch change rate, specified in degrees per second, of the lower arm relative to the base.
RobotLowerArm - Run Initialized (Add-On Process Triggers)	Process Name	Occurs when the simulation run is initialized.
RobotLowerArm - Run Ending (Add-On Process Triggers)	Process Name	Occurs when the simulation run is ending.
RobotLowerArm - Rotator Joint Length Offset (Joint Geometry)	Real	The offset of the interior rotator attachment joint measured outward from the start of the lower arm.
RobotLowerArm - Upper Joint Length Offset (Joint Geometry)	Real	The offset of the interior upper arm attachment joint measured inward from the end of the lower arm.
RobotLowerArm - Lateral Offset (Joint Geometry)	Real	The offset of the lower arm measured in distance to the right of the centerline of the robot extension.
RobotUpperArm - Pitch Change Rate	Expression	The pitch change rate, specified in degrees per second, of the upper arm relative to the lower arm.
RobotUpperArm - Run Initialized (Add-On Process Triggers)	Process Name	Occurs when the simulation run is initialized.
RobotUpperArm - Run Ending (Add-On Process Triggers)	Process Name	Occurs when the simulation run is ending.
RobotUpperArm - Lower Joint Length Offset (Joint Geometry)	Real	The offset of the interior lower arm attachment joint measured outward from the start of the upper arm.
RobotUpperArm - Hand Joint Length Offset (Joint Geometry)	Real	The offset of the interior robot hand attachment joint measured inward from the end of the upper arm.
RobotUpperArm - Hand Vertical Offset (Joint Geometry)	Real	The offset of the robot hand in the vertical direction measured going up from the centerline of the upper arm.
RobotUpperArm - Lateral Offset (Joint Geometry)	Real	The offset of the upper arm measured in distance to the right of the centerline of the robot extension.

RobotHand - Task Selection Strategy	Smallest Distance, Largest Distance, Smallest Priority, Largest Priority, First In Queue	The strategy used by the RobotHand to select its next transport pickup or dropoff task.
RobotHand - Load Time	Expression	The time required for this RobotHand to load an entity (i.e., the load time per entity).
RobotHand - Unload Time	Expression	The time required for this RobotHand to unload an entity (i.e., the unload time per entity).
RobotHand - Loading Pitch Angle	Expression	The pitch angle of the end effector relative to the floor during loading (pickup) of an entity.
RobotHand - Unloading Pitch Angle	Expression	The pitch angle of the end effector relative to the floor during unloading (dropoff) of an entity.
RobotHand - Travel Pitch Angle	Expression	The pitch angle for the Robot's end effector relative to the floor when traveling.
RobotHand - Pitch Change Rate	Expression	The pitch change rate of the end effector in degrees per second.
RobotHand - Initial Priority	Expression	The initial priority value of this RobotHand at the beginning of the simulation run.
RobotHand - Idle Action	Go to Home, Park At Home	The action of this RobotHand when it does not have any tasks to perform.
RobotHand - Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order requests waiting to seize this Robot for a process task.
RobotHand - Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' static ranking rule for ordering requests waiting to be allocated capacity of this Robot.
RobotHand - Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this Robot, when it becomes available, dynamically selects the next seize request from its statically ranked allocation queue using a dynamic selection rule.
RobotHand - Parent Cost Center	Cost Center Name	The cost center that costs allocated to robots of this type are rolled up into. If a parent cost center is not explicitly specified, then costs allocated to a robot will be automatically rolled up into the parent object that contains the robot's current location.
RobotHand - Capital Cost Per Robot	Expression	The initial one-time setup cost to add a robot of this type to the system.
RobotHand - Cost Per Rider	Expression	The cost to load and transport an entity using a transporter of this type, irrespective of the transportation time. This cost is allocated to an entity

(Transport Costs)		when it is loaded onto the transporter.
RobotHand - Transport Cost Rate (Transport Costs)	Expression	The cost per unit time to transport an entity using a transporter of this type.
RobotHand - Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of a robot of this type when it is idle.
RobotHand - Cost Per Use(Resource Costs)	Expression	The one-time cost that is accrued each time a robot of this type is used for a non-transport task, regardless of the usage duration. This cost will be charged to the cost of the owner object(i.e., task) using the robot.
RobotHand - Usage Cost Rate(Resource Costs)	Expression	The cost per unit time to use a robot of this type for a non-transport task. This cost will be charged to the cost of the owner object (i.e., task) using the robot.
RobotHand - Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized.
RobotHand - Run Ending Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is ending.
RobotHand - Allocated Add-On Process Triggers	Process Instance Name	Occurs when this RobotHand has been allocated to perform a transport tasks.
RobotHand - Released Add-On Process Triggers	Process Instance Name	Occurs when this RobotHand has been released after completing a transport task.
RobotHand - Entered Node Add-On Process Triggers	Process Instance Name	Occurs when this RobotHand has entered a node.
RobotHand - Unloaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is unloaded from this RobotHand.
RobotHand - Loaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is loaded onto the RobotHand.
RobotHand - Exiting Node Add-On Process Triggers	Process Instance Name	Occurs when this RobotHand is about to begin exiting a node.

RobotHand - Evaluating Transport Request Add-On Process Triggers	Process Instance Name	Occurs when this RobotHand is evaluating whether or not to accept the pickup of an entity. In the executed decision process, assigning the value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the potential rider is rejected.
RobotHand - Upper Joint Length Offset	Expression	The offset of the interior arm attachment joint measured outward from the start of the robot.
RobotHand - Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Rack

The Simio Extras Library contains two objects designed for modeling entity storage and retrieval - the LiftTruck object and the Rack object. These objects may be used in conjunction with the Simio Standard Library.

The Rack is comprised of the Rack, a Fixed object representing the Rack's frame, and one or more associated Shelves, which are dynamically created entities representing individual shelves on the Rack. The Rack's Shelf configuration can be generated in using two different methods – Pattern or Repeat Group.

When using the Pattern shelf configuration method, the *Number of Shelves*, *Shelf Capacity*, *Height of First Shelf*, and *Shelf Spacing* properties are used to create the shelf configuration. The first Shelf will be generated at the height specified by the *Height of First Shelf* property, and a number of Shelves corresponding to the *Number of Shelves* property will be generated at spacings designated by the Shelf Spacing property. Each Shelf will be capable of holding a number of entities equal to the *Shelf Capacity* property.

When using the Repeat Group shelf configuration method, each Shelf is generated from row in the Shelf Configuration repeat group. Each Shelves' *Shelf ID*, *Height*, and *Storage Capacity* are specified in the repeat group, allowing the generation of custom shelf configurations.

The Rack has an Input node where new entities can arrive to the Rack, and an Output node where entities can depart from the Rack once their storage requirement has been met. The entity arrivals and departures can use direct network transfers or can be picked up and dropped-off at the Rack using any type of transporter such as a Vehicle or Worker. If a LiftTruck is employed, the LiftDevice will first move to the appropriate Shelf height before the transfer can take place. This allows the explicit modeling of delays required by the moving of the LiftDevice.

When new entities arrive to the Rack, they are placed on an available Shelf. The Shelf Selection Rule is used to select between the available Shelves. The 'Specific' Shelf Selection Rule can only be used when Shelves are generated with a repeat group. When the 'Specific' Shelf Selection Rule is used, the Rack will place entities on the Shelf specified by the *Destination Shelf* property.

Each Shelf in the Rack has two animated station queues. The Shelf Storage station holds entities that are being stored on the Shelf. The Transfer Staging station holds entities that are being transferred onto a LiftTruck and are currently waiting for the LiftDevice to complete its move before the transfer can take place. Once the LiftDevice move is completed the entity transfers from the Transfer Staging station queue to the LiftTruck's ride station, which is not animated. However, the entity is also inserted into the LiftDevice's Lift Storage queue which animates the entities' ride location on the LiftDevice.

The amount of time that an arriving entity is stored on the Rack is determined by the *Hold Type*, which can be specified as either 'Specific Hold Time', or 'Release Interval'. In the default case of 'Specific Hold Time', the holding time on the Shelf is specified by the *Shelf Storage Time* expression. In the case of the 'Release Interval', an expression for the *Time Between Releases* is specified. The Rack can be initialized with entities at the beginning of the simulation by specifying the *Initial Quantity*, *Initial Entity Type*, and *Initial Destination Node* for the starting entities. This information can also be specified in a repeat group. These properties are specified under the Initial Storage category.

Entities that enter the Rack object are immediately inserted into the Rack's Entities Storage element, which allows statistics about the entities on the Rack to be obtained. Use expressions such as 'Rack.Entities.Queue.NumberWaiting' to find the number of entities currently stored in the Rack, or 'Rack.Entities.Queue.AverageTimeWaiting' to find the average time entities wait in the Rack. The number of available locations on the Rack can be determined using the expression 'Rack.RackStorageStation.Capacity.Remaining'.

Listed below are the properties of the **Rack**:

Property	Valid Entry	Description
Shelf Configuration Type	Pattern, Repeat Group	The <i>Shelf Configuration Type</i> determines how the Shelves on the Rack are created. If the <i>Shelf Configuration Type</i> is 'Pattern', the <i>Number of Shelves</i> , <i>Shelf Capacity</i> , <i>Height of First Shelf</i> , and <i>Shelf Spacing</i> properties are used to generate the Shelves on the Rack. If the <i>Shelf Configuration Type</i> is 'Repeat Group', Shelves are generated based on entries in the <i>Shelf Configuration</i> repeat group.
Shelf ID (Shelf Configuration)	String	A Shelf identification string used to differentiate different shelves on the same Rack. The Shelf ID property should be unique for all shelves on the same Rack if the 'Specific' Shelf Selection Rule is used.

Height (Shelf Configuration)	Real	The height of each Shelf above the base of the Rack.
Storage Capacity (Shelf Configuration)	Integer	The number of entities which can be stored on this Shelf.
Number of Shelves	Integer	The number of Shelves in this Rack.
Shelf Capacity	Integer	The maximum number of entities that can be placed on each Shelf.
Height of First Shelf	Real	The height of the first Shelf above the base for this Rack.
Shelf Spacing	Real	The distance between successive Shelves after the first.
Shelf Selection Rule	Lowest Available, Highest Available, Least Remaining Capacity, Most Remaining Capacity, Specific	The rule used to select a Shelf when dropping off an entity. When specified as 'Specific', the Rack will transfer entities to the Shelf with the Shelf ID that matches the Destination Shelf property. The 'Specific' Shelf Selection Rule cannot be used with the 'Pattern' Shelf Configuration Type.
Destination Shelf	Expression	The ID of the destination Shelf when the Shelf Selection Rule is set to 'Specific'.
Transfer In Time	Expression	The time required to move an entity onto a Shelf.
Hold Type	Specific Hold Time, Release Interval	The hold type for each entity specified as a Specific Time or Release Interval. In the case of a Release Interval, each entity is released one at a time based on a time between releases.
Shelf Storage Time	Expression	The holding time for an entity that is placed on a Shelf when the <i>Hold Type</i> is set to 'Specific Hold Time'. After this time, the entity will depart the Rack.
Time Between Releases	Expression	The time between each entity release when the <i>Hold Type</i> is set to 'Release Interval'. Entities are released one at a time until the Rack becomes empty.
Repeat Group (Initial Entities)	True/False	Indicates whether a repeat group data structure is used to define the initial entities on the Rack.
Initial Entity Type (Initial Entities)	Entity Instance Name	The type of entity initialized at this Rack.
Initial Quantity (Initial Entities)	Integer	The number of this type of entity initialized at this Rack.
Initial Destination Node (Initial Entities)	Node Instance Name	The initial destination node for the entities initialized at this Rack.
Report	True or False	A Boolean property may be True or False and used in expressions as a

Statistics

numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

LiftTruck

The Simio Extras Library contains two objects designed for modeling entity storage and retrieval - the LiftTruck object and the Rack object. These objects may be used in conjunction with the Simio Standard Library.

The LiftTruck is comprised of a main LiftTruck (transporter) and an associated LiftDevice (entity). The LiftTruck moves its associated LiftDevice up and down to the appropriate Shelf height when placing and removing entities onto and off a Rack. The LiftTruck is like the Vehicle and has many of the same features and properties, with the exception that a LiftTruck cannot park at a Node.

In addition to the normal Vehicle properties, the LiftTruck has two additional properties, *Vertical Lift Speed*, and the *Lift Travel Height*. These specify the speed at which the LiftDevice moves up and down, and the height that it travels when moving between locations.

By default, the LiftTruck makes a smooth transition when moving between link segments. However, the associated LiftDevice changes orientation as soon as the LiftTruck starts on the next link segment. The LiftTruck and LiftDevice can be made to move together by changing the *Link Transfer Transition Type* in the Animation category to 'Immediate'.

