SAS® Simulation Studio 14.3 User's Guide

Contents

Credits

Documentation

Writing	Emily Lada, Phillip Meanor
Editing	Anne Baxter, Ed Huddleston
Documentation Support	Tim Arnold
Technical Review	Anup Mokashi, Edward P. Hughes

Software

Simulation Studio	Phillip Meanor, Emily Lada, Thomas Wolf, Anatoli Melechko

Support Groups

Software Testing	Anup Mokashi, Yu-Min Lin
Technical Support	Tonya Chapman

Chapter 1
What's New in SAS/OR 14.3

Contents

Overview

SAS/OR 14.3 includes performance improvements in the LP, MILP, QP, and NLP solvers. If you also license SAS Optimization 8.1 (or a later release), you can invoke it and SAS/OR 14.3 from the same SAS session.

SAS Simulation Studio 14.3, a component of SAS/OR 14.3 for Windows environments, adds Linux support (on an experimental basis) and also adds an alternative user interface design.

Mathematical Optimization Updates

Solver Performance Improvements

Several optimization solvers have been updated in SAS/OR 14.3 and improve their performance. The LP, MILP, QP, and NLP solver algorithms all reduce the time they require to solve benchmark optimization problems. These improvements also include the decomposition (DECOMP) algorithm for LP and MILP. You should expect to be able to solve individual optimization problems of these types more quickly. If you invoke one of these optimization solvers repeatedly (for example, within a loop), the reduction in overall solution time should be even more pronounced.

SAS Optimization 8.1, which was introduced in March 2017 as part of SAS Viya 3.2, adds several distributed optimization capabilities and works with many of the same optimization procedures as SAS/OR 14.3. For more information, see the SAS Optimization 8.1 documentation. If you license this product along with SAS/OR 14.3, you can access the capabilities of both products from the same SAS session.

Discrete-Event Simulation Updates

SAS Simulation Studio 14.3, which provides a graphical environment for building and working with discrete-event simulation models, makes two changes.

In addition to its long-established support on Windows, SAS Simulation Studio 14.3 adds Linux support on an experimental basis. In addition, the original user interface design is now accompanied by an optional alternative design. The original design is still the default, but you can select the new design by using the Configuration dialog box. The original design is named Metal, and the new alternative design is named Nimbus.

Chapter 2
Overview of SAS Simulation Studio

Contents

What Is Simulation?

Simulation is a very broad term that is applied across many fields and industries. In its most general sense, simulation is the process of building or designing a model that mimics the behavior of a particular real-life system. These models can be either physical or logical. Examples of physical models include flight simulators, wind tunnels, and earthquake simulators. This document focuses on logical models, which can usually be represented by computer programs.

For some systems governed by logical and mathematical relationships, you can use traditional mathematical techniques such as queueing theory and differential equations to derive an analytical solution. For these systems, obtaining an exact answer is a benefit. However, you often need to make simplifying assumptions about the system being studied in order to obtain an analytical model; this simplification brings to the forefront the question of model validity. You can build a simple model of a complex system, but that does not necessarily mean that the model is valid.

Many real-world systems are composed of not only extremely complicated and intricate mathematical and logical relationships, but also a significant random component. For these systems, you simply might not be able to derive an analytical model. Instead, you can use a computer to build a model of the system and numerically generate data that you can use to foster a better understanding of the behavior of the real-world system. Part of the art of designing a computer simulation model is deciding which aspects of the real-life system are necessary to include in the model so that the data generated by the model can be used to make effective decisions.

One of the main advantages of computer simulation is the ability to model extremely complex systems that ordinarily would be impossible to model using traditional analytical techniques. On the other hand, the data generated by a computer simulation model is not exact and, to complicate matters even further, the output is random if any of the model's inputs is random. This randomness makes it more difficult to analyze the

output from computer simulations, and often advanced statistical methods are required to formulate valid conclusions about the behavior of the system.

The field of computer-based simulation is itself very broad and includes a number of different classes of modeling techniques. This document focuses primarily on discrete-event modeling methods in which the state of the model is dynamic and the state of the model changes only at countable, distinct points in time. For example, the operation of an emergency room at a hospital over a 24-hour period can be modeled using discrete-event simulation techniques. A state change in this example can be triggered by the arrival of a new patient or the departure of a nurse for a meal break. Each state change occurs at a distinct point in time, and the simulation model operates by scheduling these events and proceeds by advancing the simulation time to the next event or state change.

The popularity of simulation as a tool for design and analysis has grown over recent years, especially with the advancement of computing technology. Part of simulation's popularity is also due to the numerous and diverse areas where it can be applied. Some areas where discrete-event simulation has been successfully used include manufacturing, telecommunications, transportation, military, and health care.

What Is SAS Simulation Studio?

SAS Simulation Studio is a SAS application that uses discrete-event simulation to model and analyze systems. Simulation Studio is based on the Java programming language and provides the following user interfaces:

- a graphical user interface that requires no programming and provides all the tools for building, executing, and analyzing discrete-event simulation models

- a programmatic interface that enables you to run models in batch mode

Although having a comprehensive set of modeling tools is an important quality in a simulation package, having advanced analysis tools is arguably just as important. As mentioned in the previous section, analyzing output from discrete-event simulations often requires advanced statistical methods. Simulation Studio is designed to interact with both SAS software and JMP® software so that you can conduct sophisticated statistical analyses of your results. Data generated by the model can be saved as a SAS data set or JMP table for later analysis, or alternatively you can use a SAS block included in the basic template of modeling blocks to execute SAS or JMP code directly from Simulation Studio.

Simulation Studio includes a state-of-the-art Experiment window that gives you an organized way to investigate the effects of different parameters on your model output in addition to a place to record results. For a discrete-event simulation model in general, you might be interested in conducting the following types of experiments:

- a sensitivity analysis in which you vary a parameter in the model and you examine the effect on some recorded response. For example, you might be interested in the effect on customer waiting times of hiring an additional cashier at a store.

- a comparison of two or more systems. For example, given two different factory floor layout options, you might want to determine which one yields a higher throughput.

- an experimental design for a system that has flexibility in how several different parameters can be set. You might want to use an experimental design (such as a full factorial) to efficiently organize the testing of different parameter combinations and then study the effect on one or more results.

The Simulation Studio Experiment window can be used to conduct all these different types of simulation experiments. It can interface with JMP software to generate experimental designs and then seamlessly pass the simulated results from the design back to the JMP program for analysis. Simulation Studio is also designed to support multiple models and experiments in a single project so that you can define factors and responses once and use them for all models in the project. This is especially useful when you compare two or more systems.

No matter how advanced the available output analysis tools, they are essentially useless if you have not correctly estimated the inputs to the model. Input analysis is another important aspect of building a simulation model. In Simulation Studio input analysis can be facilitated by using JMP distribution estimation capabilities.

Simulation Studio is a flexible discrete-event simulation tool designed to provide the necessary modeling and analysis tools for both novice and advanced simulation users. Furthermore, Simulation Studio attempts to avoid being simply a black box that takes model inputs and mysteriously produces model outputs. Rather, it includes features that enable you to customize your models and tailor Simulation Studio to meet your specific needs.

A Simple M/M/1 Queueing Model

To illustrate some of the basic concepts involved in building models in Simulation Studio, consider a model of a simple banking system with one teller. Assume that customers arrive at the bank at a rate of 10 per hour (so that the interarrival time between customers is a sample from the exponential distribution with a mean of 6 minutes). Customers wait in a single line on a first-come, first-served basis. Also assume that the teller has a service rate of 12 customers per hour (so that the service time for each customer is a sample from the exponential distribution with a mean of 5 minutes). This simple banking system is an example of an M/M/1 queueing system.

For a queueing system such as this one, the following statistics might be of interest:

- average time a customer waits in line

- length of the queue

- number of customers served in one day

Figure 2.1 An M/M/1 Queueing Model
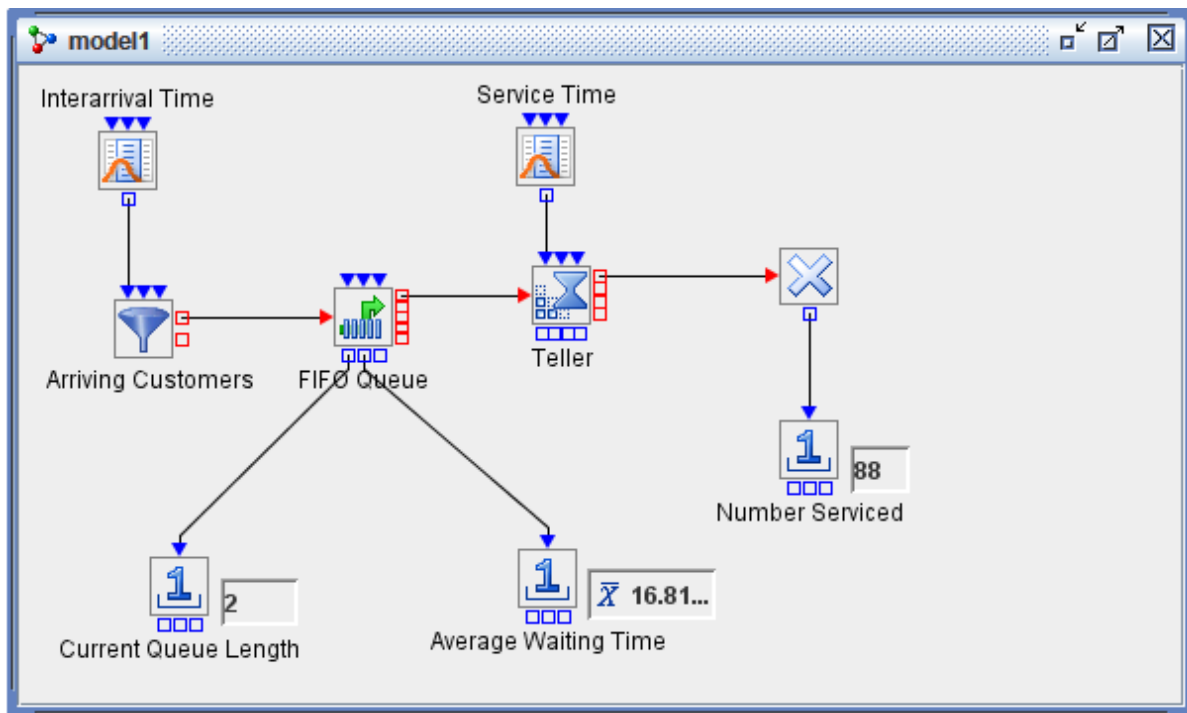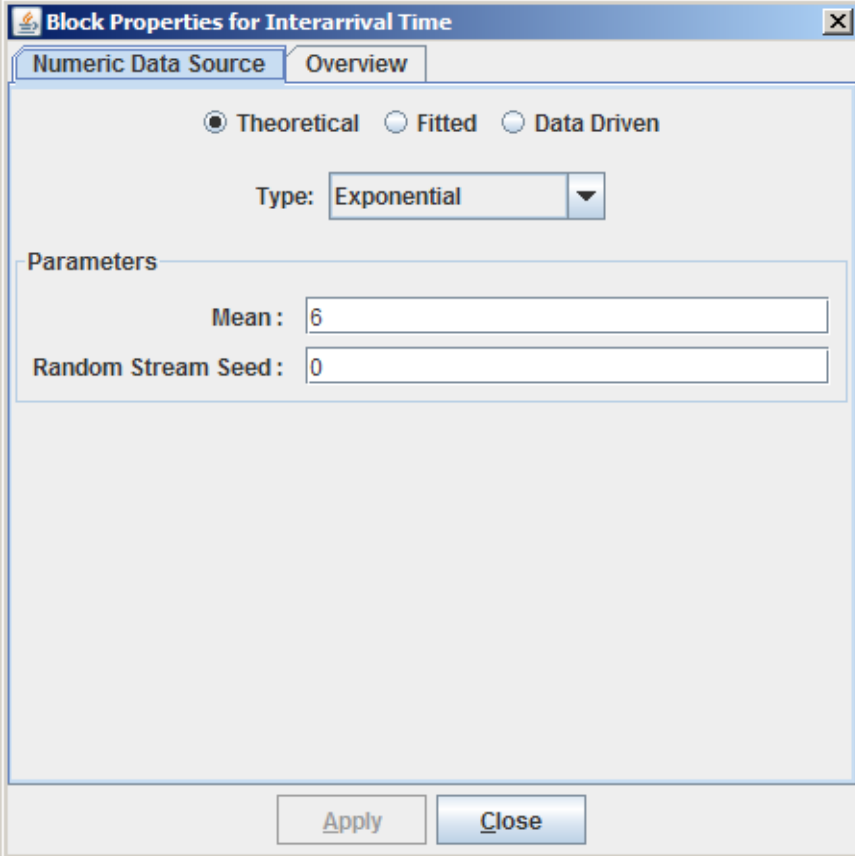
Figure 2.1 shows a Simulation Studio model of the banking system. All the blocks used in this example can be found in the basic template of blocks provided by Simulation Studio. (The labels of blocks in Figure 2.1 have been changed from their default labels to reflect their role in this model. The default labels match the block type.) Customer arrivals to the bank are modeled using an Entity Generator block ▽ labeled Arriving Customers in Figure 2.1. The Entity Generator block has an input value port for the interarrival time. (See "Ports" on page 32 for more information about ports.) The Numeric Source block ▣ labeled InterArrival Time generates a sample from the exponential distribution (representing the next interarrival time) and the Entity Generator block pulls that value through the InterArrivalTime port.

Figure 2.2 shows the dialog box for the Interarrival Time block. Since time in Simulation Studio is dimensionless, you can use hours or minutes or any other time unit in any Simulation Studio model, as long as you use the same units consistently throughout the model.

Figure 2.2 Numeric Source Block Dialog Box

When the entity that represents a customer leaves the Arriving Customers Entity Generator block, it is pushed to the FIFO Queue block ![icon]. The movement of the entity down the link does not advance the simulation clock. If the queue has a limited capacity and is full when the entity arrives, the entity is pushed out the Entity Generator block's OutBalk port. If the queue is not full, the FIFO Queue block attempts to push the entity to the Server block ![icon] labeled Teller. If the Teller is available, it accepts the entity; otherwise, the entity waits in the queue. When the Teller becomes available, it requests an entity from the queue.

When the entity arrives at the Teller block ![icon], a service time is sampled from the second Numeric Source block (labeled Service Time) and pulled by the Teller through the InServiceTime port. After the entity completes its service, it is pushed out to the Disposer block ![icon] where it leaves the system. The Teller then requests another entity from the queue.

Running the Model

Figure 2.3 shows the Experiment window for this model. A single experimental design point, called *point1*, has the number of replications set to 1 and the length of the simulation set to 540 minutes (one banking day).

Figure 2.3 M/M/1 Queueing Model: One Design Point
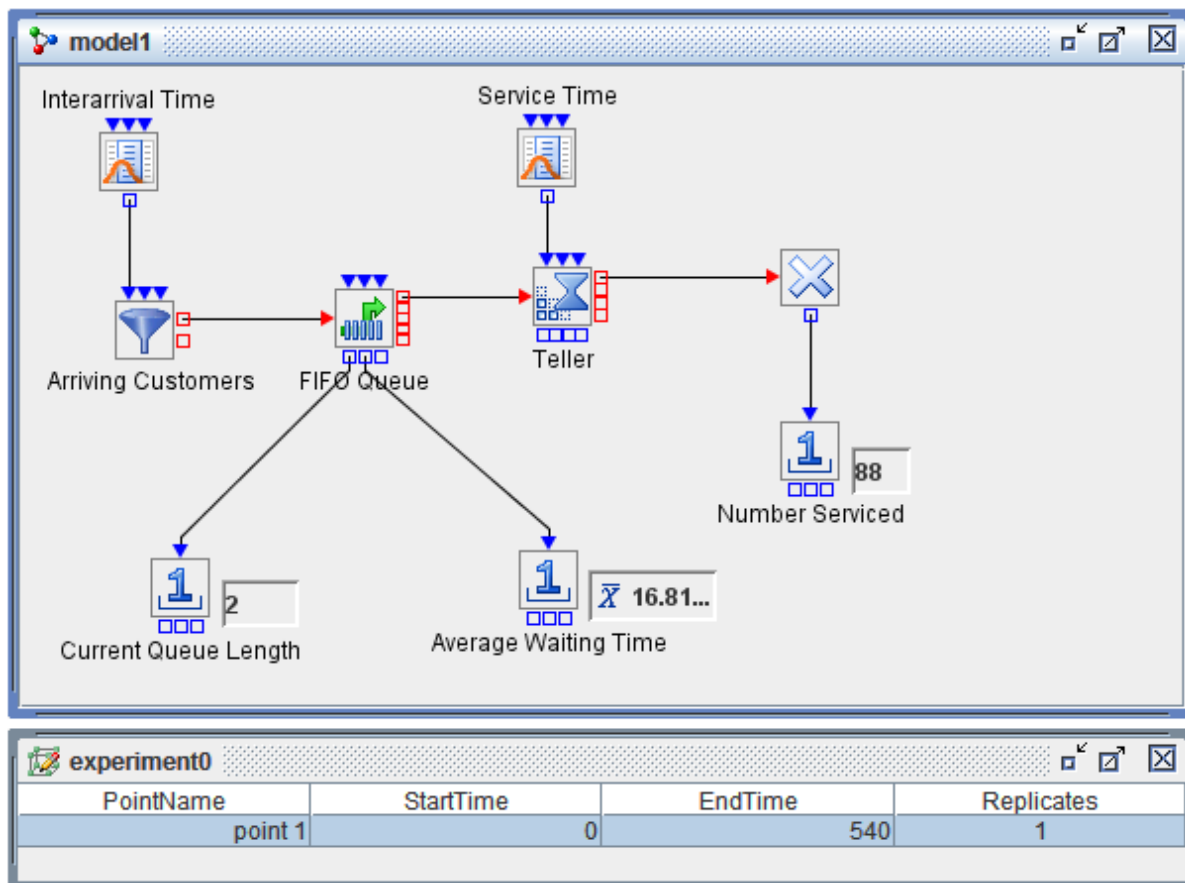
To display the simulation clock, select **Run▶Show▶Simulation Clock** from the Simulation Studio menu. To turn on the animation, click the **Animation** button ▨. To run this model, click the **Start** button ▶. To pause the model, click the **Pause** button ‖. To restart the model, click the **Start** button again. When the model finishes running, only the **Reset** button ▪ is active. You must click the **Reset** button before you make changes to the experiment window or rerun the model.

Collecting Statistics

You can use a Number Holder block [1] to collect and display statistics such as minimum, maximum, sum, and mean as the model is running. In Figure 2.3, a Number Holder block (labeled Average Waiting Time) is connected to the OutWait port on the FIFO Queue block. (Although the ports on a block are not labeled in Figure 2.3, when you rest your mouse pointer on a port, a tooltip displays the port name.)

Double-clicking any block in a model opens the properties dialog box for that block. Figure 2.4 shows the dialog box for this Number Holder block. As each entity leaves the queue, its wait time is pushed into the Number Holder block, whose **Display** field is set to Mean. The Number Holder then recomputes the average waiting time and displays the new value. In this example, the average waiting time for customers computed over one banking day is 16.81 minutes.

Figure 2.4 Number Holder Block Properties
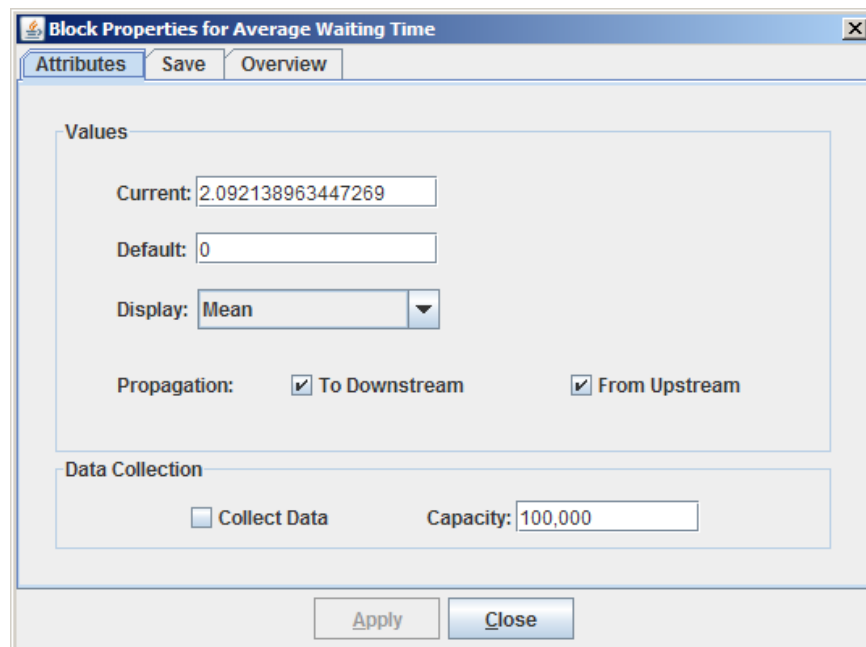
A second Number Holder block (labeled Current Queue Length), with the **Display** field set to Value, is connected to the OutLength port on the FIFO Queue block. Each time an entity enters or leaves the queue, the new queue length is pushed to the Current Queue Length Number Holder block and the updated queue length is displayed. Number Holder blocks can display only averages for observation-based statistics, such as waiting time. For time-dependent statistics such as queue length, Number Holder blocks should be used only to display the minimum, maximum, sum, or current value. In Figure 2.3, the final queue length is 2.

Finally, there is a third Number Holder block (labeled Number Serviced), with the **Display** field set to Value, connected to the OutCount port on the Disposer block ✕. Each time an entity leaves the system, the Number Serviced Number Holder block updates its value and displays the current number of entities serviced. In this example, the number of customers served by the end of one day is 88.

Repair Shop Example

This section discusses a more complicated model to demonstrate some of the additional features and capabilities of Simulation Studio, including compound blocks, branching based on probability, and using the various plotting blocks to monitor the status of the model as it is running.

Suppose parts arrive at a repair shop at a rate of four per hour. Upon arrival, a part is taken to the service desk where it is inspected. The time it takes a person to inspect the part is uniformly distributed between 5 and 15 minutes. The service desk can repair 35% of the parts. The rest require more complicated repairs and must be sent to the repair station. At the repair station, the part is worked on by a repairman. The time it takes a repairman to diagnose and fix the problem is uniformly distributed with a minimum of 10 minutes and a maximum of 45 minutes. With a probability of 0.09, a repairman cannot fix the part, and it is sent to the scrap area. Otherwise, the repaired part is sent on to a quality control manager who inspects the part to determine whether it has been repaired properly. The quality control manager sends 10% of the repaired parts back to the repair center for further repairs. The rest of the parts that pass inspection are sent on to the part pickup area. The time it takes a quality control manager to inspect a part is uniformly distributed between 6 and 18 minutes. Two people work at the service desk, and two people work at the repair desk. Assume the travel time for parts between all stations is 1 minute. The shop is open from 9:00 a.m. to 6:00 p.m., Monday through Friday. The simulation is run for one work week (45 hours).