Listed below are the properties of the **LiftTruck**:

Property	Valid Entry	Description
Initial Ride Capacity	Integer	The initial carrying capacity of this LiftTruck object.
Task Selection Strategy	Smallest Distance, Largest Distance, Smallest Priority, Largest Priority, First In Queue	The strategy used by the LiftTruck object to select its next transport pickup or dropoff task.
Load Time	Expression	The time required for this LiftTruck object to load an entity (i.e., the load time per entity).
Unload Time	Expression	The time required for this LiftTruck object to unload an entity (i.e., the unload time per entity).
Minimum Dwell Time Type	Dwell Until Event, Dwell Until Full, No Requirement, Specific Time	Specifies the minimum dwell time requirement for this LiftTruck object when loading and unloading entities at a node. A 'minimum dwell time' is a minimum amount of time the LiftTruck is required to wait, or 'dwell', at a node to load and unload entities.
Event Name	Event Name	The name of the event that will indicate the expiration of the minimum wait time requirement for this LiftTruck object when loading and unloading entities at a node.
Minimum Dwell Time	Expression	The specific minimum amount of time that this LiftTruck object is required to wait, or 'dwell', at a node to load and unload entities.
Dwell Only If	Expression	If specified, this condition will be used to indicate whether the LiftTruck must respect the minimum dwell time at a node if that requirement has not yet expired, thereby forcing the LiftTruck to wait, or 'dwell', at the node for possible additional loads. Example uses of this property might include allowing the LiftTruck to exit a node before the minimum dwell time has expired if the LiftTruck has reached full ride capacity, or if there is little to no chance of additional loads. Another example condition might be to only

		respect the minimum dwell time requirement if the LiftTruck is at a node meeting some specific criteria.
Vertical Lift Speed	Expression	The vertical speed of the LiftDevice when moving up or down.
Lift Travel Height	Expression	The height of the LiftDevice relative to the base of the LiftTruck when traveling.
Initial Desired Speed	Expression	The initial desired speed value for this LiftTruck when traveling between locations.
Initial Travel Mode	Free Space Only, Network Only, Network If Possible	<p>The initial travel mode for the LiftTruck.</p> <p>'Free Space Only' indicates that the entities are required to perform travel movements in free space.</p> <p>'Network Only' indicates that the entities are required to perform travel movements using the currently assigned network.</p> <p>'Network If Possible' indicates a preference for the entities to perform travel movement using the currently assigned network, but if no followable network path exists to the next destination then travel in free space is allowed.</p> <p>NOTE: Refer to the user-assignable 'CurrentTravelMode' state variable of an entity to dynamically change its travel mode during a simulation run. An entity's travel mode may also be changed using the 'Outbound Travel Mode' property provided by a node.</p>
Initial Network	Network Instance Name	The initial network of links used by this LiftTruck object to travel between node locations.
Network Turnaround Method	Exit & Re-enter, Rotate in Place, Reverse	When traveling on a network, the default method used by objects of this type to turn around at a node in order to move in the opposite direction on a bidirectional link. The 'Turnaround Method' property of a Network element may also be used to override this default on a network-by-network basis.
Free Space Steering Behavior	Direct To Destination, Follow Network Path If Possible	<p>The behavior used to steer an object of this type when traveling in free space to a destination.</p> <p>'Direct To Destination' will steer an entity in a straight line to its destination.</p> <p>'Follow Network Path If Possible' will prefer to steer an entity along a path following its currently assigned network, staying within the boundaries of the drawn path width. The Entity will travel based on the drawn length of the path even if a different logical length is specified. However, if no followable network path exists to the entity's destination then the entity will be steered in a straight line. Note: In order to use this steering behavior, make sure the entity's travel mode is set to 'Free Space Only'.</p>
Update Interval	Expression	The time interval between steering adjustments to maintain adherence to the network path and avoid collisions.
Avoid Collisions	True or False	Indicates whether the steering behavior will attempt to avoid collisions with other entities that are following the same network path.
Initial Priority	Expression	The initial priority value of this LiftTruck object at the beginning of the simulation run.
Initial Node (Home)	Node Instance Name	The initial node location of this LiftTruck object at the beginning of the simulation run. Also indicates the LiftTruck's initial 'Home' location if the object is specified as 'Go To Home' when idle.
Routing Type	On Demand, Fixed Route	Specifies the routing behavior of this LiftTruck. If 'On Demand', then the LiftTruck will route to locations on the network according to visit requests. An 'On Demand' LiftTruck may also be used as a moveable resource that is seized and released for non-transport related tasks by model process logic. If 'Fixed Route', then the LiftTruck will follow a fixed route sequence to

		transport riders and will only load and unload entities at locations on the specified route.
Route Sequence	Sequence Instance Name	The sequence table that defines the route sequence that this LiftTruck will loop.
Idle Action	Go to Home, Remain In Place, Roam	The action of this LiftTruck object when it does not have any tasks to perform.
Off Shift Action	Go to Home, Remain In Place	The action of this LiftTruck object when it goes 'off shift'. Note that, if the LiftTruck is busy or transporting when an off shift period begins, then this action will be taken when the LiftTruck is released from (i.e., finishes) the remaining tasks.
Capacity Type	Fixed, Work Schedule	The availability of this LiftTruck to perform tasks. 'Fixed' indicates that this type of LiftTruck is always available. 'WorkSchedule' indicates that each LiftTruck of this type follows a work schedule defining its 'On-shift', 'Off-shift' availability over time.
Work Schedule	Schedule Property	The name of the schedule that defines the 'On-shift', 'Off-shift' availability of this type of LiftTruck over time.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this LiftTruck.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this LiftTruck.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order requests waiting to seize this LiftTruck object for a process task. Note: This feature only applies to an 'On Demand' routing type LiftTruck.
Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' static ranking rule for ordering requests waiting to be allocated capacity of this object.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this LiftTruck, when it becomes available, dynamically selects the next seize request from its statically ranked allocation queue using a dynamic selection rule. Note: This feature only applies to an 'On Demand' routing type LiftTruck.
Failure Type	No Failures, Calendar Time Based, Event Count Based	Specifies whether this LiftTruck object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.

Failures		
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Uptime Between Failures	Expression	The uptime bewteen failure occurences.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs.
Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to LiftTrucks of this type are rolled up into. If a parent cost center is not explicitly specified, then costs allocated to a LiftTruck will be automatically rolled up into the parent object that contains the LiftTruck's current location.
Capital Cost Per LiftTruck	Expression	The initial one-time setup cost to add a LiftTruck of this type to the system.
Cost Per Rider (Transport Costs)	Expression	The cost to load and transport an entity using a transporter of this type, irrespective of the transportation time. This cost is allocated to an entity when it is loaded onto the transporter.
Transport Cost Rate (Transport Costs)	Expression	The cost per unit time to transport an entity using a transporter of this type.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of a LiftTruck of this type when it is idle.
Cost Per Use(Resource Costs)	Expression	The one-time cost that is accrued each time a LiftTruck of this type is used for a non-transport task, regardless of the usage duration. This cost will be charged to the cost of the owner object(i.e., task) using the LiftTruck.
Usage Cost	Expression	The cost per unit time to use a LiftTruck of this type for a non-transport task.

Rate(Resource Costs)		This cost will be charged to the cost of the owner object (i.e., task) using the LiftTruck.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized. Note: This trigger will only occur for Vehicle objects created by the Initial Number in System setting.
Run Ending Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is ending.
Allocated Add-On Process Triggers	Process Instance Name	Occurs when this LiftTruck resource has been allocated to perform some transport tasks, or when another object has seized this LiftTruck for some non-transport related task.
Released Add-On Process Triggers	Process Instance Name	Occurs when this LiftTruck resource has been released after completing a set of transport tasks, or when another object has released this LiftTruck from some non-transport related task.
Failed Add-On Process Triggers	Process Instance Name	Occurs when the LiftTruck has failed.
Repaired Add-On Process Triggers	Process Instance Name	Occurs when a failure is repaired at the LiftTruck.
Entered Node Add-On Process Triggers	Process Instance Name	Occurs when this LiftTruck has entered a node.
Unloaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is unloaded from this LiftTruck.
Loaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is loaded onto the LiftTruck.
Exiting Node Add-On Process Triggers	Process Instance Name	Occurs when this LiftTruck object is about to begin exiting a node.
Evaluating Transport Request Add-On Process Triggers	Process Instance Name	Occurs when this LiftTruck is evaluating whether or not to accept the pickup of a rider. In the executed decision process, assigning the value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the potential rider is rejected.
Evaluating Seize Request Add-On	Process Instance Name	Occurs when an 'On Demand' routing type LiftTruck is evaluating whether to accept or reject a seize allocation attempt by some process in the system. In the executed decision process, assigning a value less than or equal to 0.0 to

Process Triggers		the executing token's ReturnValue state (Token.ReturnValue) indicates that the allocation attempt is rejected.
On Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'off shift' because of a specified work schedule.
Initial Number in System	Integer	The initial number of LiftTruck objects of this type in the system. Vehicles are initially located in free-space at the object instance location.
Maximum Number in System	Integer	The maximum number of objects of this instance that can be simultaneously in the system. If this limit is exceeded then a runtime error is generated.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Edition). If not specified, then this property defaults to a color derived from the object's unique name.
Can Transfer In & Out of Objects	True or False	Indicates whether objects of this type are by default allowed to transfer into and out of fixed objects via those object's external input / output nodes.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.
Link Segment Transition Type	Smooth, Immediate	The type of link movement transitions for this object. Smooth indicates the object will move smoothly between different angles of the different link segments. Immediate will immediately change the angle of the object at the next link segment.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Elevator

The Simio Extras Library contains two objects designed for modeling elevators - the Elevator and the ElevatorNode. These objects may be used in conjunction with the Simio Standard Library.

The Elevator is a transporter that moves up and down in the vertical direction, picking up and dropping off entities at associated ElevatorNodes. Each ElevatorNode references its associated Elevator, and each Elevator has a list of associated ElevatorNodes that it can visit. When modeling a single elevator, only the Elevator object and its associated ElevatorNodes are required. When an entity arrives to an ElevatorNode, it issues a pickup request to the Elevator. The movement logic for the Elevator is to move in a single vertical direction (up or down) until all drop offs and pickups have been done while continuing movement in that direction. Entities will only enter the Elevator if it is currently moving in its desired direction. Once all pickups and drop-offs in the current direction have been completed the elevator then changes direction and continues with additional pickups and drop-offs. When the Elevator becomes idle it either remains in place or returns to its home ElevatorNode based on the specified *Idle Action* for the Elevator.

Bank of Elevators

When modeling a bank of two or more elevators, an ElevatorNode may be used at each elevator level (floor) to route arriving entities to the appropriate elevator. This selection is made based on the elevator that must travel the least distance to accomplish the pickup – given the current workload already assigned to the elevator. In the case of an approaching elevator moving in the desired direction, this is simply the current distance from the elevator. Otherwise, this is the travel distance for the elevator to complete its existing pickups and drop-offs and then return to this location. Note that a separate ElevatorNode acting as a Selection Point is required at each level of the elevator bank.

There is no limit to the number of floors (ElevatorNodes) that can be assigned to an elevator or to the number of Elevators that can be used in an elevator bank. Also, note that the Elevator supports acceleration and deceleration to accurately model movement times.

Listed below are the properties of the **Elevator**:

Property	Valid Entry	Description
Initial Ride Capacity	Integer	The initial carrying capacity of this Elevator object. NOTE: Refer to the user-assignable 'RideStationCurrentCapacity' state variable of a transporter to dynamically change its ride capacity during a simulation run.
Load Time	Expression	The time required for this Elevator object to load an entity (i.e., the load time per entity).
Unload Time	Expression	The time required for this Elevator object to unload an entity (i.e., the unload time per entity).
Minimum Dwell Time Type	Dwell Until Event, Dwell Until Full, No Requirement, Specific Time	Specifies the minimum dwell time requirement for this Elevator object when loading and unloading entities at a node. A 'minimum dwell time' is a minimum amount of time the Elevator is required to wait, or 'dwell', at a node to load and unload entities.
Event Name	Event Name	The name of the event that will indicate the expiration of the minimum wait time requirement for this Elevator object when loading and unloading entities at a node.
Minimum Dwell Time	Expression	The specific minimum amount of time that this Elevator object is required to wait, or 'dwell', at a node to load and unload entities.
Dwell Only If	Expression	If specified, this condition will be used to indicate whether the Elevator must respect the minimum dwell time at a node if that requirement has not yet expired, thereby forcing the elevator to wait, or 'dwell', at the node for possible additional loads. Example uses of this property might include allowing the elevator to exit a

		node before the minimum dwell time has expired if the elevator has reached full ride capacity, or if there is little to no chance of additional loads. Another example condition might be to only respect the minimum dwell time requirement if the elevator is at a node meeting some specific criteria.
Elevator Node List	Node List	The list of elevator nodes that this elevator may visit for picking up and dropping off entities.
Initial Desired Speed	Expression	The initial desired speed value for this Vehicle object when traveling between locations.
Acceleration	Expression	The acceleration of this vehicle object when traveling between locations.
Deceleration	Expression	The deceleration of this vehicle object when traveling between locations.
Initial Network	Network Instance Name	The initial network of links used by this Vehicle object to travel between node locations.
Network Turnaround Method	Exit & Re-enter, Rotate in Place, Reverse	When traveling on a network, the default method used by objects of this type to turn around at a node in order to move in the opposite direction on a bidirectional link. The 'Turnaround Method' property of a Network element may also be used to override this default on a network-by-network basis.
Free Space Steering Behavior	Direct To Destination, Follow Network Path If Possible	The behavior used to steer an entity of this type when traveling in free space to a destination. 'Direct To Destination' will steer an entity in a straight line to its destination. 'Follow Network Path If Possible' will prefer to steer an entity along a path following its currently assigned network, staying within the boundaries of the drawn path width. However, if no followable network path exists to the entity's destination then the entity will be steered in a straight line. Note: In order to use this steering behavior, make sure the entity's travel mode is set to 'Free Space Only'.
Update Interval	Expression	The time interval between updates checking an entity's adherence to the network path.
Avoid Collisions	True or False	Indicates whether the steering behavior will attempt to avoid collisions with other entities that are following the same network path.
Initial Priority	Expression	The initial priority value of this Vehicle object at the beginning of the simulation run.
Initial Node (Home)	Node Instance Name	The initial node location of this Vehicle object at the beginning of the simulation run. Also indicates the vehicle's 'Home' location if the object's 'Idle Action' is specified as 'Park At Home' or 'Go To Home'.
Routing Type	On Demand, Fixed Route	Specifies the routing behavior of this Vehicle. If 'On Demand', then the vehicle will route to locations on the network according to visit requests. An 'On Demand' vehicle may also be used as a moveable resource that is seized and released for non-transport related tasks by model process logic. If 'Fixed Route', then the vehicle will follow a fixed route sequence to transport riders and will only load and unload entities at locations on the specified route.
Route Sequence	Sequence Instance Name	The sequence table that defines the route sequence that this vehicle will loop.
Idle Action	Park At Node, Go to Home,	The action of this Vehicle object when it does not have any transportation tasks to perform.

	Remain In Place, Park At Home, Roam	
Off Shift Action	Park At Node, Go to Home, Remain In Place, Park At Home	The action of this Vehicle object when it goes 'off shift'. Note that, if the vehicle is busy or transporting when an off shift period begins, then this action will be taken when the vehicle is released from (i.e., finishes the remaining tasks).
Capacity Type	Fixed, Work Schedule	The availability of this vehicle type to perform tasks. 'Fixed' indicates that this type of vehicle is always available. 'WorkSchedule' indicates that each vehicle of this type follows a work schedule defining its 'On-shift', 'Off-shift' availability over time.
Work Schedule	Schedule Property	The name of the schedule that defines the 'On-shift', 'Off-shift' availability of this type of vehicle over time.
Work Day Exceptions	Repeat Group, Work Day Exceptions	Work Day schedule exceptions specific to this resource.
Work Period Exceptions	Repeat Group, Work Period Exceptions	Work Period schedule exceptions specific to this resource.
Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static ranking rule used to order requests waiting to seize this Vehicle object for a process task. Note: This feature only applies to an 'On Demand' routing type vehicle.
Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' static ranking rule for ordering requests waiting to be allocated capacity of this object.
Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	Indicates whether this vehicle, when it becomes available, dynamically selects the next seize request from its statically ranked allocation queue using a dynamic selection rule. Note: This feature only applies to an 'On Demand' routing type vehicle.
Park While Busy	True or False	Indicates whether this Vehicle object should park while busy at a node due to a seize request. Parking a vehicle can be useful to avoid blocking other travelers on the network.
Failure Type	No Failures, Calendar Time Based, Event Count Based	Specifies whether this Vehicle object has failures that affect the object's availability, and the method used to trigger the failure occurrences.
Event Name	Event Instance Name	The name of the event which determines when the next failure occurs.

Uptime Between Failures	Expression	The uptime between failure occurrences. This property may be specified using a random sample from a distribution.
Count Between Failures	Expression	The count between failure occurrences. This property may be specified using a random sample from a distribution.
Time To Repair	Expression	The time required to repair a failure when one occurs. This property may be specified using a random sample from a distribution.
Parent Cost Center	Cost Center Name	The cost center that costs allocated to vehicles of this type are rolled up into. If a parent cost center is not explicitly specified, then costs allocated to a vehicle will be automatically rolled up into the parent object that contains the vehicle's current location.
Capital Cost Per Vehicle	Expression	The initial one-time setup cost to add a vehicle of this type to the system.
Cost Per Rider (Transport Costs)	Expression	The cost to load and transport an entity using a transporter of this type, irrespective of the transportation time. This cost is allocated to an entity when it is loaded onto the transporter.
Transport Cost Rate (Transport Costs)	Expression	The cost per unit time to transport an entity using a transporter of this type.
Idle Cost Rate(Resource Costs)	Expression	The cost per unit time charged, or accrued, to the cost of a vehicle of this type when it is idle.
Cost Per Use(Resource Costs)	Expression	The one-time cost that is accrued each time a vehicle of this type is used for a non-transport task, regardless of the usage duration. This cost will be charged to the cost of the owner object(i.e., task) using the vehicle.
Usage Cost Rate(Resource Costs)	Expression	The cost per unit time to use a vehicle of this type for a non-transport task. This cost will be charged to the cost of the owner object (i.e., task) using the vehicle.
Run Initialized Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is initialized. Note: This trigger will only occur for Vehicle objects created by the Initial Number in System setting.
Run Ending Add-On Process Triggers	Process Instance Name	Occurs when the simulation run is ending.
Allocated Add-On Process Triggers	Process Instance Name	Occurs when this Vehicle resource has been allocated to perform some transport tasks, or when another object has seized this Vehicle resource for some non-transport related task.
Released Add-On Process Triggers	Process Instance Name	Occurs when this Vehicle resource has been released after completing a set of transport tasks, or when another object has released this Vehicle resource from some non-transport related task.

Failed Add-On Process Triggers	Process Instance Name	Occurs when the vehicle has failed.
Repaired Add-On Process Triggers	Process Instance Name	Occurs when a failure is repaired at the vehicle.
Entered Node Add-On Process Triggers	Process Instance Name	Occurs when this vehicle has entered a node.
Unloaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is unloaded from this vehicle.
Loaded Add-On Process Triggers	Process Instance Name	Occurs after an entity is loaded onto the vehicle.
Exiting Node Add-On Process Triggers	Process Instance Name	Occurs when this vehicle object is about to begin exiting a node.
Evaluating Transport Request Add-On Process Triggers	Process Instance Name	Occurs when this vehicle is evaluating whether or not to accept the pickup of a rider. In the executed decision process, assigning the value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the potential rider is rejected.
Evaluating Seize Request Add-On Process Triggers	Process Instance Name	Occurs when an 'On Demand' routing type Vehicle is evaluating whether to accept or reject a seize allocation attempt by some process in the system. In the executed decision process, assigning a value less than or equal to 0.0 to the executing token's ReturnValue state (Token.ReturnValue) indicates that the allocation attempt is rejected.
On Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'on shift' because of its specified work schedule.
Off Shift Add-On Process Triggers	Process Instance Name	Occurs when the object goes 'off shift' because of a specified work schedule.
Initial Number in System	Integer	The initial number of Vehicle objects of this type in the system. Vehicles are initially located in free-space at the object instance location.
Maximum Number in System	Integer	The maximum number of objects of this instance that can be simultaneously in the system. If this limit is exceeded then a runtime error is generated.
Log Resource Usage	True or False	Indicates whether usage related events for this resource are to be automatically logged. Go to Results -> Logs to view logged data. Note that using values that resolve to True at runtime but not at design time (such as table references) will

		result in the resource being added to the gantt view once it appears in the Resource Usage Log. Its gantt representation will not display Day or Downtime Exception rows, but that information can still be seen in the Work Day Exceptions and Work Period Exceptions repeat groups of the object instance.
Display Category	String	Optional text used for hierarchically arranging resources in the Resource Plan window. Use backslashes for multiple levels (e.g., One\Two\Three).
Display Color	Color	A color to be used when showing this object in various windows (currently only Gantt windows in Simio RPS Editions). If not specified, then this property defaults to a color derived from the object's unique name.
Can Transfer In & Out of Objects	True or False	Indicates whether objects of this type are by default allowed to transfer into and out of fixed objects via those object's external input / output nodes.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ElevatorNode

The Simio Extras Library contains two objects designed for modeling elevators - the Elevator and the ElevatorNode. These objects may be used in conjunction with the Simio Standard Library.

The ElevatorNode has two functions. It can act either as a pickup/dropoff point for an Elevator, or it can be used as an Elevator selection point to route entities to a particular elevator. This function is determined by its *Elevator Node Type* property.

When an ElevatorNode is used as a pickup/dropoff point, each ElevatorNode references its associated Elevator, and each Elevator has a list of associated ElevatorNodes that it can visit. When an entity arrives to an ElevatorNode, it issues a pickup request to the Elevator. There is no limit to the number of floors (ElevatorNodes) that can be assigned to an Elevator.

An ElevatorNode can also be used to select between Elevators. When modeling a bank of two or more elevators, an ElevatorNode may be used at each elevator level (floor) to route arriving entities to the appropriate elevator. This selection is made based on the elevator that must travel the least distance to accomplish the pickup – given the current workload already assigned to the elevator. In the case of an approaching elevator moving in the desired direction, this is simply the current distance from the elevator. Otherwise, this is the travel distance for the elevator to complete its existing pickups and drop-offs and then return to this location. Note that a separate ElevatorSelectorNode is required at each level of the elevator bank. There is no limit to the number of floors (ElevatorNodes) that can be assigned to an Elevator.

Listed below are the properties of the **ElevatorNode**:

Property	Valid Entry	Description
Initial Traveler Capacity	Integer	The initial maximum number of traveling entities that may simultaneously occupy this node. NOTE: An entity using the 'Ride On Transporter' option for this node will be exempted from being blocked by or counted against the node's traveler capacity limit. Also, refer to the user-assignable 'CurrentTravelerCapacity' state variable of a node to dynamically change its traveler capacity during a simulation run.
Entry Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank entry into this node among competing entities.
Entry Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' entry ranking rule.
Outbound Link Rule	Shortest Path, By Link Weight	The rule used by a traveler to select an outbound link from this node to its next destination. Note that if this rule is set to 'Shortest Path', and the traveler does not have an assigned destination, then the rule will revert to 'By Link Weight' selection.
Elevator Node Type	Pickup/Dropoff Point, Elevator Selection Point	
Elevator Node List Name	Node List	The list name for the list of ElevatorNodes associated with this bank of elevators at this level.
Select Elevator In	True, False	If 'True', the entity selects an elevator located above its current Y coordinate, otherwise, it selects an Elevator located below its current Y coordinate.

Upward Direction		
Associated Elevator	Elevator Instance Name	The elevator that is associated with this ElevatorNode.
Elevator Destination Type	By Sequence, ElevatorNode, EntityDestination	The method used to set a destination node for entities entering the elevator at this node. If the method is 'By Sequence', then the entity must have a sequence table assigned. The destination node value will be set to the next destination in the sequence.
Dropoff Node	Node Instance Name	The elevator departure node for this entity when the <i>Elevator Destination Type</i> is specified as 'ElevatorNode'.
On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the ElevatorNode object.
Assign If (State Assignments)	Custom Condition, Entity Entering, No Condition, Transporter Entering	Condition required to perform the assignment.
Condition (State Assignments)	Expression	Condition required to perform the assignment when <i>Assign If</i> is 'Custom Condition'.
State Variable Name (State Assignments)	State Name	The name of the state that will be assigned a new value.
New Value (State Assignments)	Expression	The new value to assign.
On Entering (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity enters the node object.
On Exited (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity has exited the node object.
Tally If (Tally Statistics)	Custom Condition, Entity Entering, No Condition, Transporter Entering	The condition required to record the tally statistic.
Tally Statistic Name (Tally Statistics)	TallyStatistic Element Name	The tally statistic for which the value is to be recorded.
Value Type (Tally Statistics)	Expression, TimeBetween	The type of value to record. The value type 'Expression' records the value of the specified expression. The value type 'TimeBetween' records the time between arrivals to Tally steps referencing the tally statistic.