Compound Blocks

Figure 2.5 shows the completed repair shop model. This model contains several yellow blocks labeled Arrivals, Delay, and Chance; these are compound blocks. If you double-click the yellow compound block labeled Arrivals, you see that it is made up of two blocks: a Numeric Source block and an Entity Generator block. (See Figure 2.6.) Compound blocks are a handy way to organize and streamline your model by collapsing groups of blocks into one block. Compound blocks are also useful in situations where you have the same logic repeated more than once because they can be saved to a template and later reused. For example, double-clicking a Chance compound block reveals that it is made up of three blocks. (See Figure 2.7.) By combining them into one compound block and saving it to a template, you can easily reuse this same logic at other places in your model. See Chapter 7, "Compound and Submodel Blocks," for more information about creating and saving compound blocks.

Figure 2.5 Repair Shop Model
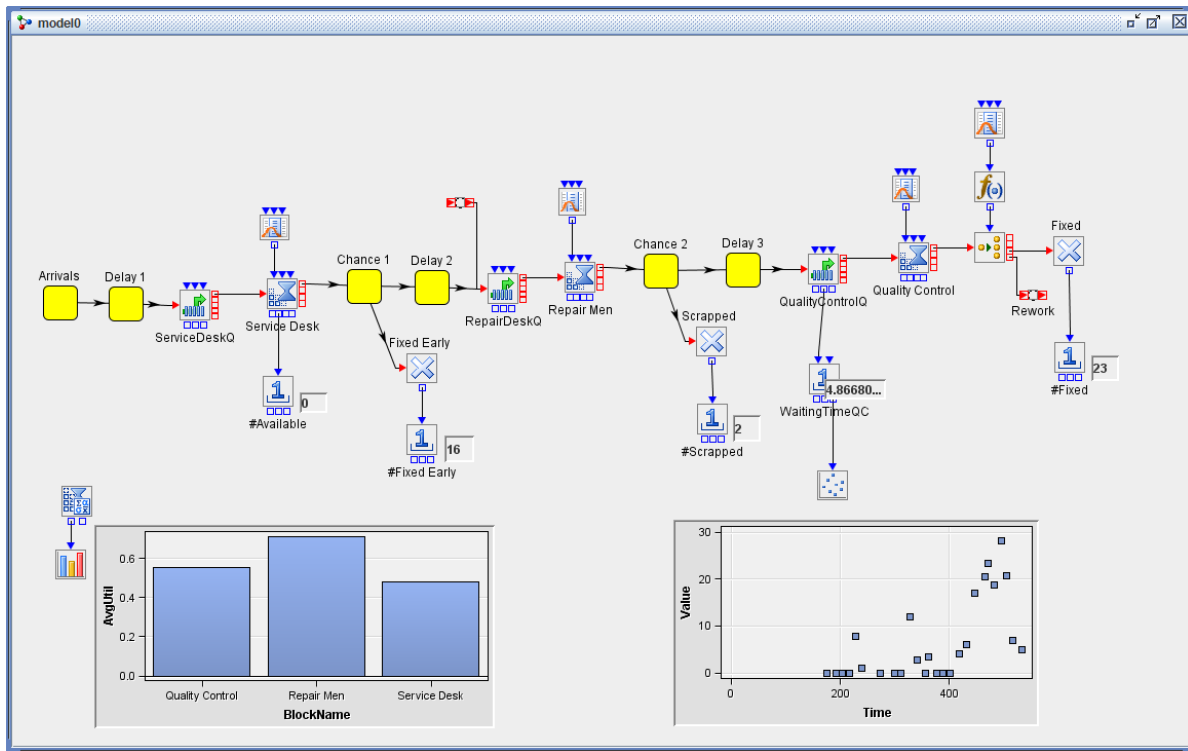
Figure 2.6 Arrivals Compound Block
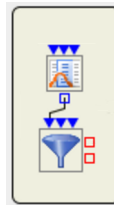
Figure 2.7 Chance Compound Block
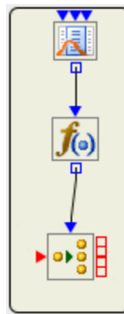
Model Logic

Entities that represent the parts are created in the Arrivals compound block and are pushed to a Delay compound block where they are held for one minute, representing the travel time between stations. Next they are pushed to the Service DeskQ Queue block where they wait for the next available associate at the service desk. After service is completed at the service desk, the entity is pushed to a Chance compound block, which is used to model branching based on probability (in particular, to model that 35% of the parts are repaired at the service desk while the rest are sent on to the repair station).

If you right-click the Switch block inside the first Chance compound block to open the properties dialog box, you see that two cases are defined: one named FurtherRepair and one named Fixed. (See Figure 2.8.) The **Port** option indicates that the switch value comes through the InSwitchValue port. After the two cases have been defined, two additional entity output ports are dynamically created on the Switch block to allow routing of entities based on the switch value. The InSwitchValue port is connected to a Formula block.

Figure 2.8 Switch Block Properties
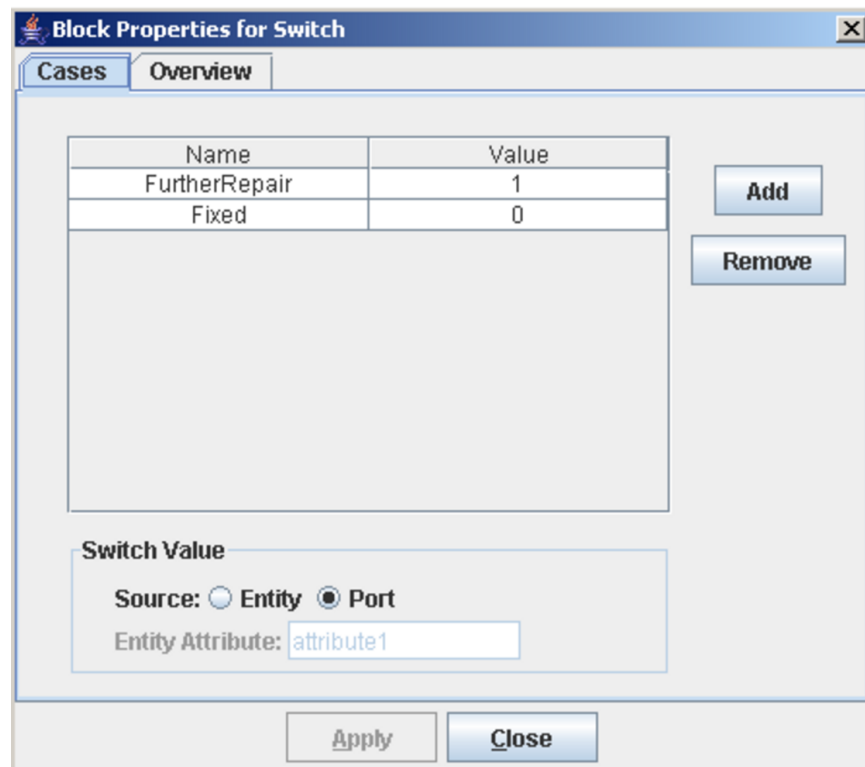
Figure 2.9 shows the properties dialog box for the Formula block. After you add one input variable named *runif* of type *Number*, the Formula block dynamically creates an input port labeled *runif*. Connected to the *runif* input port is a Numeric Source block. This Numeric Source block generates a sample from the uniform distribution with a minimum of 0 and a maximum of 1. After the value for *runif* is pulled by the Formula block, the expression `cond(runif>0.35,1,0)` is evaluated as follows: If *runif* is greater than 0.35, then the value 1 is returned and pushed out of the Formula block and into the Switch block. Otherwise, the value 0 is returned and pushed out. The Switch block then receives either the value of 1 or 0 and uses the value to determine which output port the entity should use to leave the Switch block.

Figure 2.9 Formula Block Properties

If a part is fixed at the service desk, it leaves the system. Otherwise, it is pushed on to the second Delay compound block where it is held for one minute. It then waits in the repair desk queue for the next available repairman. After being serviced by a repairman, the part is pushed into the second Chance block. Here the expression `cond(runif>0.09,1,0)` is evaluated so that with probability 0.09 the part cannot be fixed and is scrapped (that is, the entity leaves the system). Parts that are fixed move on to the third Delay compound block where they wait for one minute and then are pushed into the quality control queue. After being inspected by the quality control manager, the condition `cond(runif>0.10,1,0)` is evaluated so that with probability 0.10 the part does not pass the quality control inspection and is sent back (via the Connector labeled Rework) to the Repair Desk queue. Parts that do pass the quality control inspection leave the system.

Collecting Data

Several blocks in Simulation Studio can be used to collect data. One of these blocks used in the repair shop model is the Server Stats Collector block 📊. This block can be placed anywhere in the model window because entities do not flow through it. Figure 2.10 shows the properties dialog box for the Server Stats Collector block. A list of all blocks that implement the ServerStats interface in the model is shown, and you can select the ones for which you want to collect statistics. The data collected for each replication can be saved to a file as a SAS data set or JMP table or passed to one of the Simulation Studio plotting blocks.

Figure 2.10 Server Stats Block Properties
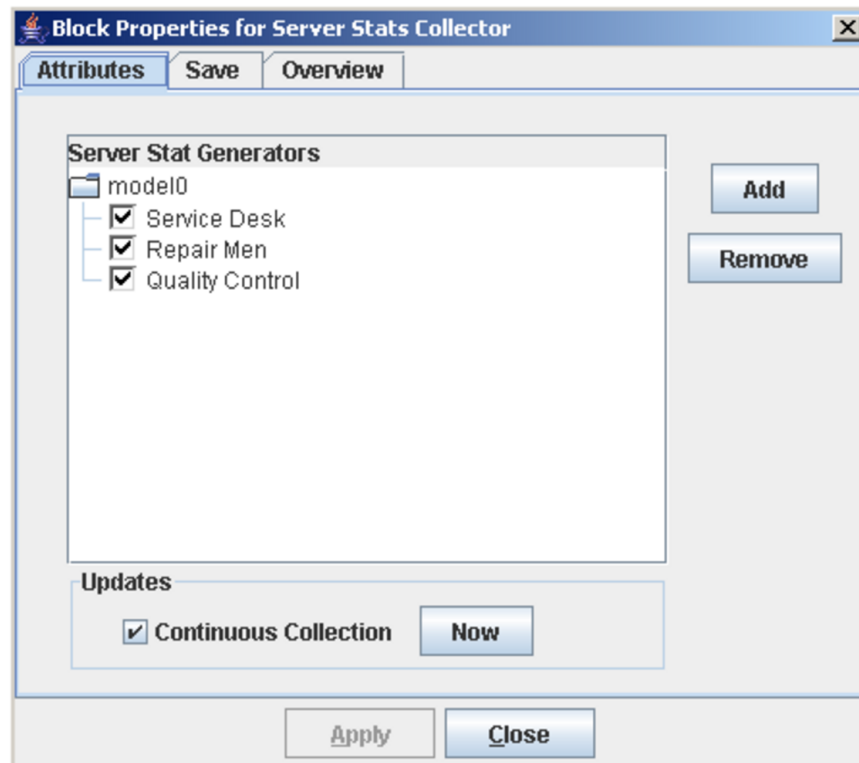
In the repair shop model, a Bar Chart block is connected to the OutData port of the Server Stats Collector block. Figure 2.11 shows the properties dialog box for the Bar Chart block, which requests a bar chart of the average utilization for each of the three servers in the model. After the model is run, the bar chart shows that the average utilization at the quality control station is significantly higher than at the repair or service desks. (See Figure 2.5.)

Figure 2.11 Bar Chart Block Properties
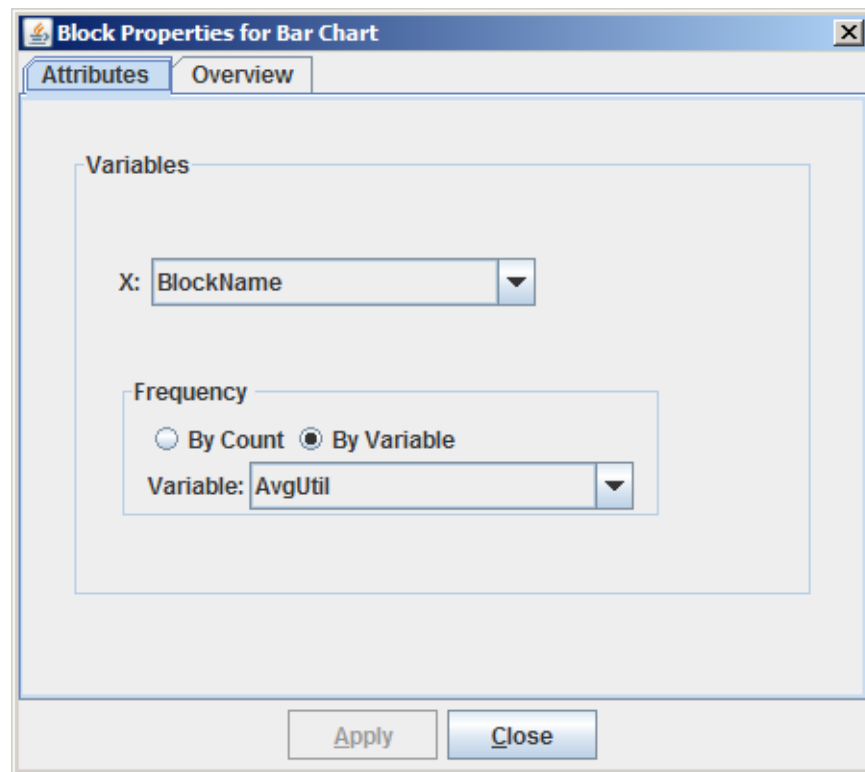
To further investigate the severity of the bottleneck at the quality control station, you can connect a Number Holder block (labeled WaitingTimeQC) to the OutWait port on the quality control queue. Then you can pass the waiting time values to a Scatter Plot block by connecting the OutCollected port of the Number Holder block to the InData port of the Scatter Plot block. For the plots to display correctly, the **Collect Data** check box in the Number Holder Block properties dialog box must be selected. (See Figure 2.4.) As the model runs, you see that the waiting time at the quality control station continues to increase. Appendix E, "Examples of Simulation Studio Models," revisits this repair shop model.

Chapter 3
Introduction to SAS Simulation Studio

Contents

Simulation Studio Graphical User Interface

As mentioned in Chapter 2, "Overview of SAS Simulation Studio," Simulation Studio provides a graphical user interface (GUI) and a batch interface. Initially, most users typically use the Simulation Studio GUI to build and execute simulation models. This chapter provides a high-level overview of Simulation Studio from the GUI perspective and discusses the major components of the application framework. The batch interface is detailed in Chapter 14, "Batch Execution."

Figure 3.1 Simulation Studio Application Framework
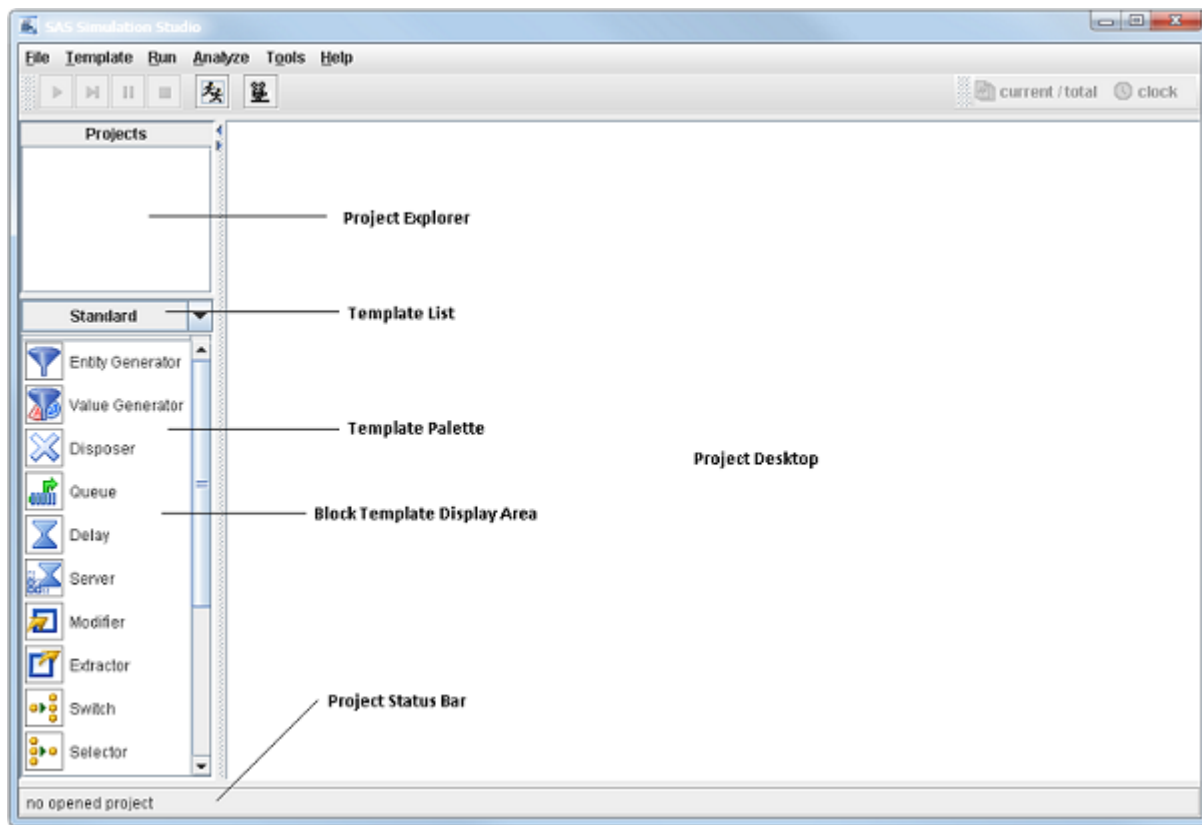
When you start the Simulation Studio application, the graphical user interface opens as shown in Figure 3.1. This window consists of six main areas: menu, toolbar, block template display area, project explorer, project desktop, and project status bar. The following sections provide details about each of these areas, as well as how to launch the application.

Installing and Starting Simulation Studio

Installing Simulation Studio

The installation program for Simulation Studio asks where you want to install the software on your computer. The default location on the Windows operating system is `\Program Files\SASHome\SASSimulationStudio\<release_number>`. If you choose the default location, the installation software loads the software and adds an entry for Simulation Studio to the **Start** menu.

The default installation location on the Linux operating system is `/usr/local/SASHome/SASSimulationStudio/<release_number>`.

Simulation Studio requires you to have a valid version of SAS/OR software installed on your computer. It also needs to know the location of `SASHome` on your system. This information is part of the Simulation Studio configuration data and must be supplied to Simulation Studio for the application to launch.

Starting Simulation Studio

To start Simulation Studio on the Windows operating system, you can either double-click the Simulation Studio desktop icon or select the Simulation Studio entry from the **Start** menu (**Start▶Programs▶SAS▶Simulation Studio 14.3.**)

For the Linux operating system, you launch Simulation Studio from the installation directory in a terminal window by using the command `./simstudio`. The default installation location is `/usr/local/SASHome/SASSimulationStudio/<release_number>`.

For information about running Simulation Studio in batch mode, see Chapter 14, "Batch Execution."

Configuring Simulation Studio

When you attempt to launch Simulation Studio for the first time, it has not yet acquired all the configuration data that it needs. The message "SAS Simulation Studio configuration data not specified" appears, because Simulation Studio does not know where on your machine to look for SAS license information. Then the SAS Simulation Studio Configuration dialog box opens so that you can enter the necessary information. (See Figure 3.2.) In the **SASHome Path** box, enter the directory for `SASHome`. A common default path for this location on the Windows operating system is `\Program Files\SASHome`, and on the Linux operating system it is `/usr/local/SASHome`. To open the SAS Simulation Studio Configuration dialog box after the initial launch, select **Tools▶Configuration** from the main Simulation Studio menu.

Figure 3.2 Configuration Dialog Box
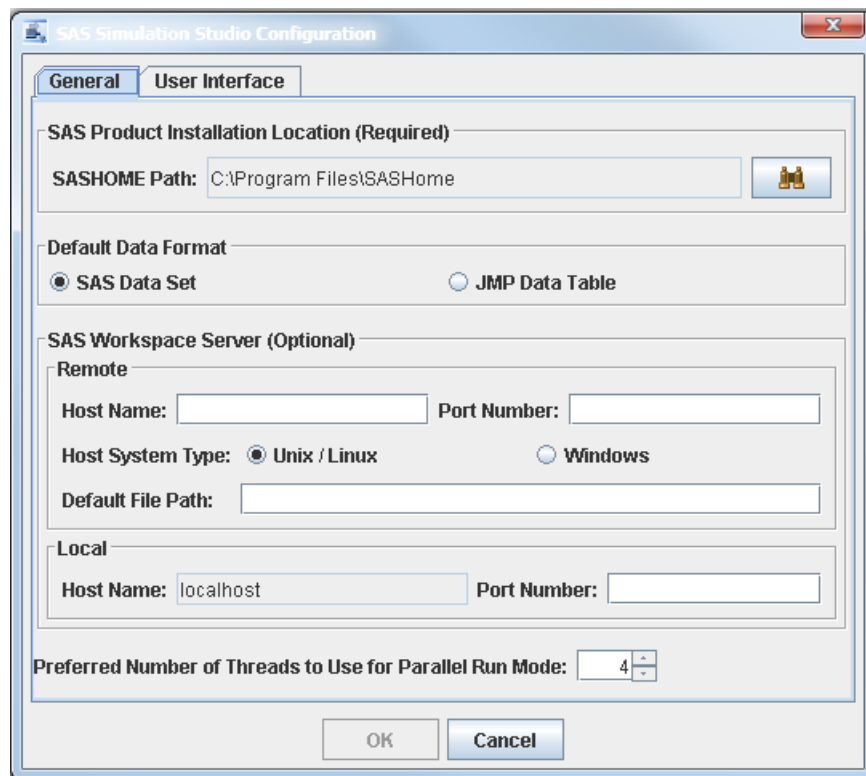
If you have both SAS and JMP software installed, you can select either **SAS Data Set** or **JMP Data Table** as your default data format. (This format information is used for reading and writing data when the filename extension is not provided in an input or output filename.)