Value (Tally Statistics)	Expression	The expression value to be recorded.
Run Initialized Add-On Process	Process Instance Name	Occurs when the simulation run is initialized.
Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs when an entity's leading edge has entered this node.
Exited Add-On Process	Process Instance Name	Occurs when an entity's leading edge has exited this node.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

ElevatorSelectorNode

Note: The ElevatorSelectorNode was combined with the ElevatorNode in Sprint 218.

The Simio Extras Library contains three objects designed for modeling elevators - the Elevator, ElevatorNode, and ElevatorSelectorNode. These objects may be used in conjunction with the Simio Standard Library.

When modeling a bank of two or more elevators, an ElevatorSelectorNode may be used at each elevator level (floor) to route arriving entities to the appropriate elevator. This selection is made based on the elevator that must travel the least distance to accomplish the pickup – given the current workload already assigned to the elevator. In the case of an approaching elevator moving in the desired direction, this is simply the current distance from the elevator. Otherwise, this is the travel distance for the elevator to complete its existing pickups and drop-offs and then return to this location. Note that a separate ElevatorSelectorNode is required at each level of the elevator bank. There is no limit to the number of floors (ElevatorNodes) that can be assigned to an Elevator.

Listed below are the properties of the **ElevatorSelectorNode**:

Property	Valid Entry	Description
Initial Traveler Capacity	Expression	The initial maximum number of traveling entities that may simultaneously occupy this node. NOTE: An entity using the 'Ride On Transporter' option for this node will be exempted from being blocked by or counted against the node's traveler capacity limit. Also, refer to the user-assignable 'CurrentTravelerCapacity' state variable of a node to dynamically change its traveler capacity during a simulation run.
Entry Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The rule used to rank entry into this node among competing entities.
Entry Ranking Expression	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' entry ranking rule.
Outbound Link Rule	Shortest Path, By Link Weight	The rule used by a traveler to select an outbound link from this node to its next destination. Note that if this rule is set to 'Shortest Path', and the traveler does not have an assigned destination, then the rule will revert to 'By Link Weight' selection.
Elevator Node List Name	Node List	The list name for the list of ElevatorNodes associated with this bank of elevators at this level.
Up Direction	True, False	If 'True', the entity selects an elevator in the up direction, otherwise it selects an elevator in the down direction.
Route Request Ranking Rule	First In First Out, Last In First Out, Smallest Value First, Largest Value First	The static rule used to rank competing entities waiting to be assigned an available destination.
Route Request Ranking	Expression	The expression used with a 'Smallest Value First' or 'Largest Value First' ranking rule.

Expression

Route Request Dynamic Selection Rule	None, or one of several Dynamic Selection Rules	The rule used to dynamically select from competing requests waiting to be assigned an available destination.
On Entering (State Assignments)	Repeat Group, Assignments	Optional state assignments when an entity is entering the ElevatorSelectorNode object.
Assign If (State Assignments)	Custom Condition, Entity Entering, No Condition, Transporter Entering	Condition required to perform the assignment.
Condition (State Assignments)	Expression	Condition required to perform the assignment when <i>Assign If</i> is 'Custom Condition'.
State Variable Name (State Assignments)	State Name	The name of the state that will be assigned a new value.
New Value (State Assignments)	Expression	The new value to assign.
On Entering (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity enters the node object.
On Exited (Tally Statistics)	Repeat Group, Tally Statistics	Optional tally statistics to be calculated when an entity has exited the node object.
Tally If (Tally Statistics)	Custom Condition, Entity Entering, No Condition, Transporter Entering	The condition required to record the tally statistic.
Tally Statistic Name (Tally Statistics)	TallyStatistic Element Name	The tally statistic for which the value is to be recorded.
Value Type (Tally Statistics)	Expression, TimeBetween	The type of value to record. The value type 'Expression' records the value of the specified expression. The value type 'TimeBetween' records the time between arrivals to Tally steps referencing the tally statistic.
Value (Tally Statistics)	Expression	The expression value to be recorded.
Run Initialized	Process Instance Name	Occurs when the simulation run is initialized.

Add-On
Process

Run Ending Add-On Process	Process Instance Name	Occurs when the simulation run is ending.
Entered Add-On Process	Process Instance Name	Occurs when an entity's leading edge has entered this node.
Exited Add- On Process	Process Instance Name	Occurs when an entity's trailing edge has exited this node.
Report Statistics	True or False	A Boolean property may be True or False and used in expressions as a numeric 1.0 (True) or 0.0 (False) value.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Visio Interface

Simio is releasing the first phase of a Simio-Visio interface that permits Visio drawings to be imported into Simio. This new capability can be found in the Shared Items forum of the Simio Insiders Forum at <https://www.simio.com/shared-items/>.

This first phase provides two ways to input Intelligent Objects, Links, and Vertices from Visio into the Simio Facility view:

1. Use Simio-Provided Visio Stencils to add Intelligent Objects and their Links, or
2. Add Shape-Data (Visio properties) to existing Visio drawings to identify the Shapes and Connectors destined for Simio.

Additional features are:

- A Visio license is not required by the Add-In (only a Visio .vsdx file)
- No COM or .NET Interop or 3rd party software is required.

Later phases will permit the import of Visio data for Simio Processes and Tasks.

[Copyright 2006-2025, Simio LLC. All Rights Reserved.](#)

Send comments on this topic to [Support](#)

Appendix

Anti-Virus

Some anti-virus software views Simio software or actions done within Simio as a threat. This can result in odd behavior which might include the model or experiments not running or loading slowly or the application becoming unresponsive. If you are experiencing abnormal issues that could potentially be contributed to your anti-virus, we recommend excluding Simio.exe or the entire Simio Programs file from your anti-virus scans.

Here is sample list of anti-virus providers with links to their instructions on how to add a file to the exceptions list:

- BitDefender - <https://www.bitdefender.com/consumer/support/answer/13427/>
- Norton - <https://support.norton.com/sp/en/us/home/current/solutions/v3672136>
- Webroot - https://docs.webroot.com/us/en/home/wsa_pc_userguide/Content/ManagingQuarantine/BlockingOrAllowingFiles.htm

If you use a different anti-virus provider, please search your corresponding site for similar instructions.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

System Requirements

Desktop Requirements

Hardware and Software Requirements

	Minimum	Recommended
Operating System	Any currently supported Microsoft Windows 10/11 64-bit	Any currently supported Microsoft Windows 10/11 64-bit
Supporting Software	Microsoft Excel	Microsoft Excel, SQL Server Express
Processor (CPU)	Intel Core i5, 4+ cores, 2.0+ GHz	Intel Core i7 or i9, 6+ cores, 3.0+ GHz with Turbo Boost
Memory (RAM)	8 or 16GB	32 or 64GB
Hard Drive Space	250GB SSD	512GB or 1TB NVMe
Graphics (GPU)	128MB Integrated, Graphics compatible with DirectX11	1+ GB Discrete Graphics Card compatible with DirectX11
Display*	Full HD (1920x1080), Built in or 17"+ external display	2K (2560x1440), 23"+ external display

Notes

The Simio UI offers touch screen support which requires the drivers provided and supported by your hardware.

For cloud based virtual machines running Simio desktop, choose a VM with similar specifications outlined above. *Not only can you use a second monitor for displaying Help and documentation, but most Simio windows can be undocked and moved to a second screen. For example, you can display the Trace or Search window on the second screen, or you can simply have the animation span two screens.

Portal Requirements

Hardware and Software Requirements

	On-Prem	Cloud (AWS, Azure)
Operating System	Any currently supported Microsoft Windows Server edition	Any currently supported Microsoft Windows Server edition
Supporting Software	SQL Server Standard 2017+ (SQL 2019 Express less than 10GB databases)	SQL Server Standard 2017+ (SQL 2019 Express less than 10GB databases)
Processor (CPU)	Intel Xeon CPU, 2.5+ GHz with Turbo Boost, 6+ cores	X86 Architecture vCPU, 2.5+ GHz per vCPU, 4+ vCPU*
Memory (RAM)	64GB for planning use cases, 128GB for experiment use cases	8GB per vCPU min, 16GB per vCPU for larger models
Hard Drive Space	Primary - 250GB SSD, Secondary - 512GB or 1TB SSD or NVMe (SQL Database on Secondary)	Primary - 250GB SSD, Secondary - 512GB or 1TB SSD or NVMe (SQL Database on Secondary)

For vCPU needs, portal administrators and users need to align on the performance targets to set vCPU count. For scheduling purposes, there should be 1 vCPU for each planner or scheduler that would be generating plans to ensure portal is always available for plan creation. For experiment-based use cases, the model should be run on desktop to estimate the

total run time on one vCPU. This time (model run time) can then be used along with the total expected scenarios for each experiment and the maximum allowable experiment run time to calculate the number of cores needed.

For a large model with a model run time of 1 hour, 48 scenarios, 10 replications each would take the following times:

vCPU	Scenarios	Replications Per Scenario	Total Replications	Total Processing Time (hours)	Total Run Hours for All Scenarios
2	48	10	480	480	240
4	48	10	480	480	120
6	48	10	480	480	80
8	48	10	480	480	60
10	48	10	480	480	48
12	48	10	480	480	40
16	48	10	480	480	30
20	48	10	480	480	24
24	48	10	480	480	20

Portal Components and Installation

Simio Portal on-premises can run on local hardware, but it is also capable of running in local cloud configurations.

Simio Portal comprises three main components:

1. A web UI (IIS),
2. The Simio Simulation Engine, and
3. An embedded database that is a combination of relational and blob storage.

The standard configuration is that all three components are located on a single machine, which can be either a real or virtual machine (VM) within the client network. When it is a VM, that VM can be in a cloud environment such as Azure, AWS, or Google.

The Simio Portal Pre-Installation Checklist outlines how the components of the Simio Portal should be set up and configured and what IT permissions are required for installation. Please contact your Simio representative to get access to the Simio Portal Pre-Installation Checklist.

Other Platforms

Mac and Windows Touch Screen Support

Note: Many Simio users run Simio on their Macs. Like other popular designed-for-Windows applications, the first step is to provide a Windows instance on your Mac.

We recommend you read our Simio for Mac Reference Manual which can be found at <https://cdn.simio.com/SimioLicenses/MacInstructions.pdf> for more information.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

64 bit Execution

Beginning in Sprint 14.230 (October 2021), Simio provides separate installations for Simio 64-bit and 32-bit. To use both bit types of Simio, each installer will need to be used. Prior to 14.230, the default was 32-bit (x86). Now, the default bit-type is 64-bit, provided in a unique installer (where 32-bit requires a separate installation). A user is limited to using the 32-bit type if their machine is exclusively a 32-bit system. In most modern machines, the system can accommodate both 32- and 64-bit applications.

Can I run Simio in 64 bit mode? Yes, but only if you are using a 64 bit OS.

Do I have to use the 64 bit version? No. If you are running a 64 bit OS you can run either the 32 bit or 64 bit version of Simio. By default you will run the 64 bit version of Simio regardless of your OS if both 64-bit and 32-bit are installed. If your OS is 32 bit, you do not have a choice and can run only 32 bit Simio.

Will it run faster? Sorry, probably not. All of the memory addressing is twice as long (i.e., 64 bits rather than 32 bits) so the CPU is working a bit harder and models will generally execute a bit slower.

Why would I use 64 bit? You may need it to run very large models or lots of concurrent replications of moderate-size models. Depending on exactly how you use it, running in 32 bit mode limits each process to using less than 2G of memory, and sometimes much less:

- Windows itself, plus Simio and every application you are running take part of that 2G.
- Sharing that 2G between 8 concurrent replications leaves less than 250M per replication.

64 bit mode replaces those limits with numbers so high they are unlimited from a practical sense. *You can load and run models that are limited only by the amount of memory you have on your computer.*

Memory is relatively inexpensive, so you can max out your system and Simio will take full advantage of all your memory and processors. *You can run models in Simio that could not be run in most other products.* One simple example is that Simio permits you to load tables that are so large they will not load into Excel.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Installation and Activation

Where to Find New Releases

The latest public software releases can be found at: www.simio.com/download. Your activation will continue to work with new releases as long as your maintenance is current. Newer releases are often available on the [Simio Insiders Forum](#).

Recommended Automatic Installation

Unzip the contents of the Simio .zip file into a temporary directory anywhere on your machine. You can then start installing Simio by opening that directory and starting setup.exe. If the requirements below have not been previously installed on your machine, the setup.exe will install them for you. **Note:** Watch for a confirmation dialog halfway through the installation to provide administrative approval. If you do not accept this within a short period of time, the install will fail.

The *Simio Release Notes*, *Simio Reference Guide* and other useful documentation is available using the links on the [Support](#) ribbon. Or you can find them in the install folder (by default `\Program Files\Simio LLC\Simio` for 32-bit computer and `\Program Files\Simio LLC\Simio` for 64-bit computer).