SAS Simulation Studio communicates with SAS Workspace Servers to process the input and output requests of SAS data sets in data streaming and collecting blocks, such as the Numeric Source and Bucket blocks. It also supports the submission of SAS programs to the SAS Workspace Servers from the SAS Program block. Currently, you can use up to two SAS Workspace Servers:

- One server can reside on the local machine where Simulation Studio is installed and running.

- Another SAS server can reside on a remote machine and be used as a remote server.

NOTE: In the SAS Simulation Studio Configuration dialog box (shown in Figure 3.2), the only required field is **SASHOME Path**. The remote and local **SAS Workspace Server** fields are optional. For information about configuring and using the local and remote SAS servers, see the sections "Launching Local SAS and JMP Servers" on page 21 and "Using a Remote SAS Server" on page 23.

When you run a Simulation Studio model on a multicore machine by selecting **Parallel Mode** from the **Run** menu, different design points and replications are distributed on different cores simultaneously. The actual distribution of design points and replications depends on the number of cores available and the total number of design points and replications that are selected to run. You can select **Preferred Number of Threads to Use for Parallel Run Mode** to specify the number of threads to use in parallel mode. The maximum value is machine-dependent, the minimum value is 2, and the default is less than the actual number of cores. If the machine is not multicore, then the **Preferred Number of Threads to Use for Parallel Run Mode** option is not available. For more information about using the **Parallel Mode** option, see the section "Running a Model in Parallel Mode" on page 38 in Chapter 4, "Simulation Models."

The **User Interface** tab in the SAS Simulation Studio dialog box presents two choices for the application's look-and-feel: Metal (the default, or original, look-and-feel) and Nimbus. The look-and-feel changes the appearance of scroll bars, buttons, tables, progress indicators, and fonts. To change the look-and-feel, select **Metal (original)** or **Nimbus**, click **OK**, and then restart the application.

Launching Local SAS and JMP Servers

If you want to save data, load data, submit SAS code, or interact with JMP software locally during a Simulation Studio session, you need to have a SAS Workspace Server or JMP server running locally on your computer.

Simulation Studio provides two different methods of connecting to a local SAS Workspace Server. First, on the Windows operating system, you can use the default local workspace server that is launched when Simulation Studio requires it. This default local workspace server does not require configuration or credentials and is also available when you are running in batch mode. No action is required to use the default local SAS Workspace Server in either interactive or batch mode. **NOTE:** The default local workspace server is not available on the Linux operating system.

As an alternative, you can connect to the SAS Simulation Studio Object Spawner for SAS services by using credentialing. This approach works on both the Windows and Linux operating systems. To use this approach, you specify a port number in the Local SAS Workspace Server section of the SAS Simulation Studio Configuration dialog box (shown in Figure 3.2). After you enter a port number for the local SAS Workspace Server, SAS Simulation Studio no longer uses its default local SAS Workspace Server. You are also required to provide local credentials in the User Credentials for SAS Workspace Server dialog box (shown in Figure 3.3), which you can open by selecting **Tools▶Server Credentials** from the main Simulation Studio menu and then clicking the **LocalHost** tab. For the Linux operating system, it is not necessary to specify a value in the **Domain** field of the User Credentials for SAS Workspace Server dialog box.

Figure 3.3 User Credentials for SAS Workspace Server Dialog Box
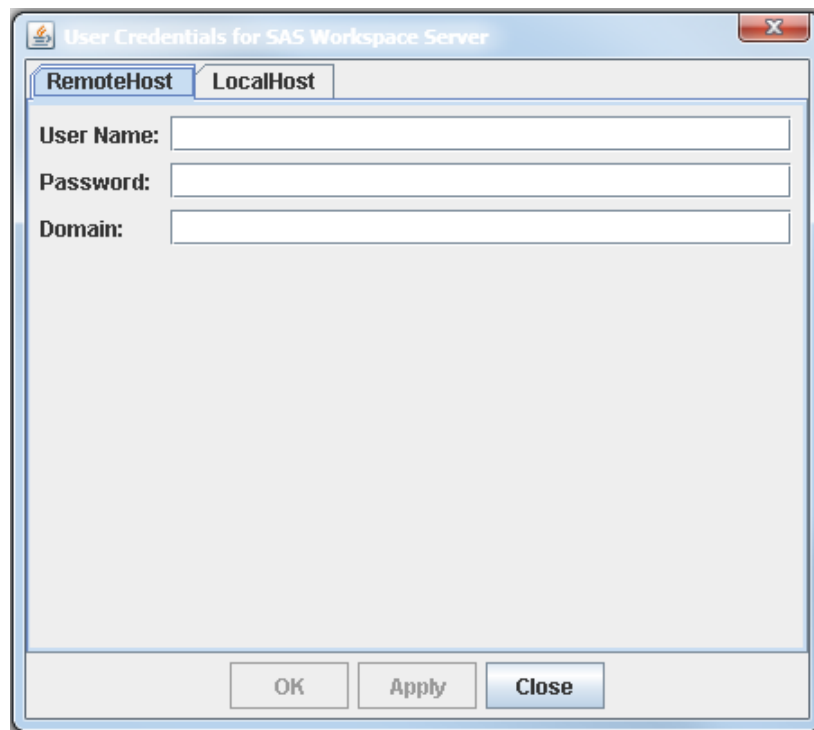
The **Configure SAS Simulation Studio** step of the SAS Simulation Studio installation process provides the default port number 58591 for you to use to connect to the SAS Simulation Studio Object Spawner for SAS services. If you skipped the **Configure SAS Simulation Studio** step during the installation process, then you will not be able to connect to the SAS Simulation Studio Object Spawner for SAS services. If you are using the SAS Simulation Studio Object Spawner for SAS services and want to run Simulation Studio in batch mode, then you must first launch Simulation Studio, provide the local port number and your local credentials, and leave Simulation Studio open.

On the Linux operating system, you must restart the SAS Simulation Studio Object Spawner for SAS services if Linux is shut down and restarted. You use the following command in a Linux terminal window to restart the object spawner: `sudo /sas/simstudioconfig/simstudio.sh start`. The directory `/sas/simstudioconfig` is the default location that is specified in the **Configure SAS Simulation Studio** step during the Simulation Studio installation process.

On the Windows operating system, the local JMP server must be launched manually. The server script is installed in the `launchSASServer` folder in your Simulation Studio installation directory. To launch the Simulation Studio JMP server, you open and run the JMP script file named `JMP_Sim_Server.JSL`. If you double-click the file `JMP_Sim_Server.JSL`, the encrypted script file opens in JMP. You then right-click in the JMP script window and select **Run Script**. You can minimize the JMP window, but it must remain open for Simulation Studio to communicate with JMP. **NOTE:** JMP software is not available on the Linux operating system.

Using a Remote SAS Server

In the Remote SAS Workspace Server section of the SAS Simulation Studio Configuration dialog box (shown in Figure 3.2), you use the **Host Name** field to specify either the host name or the Internet Protocol (IP) address of the remote server. In the **Port Number** field, specify the TCP/IP port number for the remote SAS Workspace Server session on the remote server. Simulation Studio uses this port number to access the services provided by the remote SAS Workspace Server. Select either **Unix** or **Windows** as the **Host System Type** of the remote server. In the **Default File Path** field, specify the default file input and output root directory or folder path for the data input and output requests. Use the appropriate UNIX or Windows path format convention to specify the path, and make it consistent with the specified **Host System Type** of the remote server. When using a remote server, you specify all file paths in the model for loading and saving data as relative to the path specified in the **Default File Path** field. For example, if the default file path for a remote UNIX server is specified as `/username/simstudio/projects` and the file path for an input data set is specified as `project1/data1.sas7bdat,` then at run time Simulation Studio looks for the file `data1.sas7bdat` in the folder `/username/simstudio/projects/project1.`

All data streaming blocks, such as the Numeric Source and Observation Source blocks, have the option **Load from Remote SAS Workspace Server**. When this option is selected, data are read from the specified location on the remote SAS Workspace Server. All data collecting blocks, such as the Number Holder and Bucket blocks, have the option **Submit to Remote SAS Workspace Server**. Selecting this option saves any collected data to a location on a remote SAS Workspace Server. This option is also available in the SAS Program block. When this option is selected, the specified SAS program on the local machine is executed on the remote SAS Workspace Server. You specify the remote SAS Workspace Server information in the SAS Simulation Studio Configuration dialog box, available when you select **Configuration** from the **Tools** menu. (See Figure 3.2.)

The options **Submit to Remote SAS Workspace Server** and **Load from Remote SAS Workspace Server** can be set in the Block Properties dialog box for a specific block. Also, right-clicking on a model name in the Project Explorer display area and clicking **Remote Service Host** opens a dialog box where you can select the blocks in the model for which the **Submit to Remote SAS Workspace Server** and **Load from Remote SAS Workspace Server** options should be selected. If any block in a model has either of these options selected, then you are also required to provide the credentials necessary to access the remote SAS Workspace Server. Select **Tools▶Server Credentials** from the main Simulation Studio menu to open the User Credentials for SAS Workspace Server dialog box (shown in Figure 3.3). For the Linux operating system, it is not necessary to specify a value in the **Domain** field of the User Credentials for SAS Workspace Server dialog box. If you are running in batch mode and using a remote SAS Workspace Server, then you must first launch Simulation Studio, provide your remote host credentials, and leave Simulation Studio open.

Simulation Studio Menu and Toolbar

The main Simulation Studio menu consists of six items: **File**, **Template**, **Run**, **Analyze**, **Tools**, and **Help**. Use the **File** menu (shown in Figure 3.4) to open, create, close, and save projects, models, and experiments in Simulation Studio. When you open or create a new project, Simulation Studio opens a new project window in the Project Desktop area of the GUI and updates the Project Explorer accordingly. If this is a new project, a new (empty) model and experiment are also created. When an existing project is opened, all models and experiments in that project are opened and entries are created for them in the Project Explorer.

Figure 3.4 File Menu
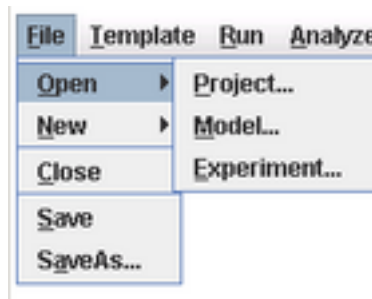
A Simulation Studio template stores information about a collection of Simulation Studio blocks. Template details are provided in Chapter 12, "Block Templates." Use the **Template** menu to open, create, close, and save Simulation Studio templates. Opening a template adds the template name to the Template list box in the Block Template Display area of the application. You can use this list box to determine which template palette is visible in the Block Template Display area of the application. Select **Template►Close** to remove the current template name from the Template list box and also remove the associated blocks from the Template palette. Select **Save** or **SaveAs** to save the current template to disk storage. More details about templates are provided in Chapter 12, "Block Templates."

The **Run** menu (Figure 3.5) controls the model execution and animation. Many of the controls are also found on the toolbar. The functionality that is associated with the model execution controls **Start**, **Augment**, **Pause**, **Reset**, and **Parallel Mode** is discussed in Chapter 4, "Simulation Models." Select **Show** to enable or disable the simulation clock, replication count, or animation for the current model. When visible, the simulation clock and replication count appear in the upper right corner of an individual Project window.

Figure 3.5 Run Menu
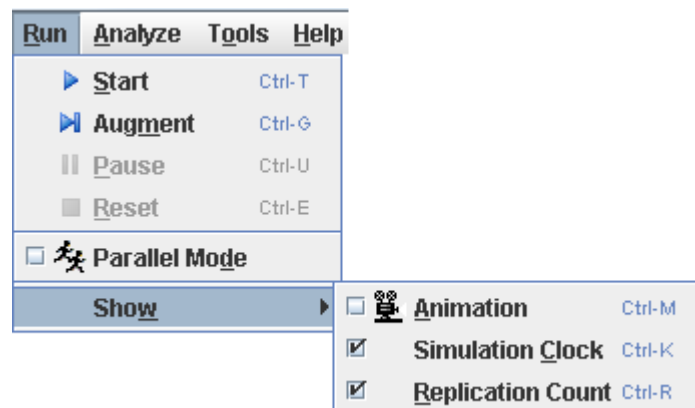
To access the JMP distribution-fitting platform, select **Analyze►Fit Distribution**. To open the configuration dialog box (Figure 3.2), select **Tools►Configuration**. You can use the **Help** menu to access online documentation for Simulation Studio and SAS Technical Support.

The toolbar (Figure 3.6) provides quick access to most of the functionality on the **Run** menu. The animation icon 🎥 acts as a toggle switch for turning execution animation on and off. The parallel mode icon 🏃 acts as a toggle switch for turning parallel mode on and off. The remaining toolbar buttons are discussed in Chapter 4, "Simulation Models."

Figure 3.6 Toolbar

Block Template Display Area

The Block Template Display area (Figure 3.7) consists of two components. The Template list box contains the names of all the templates currently loaded into Simulation Studio. The selection displayed in the Template list box represents the currently active template. In Figure 3.7 the template labeled Standard is active. The area immediately below the list box, called the Template Palette area, displays the templates for the individual blocks contained in the currently active template.

You can change the format of the displayed items in the Template Palette area by using the pop-up menu available on the Block Template Display area background. Display options include **Large Icons**, **Small Icons**, **List**, **Text Only**, and **Icons Only**. You can also use this pop-up menu to view specific information about an individual block. Selecting **Block Info** from the pop-up menu opens a dialog box to display information about the corresponding block. This information includes the block name, class path, icon, and tooltip associated with the block. Menu options are also available via the pop-up menu to remove blocks from and import blocks to a custom template.

When you rest the pointer on an individual block icon in the Template Palette, a tooltip appears that contains a brief description of the block. The Template Palette area is your source for blocks when you are using the Simulation Studio GUI to build your simulation model. To add blocks to your simulation model, drag template icons from the Template Palette into a Model window. This process creates an instance of the associated block in your model. Templates are discussed in detail in Chapter 12, "Block Templates."

Figure 3.7 Template Display Area
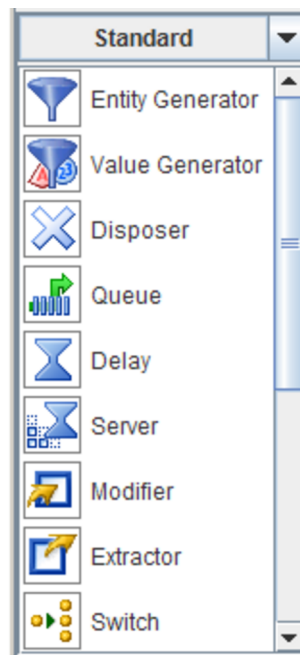
Simulation Studio Projects

A project in Simulation Studio (ideally) contains models and experiments that are in some way associated with each other and helps to provide organizational structure to all of your models and experiments. A project must contain at least one model and experiment, but there is no limit to how many models and experiments can be in a project. Any number of projects can be loaded into Simulation Studio at one time. In addition to organizing models and experiments, projects provide storage for factor and response definitions that can be shared across models and experiments in that project. Factors and responses are discussed in Chapter 5, "Experiments."

Project Explorer

The Project Explorer (located on the top left side of the GUI in Figure 3.1) uses a tree structure to display the projects, associated models, and experiments that are currently loaded into the application. Figure 3.8 shows a Project Explorer with two projects loaded: crane and repairshopDOE, each with one model and one experiment.

Figure 3.8 Sample Project Explorer

Selecting a project, model, or experiment name listed in the Project Explorer hierarchy causes the windows associated with that item to activate and pop to the top in the Project window. The activated model and experiment names are shown in **bold**. Up to one model and one experiment in a project can be activated.

Context-sensitive pop-up menus are available on the items displayed in the Project Explorer. You can right-click a project to open a dialog box to edit factors and responses associated with the project and also to change the base directory location where any simulation results are stored. You can right-click a model to open the Anchors dialog box to associate block parameters in a model to project factors and responses and also to set flags in blocks for automatically saving results. You can right-click an experiment to open a dialog box to include factors and responses. Factors, responses, and anchors are detailed in Chapter 5, "Experiments."

Project Window

Each project loaded into Simulation Studio has a Project window associated with it in the Project Desktop area of the GUI. A newly created Project window is displayed in Figure 3.9. Each Project window has a desktop area at the top and a tabbed window at the bottom. The Project Desktop area contains any Model windows and Experiment windows associated with the project. When a new Project window is first created, it has one (empty) Model window, one Experiment window, and at the bottom of the frame, **Log**, **Trace**, and **Animation** tabs. Project windows can be minimized as needed using controls on the Project window frame. To open the Factor and Response definition dialog boxes, right-click on the background of a Project window. See Chapter 5, "Experiments," for details about factors and responses.

Figure 3.9 Sample Project Window
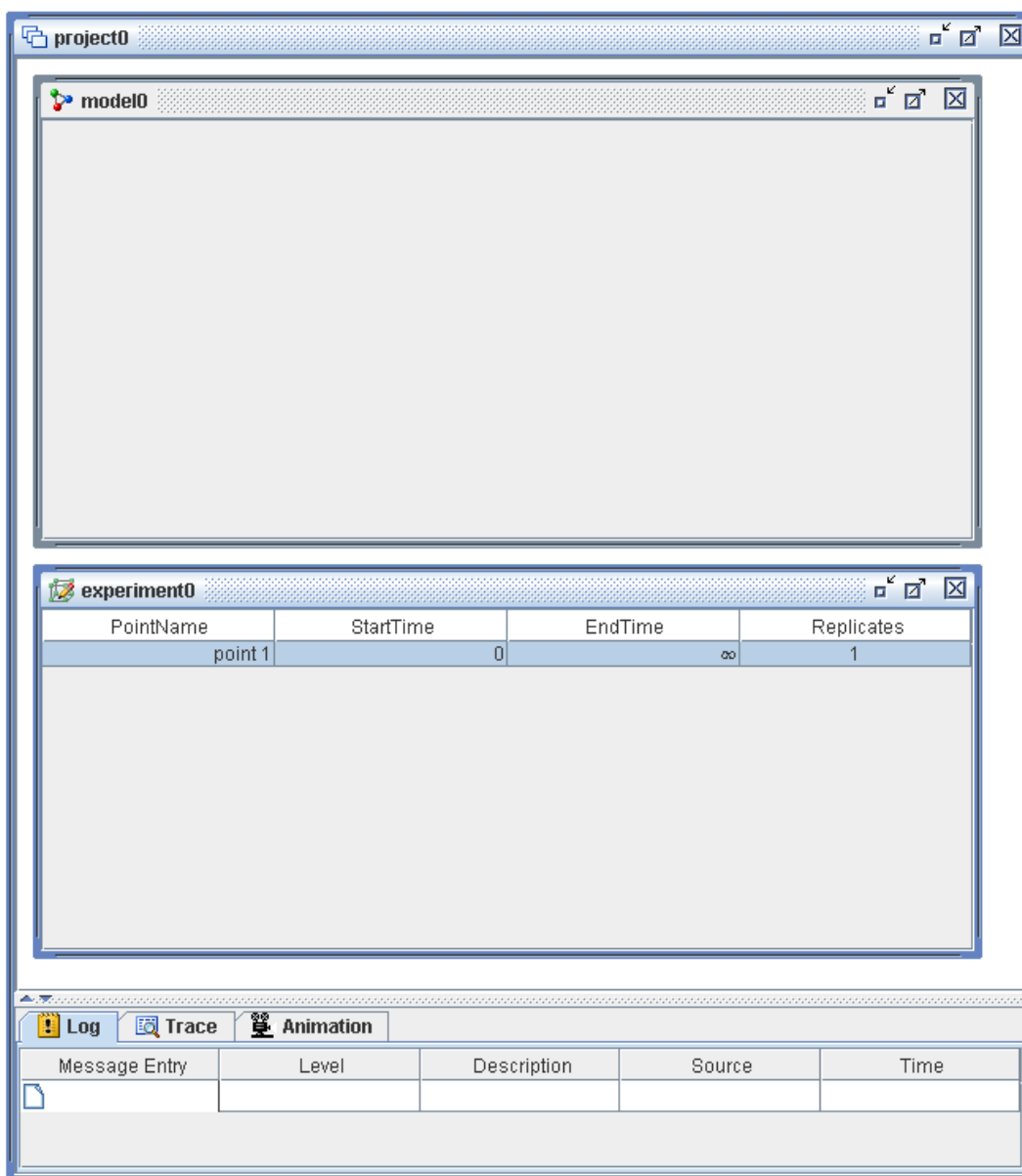
Model Window

Each model in a project has a Model window associated with it. You use this window to graphically construct and display a simulation model. You drag a block from the Template Palette into a Model window to create an instance of the block associated with that template in your model. You connect the blocks in your model by creating links between ports on the various blocks. You can right-click or double-click an individual block to open a dialog box where you can modify parameters associated with the block. Interacting with blocks and models is discussed in detail in Chapter 4, "Simulation Models." Closing a Model window deletes the window and removes the model entry from the Project Explorer. **NOTE:** Modified models are not automatically saved upon closing. You will however be prompted and asked whether you want to save the model before it is closed. Figure 3.10 displays a sample Model window that contains a simple M/M/1 model.

Figure 3.10 Sample Model Window
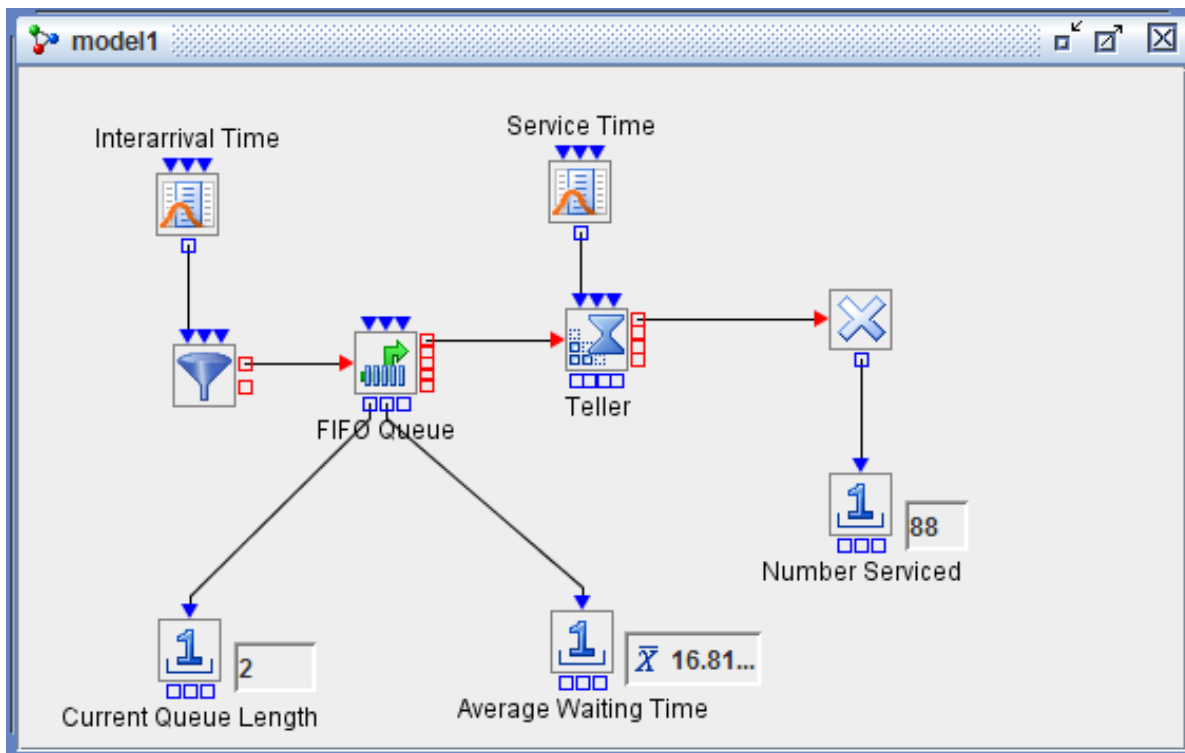
Experiment Window
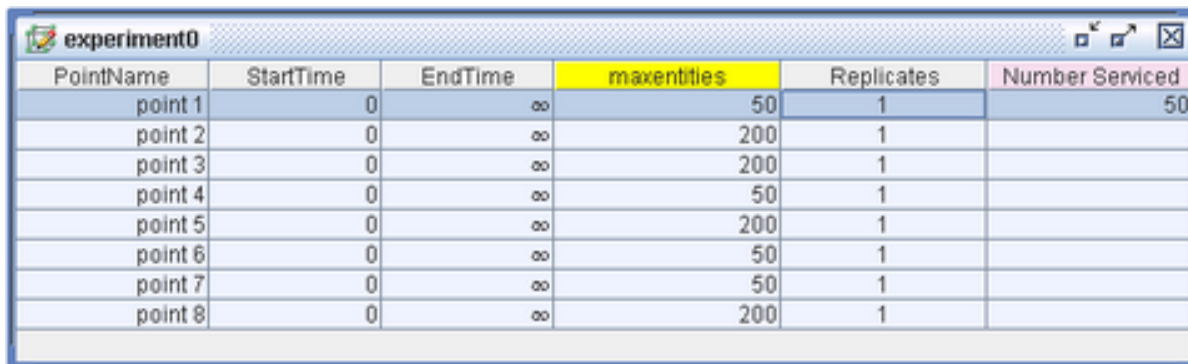
You use Experiment windows to control the initialization and running of simulation models. Each experiment in a project has an Experiment window in the Project window. By default, each Experiment window contains columns for controlling the start and end times of a simulation run (or design point) along with a column for designating how many times you want to repeat this run. An experiment must have at least one design point in order to run an associated simulation model.

You can use an Experiment window to control initialization of block parameters before running a simulation model. Any factor or response defined on the project can be included in an experiment. Using this and other features of experiments is discussed in Chapter 5, "Experiments."

As with a Model window, closing an Experiment window deletes the window and removes the experiment entry from the Project Explorer. Figure 3.11 shows a sample Experiment window including a factor labeled

maxentities and a response labeled Number Serviced. To modify the content of an Experiment window, right-click on the background of the Experiment window and select the appropriate item from the pop-up menu.

Figure 3.11 Sample Experiment Window

PointName	StartTime	EndTime	maxentities	Replicates	Number Serviced
point 1	0	∞	50	1	50
point 2	0	∞	200	1	
point 3	0	∞	200	1	
point 4	0	∞	50	1	
point 5	0	∞	200	1	
point 6	0	∞	50	1	
point 7	0	∞	50	1	
point 8	0	∞	200	1	

Log, Trace, and Animation Tabs

Each Project window also contains a Log, Trace, and Animation window in a tabbed format along the bottom of the Project window frame. (See Figure 3.9.) The **Log** tab displays log messages from either the currently running or most recently run model. Each log message has a Severity Level associated with it along with the source and simulation time of the message. If you click on a message in the log, the block in the model that generated the message will be highlighted.

The Tracer feature must be enabled for trace messages to appear in the **Trace** tab. You enable the Tracer feature by using a pop-up menu available on the **Trace** tab background. Trace messages are generated by individual blocks during the execution of a model and are intended to provide details about events and execution flow within the blocks.

The **Animation** tab provides options for controlling the simulation animation as a model runs. You can enable animation for different regions of the model, as well as adjust the animation speed, start time, and end time for each selected region.

Additional details about the log, trace, and animation facilities in Simulation Studio are provided in Chapter 11, "Model Debugging and Verification."

Project Status Bar

The Project Status bar, located at the bottom of the GUI, displays the path name of the currently active project.

Chapter 4
Simulation Models

Contents

Overview of Models

In Simulation Studio, the term *model* refers to an abstraction or representation of a system that you want to investigate or study. Most models represent a simplified version of the real system, but they still must capture the essence of the system under investigation to be useful. In Simulation Studio, models are composed of *blocks*, and blocks communicate with each other through *ports*. In the Simulation Studio GUI, blocks are said to be connected if a port on one block has a *link* to a port on another block, creating a path for information to flow between them. A model in Simulation Studio is usually a series of blocks arranged or connected in a configuration that represents the system under investigation.

As described in Chapter 3, "Introduction to SAS Simulation Studio," models in Simulation Studio are organized into projects. Each project has at least one Model window, and you use this window to construct your simulation model. This chapter provides an overview of the components of a model, including blocks, ports, and the types of information that flow between them. The details about each of these subjects are provided in later chapters. This chapter also discusses how to use the Simulation Studio GUI to build, run, and save a simulation model.

Blocks

In Simulation Studio, blocks are the most fundamental units used to build a simulation model. Each block usually encapsulates some well-defined and specialized functionality. Communication between blocks is accomplished through the ports defined on the individual blocks. In the Simulation Studio GUI, you manually create a link between the ports on blocks to provide a path for information to flow when the simulation is running.

Simulation Studio provides a default collection of basic blocks for model construction. These blocks appear in, and can be accessed through, the Standard template of the application. The details about these individual blocks is provided in Appendix A, "Templates." Each block has a pop-up menu associated with it that you can open by right-clicking the block in the Model window. This menu looks similar to the one displayed in Figure 4.1.

Figure 4.1 Sample Block Menu
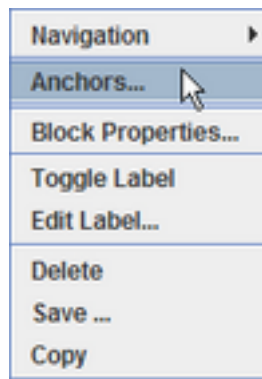
This menu provides various functionalities, including access to the Block Properties dialog box. The Block Properties dialog box displays any individual parameters for the block along with a block functionality overview page. You can also open the Block Properties dialog box by double-clicking the block.

In addition to the basic blocks provided by Simulation Studio, you can create compound and submodel blocks by aggregating a series of blocks. For more information about blocks and compound and submodel blocks, see Chapter 6, "Blocks," and Chapter 7, "Compound and Submodel Blocks."

Ports

Ports represent the basic interface to blocks. Blocks usually have multiple ports. Depending on the functionality of the block, a block can have static ports (the same ports are always available for this type of block) or dynamic ports (ports can be dynamically created or deleted based on various properties of the block). An example of a block with static ports is the Entity Generator block used to create entities. (See Figure 4.2.) This block always has an InterArrivalTime, BatchSize, Signal, OutEntity, and OutBalk port.

Figure 4.2 Entity Generator Block with Five Static Ports

The Modifier block has static ports and optional dynamic ports. You use a Modifier block (Figure 4.3) to assign attributes to an entity that flows through a model. The number of ports available on the Modifier block is dependent on the number of attributes you have decided to set using that block. In Figure 4.3, three attributes are assigned using this Modifier block, so the block's icon displays three dynamic attribute input ports along with the Modifier block's standard static input and (two) output entity ports.

Figure 4.3 Modifier Block with Three Dynamic Attribute Ports

Blocks have two types of ports: value ports and entity ports. The ports are color-coded with value ports displayed in blue and entity ports in red. Value ports are always located on the top and bottom of the block icon, and entity ports are displayed on the right and left sides of the block icon. In general, values are data-oriented information such as numbers, character strings, observation objects, and data model objects, while entities represent special objects that flow through the model during a simulation, potentially carrying additional information or properties along with them.

Each value port can either be an input value port or an output value port. Similarly, entity ports can be input or output ports. An input port is used to get information into the block, and an output port is either used by the block to push information out or used by another block to pull information from the block. Input ports are drawn as triangles on the perimeter of a block, and output ports are represented by squares. In Figure 4.2, the InterArrivalTime port represents an input value port and the OutEntity port is an output entity port. When you first rest the pointer over a port, a tooltip with a brief label for the port appears. The ports for each block are described on the **Overview** tab in the block's properties dialog box.

Each port has associated with it a Port Connections dialog box that is accessed by right-clicking the port. The Port Connections dialog box shows a list of all ports (and associated blocks) connected to the selected port. (See Figure 4.4.) The Order column in the dialog box indicates the priority of each port that is connected to the selected port. The order in which the ports appear in the dialog box is the order in which they are

activated when the selected port needs to communicate with another port. The order can be changed by selecting a row in the Port Connections dialog box and then clicking the Up or Down arrow button to move the selected row in the list. Connections can also be deleted by selecting a row in the dialog box and clicking the Delete button located under the arrow buttons.

Figure 4.4 Port Connections Dialog Box
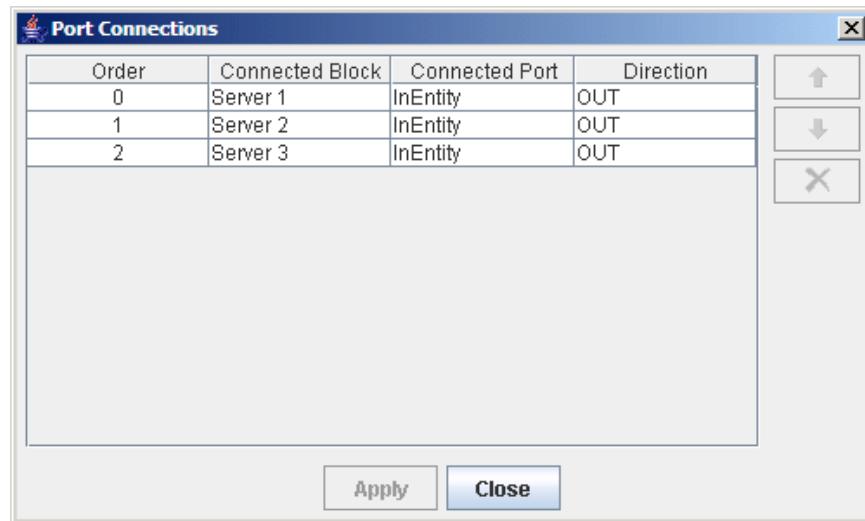
Entities and Values

Two general types of information are communicated between blocks: values and entities. *Values* can be numbers, strings, Boolean values, or objects. Frequently used object types include data model objects and observation objects. Data model objects can be used to store SAS data sets and JMP tables in a simulation model during a simulation run, and they are also used to store data created within a simulation model run. Observation objects contain a row of information from a data model, SAS data set, or JMP table. Number and string values in Simulation Studio are often associated with state information or properties on blocks, whereas data model and observation object values contain a collection of related data values. Value ports are used to access or set value information associated with blocks.

As an example, the Queue block has a numeric property called *capacity* that represents the maximum number of entities the queue can hold at one time. Although you can use the properties dialog box of a Queue to set its capacity, you can also connect the InCapacity port (as shown in Figure 4.5) to a numeric output port of another block that sends out a numeric value while the simulation is running. The Queue block also has a numeric value output port called OutLength. Every time the number of entities in the queue changes, the length of the queue is pushed out its OutLength port. It is also possible to query the length of the queue via a connection to the OutLength port.

Figure 4.5 Queue Block
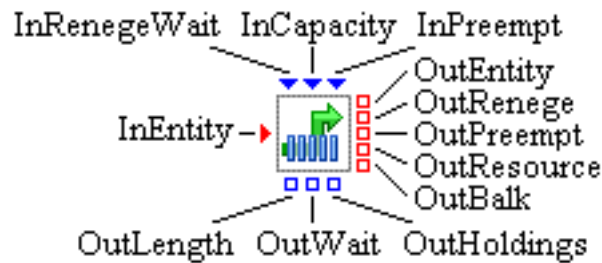
Although most values are simply numbers or strings, as mentioned previously, values can also be Java objects. Some blocks collect information or statistics and store the data in a custom Java object called a data model object. This data model object can be shared with other blocks. An example of this functionality is the Number Holder block. The Number Holder block provides an option to store data values that pass through the block in a data model object, and the Number Holder block has an output port called OutCollected to provide access to this data model object. For example, you might be interested in displaying the values being collected in the Number Holder block while the simulation is running, so you could connect the DataIn input port (of value type Object) of a Histogram block to the OutCollected port of the Number Holder block. The Histogram block would then display a representation of the data passing through the Number Holder block when the simulation is running.

The second type of information passed between blocks is called an *entity*. Entities are special objects in Simulation Studio and have a unique role in this discrete-event simulation application. Although value information tends to flow between two blocks and is immediately consumed, entities usually flow through the model and have a much longer life span. Entities can have properties or attributes assigned to them, and these properties might be modified during the simulation execution. Blocks can use property information assigned to entities in their internal processing and logic to affect simulation execution.

Consider the simple M/M/1 example presented in Chapter 2, "Overview of SAS Simulation Studio." Suppose this example represents customers waiting to check out at a cashier, and the entities that flow through the model represent customers. You could modify this model to assign a property to the customers (entities) to represent how many items the customer is purchasing and then use this information to determine how long it takes the customer to check out at the cashier.

Additional information about entities and their role in Simulation Studio is given in Chapter 8, "Entities."

Building a Model

Using the Simulation Studio GUI to build a simulation model is straightforward. It consists of dragging icons from the Template Palette window into a Model window and then creating links between the appropriate ports on the various blocks in the model. (Of course, the hard part is actually determining the appropriate composition of the model.)

To create an instance of a specific block in a Model window, you drag the appropriate icon or text in the Template Palette window into the Model window to the location where you want the block to be created. During the drag, a transparent icon is attached to the pointer. (You can always drag the new block instance around in the Model window to move it to another position.) You can modify the properties on any block

in your model by using the properties dialog box associated with each block. You can open this dialog by right-clicking the block and selecting the **Block Properties** menu item. Double-clicking the block icon also causes the properties dialog box to open.

One-step undo and redo options are available immediately after you insert or delete a block. If you perform another action after adding or deleting a block, such as forming a link, then the undo and redo options are no longer available for that block. The options are available from the model pop-up menu, which you access by right-clicking in the Model window. To remove the last block added to the model, select **Undo Block Insertion**. To insert the block back into the model, select **Redo Block Insertion**. To insert a deleted block back into the model, select **Undo Block Deletion**.

Creating Links

After you have blocks in your Model window, you can begin to create links between the ports on the various blocks to enable the flow of values and entities between blocks. To create a link, position the pointer over a port that you want to be an endpoint of the link and hold down the left mouse button. (Note that the port size enlarges when the pointer is over it, indicating port selection.) A rubber-band line appears that connects the selected port with the pointer. While holding down the left mouse button, move the pointer until it is positioned over the port you want to connect to and then release the left mouse button. If the port types that are associated with the two selected ports are compatible, a link is created between them.

Simulation Studio also offers a two-click method for creating links that is especially useful when the blocks you want to connect are far apart in the model. You can use the two-click method to create a link by moving the pointer until it is positioned over the first port to connect and then clicking the left mouse button. Move the pointer until it is positioned over the compatible port that you want to connect to (scrolling the Model window if necessary), and then click the left mouse button again to form the link. If you hold down the CTRL key while performing the second click, then you can continue to click on other input ports to form additional links. However, you cannot use the two-click method to connect tunnels on compound and submodel blocks. For information about tunnels, compound blocks, and submodel blocks, see Chapter 7, "Compound and Submodel Blocks."

One-step undo and redo options are available for the most recent link insertion or deletion action. These options are available from the Model window pop-up menu, which you access by right-clicking in the Model window. To remove the last link formed from the model, select **Undo Link Connection**. To insert a deleted link back into the model, select **Redo Link Connection**.

Connectors

Connecting blocks that are far apart in a model usually results in a link that crosses over other blocks and links. This can be visually distracting and confusing. You can use Connectors to create invisible links between blocks. To create an inter-Connector link:

1. Drag a separate Connector block from the Advanced template into the model window, and place it adjacent to each of the blocks in the model that you want to directly link. When you drag a Connector into a model, the Connector Type dialog box appears. In this dialog, you select the type of Connector: Entity, Number, String, Boolean, Observation, DataModel, or Entity Group. As discussed in the section "Ports" on page 32, each block port has a type associated with it. You base your selection of the type

of Connector on the type of the block ports between which you want to create a hidden link. Only Connectors of the same type can be connected. The ports on an entity-type Connector are red, and the ports on a value-type Connector are blue.

2. Create a link from the appropriate port on each block to the adjacent Connector block.

3. Create a link between the two Connector blocks. When the link is created, it briefly flashes and then disappears.

To show all inter-Connector links in the model, select **Navigation►Show Connector Links** from the pop-up menu either on an individual block or on a Model window. Links between entity-type Connectors are displayed in red when they are visible, and links between value-type Connectors are displayed in blue when visible. To hide all inter-Connector links, select **Navigation►Show Connector Links** again.

You can view a list of all input (or output) links by right-clicking on the input (or output) port of a Connector to open the Port Connections dialog box. To display a Connector link in the model, click on a row in the Port Connections dialog box (where each row represents a link with another Connector). You can also remove links or change the precedence order of the links in the Port Connections dialog box.

Running a Model

Before you can run your simulation model, you must have an active model and an active experiment, and the active experiment must have at least one design point selected or highlighted in the Experiment window. (For more information about experiments, see Chapter 5, "Experiments.") Although a project can have multiple models and experiments associated with it and multiple windows visible in the Project window, only one model and one experiment are considered active at any particular time. The active model and active experiment are identified by bold text for their names in the Project Explorer window. To activate a model or experiment, you can either select the name in the Project Explorer window or select the window that is associated with the model or experiment in the Project window.

After you have a valid model and experiment selected (that is, *active*), you can use any of the following methods to start the simulation running. You can select the ▶ icon on the toolbar or select **Run►Start** from the main menu. You can also select the ▶ icon on the toolbar or select **Run►Augment**. For more information about augment run, see Chapter 5, "Experiments." After you start a simulation run, Simulation Studio attempts to synchronize the currently active model and experiment. This process involves checking for correct factor and response anchors and for any other run-time errors (such as a required input port with no connection). If there is an error, the model execution does not begin and a severe-level message is posted to the Log window. For more information about the Log window, see Chapter 11, "Model Debugging and Verification." If this process is successful and there are no initial run-time errors, then the active model and experiment transition into the running state and their labels in the Project Explorer are displayed in red.

As a model is executing, a progress bar is displayed and animated in the upper right corner of the SAS Simulation Studio window, as shown in Figure 4.6. The progress bar displays the percentage of the total number of replications across all selected design points that have completed execution.

Figure 4.6 Progress Bar

By default, the simulation clock and replication count for the current design point are also displayed in the upper right corner of the SAS Simulation Studio window. As a model is executing, the clock icon ⏲ is animated. Controls for displaying the clock and replication count are provided under the **Run►Show** menu.

While a simulation model is running, only the ‖ Pause and ▪ Reset icons are selectable on the toolbar. You can stop or pause a running simulation at any point either by clicking the ‖ icon on the toolbar or by selecting **Run►Pause** from the main menu. When the execution of a model is paused, the paused clock icon 🕒 is displayed. When the simulation has finished running, the ‖ Pause icon is not selectable and the run completed clock icon 🕒 is displayed. The progress bar also displays 100%. **NOTE:** If a large amount of data is saved or a SAS or JMP script is executed at some point during a simulation run, then there might be a visible pause in the model execution during which the simulation clock might not be advancing and no animation might be visible. However, the simulation engine is still processing data.

During the early stages of developing and validating your simulation model, it is often helpful to use the animation feature in Simulation Studio. You can switch animation on and off by clicking the 🖳 toolbar icon or by selecting **Show►Animation** on the **Run** menu. When animation is activated during model execution, the Animate toolbar icon 🖳 flashes green and the flow of information is graphically depicted in the Model window; value movement is visualized by blue icons, and entity movement is visualized by red icons. These blue and red icons are shown traversing the various links in the model. Although animation slows the execution of the model, it can provide valuable insight when you are debugging your model or demonstrating the mechanics of your model to others. You can control the animation speed by using the slider control that is located next to the 🖳 toolbar icon. You can use the **Track Animation** option on the Model window pop-up menu, as described in the section "Navigation" on page 41 to ensure that the part of the model currently being animated is always visible in the Model window. The **Animation** tab (which you can expand at the bottom of the Project window) enables you to select which regions of the model to animate during model execution. For more information, see Chapter 11, "Model Debugging and Verification."

Finally, selecting the ▪ icon or **Run►Reset** reinitializes the states of the simulation clock and random number streams; it also invokes a reset method for certain blocks in the active model. In the reset state, the clock and replication count are unavailable.

Running a Model in Parallel Mode

If you are building a simulation model on a machine that has a multicore processor, then you can select the **Parallel Mode** option from the **Run** menu or the parallel mode icon 🏃 on the toolbar to improve the run-time performance of the model. When you select **Parallel Mode**, different design points and replications are distributed on different cores simultaneously. The actual distribution of design points and replications depends on the number of cores available along with the number of design points and replications that are selected to run. If you do not select **Parallel Mode**, then one single core is used to run the model. When **Parallel Mode** is selected, the progress bar displays the percentage of replications that have completed execution, and you can use the **Start**, **Augment**, **Pause**, and **Reset** run options on the toolbar or on the **Run** menu in the same way as in nonparallel mode (see the section "Running a Model" on page 37).