If you have difficulties during download or installation, you can go to our Simio Users Forum and search on the keyword **install** for many useful articles: <http://www.simio.com/forums/>

Optional Manual Installation Procedure

The above automatic installation procedures work for most people, but if you have system-specific problems you may manually install the required software by following the instructions below. **Universities & Network Activations – see important note below.**

Step 1 - Install .NET 4.7.2

Simio requires the *.NET Framework Version 4.7.2*. If the framework is not already installed on your computer and you want to perform a manual install, you can download it from the *.NET Framework Downloads* page: <http://msdn2.microsoft.com/en-us/netframework/aa569263.aspx>.

It may also be installed by doing a Windows Update, depending upon your particular system configuration.

Step 2 - Install DirectX

In order to use the 3D features, you may also need to install the DirectX End-User Runtimes from March 2009. (<http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=0cf368e5-5ce1-4032-a207-c693d210f616>) First create a new directory on your desktop called *DirectXInstall*. When you have downloaded *directx_mar2009_redist.exe*, double click to open it. When prompted, choose the *DirectXInstall* directory on your desktop as the location to extract the contents of the archive. Now, go into the DirectXInstall directory, and double click on DXSETUP.exe. This should install the proper DirectX components onto your machine.

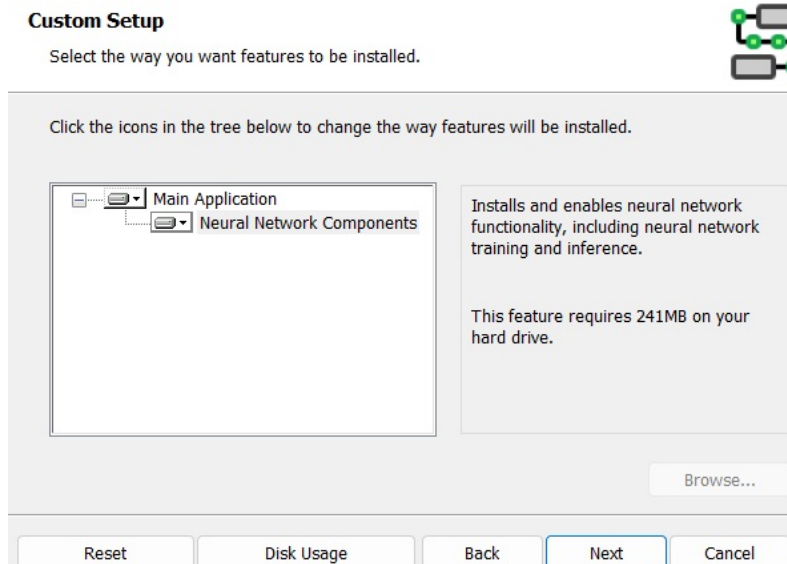
Once DXSETUP.exe is finished, you can delete both the new directory on your desktop and *directx_mar2009_redist.exe*.

Step 3

If you have not already installed Simio, you can now install the Simio software itself by starting setup.exe from the downloaded Simio zip file.

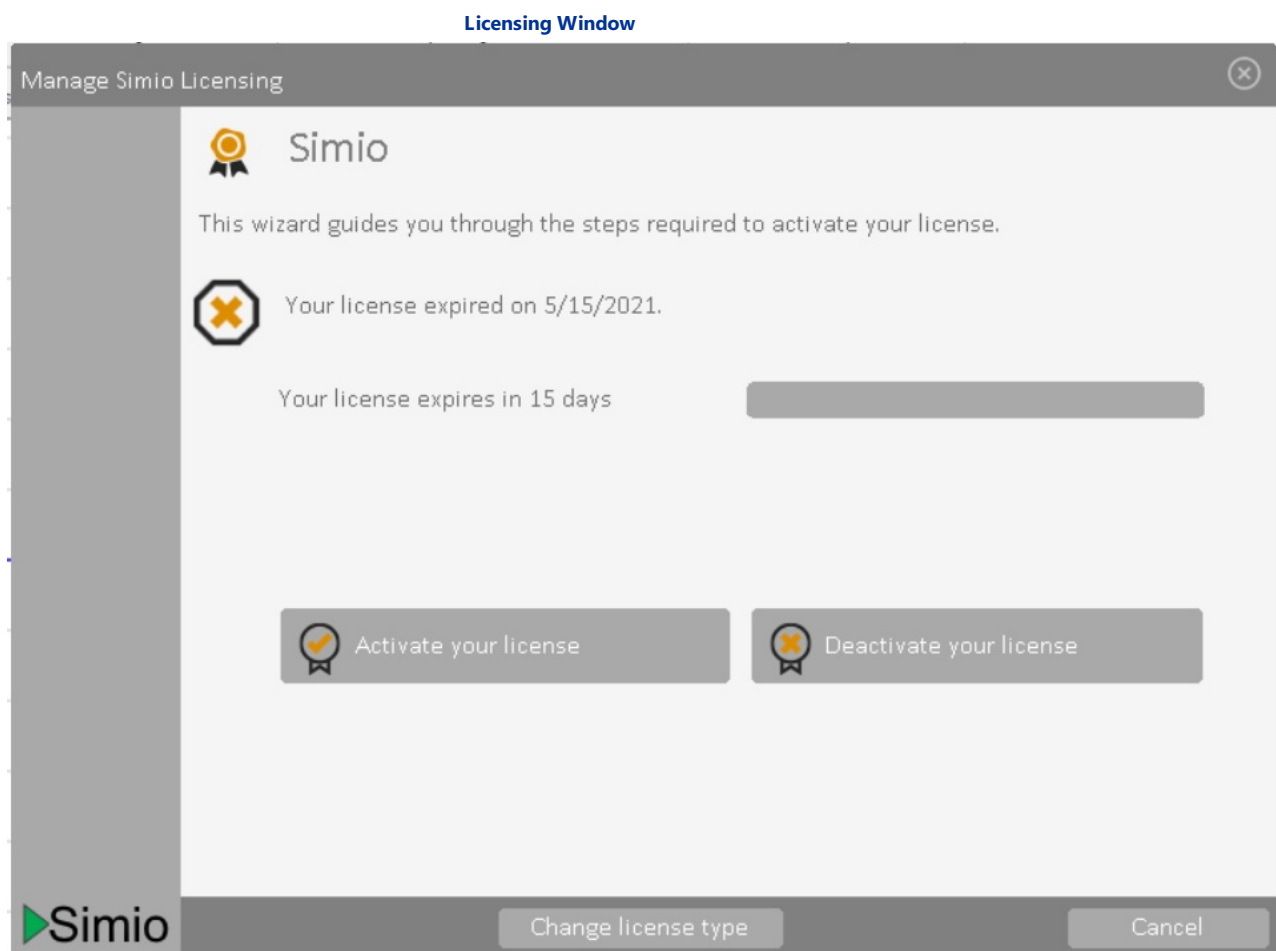
Neural Network Components

Optionally, you can decide to install or not install the neural network components of the Simio software. Choosing to install installs and enables newral network functionality, including neural network training and inference.



Switching Between Network and Local License

If you would like to switch between using a local license and using a license on the network, you may do so by bringing up the Simio Licensing window by clicking on File tab in the top left side of Simio and clicking on the Licensing button. To change the type of license you'd like to use, click on the link at the bottom of this window that reads Change License Type.



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Simio Command Line Options

Simio Command Line Arguments

Loading Simio in Other Languages

The Simio software interface can be displayed in several different languages. This capability is already installed with your software and can be loaded using the **-language:XX** command line switch where XX is your country code. This is easiest to use if you define a shortcut like:

```
"C:\Program Files\Simio LLC\Simio\Simio.exe" -language:de
```

to load German (de). If you provide SYSTEM as the VALUE here, we will use the language to which the OS is set.

Some of these are professionally translated and others are currently a Google "machine" translation. Here is what is available as of this writing and the 2 letter code:

Bulgarian - bg

Chinese - zh

Czech- cs

French - fr

German - de

Japanese - ja

Korean - ko

Portuguese - pt

Russian - ru

Spanish - es

Turkish - tr

You can look in your Simio install folder (typically C:\Program Files\Simio LLC\Simio) for 2 letter subfolders (such as "de") to see what is currently supported in your version.

If you would like to see a Google translation of a particular language or you are interested in helping to manually translate (or clean up an automatic translation) contact Simio support.

Activating Simio Remotely for Multi-Machine Deployment

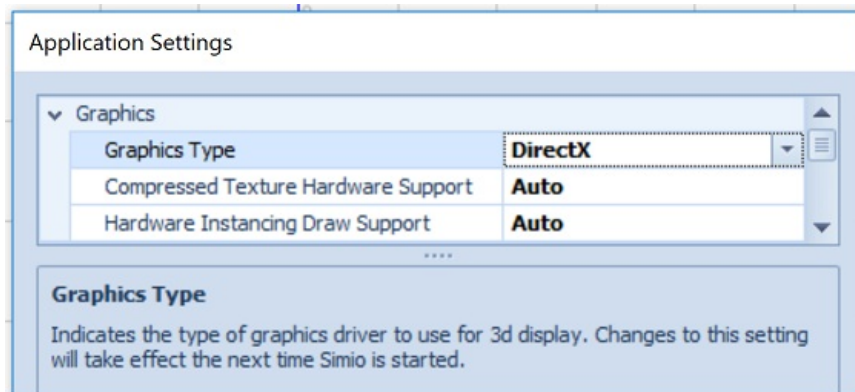
Simio can be activated with a command-line argument which can simplify the installation process in a multiple machine deployment with multi-use activation. After deploying the image, the user can run a post-deployment step that looks something like this:

```
C:\Program Files\Simio LLC\Simio\Simio.exe -authenticate:THIS-IS-MY-AUTH-KEY
```

Simio will set the "errorlevel" value to 0 if the authentication is successful, or 1 if not. Note that in order to authenticate, Simio must be able to connect to the internet to access our authentication web service.

Overriding graphics options

Users can set graphics options in the File > Settings menu:



It is also possible to control that with the following command line arguments:

C:\Program Files\Simio LLC\Simio\Simio.exe **-force-software-mode**

This forces software mode 3d rendering, meant to be used for diagnosing issues, not to run Simio normally

Miscellaneous options - used only at direction of Simio technical support

Many of these are for very specific purposes or compatibility issues, should be **used only at the direction of technical support**, and are not for routine use by modelers.

-default-network-license:VALUE - Sets the default network license to search for on startup to some VALUE, if VALUE isn't provided, it removed the default setting

-no-splash - Disable splash screen

-avoid-loh - Possibly improves performance or memory by making pains to avoid large contiguous allocation in the .NET runtime

-use-old-dijktras - Use the (very) old way of computing shortest paths between network nodes

-use-min-3ddevice-memory - Rendering option that may increase available memory in nonmulti-monitor setups

-dont-use-mmf-for-geometry - Don't use Memory Mapped Files to read 3d geometry from disk, instead use standard file operations

-use-linear-fk-search - Use the old linear search for finding a table row by key value

-use-linear-table-relation-search - Use the old linear scan for computing what rows in a table point to a specific key

-load-ogre-plugins-the-old-way - Use the old way of loading our 3d subsystem

-max-expression-stack-size:VALUE - Sets the maximum size of the "expression stack" which is a stack of expression data we utilize as we are evaluating an expression

-max-world-size:VALUE - Sets the maximum size of the visible world space (in meters). Defaults to 50,000 km

-camera-far-clip-distance:VALUE - Sets the far clip distance of the 3d camera, defaults to 10 km

-ogre-plugin:VALUE - Loads the specified Ogre 3d subsystem implementation

-showdocumentation - Shows the documentation for the project after loading the file specified on the command line

-run-and-exit - Runs the project specified on the command line then immediately exits. Returns 1 if the runs fails, -2 if the file can't open

-update-libraries:VALUE - If VALUE is "true" then updates the libraries as part of the run-andexit

-report-warnings-as-runtime-errors:VALUE - If VALUE is "true" then makes warnings runtime errors for run-and-exit

-newwindow - Does not load any existing window settings (like size or whatnot) for the application window

-count-the-things:FILENAME - Generates a report at FILENAME of the number of Steps, Objects, Logged resources, and Targets in the specified project

-dpiAware - Opens Simio in dpi Aware mode.

Send comments on this topic to [Support](#)

Saving Project File to XML

For multi-file save targets (simproj and the internals of the zip archive of the spfx), Simio will save each Object Definition to its own file and each Table to its own file *IF* the project level property setting of *Save Project As Multiple Files* property is set to 'True' (Default for projects created in Simio sprint 156+). If it's set to 'False' (Setting for projects created prior to Simio sprint 156), Simio will save the object information in a single file.