The **Parallel Mode** option is intended to decrease the amount of time it takes to run a model. Because in parallel mode more than one design point or replication is being run at the same time, the simulation clock, replication count, animation, and trace are disabled. If you want to run a model in parallel mode, then you have to ensure that your model does not attempt to save output data to the same location while multiple

replications or design points are executing. For example, if you use the Dataset Writer block to write a data set during the simulation run, Simulation Studio might attempt to open and write to the same data set simultaneously for different replications in parallel mode, resulting in a SEVERE log message. Similarly, using the SAS Program block during a simulation run can cause an error if multiple replications attempt to write to the same data set during the execution of the SAS program. Because the data collection blocks (such as the Number Holder, Bucket, and Stats Collector blocks) automatically save data to a different location for each replication and design point at the end of a run, using these blocks in a model does not cause any problems in parallel mode.

NOTE: In parallel mode, SAS should be used to read or write data during a simulation run and to execute code by using the SAS Program block. JMP should not be used in parallel mode to load data, write data, or execute a JSL script.

Searching a Model

Simulation Studio includes a search feature that you can use to locate blocks, compound blocks, or submodels in a particular model that satisfy specified criteria. You can open the Search dialog box, shown in Figure 4.7, by pressing CTRL-F when the Model window is active. If you click anywhere outside the Search dialog box, then the Search dialog box automatically closes. You can press CTRL-F to open it again.

Figure 4.7 Sample Search Dialog Box
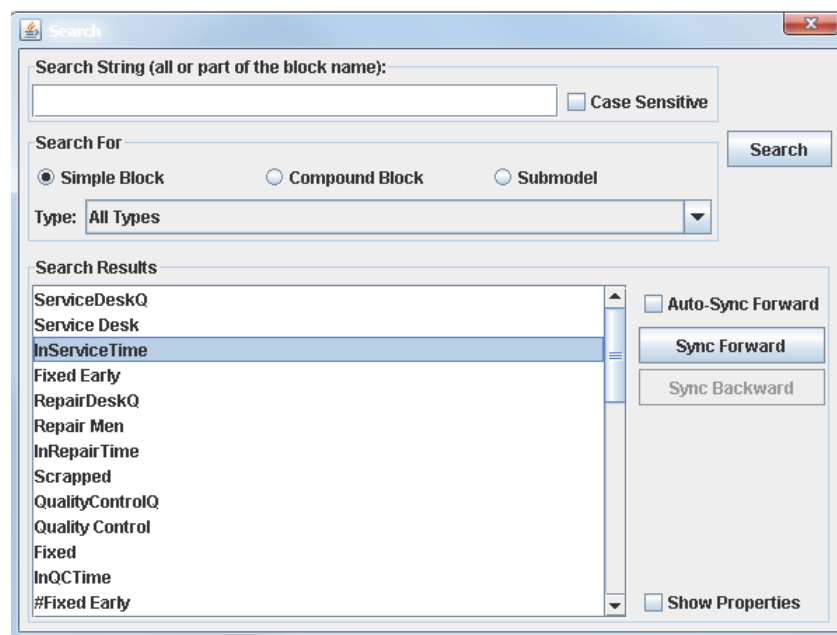
In the Search dialog box, you can specify a whole or partial block label name in the **Search String** box. This search string is used at the label-matching phase of the search process to select any blocks that have a label that matches the specified string. Select the **Case Sensitive** check box to indicate that the search should find only blocks with a label that matches the specified case-sensitive string. If the **Search String** box is blank, then the block label is not used during the search.

The **Search For** section of the dialog box enables you to select one of three categories of blocks to search: simple block, compound block, or submodel. If you select the **Simple Block** option, then you can specify the type of simple block (Entity Generator, Delay, Formula, and so on) in the **Type** list. Select **All Types** to indicate that the search should be applied to all simple block types. If you select the **Compound Block** option, then the **Type** list is not available.

After you select a block category and specify a search string (if necessary), click **Search** to execute the search process. When the search is complete, the **Search Results** box is populated with a list of block labels that satisfy the specified search criteria. You can then select block labels that are listed in the **Search Results** box to locate the corresponding block in your model. If the **Show Properties** check box is selected when you execute a search, then the Block Properties dialog box for a selected block in the **Search Results** field automatically appears, and you can inspect or edit the block properties as needed. If the **Auto Sync Forward** check box is selected when you execute a search, then the Model window scrolls to display the selected block in the **Search Results** field. If the block is one or more levels deep inside a collapsed compound block, then the compound blocks are automatically expanded to show the selected block.

If the **Auto Sync Forward** check box is not selected, then you can click **Sync Forward** to manually scroll the Model window until the selected block is displayed. If you click **Sync Forward** and the selected block is inside a collapsed compound block, then the compound block is automatically expanded to display the selected block. You can then click **Sync Backward** to automatically undo the sync forward operation and restore the collapsed status of the selected block's parent compound block.

Saving a Project

Saving models, experiments, and factor/response definitions is currently done on a project basis. That is, only entire projects can be saved. All models, experiments, and factor and response definitions that are associated with a project are saved when a project is saved. To save a project, select **File▶Save** or **File▶Save As** from the main menu. A File dialog box opens where you select the folder or directory location in which to save the project.

To improve security, models that you create using version 14.3 and later are saved in a different format than models that you create using previous versions of Simulation Studio. To save a model created in version 14.3 so that it is compatible with previous versions of Simulation Studio, select **File▶Export As** from the main menu.

Opening a Project, Model, or Experiment

Although you can save only an entire project, Simulation Studio enables you to open an individual project, model, or experiment.

Opening a project results in opening all models and experiments associated with that project. To open a project, model, or experiment, select **File** from the main menu. There are options to open a project, model, or experiment. When you open a project, a new entry is created in the Project Explorer tree for that project along with subentries for any models and experiments that reside in that particular project.

When you open an individual model or experiment, that item is included in the currently active project and a new leaf is created in the Project Explorer under the appropriate project node.

To improve security, models that you create using version 14.3 and later are saved in a different format than models that you create using previous versions of Simulation Studio. If you open a project or model in version 14.3 that was created in a previous version of Simulation Studio, then you will see the message shown in Figure 4.8. This message is also displayed if you open a model in version 14.3 that contains a submodel that was created in a previous version of Simulation Studio. After you save the old model (or submodel) in version 14.3, this warning message will no longer appear when you open it.

Figure 4.8 Warning Message for Old Model Format
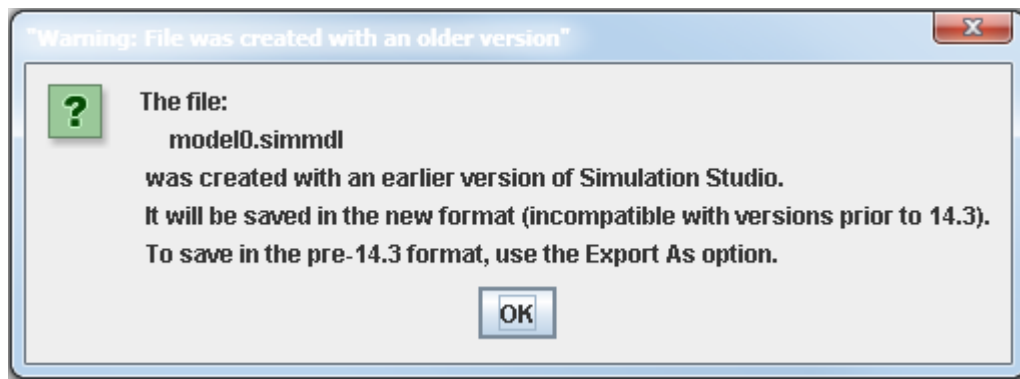
Model Window Pop-Up Menu

Each model window has a pop-up menu that includes **Navigation** and **Anchors** options, as shown in Figure 4.9. These two options are described in the sections "Navigation" on page 41 and "Managing Anchors" on page 42. As described in the section "Building a Model" on page 35, the model pop-up menu can also contain the **Undo Block Insertion**, **Redo Block Insertion**, **Undo Link Connection**, **Redo Link Connection**, and **Undo Block Deletion** options. If a block or compound block has been copied to the clipboard, then the **Paste** option is also available on the Model window pop-up menu. For more information about copying blocks see Chapter 6, "Blocks."

Figure 4.9 Model Window Pop-Up Menu
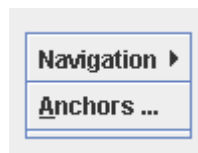
Navigation

The **Navigation** menu item on the block pop-up menu has three submenu items: **Show Snapshot**, **Track Animation**, and **Show Connector Links**. The features that are associated with these items are particularly

useful when you work with large simulation models. Selecting **Show Snapshot** creates a new window that displays a scaled-down version of the entire model in the currently active model window. This new window is called the Snapshot window. The portion of the model visible in the original model window is depicted by a highlighted rectangle in the Snapshot window. You can drag the highlighted region in the Snapshot window to reveal other regions of the model in the associated model window.

When you run a large simulation model, the area of the model where the animation is occurring might not be visible in the model window. You can select **Track Animation** to enable the model window to scroll (when the simulation is running with animation turned on) to ensure that the part of the model currently being animated is always visible.

By default, the links between Connector blocks are invisible. You can select **Show Connector Links** from the **Navigation** submenu to display all connector links in the model.

Managing Anchors

As mentioned in Chapter 5, "Experiments," anchors are used in association with experiments. Each project has a collection of factors and responses that are defined on it, and each experiment in that project might include any of these factors and responses. Simulation Studio must know how to map the factors and responses in an experiment to block parameters in the simulation model under investigation. This is accomplished through *anchors*. The Anchors dialog box (Figure 4.10) shows all factor and response anchors that are defined on the model. Use the **New**, **Edit**, and **Remove** buttons in this dialog box to manipulate the entries in these tables.

Figure 4.10 Sample Anchors Dialog Box
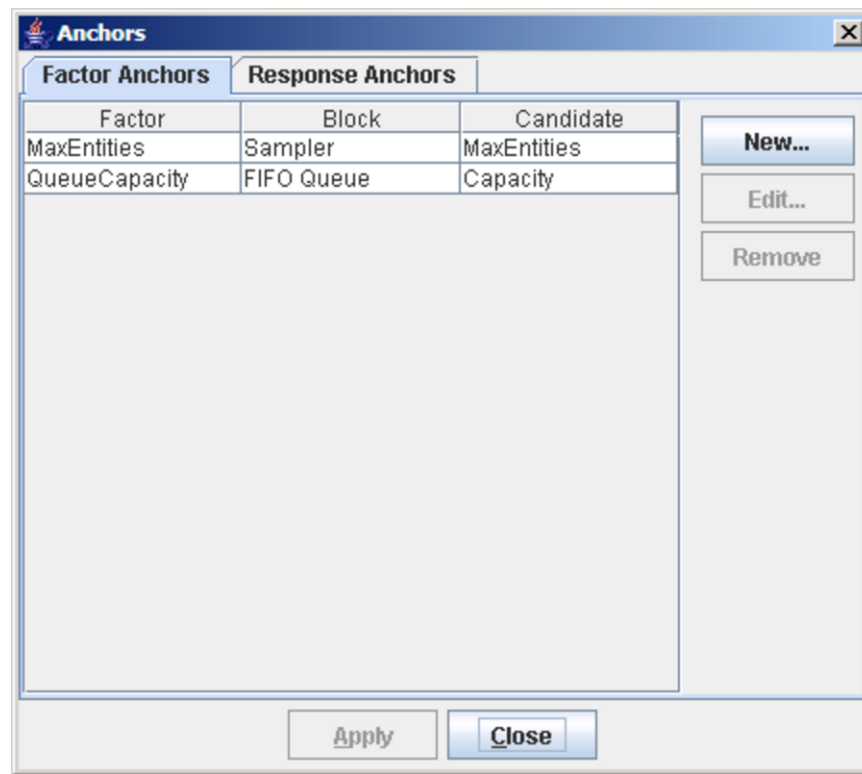
Clicking **New** opens the New Anchor dialog box (Figure 4.11), where you can associate block parameters in your model to factor and response definitions in the project. Selecting a block label in the model tree on the left side of the New Anchor dialog box causes the names of any factor (or response) parameter candidates that are associated with the block to be displayed in the center **Candidates** list. To match a parameter with a project factor (or response), you must select both the parameter name in the **Candidates** list and the project factor name in the **Factors** list, and then click **OK**. A new entry is then created in the Anchors dialog box. The anchors are matched to the experiment factors and responses when you attempt to run an experiment.

Figure 4.11 New Anchor Dialog Box
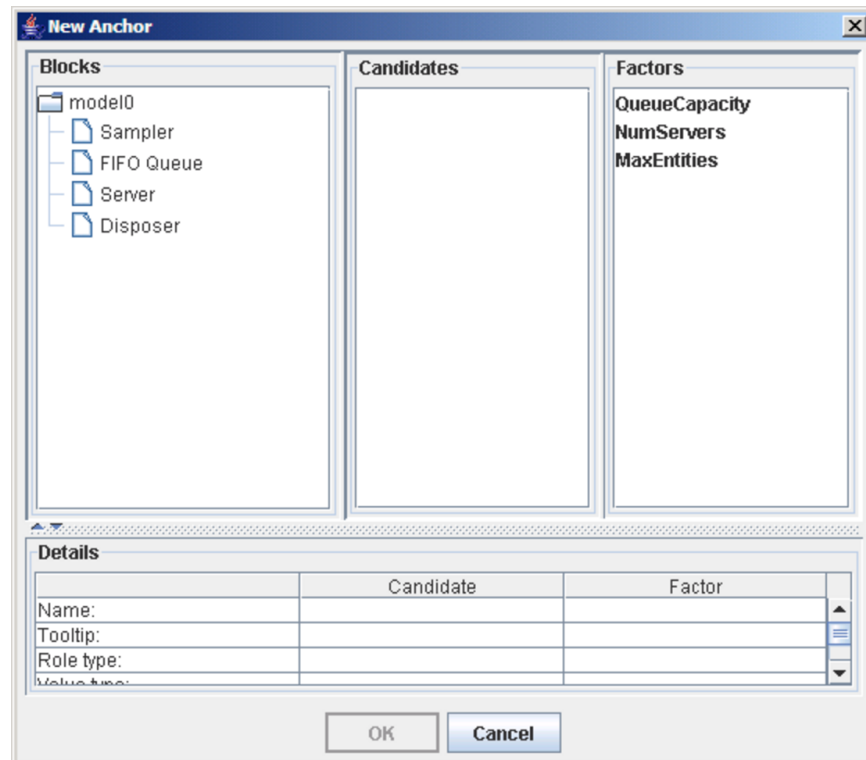
Chapter 5
Experiments

Contents

Overview of Experiments

The concept of an *experiment* can have a variety of meanings in different contexts and fields of study. In SAS Simulation Studio, an experiment provides a facility to automate the initialization and running of simulation models and also to record measures from a simulation run. That is, you can use an experiment to set parameters for blocks in your model before you run the model (without manually editing the individual blocks) and also to extract and record measures from blocks at the end of each simulation run. This feature provides you with a powerful facility to automate running a wide range of simulation scenarios and the capability to generate simulated results for experimental designs (such as full or fractional factorial designs). The process of creating an experimental design includes the following steps: defining factors and responses, creating model anchors for the factors and responses, and including factors and responses in the Experiment window.

Defining Factors, Responses, and Anchors

In keeping with traditional experimental design terminology, in this chapter the term *factor* describes a variable or parameter that is manipulated or changed for each experimental design point. The term *response* refers to a measure that is recorded for each experimental run. In Simulation Studio, factors and responses are defined for each project. You use the Factors and Responses dialog boxes (Figure 5.1 and Figure 5.2, respectively) to define factors and responses for a specific Simulation Studio project. To open the Factors or Responses dialog box, right-click in the Project window background (or right-click the project name in the Project Explorer) and select **Factors** or **Responses**.

Figure 5.1 Factors Dialog Box

Figure 5.2 Responses Dialog Box

In the Factors dialog box, you specify a unique name for a factor in the **Name** field, and you use the optional **Label** field to specify a detailed description of the factor. The **Value Type** field specifies the data type of

the factor in the simulation model (double, integer, text, or Boolean), and the **Default Value** field is used to set the initial value of an included factor when you add a new design point in the Experiment window. You can add values to the **Values** table to define the levels of the factor. If you use JMP to create a design, as described in Appendix C, "Design of Experiments," then the **Role Type**, **Value Type**, and **Values** fields are used in the design generation process. Even if you do not use JMP to generate a design, you can still use the **Role Type** and **Values** fields to define a suitable range of values for a factor, so the Factors dialog box essentially becomes a repository of information about the factors for a project.

To create a response, you specify a unique name in the **Name** field of the Responses dialog box, and you use the optional **Label** field to specify a detailed description of the response. If you are not using JMP to generate an experimental design, then you can leave **Goal Type** as 'None' and **Values** as 'NaN'. If you are using JMP to generate an experimental design, then the **Goal Type** field tells JMP whether to maximize your response, minimize your response, or match a target, and the **Values** field specifies lower and upper bounds. For more information about experimental design using JMP, see Appendix C, "Design of Experiments."

After you define factors and responses, you need to map the factors and responses to block parameters and measures in the simulation model before including them in an experiment. You do this by using an *anchor*, which defines the link between a factor or response defined for a project and an actual block parameter or measure in a specific model. It is possible to link a single factor to multiple blocks in a model.

For an illustration of the anchor definition process, consider the simple queueing model depicted in Figure 5.3 of a banking system with one teller. To open the Anchors dialog box (Figure 5.4), which you use to define both factor and response anchors, right-click in the Model window and select **Anchors** from the pop-up menu. The Anchors dialog box has separate tabs for displaying the factor anchors and the response anchors. In Figure 5.4, no anchors have been defined for this model yet.

Figure 5.3 Simple Bank Teller Model
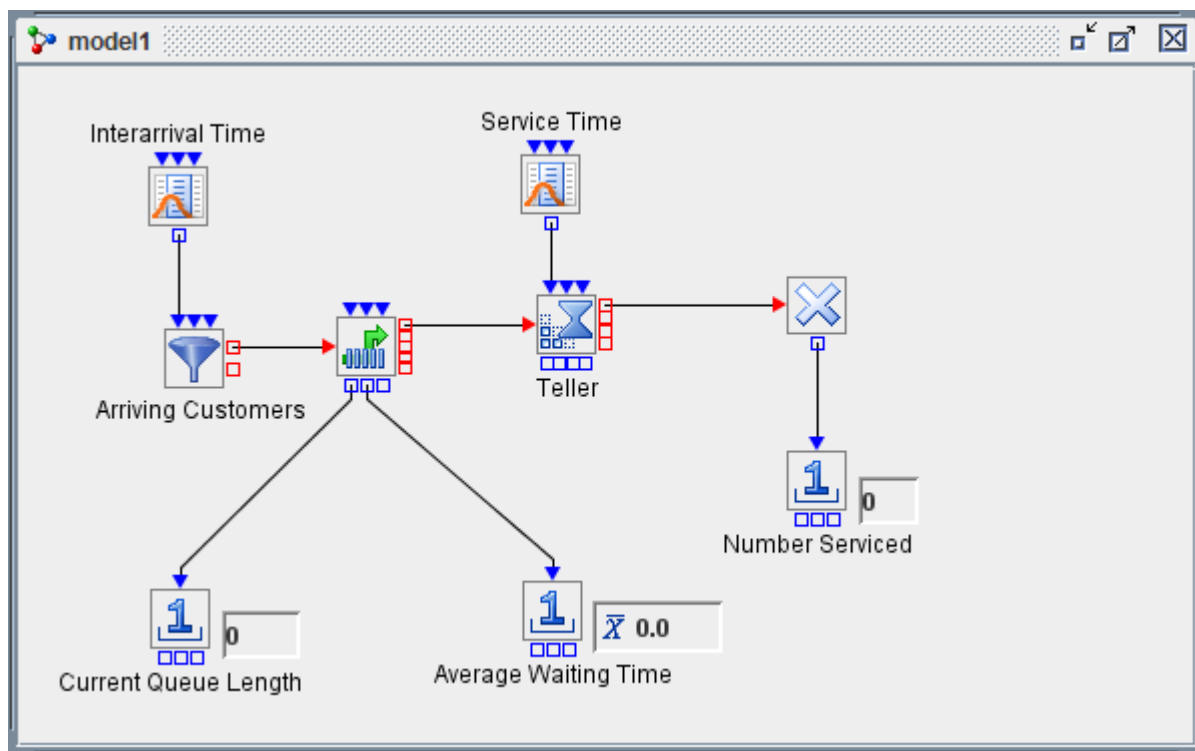
Figure 5.4 Sample Anchors Dialog Box
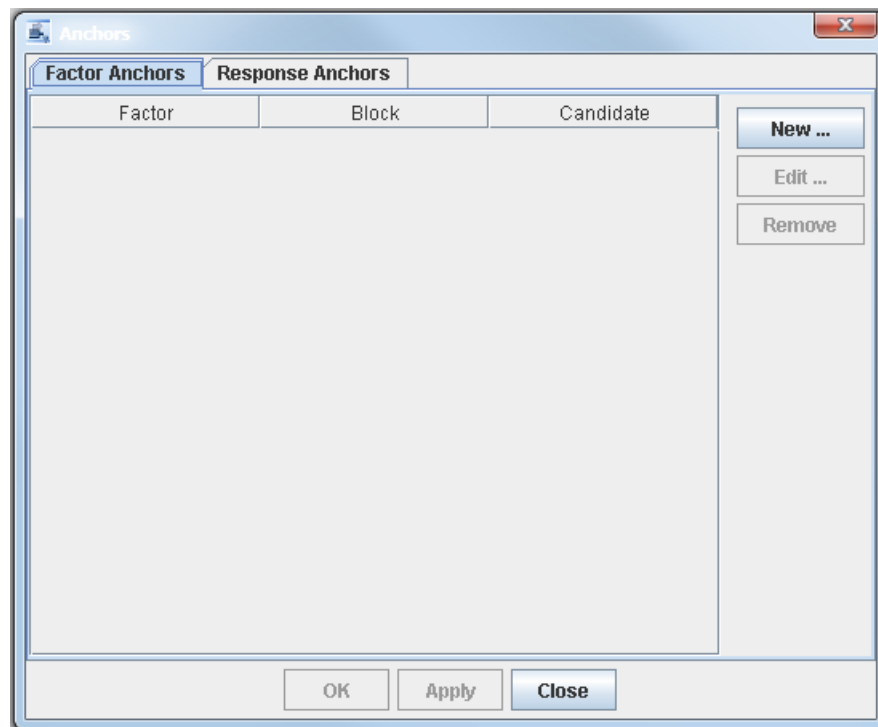
To create a factor anchor, click **New** on the **Factor Anchors** tab in the Anchors dialog box to open the New Anchor dialog box for factors. (See Figure 5.5.) The **Factors** list in the upper right area of the New Anchor dialog box displays the names of all the factors defined for the project. In this example, the project has one defined factor: NumTellers. When you select a factor name from this list, the details that are associated with that factor are displayed in the **Details** table of the dialog box below the **Factors** list.