Navigation: MySimioProject

MySimioProject

Mod

New Model

Mod

New Entity

Rename Project

Project Properties

Properties: MySimioProject (Project)

Show Commonly Used Properties Only

Data

CSV Separator

Disable Grid Data Cache

False

Advanced Options

Save Project As Multiple Files

True

General

Project Name

MySimioProj...

Icon

(none)

File name:

Model

Save as type:

Project Files (*.spfx)















Project Files (*.spfx)

Multi-file Projects (*.simproj)

XML Files (*.xml)

Save Folders

To make use of the multi-file feature, saving the project as a .simproj should be used. When the model is saved with this target, each object in the model will has its information stored in a folder by the same name located in the Model.Files\Models folder. For example, the HierarchyWithTables simbit would look as follows:

- ▼  HierarchyWithTables.Files
 - >  Data
 - ▼  Models
 -  BasicNode
 -  Connector
 -  ModelEntity
 -  Path
 -  Resource
 - >  SearchTable
 -  Server
 -  Sink
 -  Source
 - >  TopLevelModel
 -  TransferNode

Each folder will contain an XML file with information pertaining to that object type. If multiple modelers are working on the same project, but different objects, the updated object folder can be copied over the existing object folder. In the above example, if a modeler wanted to make changes only to the SearchTable object, the modeler can make the edits in a separate copy of the model, save the model as a simproj and then replace the old SearchTable folder with the updated folder.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

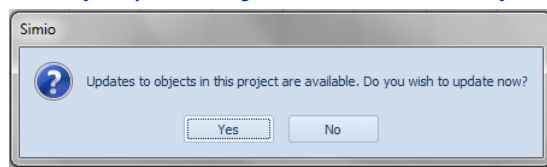
Send comments on this topic to [Support](#)

Check For Updates

Standard Library Updates

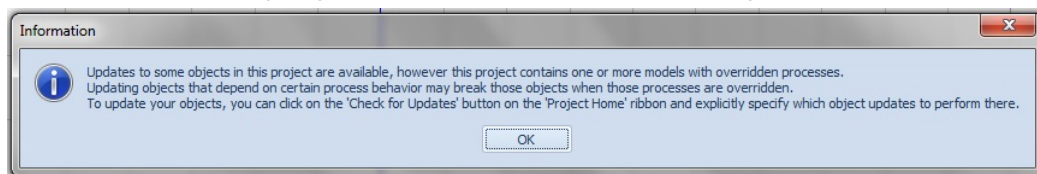
Over time, Simio might make changes to the objects that are part of the Standard Library. When this occurs, we give users the option of updating the object definitions in their existing models or the option to not use the new changes to the object definition. If a project has no objects with **process** overrides, the below dialog is displayed. As a general rule, we suggest that you update all of the objects that have been changed.

Object Updates Dialog - No Process Overrides in Objects



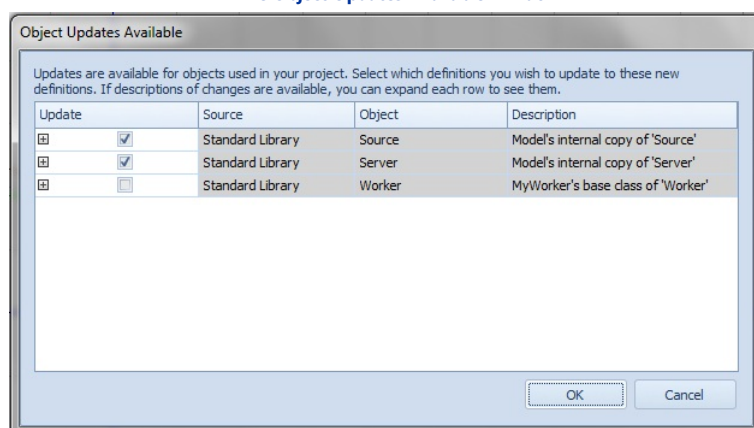
A scenario when you might consider not accepting an update, is if you've created a new object by subclassing one of our standard objects and you do not want your custom logic to be affected. If Simio detects that there are some objects with process overrides, the below dialog will be displayed instead:

The Object Updates Available Window - Process Overrides in Objects

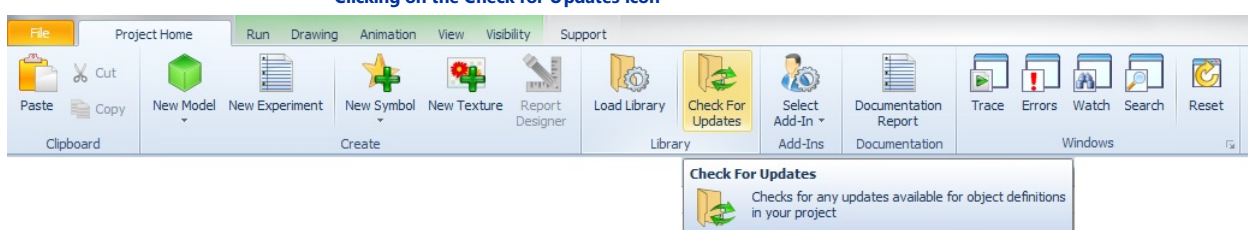


When the above dialog has been shown, the user then has an opportunity to open the Object Updates Available window from within the project. To open the update window after you've looked more closely at your objects, you can bring it up by clicking on the *Check for Updates* icon in the Project Home ribbon. As you can see in the below example, the Standard Library updates are suggested by a check in the update checkbox; however, it is not suggested that users automatically update any subclassed objects that have process overrides, as it may incorporate some object logic updates, but will not update the overridden processes. As shown below, those custom objects with process overrides do not have a check in the update checkbox.

The Object Updates Available Window



Clicking on the Check for Updates icon



Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Copyright Notices and Acknowledgements



Simio - Patent No. 8,156,468 C1: A System and Method for Creating Intelligent Simulation Objects using Graphical Process Descriptions.

Simio - Patent No. 8,682,707 B2: Simulation-Based Risk Analysis for Finite Capacity Scheduling

Certain icons provided by VisualPharm - <http://www.visualpharm.com>

Certain icons provided by Sprial Graphics - <http://www.spiralgraphics.biz>

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Third Party Notices

See the file named 'ThirdPartyNotices_SimioDesktop.txt' in the main Simio folder. This file contains the licenses for all the 3rd party dependencies Simio uses.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

Terms of Use (EULA)

Software End User License Agreement

This End User License Agreement, including the Quotation which by this reference is incorporated herein (this “**Agreement**”), is a binding agreement between Simio, LLC (“**Licensor**”) and the person or entity identified on the Quotation as the licensee of the Software (“**Licensee**”).

LICENSOR PROVIDES THE SOFTWARE SOLELY ON THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT AND ON THE CONDITION THAT LICENSEE ACCEPTS AND COMPLIES WITH THEM. BY CLICKING THE “ACCEPT” BUTTON IN THE SOFTWARE OR USING THE SOFTWARE, YOU (A) ACCEPT THIS AGREEMENT AND AGREE THAT LICENSEE IS LEGALLY BOUND BY ITS TERMS; AND (B) REPRESENT AND WARRANT THAT IF LICENSEE IS A CORPORATION, GOVERNMENTAL ORGANIZATION, OR OTHER LEGAL ENTITY, YOU HAVE THE RIGHT, POWER, AND AUTHORITY TO ENTER INTO THIS AGREEMENT ON BEHALF OF LICENSEE AND BIND LICENSEE TO ITS TERMS. IF LICENSEE DOES NOT AGREE TO THE TERMS OF THIS AGREEMENT, LICENSOR WILL NOT AND DOES NOT LICENSE THE SOFTWARE TO LICENSEE AND YOU MUST NOT USE THE SOFTWARE AND MUST DEINSTALL THE SOFTWARE AND DOCUMENTATION IMMEDIATELY.

NOTWITHSTANDING ANYTHING TO THE CONTRARY IN THIS AGREEMENT OR YOUR OR LICENSEE’S ACCEPTANCE OF THE TERMS AND CONDITIONS OF THIS AGREEMENT, NO LICENSE IS GRANTED (WHETHER EXPRESSLY, BY IMPLICATION, OR OTHERWISE) UNDER THIS AGREEMENT, AND THIS AGREEMENT EXPRESSLY EXCLUDES ANY RIGHT, CONCERNING ANY SOFTWARE THAT LICENSEE DID NOT ACQUIRE LAWFULLY OR THAT IS NOT A LEGITIMATE, AUTHORIZED COPY OF LICENSOR’S SOFTWARE.

THE TERMS OF THIS AGREEMENT WILL SUPERSEDE AND GOVERN THE RELATIONSHIP BETWEEN LICENSEE AND LICENSOR, AND ALL OTHER TERMS PROVIDED BY LICENSEE, WHETHER IN A PURCHASE ORDER, REQUEST FOR PROPOSAL, OR OTHER DOCUMENT ARE HEREBY REJECTED.

1. Definitions. For purposes of this Agreement, the following terms have the following meanings:

“**Authorized Users**” means solely those individuals employed by Licensee and authorized to use the Software pursuant to the license granted under this Agreement, as set forth on the Quotation.

“**Documentation**” means Licensor’s user manuals, handbooks, and installation guides relating to the Software provided by Licensor to Licensee either electronically or in hard copy form.

“**Intellectual Property Rights**” means any and all registered and unregistered rights granted, applied for, or otherwise now or hereafter in existence under or related to any patent, copyright, trademark, trade secret, database protection, or other intellectual property rights laws, and all similar or equivalent rights or forms of protection, in any part of the world.

“**Fees**” means the fees, including all taxes thereon, paid or required to be paid by Licensee for the license granted under this Agreement.

“**Person**” means an individual, corporation, partnership, joint venture, limited liability company, governmental authority, unincorporated organization, trust, association, or other entity.

“**Quotation**” means the sales document provided by Licensor to Licensee which constitutes an offer to sell and license to Licensee, and accepted by Licensee, for Licensee’s purchase of the license for the Software granted under this Agreement.

“**Software**” means the product described in the Quotation in object code format, including any Updates provided to Licensee pursuant to this Agreement.

“**Third Party**” means any Person other than Licensee or Licensor.

“**Updates**” means any updates, bug fixes, patches, or other error corrections to the Software that Licensor generally makes available free of charge to current licensees of the Software.

For purposes of this Agreement, (a) the words “include,” “includes,” and “including” shall be deemed to be followed by the words “without limitation”; (b) the word “or” is not exclusive; and (c) the words “herein,” “hereof,” “hereby,” “hereto,” and “hereunder” refer to this Agreement as a whole. Unless the context otherwise requires, references herein: (x) to Sections, Annexes, Schedules, and Exhibits refer to the Sections of, and Annexes, Schedules, and Exhibits attached to, this Agreement; (y) to an agreement, instrument, or other document means such agreement, instrument, or other document as amended, supplemented, and modified from time to time to the extent permitted by the provisions thereof and (z) to a statute means such statute as amended from time to time and includes any successor legislation thereto and any regulations promulgated thereunder. This Agreement shall be construed without regard to any presumption or rule requiring construction or interpretation against the party drafting an instrument or causing any instrument to be drafted.

The Quotation and all Annexes, Schedules, and Exhibits referred to herein shall be construed with, and as an integral part of, this Agreement to the same extent as if they were set forth verbatim herein.

2. License Grant and Scope. Subject to and conditioned upon Licensee's payment of the Fees and Licensee's compliance with all terms and conditions set forth in this Agreement, Licensors hereby grants Licensee a non-exclusive, non-sublicensable, non-transferable (except in compliance with Section 16(e)), license, during the Term and solely by and through its Authorized Users, to:

(a) Download and install the Software in accordance with the Documentation. In addition to the foregoing, Licensee has the right to make one copy of the Software solely for backup purposes, provided that Licensee shall not allow any Person to, install or use any such copy. All copies of the Software made by the Licensee:

(i) will be the exclusive property of the Licensors;

(ii) will be subject to the terms and conditions of this Agreement; and

(iii) must include all trademark, copyright, patent, and other Intellectual Property Rights notices contained in the original.

(b) Use and run the Software as properly installed in accordance with this Agreement and the Documentation, solely as set forth in the Documentation and solely for Licensee's business purposes.

(c) Download or otherwise make copies of the Documentation and use such Documentation, solely in support of its licensed use of the Software in accordance herewith. All copies of the Documentation made by Licensee:

(i) will be the exclusive property of Licensors;

(ii) will be subject to the terms and conditions of this Agreement; and

(iii) must include all Intellectual Property Rights notices contained in the original.

(d) If the Software was obtained for academic or research use by a student or educational institution, it may be used for teaching and research purposes only. Commercial use of the Software is prohibited if the software was intended for educational use.

3. Third Party Materials. The Software may include software, content, data, or other materials, including related documentation, that are owned by Persons other than Licensors and that are provided to Licensee on licensee terms that are in addition to and/or different from those contained in this Agreement ("Third Party Licenses"). A list of all materials, if any, included in the Software and provided under Third Party Licenses is set forth on Licensors' website and updated from time to time, said updates to be binding upon publication on Licensors' website. Licensee is bound by and shall comply with all Third Party Licenses. Any breach by Licensee or any of its Authorized Users of any Third Party license is also a breach of this Agreement.

4. Use Restrictions. Licensee shall not, and shall require its Authorized Users not to, directly or indirectly:

(a) use (including make any copies of) the Software or Documentation beyond the scope of the license granted under Section 2 and use the number of simultaneous licenses indicated on the Quotation;

(b) provide any other Person, including any subcontractor, independent contractor, affiliate, or service provider of Licensee, with access to or use of the Software or Documentation unless Licensee is responsible for the foregoing parties in accordance with Section 5 and such subcontractor, independent contractor, affiliate, or service provider: (i) uses the Software or Documentation solely for the benefit of Licensee; and (ii) is subject to the terms and conditions herein;

(c) modify, translate, adapt, or otherwise create derivative works or improvements, whether or not patentable, of the Software or Documentation or any part thereof;

(d) combine the Software or any part thereof with, or incorporate the Software or any part thereof in, any other programs;

(e) reverse engineer, disassemble, decompile, decode, or otherwise attempt to derive or gain access to the source code of the Software or any part thereof;

(f) remove, delete, alter, or obscure any trademarks or any copyright, trademark, patent, or other intellectual property or proprietary rights notices provided on or with the Software or Documentation, including any copy thereof;

(g) except as expressly set forth in Section 2(a) and Section 2(c), copy the Software or Documentation, in whole or in part;

(h) rent, lease, lend, sell, sublicense, assign, distribute, publish, transfer, or otherwise make available the Software, or any features or functionality of the Software, to any Third Party for any reason, whether or not over a network or on a hosted basis, including in connection with the internet or any web hosting, wide area network (WAN), virtual private network (VPN), virtualization, time-sharing, service bureau, software as a service, cloud, or other technology or service;

(i) use the Software or Documentation in violation of any law, regulation, or rule; or

(j) use the Software or Documentation for purposes of competitive analysis of the Software, the development of a competing software product or service, or any other purpose that is to the Licensors' commercial disadvantage.

5. Responsibility for Use of Software. Licensee is responsible and liable for all uses of the Software and Documentation through access thereto provided by Licensee, directly or indirectly. Specifically, and without limiting the generality of the foregoing, Licensee is responsible and liable for all actions and failures to take required actions with respect to the Software and Documentation by its Authorized Users or by any other Person to whom Licensee or an Authorized User may provide access to or use of the Software and/or Documentation, whether such access or use is permitted by or in violation of this Agreement. Licensee shall make all Authorized Users aware of the terms set forth herein, and shall advise each Authorized User of the binding nature of these terms.

6. Term and Termination.

(a) This Agreement and the license granted hereunder shall remain in effect for the term set forth on the Quotation or until earlier terminated as set forth herein (the "Term").

(b) Either party may terminate this Agreement, effective upon written notice to the defaulting party, if the defaulting party, materially breaches this Agreement and such breach: (i) is incapable of cure; or (ii) being capable of cure, remains uncured 15 business days after Licensor provides written notice thereof.

(c) Licensor may terminate this Agreement, effective immediately, if Licensee files, or has filed against it, a petition for voluntary or involuntary bankruptcy or pursuant to any other insolvency law, makes or seeks to make a general assignment for the benefit of its creditors or applies for, or consents to, the appointment of a trustee, receiver, or custodian for a substantial part of its property.

(d) Upon expiration or earlier termination of this Agreement, the license granted hereunder shall also terminate, and Licensee shall cease using and destroy all copies of the Software and Documentation. No expiration or termination shall affect Licensee's obligation to pay all Licensee Fees and Support Fees that may have become due before such expiration or termination, or entitle Licensee to any refund, in each case except as set forth in 12.1(c)(ii).

7. Maintenance and Support.

(a) Maintenance and support services will include provision of Updates and support for current (non-expired) subscription licenses and perpetual licenses with active (non-expired) maintenance subscriptions (as set forth in the Quotation). Support services provided by Licensor will include reasonable telephone, e-mail or web-based support. Licensor reserves the right to condition the provision of maintenance and support services, including all or any Updates, on Licensee's registration of the copy of Software for which support is requested. Licensor has no obligation to provide maintenance and support services, including Updates: (i) for any but the most current version or release of the Software; (ii) for any copy of Software for which all previously issued Updates have not been installed; (iii) if Licensee is in breach under this Agreement; or (iv) for any Software that has been modified other than by Licensor, or that is being used with any hardware, software, configuration, or operating system not specified in the Documentation.

(b) Licensor may develop and provide Updates at its sole discretion. Licensee further agrees that all Updates will be deemed Software, and related documentation will be deemed Documentation, all subject to all terms and conditions of this Agreement. Licensee acknowledges that Licensor may provide some or all Updates via download from a website designated by Licensor and that Licensee's receipt thereof will require an internet connection, which connection is Licensee's sole responsibility. Licensor has no obligation to provide Updates via any other media. Maintenance and support services do not include any new version or new release of the Software that Licensor may issue as a separate or new product, and Licensor may determine whether any issuance qualifies as a new version, new release, or Update in its sole discretion.

8. Collection and Use of Information.

(a) Licensee acknowledges that Licensor may, directly or indirectly through the services of Third Parties, collect and store information regarding use of the Software and about equipment on which the Software is installed or through which it otherwise is accessed and used, through:

- (i) the provision of maintenance and support services; and
- (ii) security measures included in the Software as described in Section 10.

(b) Licensee agrees that the Licensor may use such information for any purpose related to any use of the Software by Licensee or on Licensee's equipment, including but not limited to:

- (i) improving the performance of the Software or developing Updates; and
- (ii) verifying Licensee's compliance with the terms of this Agreement and enforcing the Licensor's rights, including all Intellectual Property Rights in and to the Software.

9. Intellectual Property Rights. Licensee acknowledges and agrees that the Software and Documentation are provided under license, and not sold, to Licensee. Licensee does not acquire any ownership interest in the Software or Documentation under this Agreement, or any other rights thereto, other than to use the same in accordance with the license granted and subject to all terms, conditions, and restrictions under this Agreement. Licensor and its licensors and service providers reserve and shall retain their entire right, title, and interest in and to the Software and all Intellectual Property Rights arising out of or relating to the Software, except as expressly granted to the Licensee in this Agreement.

Licensee shall safeguard all Software (including all copies thereof) from infringement, disclosure to third parties, misappropriation, theft, misuse, or unauthorized access. Licensee shall promptly notify Licensor if Licensee becomes aware of any infringement of the Licensor's Intellectual Property Rights in the Software and fully cooperate with Licensor, at Licensor's sole expense, in any legal action taken by Licensor to enforce its Intellectual Property Rights.

10. Compliance Measures.

(a) The Software may contain technological copy protection or other security features designed to prevent unauthorized use of the Software, including features to protect against any use of the Software that is prohibited under Section 4. Licensee shall not attempt to, remove, disable, circumvent, or otherwise create or implement any workaround to, any such copy protection or security features.

(b) During the Term, Licensor may, in Licensor's sole discretion, audit Licensee's use of the Software to ensure Licensee's compliance with this Agreement, provided that (i) any such audit shall be conducted not less than 15 business days prior notice to Licensee, and (ii) no more than one audit may be conducted in any 12 month period except for good cause shown. Licensor also may, in its sole discretion, audit Licensee's systems within 6 months after the end of the Term to ensure Licensee has ceased use of the Software and removed all copies of the Software from such systems as required hereunder. The Licensee shall fully cooperate with Licensor's personnel conducting such audits and provide all access requested by the Licensor to records, systems, equipment, information, and personnel, including machine IDs, serial numbers, and related information. Licensor shall only examine information directly related to the Licensee's use of the Software. Licensor may conduct audits only during Licensee's normal business hours and in a manner that does not unreasonably interfere with the Licensee's business operations.

(c) If any of the measures taken or implemented under this Section 10 determines that the Licensee's use of the Software exceeds or exceeded the use permitted by this Agreement then:

(i) Licensee shall, within 15 business days following the date of such determination by Licensee or Licensor's written notification thereof, pay to Licensor the retroactive Fees for such excess use and, unless Licensor terminates this Agreement pursuant to Section 10.1(c)(ii), obtain and pay for a valid license to bring Licensee's use into compliance with this Agreement. In determining the Licensee Fee payable pursuant to the foregoing, (x) unless Licensee can demonstrate otherwise by documentary evidence, all excess use of the Software shall be deemed to have commenced on the commencement date of this Agreement or, if later, the completion date of any audit previously conducted by Licensor hereunder, and continued uninterrupted thereafter, and (y) the rates for such licenses shall be determined without regard to any discount to which Licensee may have been entitled had such use been properly licensed prior to its commencement (or deemed commencement).

(ii) If the use exceeds or exceeded the use permitted by this Agreement, Licensor shall also have the right to terminate this Agreement and the license granted hereunder, effective immediately upon written notice to Licensee.

Licensor's remedies set forth in this Section 10(c) are cumulative and are in addition to, and not in lieu of, all other remedies the Licensor may have at law or in equity, whether under this Agreement or otherwise.

11. Payment. The payment of all Fees set forth in the Quotation are non-refundable, except as may be expressly set forth herein. Any renewal of the license or maintenance and support services hereunder shall not be effective until the fees for such renewal have been paid in full. Any payment overdue will be subject to interest charges which shall accrue at a rate of 1.5% per month (18% annually), or the maximum rate permitted by law, whichever is less, from the due date until the total invoice amount has been paid in full. Further, if Licensee fails to timely pay the fee, Licensor may terminate ongoing access to the products and/or services listed in the accompanying Quotation. Licensee shall be liable to Licensor for any costs incurred in Licensor's pursuit of payment for unpaid fees, including reasonable attorney fees and court costs (which shall include the costs of appeal or of executing under any judgment).

12. Limited Warranties, Exclusive Remedy, and Disclaimer.

(a) Solely with respect to Software for which Licensor receives a License Fee, Licensor warrants that, for a period of 90 days following the "start date" date set forth on the Quotation, the Software will substantially contain the functionality described in the Documentation, and when properly installed on a computer meeting the specifications set forth in, and operated in accordance with, the Documentation, will substantially perform in accordance therewith. The foregoing warranty does not apply, and Licensor strictly disclaims all warranties, with respect to any Third Party materials.

THE FOREGOING WARRANTY DOES NOT APPLY, AND LICENSOR STRICTLY DISCLAIMS ALL WARRANTIES, WITH RESPECT TO ANY THIRD PARTY MATERIALS.

(b) The warranty set forth in Section 12(a) will not apply and will become null and void if Licensee materially breaches any provision of this Agreement, or if Licensee, any Authorized User, or any other Person provided access to the Software by Licensee or any Authorized User, whether or not in violation of this Agreement:

(i) installs or uses the Software on or in connection with any hardware or software not specified in the Documentation;

(ii) modifies or damages the Software, or the media on which it is provided, including abnormal physical or electrical stress; or

(iii) misuses the Software, including any use of the Software other than as specified in the Documentation.

(c) If, during the period specified in Section 12(a), any Software covered by the warranty set forth in such Section fails to perform substantially in accordance with the Documentation, and such failure is not excluded from warranty pursuant to Section 12(b), Licensor will, subject to Licensee's promptly notifying Licensor in writing of such failure, at its sole option, either:

(i) repair or replace the Software, provided that Licensee provides Licensor with all information Licensor requests to resolve the reported failure, including sufficient information to enable the Licensor to recreate such failure; or

(ii) refund the Fees paid for such Software, subject to Licensee's ceasing all use of and, if requested by Licensor, returning to Licensor all copies of the Software.

If Licensor repairs or replaces the Software, the warranty will continue to run from the initial "start date" specified on the Quotation, and not from Licensee's receipt of the repair or replacement.

The remedies set forth in this Section 12(c) are Licensee's sole remedies and Licensor's sole liability under this Agreement.