Figure 5.5 Sample New Anchor Dialog Box for Factors

The **Blocks** area displays a hierarchical representation of the names of all blocks in the model that have potential factor anchors. Selecting a block name populates the **Candidates** list with the names of possible factor anchors associated with the selected block. In Figure 5.6, the Server block labeled Teller is selected, and all of its potential factor candidate anchors appear in the **Candidates** list. The details for any item that you select in the **Candidates** list are displayed in the **Details** table below the **Candidates** list.

Figure 5.6 Candidate Factor Anchors for the Teller Block

To create an anchor link between a factor anchor candidate from a block in your model and a factor defined for your project, select an item in the **Candidates** list and an item in the **Factors** list, and click **Apply**. This action adds an entry to the Anchors dialog box, which opens after you click **Close** in the New Anchor dialog box. (See Figure 5.7.) You can edit an entry or remove it from this table by selecting the appropriate row in the table and clicking either **Edit** or **Remove**.

Figure 5.7 Anchors Dialog Box with a Factor Anchor Defined
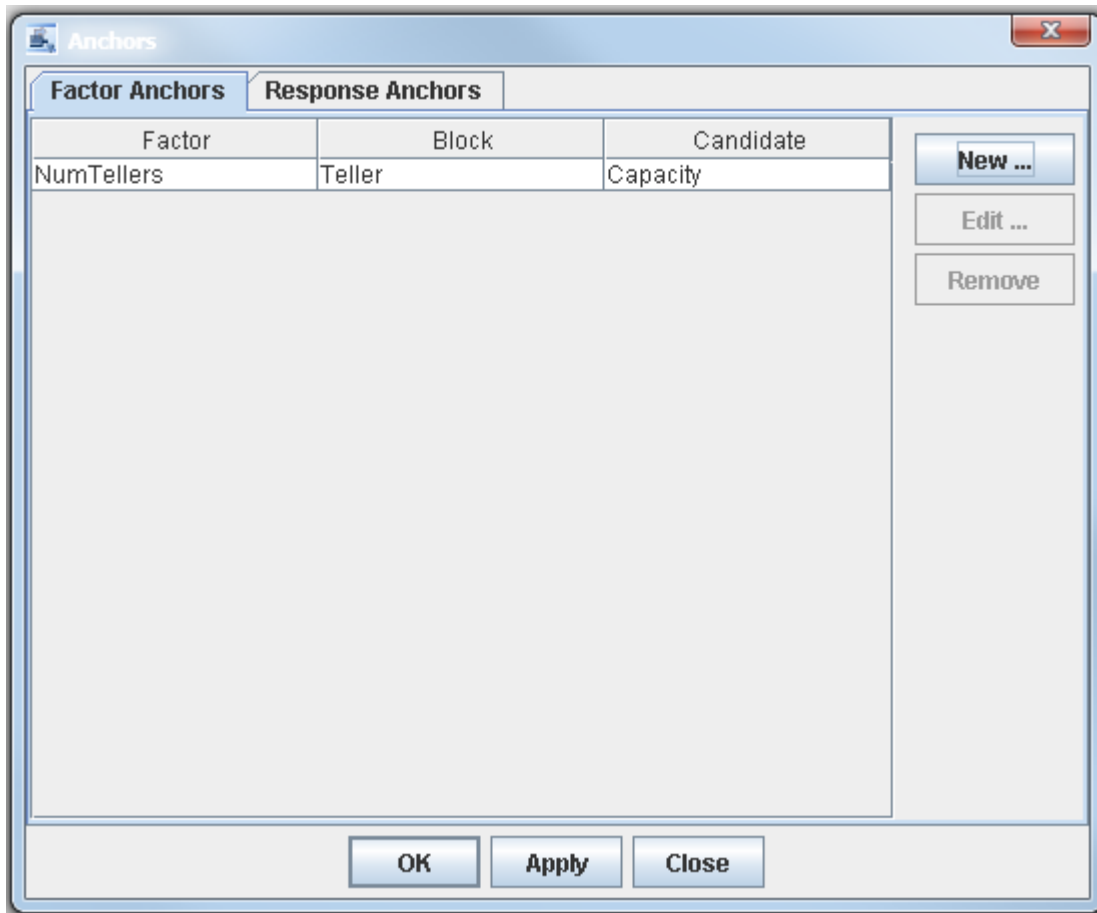
The process of creating response anchors is analogous to the process of creating factor anchors. To create a response anchor, click the **Response Anchors** tab in Figure 5.4 and click **New** to open the New Anchor dialog box for responses. (See Figure 5.8.) The **Responses** list in the upper right area of the New Anchor dialog box displays the names of all responses defined for this project. In this example, the project has five defined responses. When you select a response name in this list, the details that are associated with that response are displayed in the **Details** table of the dialog box below the **Responses** list.

Figure 5.8 Sample New Anchor Dialog Box for Responses
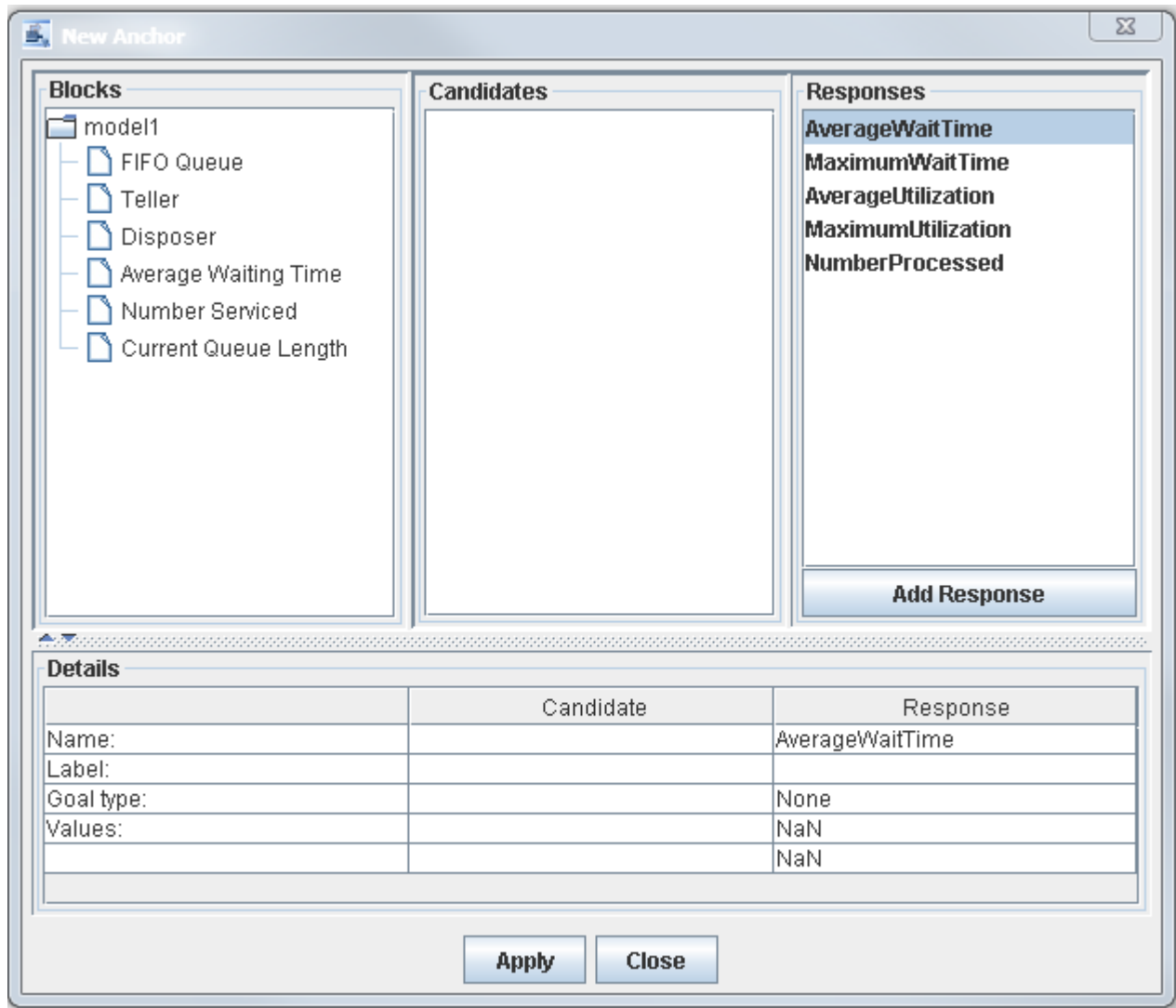
The **Blocks** area displays a hierarchical representation of the names of all blocks in the model that have potential response anchors. Selecting a block name populates the **Candidates** list with the names of the possible response anchors associated with the selected block. In Figure 5.9, the block named FIFO Queue is selected, and all of its potential response anchors appear in the **Candidates** list. The details for any item that you select in the **Candidates** list are displayed in the **Details** table below the **Candidates** list.

Figure 5.9 Candidate Response Anchors for the FIFO Queue Block
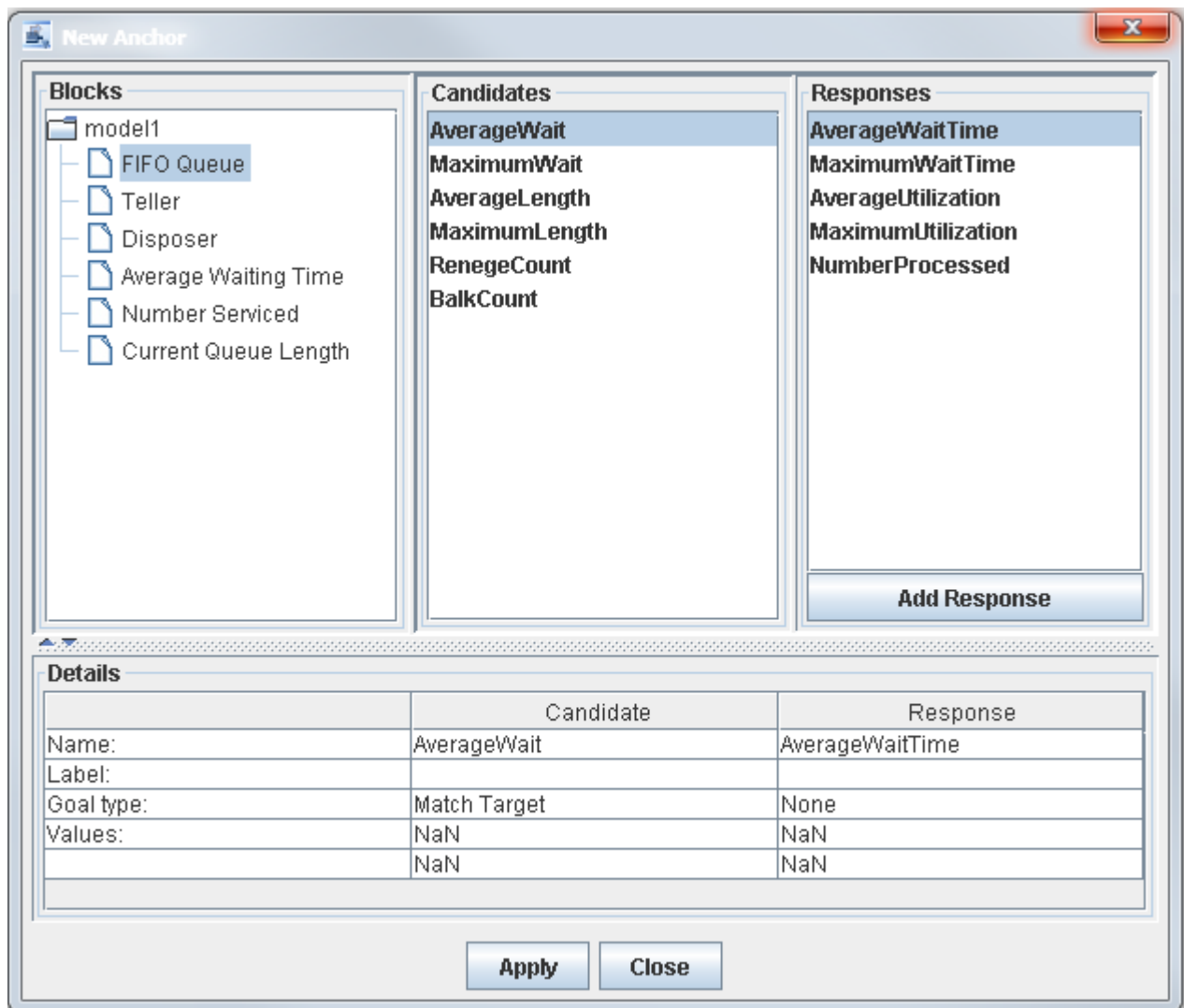
To create an anchor link between a response anchor candidate from a block in your model and a response defined in your project, select an item in the **Candidates** list and also an item in the **Responses** list, and click **Apply**. This action adds an entry to the Anchors dialog box, which opens after you click **Close** in the New Anchor dialog box. (See Figure 5.10.) You can edit an entry or remove it from this table by selecting the appropriate row in the table and then clicking either **Edit** or **Remove**.

Figure 5.10 Anchors Dialog Box with a Response Anchor Defined
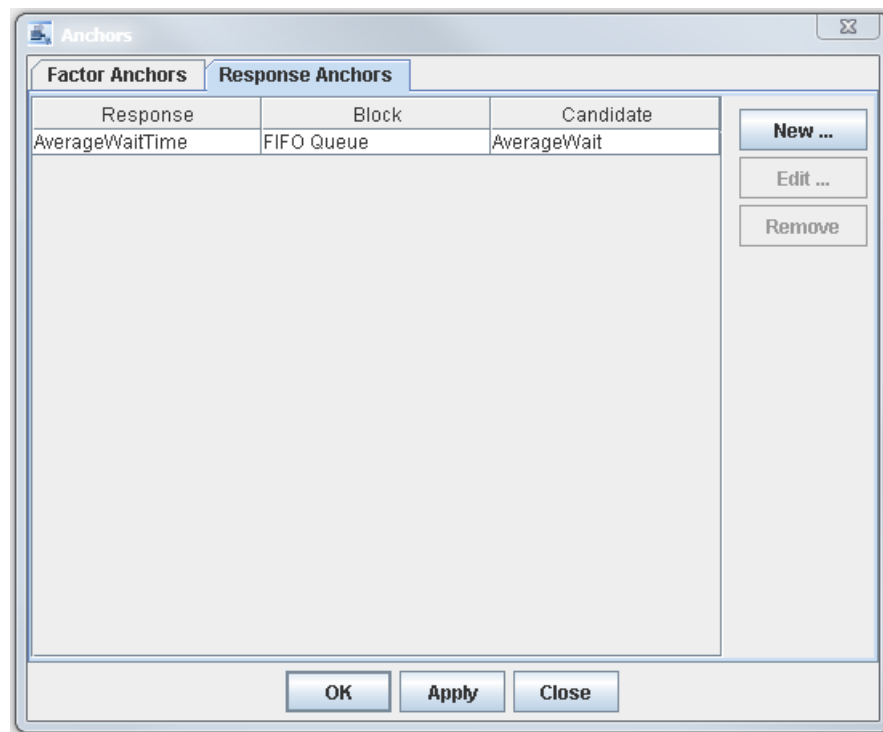
The process of creating factor and response definitions essentially creates a factor and response database for the project. This database facilitates the reuse of these definitions across experiments and models within the same project. Any factor or response that you define in a project can be included in any experiment in the project. To include a factor or response in (or remove it from) an experiment, right-click in an Experiment window and select **Factor Inclusion** or **Response Inclusion** from the pop-up menu, as shown in Figure 5.11. In the dialog box that opens, select or clear the factor or response as desired. When you select the factor or response name, the details that are associated with that factor or response are displayed on the right side of the Factor Inclusion or Response Inclusion dialog box. Figure 5.12 shows the inclusion of the NumTellers factor. After you click **OK** in the Factor Inclusion or Response Inclusion dialog box, a column is added to the Experiment window for the included factor or response.
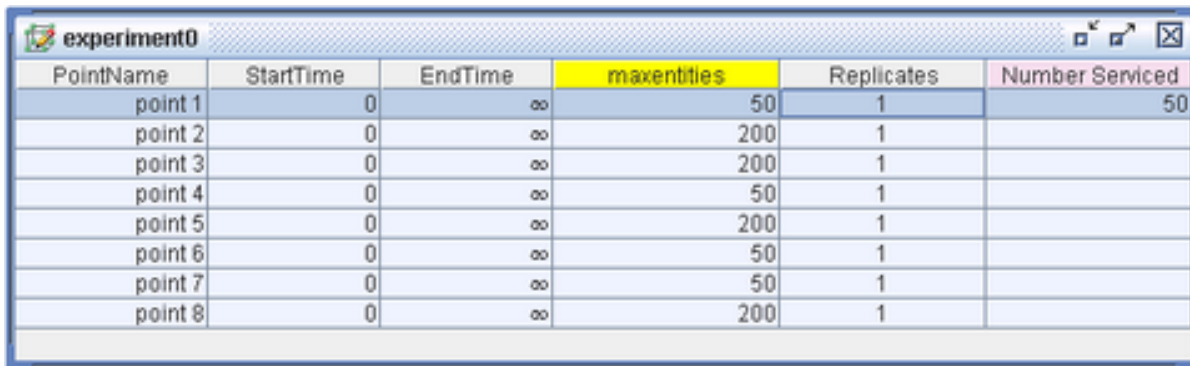
Figure 5.11 Experiment Menu
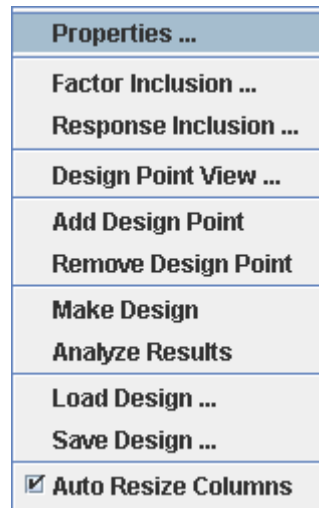
Figure 5.12 Factor Inclusion Dialog Box
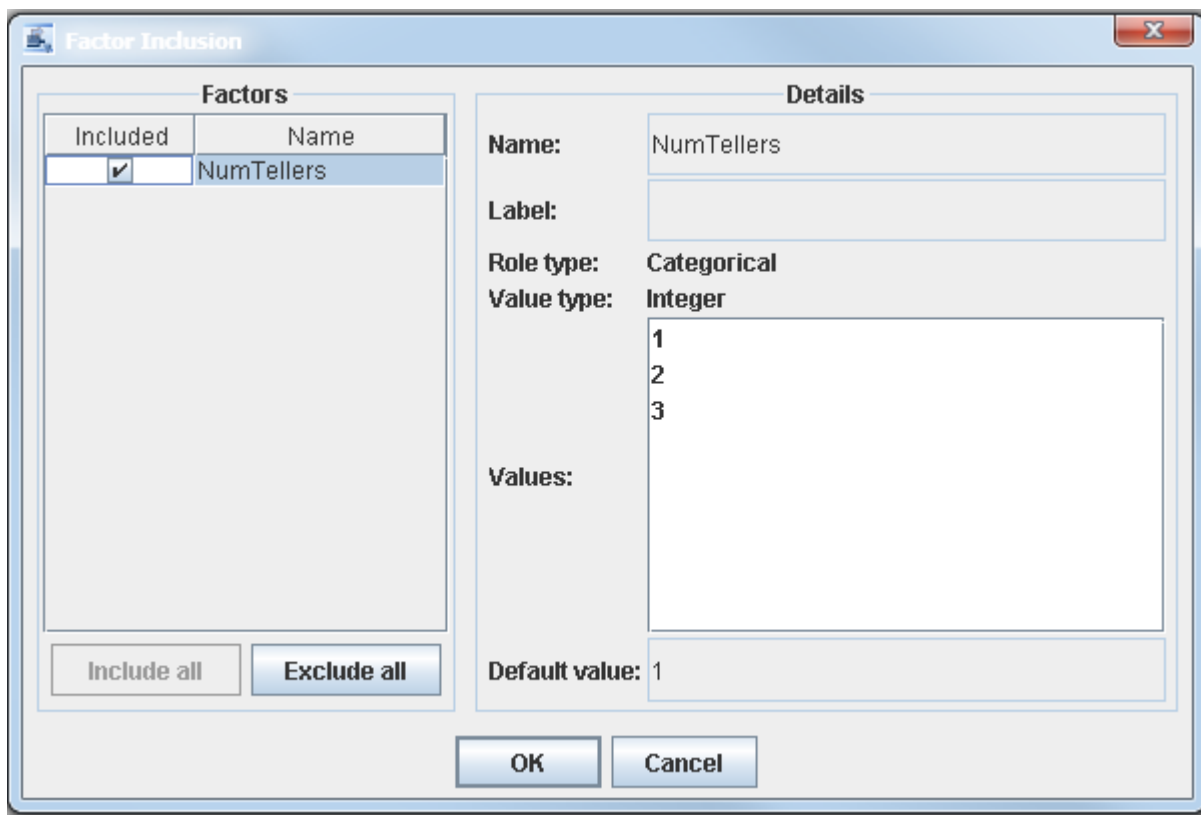
When you run a simulation model–experiment combination, Simulation Studio attempts to map the factor and response anchors that you defined for the model to the factors and responses included in the experiment. If a mismatch exists, Simulation Studio displays an error message in the Log window and stops the model execution.

The Experiment Window

Figure 5.13 shows a sample Experiment window. The table in this window is sometimes referred to as the *design matrix* for an experiment. All the information that is associated with an experiment is displayed in the design matrix. Each row in the matrix is called a *design point*, and it contains values for a label, execution controls, factor settings, and responses measured during model execution. You can add or remove design points by using the pop-up menu that opens when you right-click in the Experiment window.

Figure 5.13 Sample Experiment Window

PointName	StartTime	EndTime	maxentities	Replicates	Number Serviced
point 1	0	∞	50	1	50
point 2	0	∞	200	1	
point 3	0	∞	200	1	
point 4	0	∞	50	1	
point 5	0	∞	200	1	
point 6	0	∞	50	1	
point 7	0	∞	50	1	
point 8	0	∞	200	1	

The design matrix initially has four default columns, labeled PointName, StartTime, EndTime, and Replicates. The PointName column assigns a label to the design point. Although a default value (*pointN*) is generated for this field when you create the design point, you can edit this field to be any text you choose. The StartTime and EndTime values control the execution start time and end time (on the simulation clock) for an individual simulation run. The simulation clock begins running at time displayed in the StartTime field for the design point, and the simulation ends when the simulation clock reaches the time entered in the EndTime field. (Note that time has no unit in Simulation Studio.) The final default column in the design matrix, Replicates, contains a value that represents how many times you want to run this particular design point.

You can edit the default values for StartTime, EndTime, and Replicates by selecting **Properties** from the Experiment window pop-up menu and modifying the values in the Experiment Properties dialog box. The default values in the Experiment Properties dialog box are applied to new design points that are subsequently added to the Experiment window. Changes to the default values for StartTime, EndTime, and Replicates are not applied to existing design points. The StartTime, EndTime, and Replicates columns each have a pop-up menu (which opens when you right-click the column name) that has a menu item called **Set Value**.

To change default values for new and existing design points, you use the **Set Value** option from the pop-up menu that opens when you right-click the StartTime, EndTime, or Replicates column name. For example, Figure 5.14 shows the Set Value: Replicates dialog box, which you open by right-clicking the Replicates column name and selecting **Set Value**. The **New Value** text box enables you to specify a value for the number of replications. The **All** and **Selected** options enable you to specify whether the new value should be applied to all design points or to only the selected ones. The value that initially appears in the **New Value** field is the default replicates value, which is set in the Experiment Properties dialog box. The Set Value: EndTime and Set Value: StartTime dialog boxes have functionality similar to that of the Set Value: Replicates dialog box.

Figure 5.14 Set Value: Replicates Dialog Box
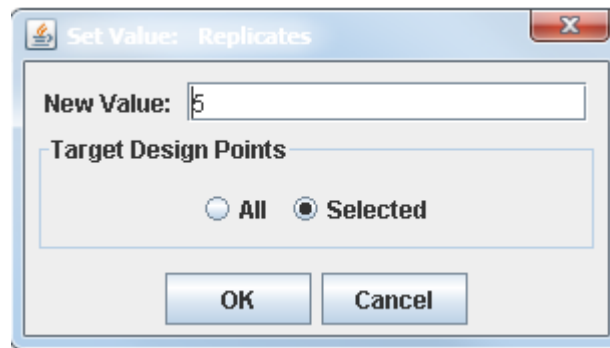
Only the default columns in the design matrix are necessary for you to actually run a simulation model. Figure 5.13 shows a design matrix with one factor, *maxentities*, and one response, *Number Serviced*. You can add any factors or responses defined for a project to the design matrix by using the Factor Inclusion or Response Inclusion dialog boxes, respectively, as described in the previous section. Color coding in the column header of the design matrix indicates which role that column plays in the experiment. The default columns have gray headers, any added factors are denoted by yellow background headers, and a pink background header is used for response columns.

Design Points

You can run an experiment by using only one design point and only the default execution control parameters (StartTime, EndTime, and Replicates). In fact, this is often the case when you are first building and debugging your model. However, after you have confidence in your simulation model and you want to use it to investigate the process that you are trying to model, automating the manipulation of block parameters and running the corresponding simulations become very important and useful.

After you determine what factors and responses to include in your experiment, the next step is to determine how many design points you need in your experiment and what values are assigned to the individual cells in your design matrix. Options in the Experiment window pop-up menu enable you to add (and delete) design points to the design matrix. You can manually create new rows in the matrix and then edit the individual cells to enter the desired parameter or factor values.

If you are interested in an automated approach to designing your experiment, Simulation Studio provides a link to the JMP Custom Design platform. To access this functionality, select **Make Design** from the Experiment window pop-up menu to automatically generate a design by using the JMP Custom Design platform. Select **Analyze Results** to conduct statistical analyses of the simulated results in JMP (including metamodel estimation) after running the design. For more information about using JMP software to populate your design matrix and analyze the results for your simulation model, see Appendix C, "Design of Experiments." The **Make Design** and **Analyze Results** options in the Experiment window are for use only with JMP, and before you can use these options, you must launch the JMP server. For information about launching the JMP server, see Chapter 3, "Introduction to SAS Simulation Studio." If you do not have JMP installed, you will get an error if you select the experiment option **Make Design** or **Analyze Results**. You can also use SAS to generate a design, as described in Appendix C, "Design of Experiments."

If your experiment contains a large number of factors or responses, then you might need to scroll through and possibly expand the collapsed columns in the Experiment window to see the desired factor or response

value. As an alternative, the Experiment window provides a design point view that organizes the factors and responses in rows rather than columns, making it easier to access factor and response values.

To view a single design point, select **Design Point View** from the Experiment window pop-up menu. A sample design point view is shown in Figure 5.15. This view enables you to edit the parameters for one design point at a time. You can switch to a different design point by using the **Index** field. You can use the **Value** column in the **Factors** area of the dialog box to change factor values. After you run a design point, the summary is displayed for each response (by default, the mean) over all replications in the **Responses** area. You can view individual replicate responses by clicking **Replicate Values**.

Figure 5.15 Experiment Window Design Point View
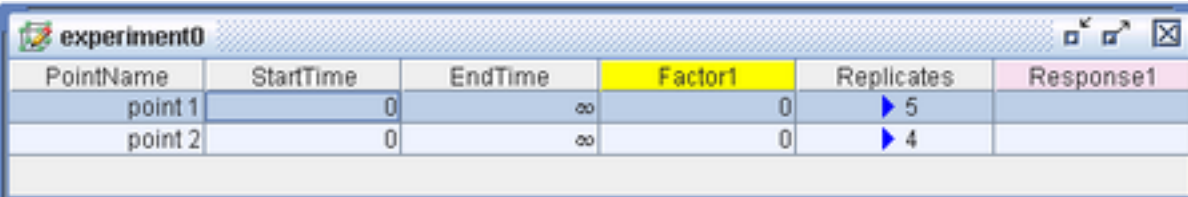
Replicate Rows

One of the default columns in the experiment design matrix is labeled Replicates. The value in this column represents the number of times you want to run the associated design point in the experiment. The default value for the entire column is one replicate. To edit the default replicate value, right-click in the Experiment window and select **Properties** to open the Experiment Properties dialog box. You can also right-click the Replicates column name and select **Set Value** to open the Set Value: Replicates dialog box, as shown in Figure 5.14.

Each replication run for a particular design point uses the same factor values. It is possible that a particular block property that a factor is anchored to could change dynamically during the course of a simulation run. If that occurs, the value of the particular block property will be reset to the specified factor value at the start of the next replication. For example, suppose the capacity of the Server block is anchored to a factor, and during the simulation run the capacity of the Server block is changed using the InCapacity port. At the start of the next replication, the Server capacity will be reset to the factor value that is specified in the Experiment window.

At the start of a new replication of a design point, all random streams advance automatically to their next substream so that each replication of a design point uses different random numbers. By default, each source of randomness in a model is assigned a different random stream (the most common source of randomness in a model is a Numeric Source block with a distribution specified). However, when you run a new design point, all random streams reset to their initial substream so that from design point to design point, the same random numbers are used. For example, if you create two design points that have the exact same start time, end time, number of replicates, and factor values, then the model outputs for the two design points will be identical.
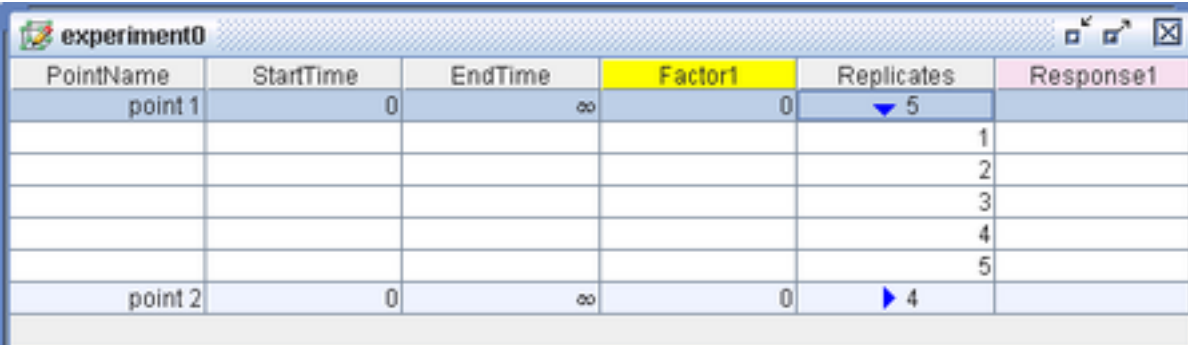
If the number of replicates for a particular design point is greater than one (and you have included responses in your experiment), a small blue triangle precedes the replication value in the design matrix Replicates cell. You click this triangle to expand or collapse the replication rows in the matrix. Figure 5.16 and Figure 5.17 show the different display states.

Figure 5.16 Replicate Rows Collapsed

Figure 5.17 Replicate Rows Expanded

The factor level values are not displayed for each of the replicate rows—only the replication number and any measured response values are displayed. If a design point has replicate rows and it is in a collapsed state, the value displayed for any response value is a summary statistic calculated from all the values collected for that design point for that response. The average value is displayed by default. You can change the summary statistic displayed by right-clicking the appropriate response column heading in the Experiment window and selecting from the statistics available. (See Figure 5.18.)

Figure 5.18 Response Summary Menu
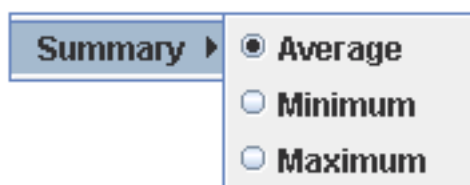
Running an Experiment

After you create and populate the design points in your Experiment window, you are ready to run your simulation model. (You must have identified the active experiment, and it must have at least one valid design point.) You can select one or more design points to automatically run in sequence. You can select a collection of design points either by dragging the mouse pointer over the desired design points or by using the Control or Shift key in combination with clicking to perform standard extended selection of design points. Selected design points are highlighted in the design matrix.

You can start running the selected design points in any of the following ways. You can select the ▶ icon from the toolbar or select **Run▶Start** from the main menu. You can also select the ⏭ icon from the toolbar or select **Run▶Augment** from the main menu. (Augmenting a run is discussed in more detail in the following section.)

When the **Parallel Mode** option on the **Run** menu is not selected, design points are run in the order in which they appear in the design matrix. The currently running design point is indicated by a red font in the design matrix. You can pause or suspend a simulation run by selecting the ⏸ icon from the toolbar or by selecting **Run▶Pause** from the main menu. To restart the simulation execution, simply select the ▶ icon or **Run▶Start** again.

When the **Parallel Mode** option on the **Run** menu is selected, different design points and replications will be distributed on different cores simultaneously. For more information about the **Parallel Mode** option, see Chapter 4, "Simulation Models."

Consider the following when you run an experiment:

- Random number streams (substreams) advance with each replication of a design point but reset when the next design point begins execution.

- Messages can be written to the Log window while the simulation is running. Messages with a SEVERE level will stop model execution.

Augmenting a Run

Suppose you create and run an experiment and then decide that you need additional design points or replicates for individual design points. Simulation Studio provides the augment run option to facilitate this simulation scenario. When you select the ⏭ icon from the toolbar or select **Run▶Augment** from the main menu, Simulation Studio attempts to run any new design points and replication rows in the experiment while preserving results (response values and saved data sets) from the previous run. This unique approach provides you with an option to incrementally expand your design matrix results. Furthermore, the simulated results from augmenting a run will be identical to the results that you would get if you ran the entire experiment again (old and new design points and replicates together). When you use the **Augment** option to run your model, any previously saved data sets in the project results folder will not be deleted.

Saving and Loading Design Data

Options available on the Experiment window pop-up menu (Figure 5.11) enable you to save and load experiment design data. Selecting **Save Design** from this menu opens a File Chooser dialog box, where you select or type the name of the file where you want to save the data. The data in the experiment design matrix are saved to either a SAS data set or a JMP data table, depending on your configuration data settings or the filename extension that is specified. Unlike the data that are displayed in the Experiment window, all individual replicate rows are completely populated with all the factor values along with the response values. Any rows that contain summary information for replication rows are not included in the saved data. The resulting saved data are in a format that can be passed to SAS procedures or JMP routines.

You can also load saved experiment data into a Simulation Studio Experiment window. Selecting **Load Design** from the Experiment window pop-up menu opens a File Chooser dialog box, where you enter the filename for the previously saved data. Simulation Studio attempts to load the data into an Experiment window. Do not use the **Load Design** option to load a data set that was not initially created using the **Save Design** option in Simulation Studio. You should first include all factors and responses in your experiment window and then select the **Save Design** option to create an initial experiment data set to work from. After modifying your saved experiment data set, you can use the **Load Design** option to load it back into Simulation Studio. You can also use the **Save Design** and **Load Design** options to generate experimental designs, as described in Appendix C, "Design of Experiments."

Chapter 6
Blocks

Contents

Overview of Blocks

The block represents the fundamental component of Simulation Studio models. All blocks are either directly or indirectly derived from a base block class. The base block class implements the message handling protocols and defines how blocks communicate with each other through ports. Each block is responsible for creating any ports it needs to perform the functionality required by the block. From a GUI perspective, ports completely define the interface to most blocks. Port management and related port message handling are two vital functions required of all blocks.

For the most part, blocks communicate with other blocks by using links between ports. When using the Simulation Studio GUI, you can create a link from an output port only to an input port of compatible type. If you have multiple links on a given port, logic in the port determines which connections are used and in what order. The default behavior is to use the connections in the order in which they were created.

Block Labels

A default label is generated for each instance of a block. If only one instance of a particular block appears in a model, the block name is used as the default label. Otherwise, an integer is appended to the default label to indicate how many instances of the block have previously been created in the model. The block label is used to identify individual blocks in any lists or dialogs. The label can be displayed next to the block icon in the Model window, but by default, a block label is not visible. The section "Managing Block Properties" on page 64 describes how you can edit a block's label.

Managing Block Properties

The **Block Properties** dialog box contains a description of a block's functionality; it is used to edit block parameters and display properties. You open the **Block Properties** dialog box for a block by double-clicking on the block icon in a model window. Figure 6.1 shows the **Block Properties** dialog box for the Queue block. Blocks that have specific properties or parameters that can be edited have at least one tab for that purpose. For example, you use the **Attributes** tab in Figure 6.1 to edit the Queue block parameters. If a block supports the saving of data (such as the Number Holder block), then the **Block Properties** dialog box contains another tab labeled **Save** where you can specify options that are related to saving data.

Figure 6.1 Queue Block Properties Dialog Box
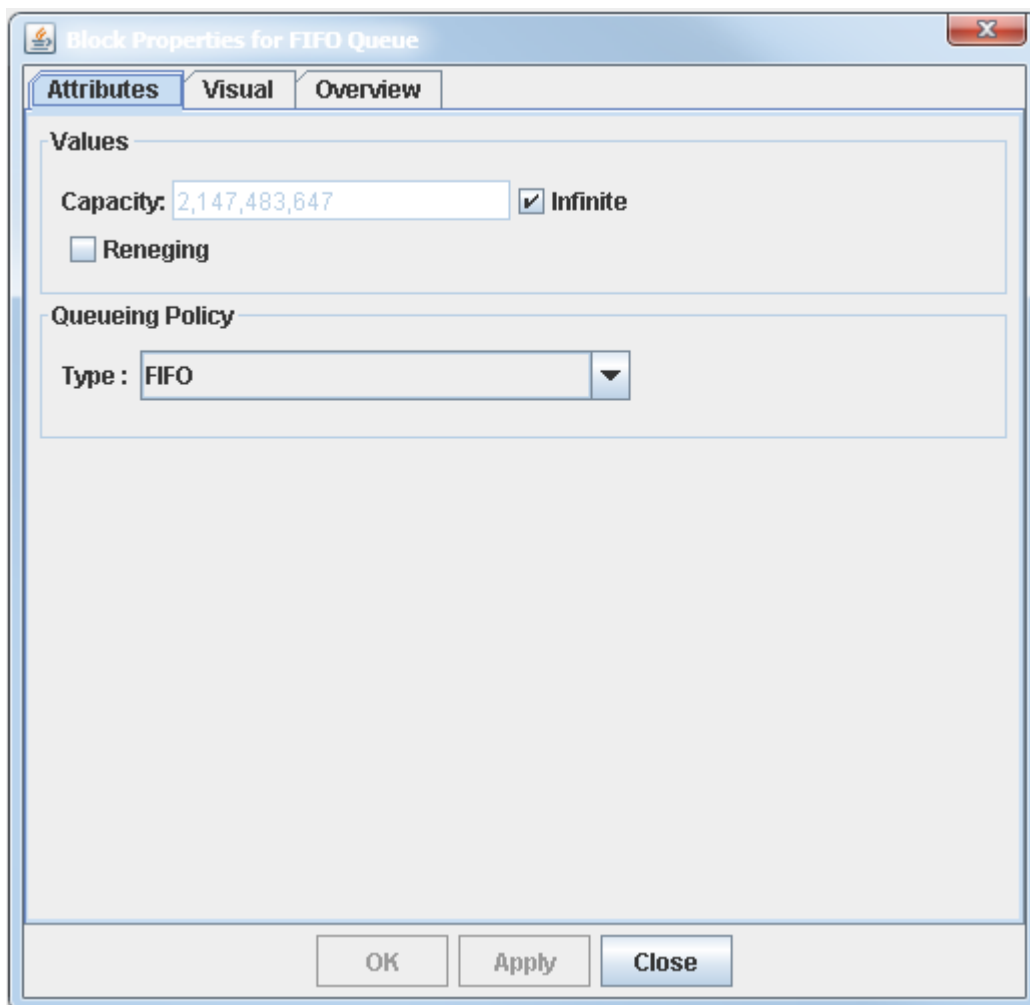
All **Block Properties** dialog boxes have a **Visual** tab and an **Overview** tab. The **Overview** tab provides a description of the block's functionality and includes a description of the block's parameters and ports. The **Visual** tab provides controls to manage the display properties of a block. Figure 6.2 shows the **Visual** tab for

the Queue block. You use the **Block Label** controls to edit the block label (using the **Text** field), to show or hide the label in the model window (using the **Visible** option), and to position the label around the block icon. For example, a position of −90° places the label at the bottom of the icon, and a position of 90° places it at the top of the icon. You use the **Block Image** controls to specify the block's icon image. Each block has a default icon image. If you select the **Custom** option, then you need to specify the file path of the image file.

Figure 6.2 Queue Block Properties Visual Tab
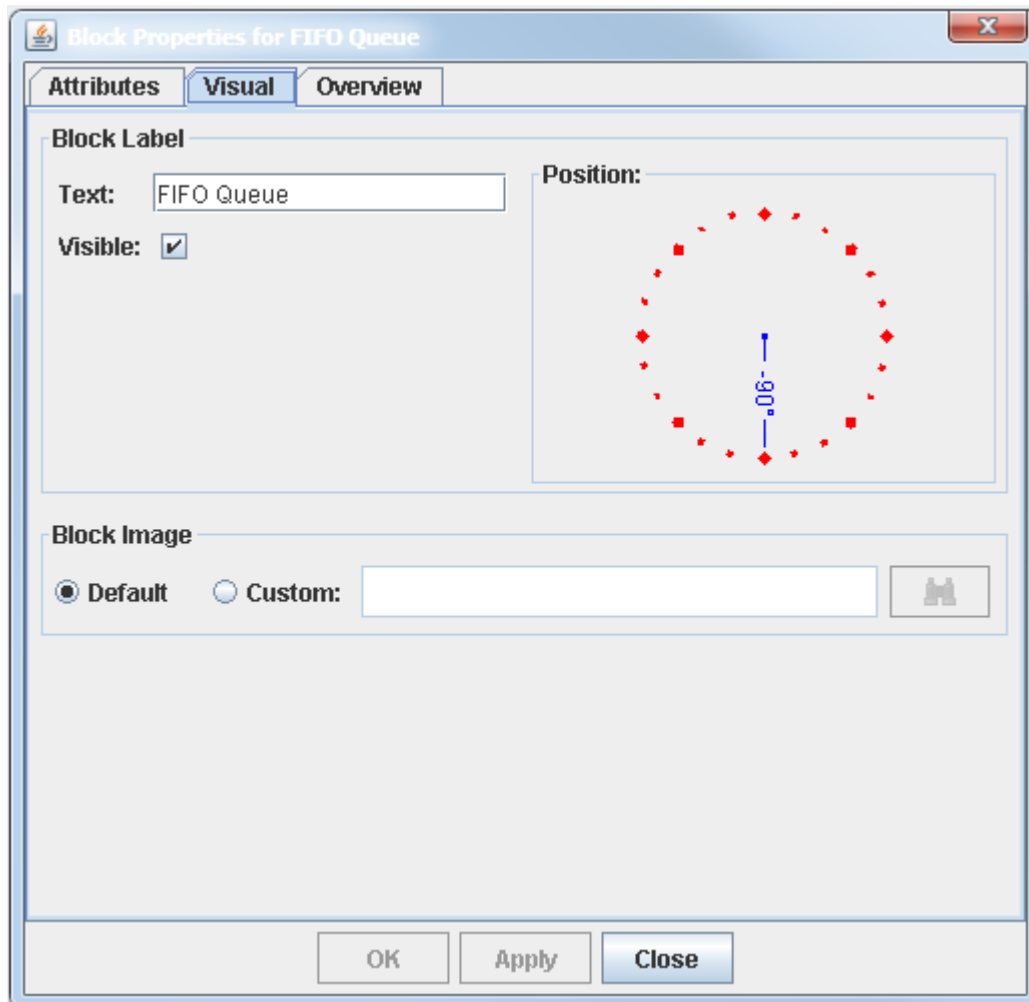
For blocks that have a display component, such as the Number Holder and the Histogram blocks, the **Visual** tab contains the **To Display Component** option, which you can use to show or hide the block's display component. Figure 6.3 shows the **Visual** tab for the Number Holder block with the **To Display Component** option selected.