(d) EXCEPT FOR THE LIMITED WARRANTY SET FORTH IN SECTION 12(a), THE SOFTWARE AND DOCUMENTATION ARE PROVIDED TO LICENSEE "AS IS" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, LICENSOR, ON ITS OWN BEHALF AND ON BEHALF OF ITS AFFILIATES AND ITS AND THEIR RESPECTIVE LICENSORS AND SERVICE PROVIDERS, EXPRESSLY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE, WITH RESPECT TO THE SOFTWARE AND DOCUMENTATION, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, AND WARRANTIES THAT MAY ARISE OUT OF COURSE OF DEALING, COURSE OF PERFORMANCE, USAGE, OR TRADE PRACTICE. WITHOUT LIMITATION TO THE FOREGOING, THE LICENSOR PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE LICENSED SOFTWARE WILL MEET THE LICENSEE'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE, OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS, OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE, OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

(i)

13. Limitation of Liability. TO THE FULLEST EXTENT PERMITTED UNDER APPLICABLE LAW:

(a) IN NO EVENT WILL LICENSOR OR ITS AFFILIATES, OR ANY OF ITS OR THEIR RESPECTIVE LICENSORS OR SERVICE PROVIDERS, BE LIABLE TO LICENSEE OR ANY THIRD PARTY FOR ANY USE, INTERRUPTION, DELAY, OR INABILITY TO USE THE SOFTWARE; LOST REVENUES OR PROFITS; DELAYS, INTERRUPTION, OR LOSS OF SERVICES, BUSINESS, OR GOODWILL; LOSS OR CORRUPTION OF DATA; LOSS RESULTING FROM SYSTEM OR SYSTEM SERVICE FAILURE, MALFUNCTION, OR SHUTDOWN; FAILURE TO ACCURATELY TRANSFER, READ, OR TRANSMIT INFORMATION; FAILURE TO UPDATE OR PROVIDE CORRECT INFORMATION; SYSTEM INCOMPATIBILITY OR PROVISION OF INCORRECT COMPATIBILITY INFORMATION; OR BREACHES IN SYSTEM SECURITY; OR FOR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, EXEMPLARY, SPECIAL, OR PUNITIVE DAMAGES, WHETHER ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT, BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, REGARDLESS OF WHETHER SUCH DAMAGES WERE FORESEEABLE AND WHETHER OR NOT THE LICENSOR WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

(b) IN NO EVENT WILL LICENSOR'S OR ITS AFFILIATES', INCLUDING ANY OF ITS OR THEIR RESPECTIVE LICENSORS' AND SERVICE PROVIDERS', COLLECTIVE AGGREGATE LIABILITY UNDER OR IN CONNECTION WITH THIS AGREEMENT OR ITS SUBJECT MATTER, UNDER ANY LEGAL OR EQUITABLE THEORY, INCLUDING BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, AND OTHERWISE, EXCEED THE TOTAL AMOUNT PAID TO THE LICENSOR PURSUANT TO THIS AGREEMENT FOR UP TO TWELVE (12) MONTHS OF THE LICENSEE FEES PAID TO LICENSOR FOR THE SOFTWARE OR \$5,000, WHICHEVER IS GREATER.

(c) THE LIMITATIONS SET FORTH IN SECTION 13(a) AND SECTION 13(b) SHALL APPLY EVEN IF THE LICENSEE'S REMEDIES UNDER THIS AGREEMENT FAIL OF THEIR ESSENTIAL PURPOSE.

(d) As an accommodation to Licensee, Licensor may provide Licensee with a pre-production release of the Software (often labeled a "**beta release**"). IN ADDITION TO ALL OTHER WAIVERS, LIMITATIONS, AND DISCLAIMERS IN THIS AGREEMENT, SUCH RELEASES ARE PROVIDED ON AN "AS IS" BASIS. LICENSEE DOES NOT WARRANT PRE-PRODUCTION RELEASES IN ANY MANNER WHATSOEVER. Licensee acknowledges and agree that Licensor shall have no obligation to maintain, correct, update, change, modify, or otherwise support the beta release. Licensor makes no guarantee or commitment as to the success of the beta software. Use of the beta release is at the entire risk of Licensee.

(e) The Software may contain samples that are provided as an accommodation to Licensee ("**Sample Project Files**"). These Sample Project files are intended to be used for example purposes only. The Sample Project Files may be contained in the Software, Documentation, or downloaded from the Licensor website. Licensor and its Third Party licensors make no representations or warranties regarding Licensee's use of the Sample Project files and related Documentation. In addition to all other waivers, limitations, and disclaimers set forth in this Agreement, such Sample Project Files are provided "as-is," and Licensor disclaims all warranties with regard to this information, including all implied warranties and conditions of merchantability, fitness for a particular purpose, title and non-infringement.

14. Export Regulation. The Software is subject to United States export control laws, including the Export Control Reform Act and its associated regulations. Licensee shall not, directly or indirectly, export, re-export, or release the Software to, or make the Software accessible from, any jurisdiction or country to which export, re-export, or release is prohibited by law, rule, or regulation. Licensee shall comply with all applicable United States or international laws, regulations, and rules, and complete all required undertakings (including obtaining any necessary export license or other governmental approval), prior to exporting, re-exporting, releasing, or otherwise making the Software available outside the United States.

15. United States Government Rights. Each of the Documentation and the Software is a "commercial product" as that term is defined at 48 C.F.R. § 2.101, consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in 48 C.F.R. § 12.212. Accordingly, if Licensee is an agency of the United States Government or any contractor therefor, Licensee only receives those rights with respect to the Software and Documentation as are granted to all other end users under license, in accordance with (a) 48 C.F.R. § 227.7201 through 48 C.F.R. § 227.7204, with respect to the United States Department of Defense and their contractors, or (b) 48 C.F.R. § 12.212, with respect to all other United States Government licensees and their contractors.

16. Miscellaneous.

(a) All matters arising out of or relating to this Agreement shall be governed by and construed in accordance with the internal laws of the state of Delaware without giving effect to any choice or conflict of law provision or rule. Any legal suit, action, or proceeding arising out of or relating to this Agreement or the transactions contemplated hereby shall be instituted in the federal courts of the United States or the courts of the state of Delaware, and each party irrevocably submits to the exclusive jurisdiction of such courts in any such legal suit, action, or proceeding. Service of process, summons, notice, or other document by mail to such party's address set forth herein shall be effective service of process for any suit, action, or other proceeding brought in any such court.

(b) In no event shall either party be liable to the other party, or be deemed to have breached this Agreement, for any failure or delay in performing its obligations under this Agreement, (except for any obligations to make payments), if and to the extent such failure or delay is caused by any circumstances beyond such party's reasonable control, including but not limited to: (i) acts of God; (ii) flood, fire, earthquake, or explosion; (iii) war, invasion, hostilities (whether war is declared or not), terrorist threats or acts, riot or other civil unrest; (iv) government order, law, or actions; (v) embargoes or blockades in effect on or after the date of this Agreement; (vi) national or regional emergency; (vii) strikes, labor stoppages or slowdowns, or other industrial disturbances; and (viii) shortage of adequate power or transportation facilities.

(c) All notices, requests, consents, claims, demands, waivers, and other communications hereunder shall be in writing and shall be deemed to have been given: (i) when delivered by hand (with written confirmation of receipt); (ii) when received by the addressee if sent by a nationally recognized overnight courier (receipt requested); (iii) on the date sent by facsimile or email (with confirmation of transmission) if sent during normal business hours of the recipient, and on the next business day if sent after normal business hours of the recipient; or (iv) on the seventh day after the date mailed, by certified or registered mail, return receipt requested, postage prepaid. Such communications must be sent to the respective parties at the addresses set forth on the Quotation (or to such other address as may be designated by a party from time to time in accordance with this Section 16(c)).

(d) This Agreement, together with the Quotation, all annexes, schedules, and exhibits attached hereto, and all other documents that are incorporated by reference herein, constitutes the sole and entire agreement between Licensee and Licensors with respect to the subject matter contained herein, and supersedes all prior and contemporaneous understandings, agreements, representations, and warranties, both written and oral, with respect to such subject matter. **The terms of this License shall supersede any terms set forth in Licensee's purchase order, request for proposal, or any other sales document provided by Licensee to Licensors.**

(e) Licensee shall not assign or otherwise transfer any of its rights, or delegate or otherwise transfer any of its obligations or performance, under this Agreement, in each case whether voluntarily, involuntarily, by operation of law, or otherwise, without Licensors' prior written consent, which consent Licensors may not unreasonably withhold. For purposes of the preceding sentence, and without limiting its generality, any merger, consolidation, or reorganization involving Licensee (regardless of whether Licensee is a surviving or disappearing entity) will be deemed to be a transfer of rights, obligations, or performance under this Agreement for which Licensors' prior written consent is required. No delegation or other transfer will relieve Licensee of any of its obligations or performance under this Agreement. Any purported assignment, delegation, or transfer in violation of this Section 16(e) is void. Licensors may freely assign or otherwise transfer all or any of its rights, or delegate or otherwise transfer all or any of its obligations or performance, under this Agreement without Licensee's consent. This Agreement is binding upon and inures to the benefit of the parties hereto and their respective permitted successors and assigns.

(f) This Agreement is for the sole benefit of the parties hereto and their respective successors and permitted assigns and nothing herein, expressed or implied, is intended to or shall confer on any other Person any legal or equitable right, benefit, or remedy of any nature whatsoever under or by reason of this Agreement.

(g) This Agreement may only be amended, modified, or supplemented by an agreement in writing signed by each party hereto. No waiver by any party of any of the provisions hereof shall be effective unless explicitly set forth in writing and signed by the party so waiving. Except as otherwise set forth in this Agreement, no failure to exercise, or delay in exercising, any right, remedy, power, or privilege arising from this Agreement shall operate or be construed as a waiver thereof; nor shall any single or partial exercise of any right, remedy, power, or privilege hereunder preclude any other or further exercise thereof or the exercise of any other right, remedy, power, or privilege.

(h) If any term or provision of this Agreement is invalid, illegal, or unenforceable in any jurisdiction, such invalidity, illegality, or unenforceability shall not affect any other term or provision of this Agreement or invalidate or render unenforceable such term or provision in any other jurisdiction.

(i) The headings in this Agreement are for reference only and do not affect the interpretation of this Agreement.

17. Confidentiality.

From time to time during the term of this Agreement, Licensor (as the "**Disclosing Party**") may disclose or make available to the Licensee (as the "**Receiving Party**") information about its business affairs, products, services, the Software, confidential intellectual property, trade secrets, Third Party confidential information and other sensitive or proprietary information, whether orally or in written, electronic, or other form or media, and whether or not marked, designated, or otherwise identified as "confidential" (collectively, "**Confidential Information**"). Confidential Information shall not include information that, at the time of disclosure and as established by documentary evidence: (i) is or becomes generally available to and known by the public other than as a result of, directly or indirectly, any breach of this Section 17 by the Receiving Party or any of its Representatives; (ii) is or becomes available to the Receiving Party on a non-confidential basis from a Third Party source, provided that such Third Party is not and was not prohibited from disclosing such Confidential Information; (iii) was known by or in the possession of the Receiving Party or its agents, employees, or professional advisors, before being disclosed by or on behalf of the Disclosing Party; (iv) was or is independently developed by the Receiving Party without reference to or use, in whole or in part, of any of the Disclosing Party's Confidential Information; or (v) is required to be disclosed under applicable federal, state or local law, regulation, or a valid order issued by a court or governmental agency of competent jurisdiction. The Receiving Party shall: (A) protect and safeguard the confidentiality of the Disclosing Party's Confidential Information with at least the same degree of care as the Receiving Party would protect its own Confidential Information, but in no event with less than a commercially reasonable degree of care; (B) not use the Disclosing Party's Confidential Information, or permit it to be accessed or used, for any purpose other than to exercise its rights or perform its obligations under this Agreement; and (C) not disclose any such Confidential Information to any person or entity, except to the Receiving Party's Representatives who need to know the Confidential Information to assist the Receiving Party, or act on its behalf, to exercise its rights or perform its obligations under the Agreement. The Receiving Party shall be responsible for any breach of this Section 17 caused by any of its Representatives. On the expiration or termination of the Agreement, the Receiving Party shall promptly return, and shall require its Representatives to return to the Disclosing Party all copies, whether in written, electronic or other form or media, of the Disclosing Party's Confidential Information, or destroy all such copies and certify in writing to the Disclosing Party that such Confidential Information has been destroyed. Receiving Party shall be liable for any breaches of this Section 17 by its agents, employees, or professional advisors.

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)

RunTime Capability

Description of RunTime Capability

Simio Professional Runtime Edition provides the ability to view and run models without acquiring the full package software. In Simio Professional Runtime Edition, you can see the Model and its Results, as well as Experiments. With a Simio Professional Runtime Edition license, model editing functionality is limited, but this license allows for complete model viewing based on the Simio Edition of software the model was saved in.

Scheduling-specific features from RPS Edition are not available in the Runtime Edition. Scheduling models can only be deployed by using the same software edition used to build the model. However there is a "Scheduling Mode" in both of these software editions that provides a simpler and more protected interface for use by schedulers who will be *generating and deploying the schedules* rather than *building* scheduling models. Simio Portal is often used as a runtime solution for scheduling applications.

For any questions on Runtime, any Simio Editions, or Portal, please contact sales@simio.com

Copyright 2006-2025, Simio LLC. All Rights Reserved.

Send comments on this topic to [Support](#)