Figure 6.3 Number Holder Block Properties Visual Tab
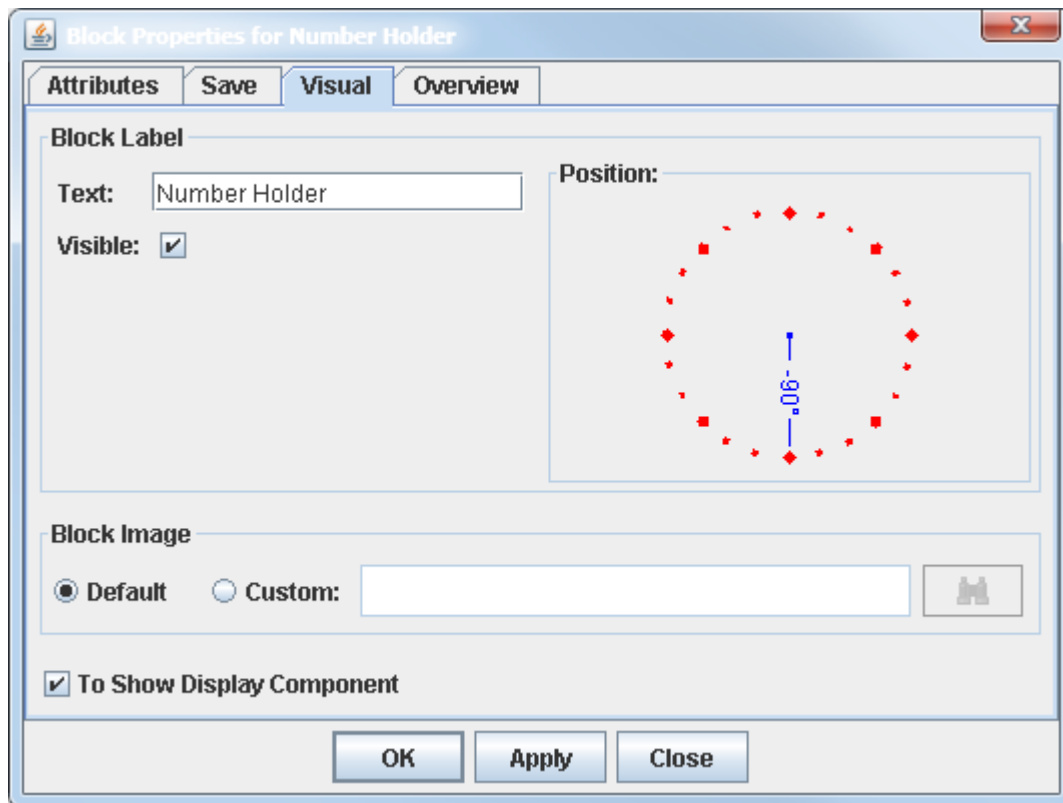
Block Pop-up Menu

Each block has a pop-up menu that includes the options **Block Properties**, **Toggle Label**, **Edit Label**, **Delete**, **Save**, and **Copy**, as shown in Figure 6.4. As an alternative to double-clicking the block icon, you can select the option **Block Properties** from the block pop-up menu to open the **Block Properties** dialog box. As an alternative to the **Block Properties** dialog box's **Visual** tab, you can use the **Toggle Label** and **Edit Label** options to show or hide the block label and to edit and position the label. You can select the **Delete** option to remove the corresponding block from the model, and you can select the **Copy** option to create a copy of the associated block (or compound block) on the clipboard. When the clipboard contains information, a **Paste** menu item is available on the block and model window pop-up menus. Selecting **Paste** copies the clipboard's contents into any model window (when the content is appropriate for a simulation model).

Figure 6.4 Sample Block Menu
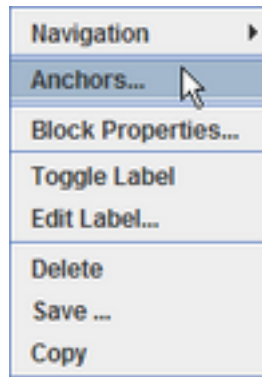
Selecting **Save** from the block pop-up menu opens a dialog box that provides options for saving the instance of the block to disk. This dialog box contains a file selection control for choosing the path and filename for the saved block. By default, blocks are saved into the `Resources\blocks` folder and the default filename extension `.blk` is automatically added to the saved block name for the default filename. A saved block can be imported into a new or existing template. For more information, see Chapter 12, "Block Templates."

The features that are associated with the **Navigation** menu item are useful for viewing and running large-scale simulation models. For more information, see Chapter 4, "Simulation Models."

Anchors

The **Anchor** option on the block pop-up menu is used to define the link between a factor or response that is defined on a project and an actual block parameter or measure in a model. Anchor information is used to associate factors and responses in an Experiment window with simulation models in a particular project at run time. For more information about anchors, factors, and responses, see Chapter 4, "Simulation Models," and Chapter 5, "Experiments."

Block Ranking

Some blocks in Simulation Studio schedule events in the application's event queue. For example, an Entity Generator block schedules when to generate its next entity, and a Server block schedules when service will be completed for an entity that it is holding. Events are placed in the event queue based on the time the event is scheduled to occur. It is possible for different blocks to schedule events that occur at exactly the same simulation time, resulting in an event scheduling tie.

If an event that is put in the event queue has the same scheduled time as an event already in the event queue, the relative order in which those events actually occur is unpredictable, although reproducible. To address this potential event timing issue, some blocks provide a **RankValue** property or factor. The **RankValue** property of a block is an integer value that can be used to resolve ties for events that are scheduled in the

application's event queue. For events that are scheduled to occur at the same simulation time, the event that has a higher rank value has precedence over an event that has a lower rank value. You can set the **RankValue** property of a specific block in your model at run time by including it as a factor in an Experiment window and assigning the desired value in the Experiment window. For information about experiments and factors, see Chapter 5, "Experiments."

The *rankable* blocks, which support a **RankValue** property, include Delay, Entity Generator, Queue, Server, Stopper, and Resource Scheduler. Most blocks have a default rank value of 0. However, the Resource Scheduler block has a default rank value equal to the Java programming language value Integer.MAX_VALUE so that it has the highest priority available.

You can also use the Block Ranking dialog box to set the rank value for a block. You open the Block Ranking dialog box for a model by right-clicking the model name in the project tree and selecting **Block Ranking**, as shown in Figure 6.5.

Figure 6.5 Block Ranking Option on the Model Menu
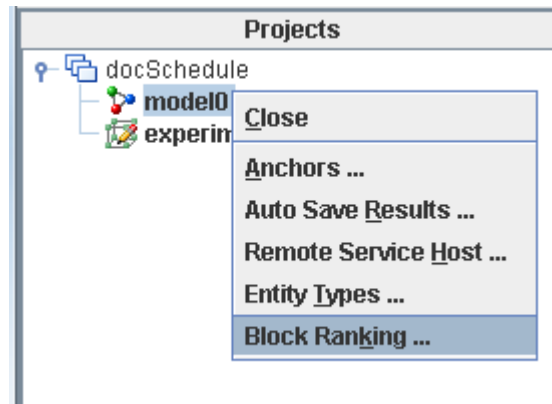
The left side of the Block Ranking dialog box (shown in Figure 6.6) displays a hierarchical check-box tree that includes blocks that have an adjustable rank value in the current model. For each block in the model that has an adjustable rank value, there is a horizontal bar that represents the current relative rank value. Initially, all blocks in the model have the same rank value, unless there is a Resource Scheduler block. By default, a Resource Scheduler block has a higher rank value than any other block in the model. The horizontal bars in the Block Ranking dialog box do not show the exact value of the rank for each block, but rather they show the rank value of a block *relative* to the other blocks in the model.

To change the rank value of a single block, you can drag the end of the bar representing the rank value to the right (to increase the rank) or to the left (to decrease the rank). If you select the check box next to the block name in the check-box tree, the bar representing the rank value is highlighted. You can still drag the bar to set the rank. Also, you can right-click in either the project tree or the area with the horizontal bars to access a pop-up menu with options for the selected block. If you select **Set Rank Value** from the pop-up menu, then the Set Rank Value dialog box is displayed. You can use the **Rank Value** field in the Set Rank Value dialog box to change the current absolute rank value of the selected block. Note that the **Rank Value** field in the Set Rank Value dialog box corresponds to the **RankValue** factor for the block. If you set the **RankValue** factor in the Experiment window for the project, then that value overrides the value in the **Rank Value** field in the Set Rank Value dialog box at run time.

The pop-up menu for a selected block also includes options to lock or unlock the rank value for the selected block. If you select the **Lock** option, then the rank value for the selected block is locked and cannot be adjusted. The color of the horizontal bar for the block changes from blue to red. You use the **Unlock** option to enable editing of the rank value for the selected block. You can select multiple blocks in the check-box tree and lock or unlock the rank value for all selected blocks simultaneously. The **Unlocked** and **Locked** sections at the top of the Block Ranking dialog box display tick marks for the rank values of all selected blocks. You can also use the **Set Rank Value** option to set the rank value for all selected blocks simultaneously.

Figure 6.6 shows the Block Ranking dialog box for the `docSchedule` project in the `examples` directory under your Simulation Studio installation directory (see the section "Extending the Banking System Model" on page 100 for a description of the `docSchedule` example). In this example, three blocks are selected. The **Unlocked** and **Locked** sections show the rank values of the selected blocks. The blocks that are labeled Arriving Customers and Create Teller have their rank values locked. The Resource Scheduler block has the highest rank value; the other blocks have the same rank value.

Figure 6.6 Block Ranking Dialog Box
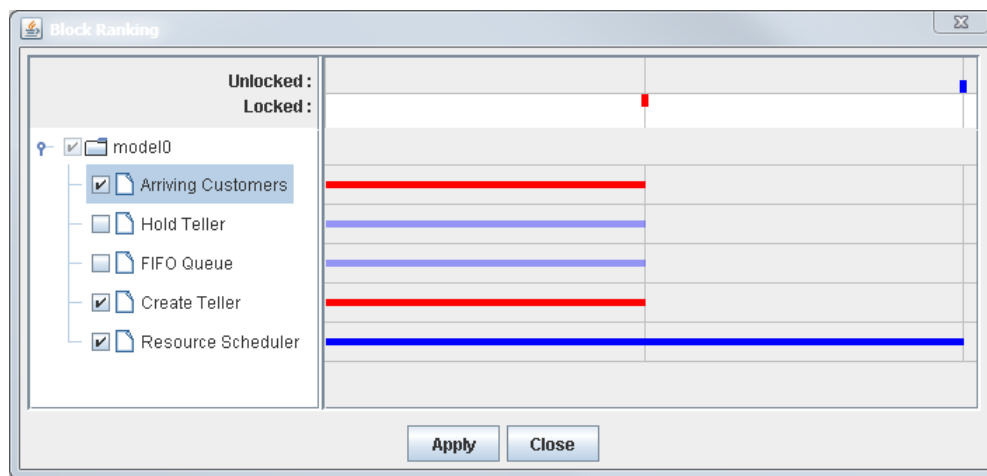
Chapter 7
Compound and Submodel Blocks

Contents

Overview of Compound and Submodel Blocks

Simulation models can become very large, possibly incorporating hundreds of blocks. Simulation Studio enables you to assemble blocks into larger aggregates by using *compound* and *submodel* blocks. These blocks encourage hierarchical model building and information hiding, and they facilitate component reuse. For very large and complicated models, compound and submodel blocks can also greatly reduce the visual complexity of the model.

Assembling and Disassembling a Compound Block

Figure 7.1 shows a small model that contains a compound block which encapsulates a Numeric Source and Server block. Figure 7.2 displays the same model, this time with the compound block collapsed.

Figure 7.1 Sample Compound Block
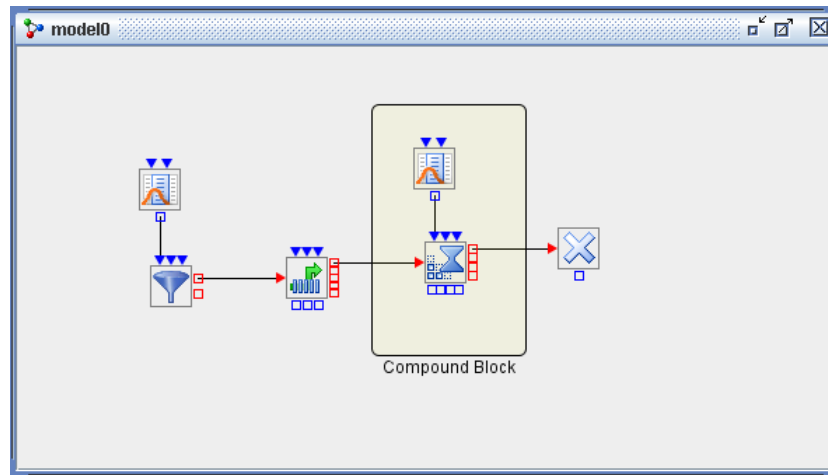
Figure 7.2 Sample Compound Block—Collapsed
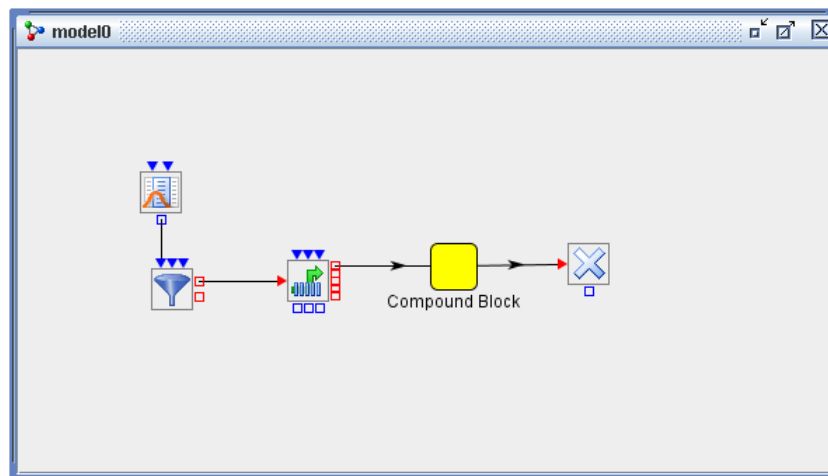
To create or assemble blocks into a compound block:

1. With the pointer positioned in the appropriate Model window, hold down the Shift + Ctrl keys while simultaneously holding down the left mouse button and sweep out a rectangular area in the Model window encompassing the blocks you want to assemble into a compound block. When you release the mouse button and Shift + Ctrl keys, a red rectangle appears in the Model window replacing the sweep rectangle. All the blocks entirely within the rectangle also have a red highlight or selection box around them.

2. Right-click within this red rectangle, and select **Assemble Compound**. Selecting this menu item creates a new compound block that contains all the highlighted blocks.

To disassemble a compound block, right-click a compound block and select **Disassemble Compound** from the resulting pop-up menu.

Collapsing and Expanding a Compound Block

To hide the contents of a compound block, double-click the compound block. This causes the visual representation of the compound block to be replaced with a small yellow square similar in size to a basic block icon. This is sometimes referred to as a collapsed compound block.

To expand a collapsed compound block to its original size, double-click the collapsed compound block.

Labeling and Saving a Compound Block

As with basic blocks, you can edit the label associated with a compound block and switch between displaying and hiding it. It is often useful to give a compound block a descriptive, concise label that reflects its functionality or usage. Because the contents of a compound block are usually not visible, a good label can be particularly helpful for understanding its functionality in a simulation model.

The procedure for saving a compound block to disk and also for adding it to a template for reuse is identical to the procedure described in Chapter 6, "Blocks," for basic blocks. The default filename extension for a saved compound block is `.cblk`, and the default save folder is `Resources\block`.

To improve security, models that you create using version 14.3 and later are saved in a different format than models that you create using previous versions of Simulation Studio. To save a compound block created in version 14.3 so that it is compatible with previous versions of Simulation Studio, right-click on the compound block and select **Export As**.

Tunnels

Usually when you create a compound block you have some specific functionality you want your group of blocks to perform, and you connect them in a very specific manner. You also know which blocks require inputs from outside of the compound block and which blocks pass output from the compound block. That is, you know which ports you are going to use as input ports and which ports you are going to use as output ports for the compound block to make it function as designed. Additional connections could potentially alter the desired functionality of your compound block in unpredictable ways.

Simulation Studio provides a feature called a *tunnel* to facilitate input and output for a compound block. You can think of tunnels as special ports for a compound block. Blocks (and their ports) located outside the compound block that need to send information to the compound block connect to input tunnels defined on the compound block. Similarly, blocks expecting information from the compound block connect to its output tunnels. Blocks internal to the compound block that need connections outside the compound block also connect through tunnels. You add or remove tunnels for a compound block using the Add/Remove Tunnels dialog box (Figure 7.3) available through the pop-up menu on a compound block. This dialog box provides options for creating various types of ports on a compound block. You can also edit the default name given to a tunnel to something more meaningful for your compound block. The name cannot contain spaces.

Figure 7.3 Tunnels Dialog Box
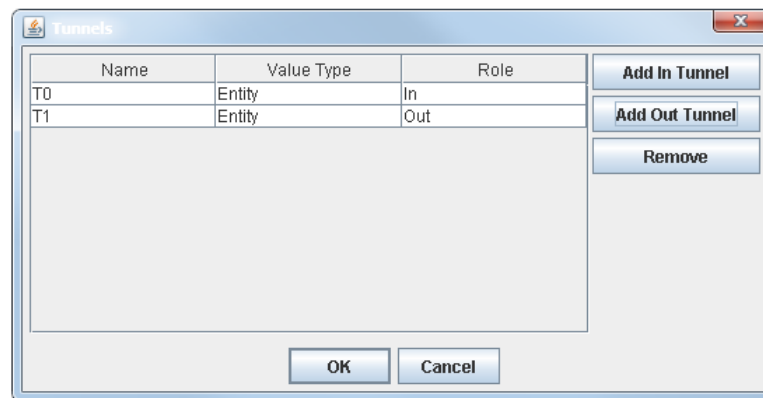
Figure 7.4 depicts a compound block with an input entity tunnel, an output entity tunnel, and an input value tunnel. The placement of the various tunnel types around the compound block is automatic and is analogous to that of ports on basic blocks. The graphic used to depict a tunnel is a combination of the graphics used for an input port and output port because the tunnel serves both functions depending on whether your perspective is from inside the compound block or outside it.

Figure 7.4 Compound Block with Tunnels
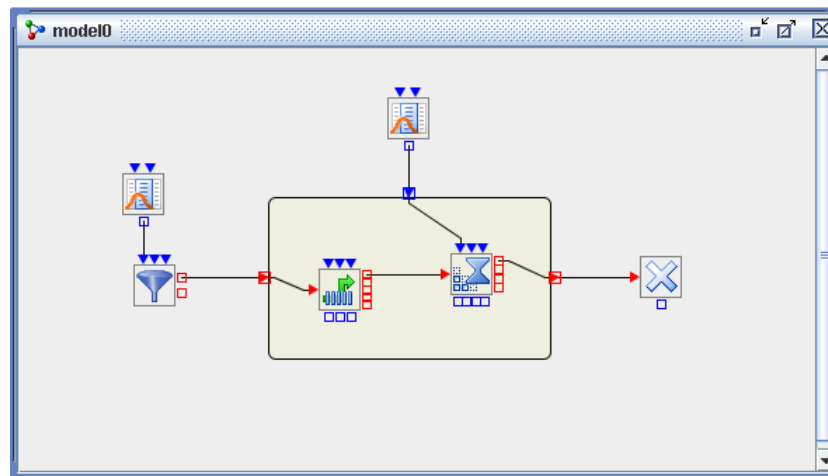
If you collapse the compound block, as shown in Figure 7.5, the compound block appears much the same as a basic block. A compound block can be treated essentially as a "black box" in terms of functionality.

Figure 7.5 Collapsed Compound Block with Tunnels
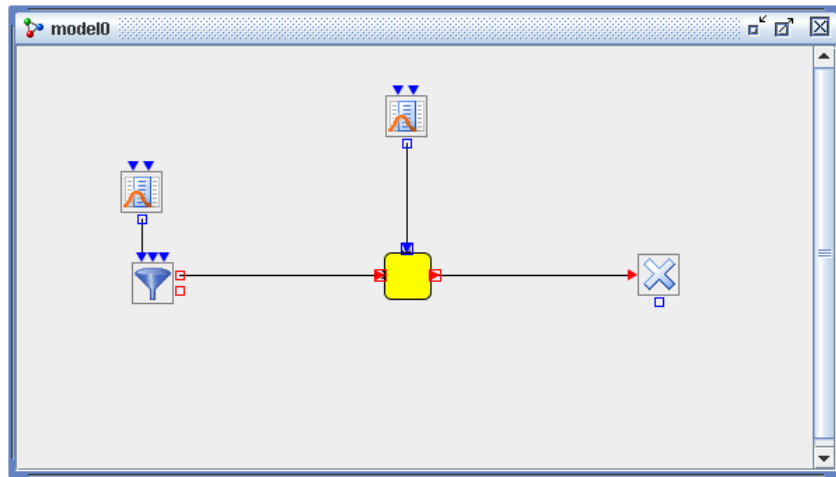
If a compound block is disassembled, the tunnels turn into Connector blocks so that no functionality is lost in the simulation model. Figure 7.6 shows the result of disassembling the compound block in Figure 7.5.

Figure 7.6 Result of Disassembling a Compound Block with Tunnels
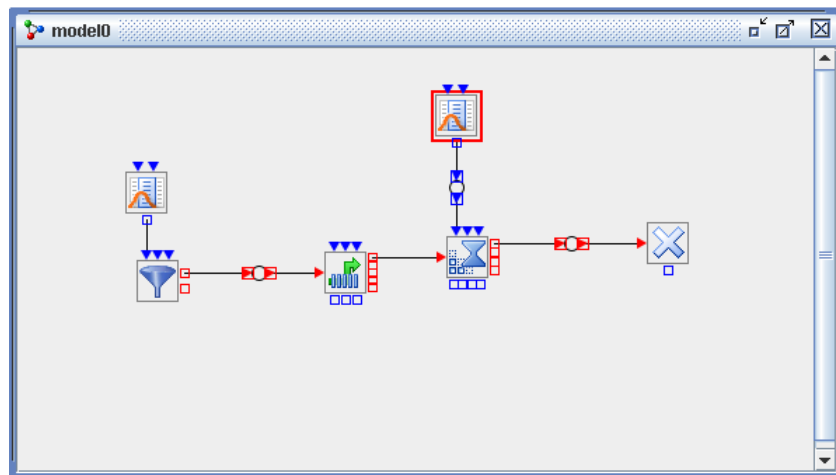
Submodel Blocks

Like a compound block, a submodel block can be used for hierarchical simulation modeling and for facilitating component reuse. The key difference between a submodel block and a compound block is that the contents of a compound block are embedded in the model when it is created. Although a compound block can be saved and reused in the same model or different models, each instance of the compound block must be edited separately if changes need to be made. All instances of a compound block, whether in the same model or different models, are independent from each other. On the other hand, a submodel block provides a linkage to

its contents from the simulation model. The definition of the submodel contents can be stored as a compound block file (with file extension *.cblk*) and modified independently. When the content definition is changed, all instances of the submodel, whether in the same model or different models, contain the changes.

Creating a Submodel

There are two ways to create a submodel block in a model:

- You can convert an existing compound block to a submodel block by selecting **Convert to Submodel** from the compound block pop-up menu (available by right-clicking on a compound block), as shown in Figure 7.7. If the compound block to be converted does not have tunnels defined, then all input and output connections to the compound block are converted into tunnels automatically after the submodel is created.

Figure 7.7 Converting a Compound Block to a Submodel Block
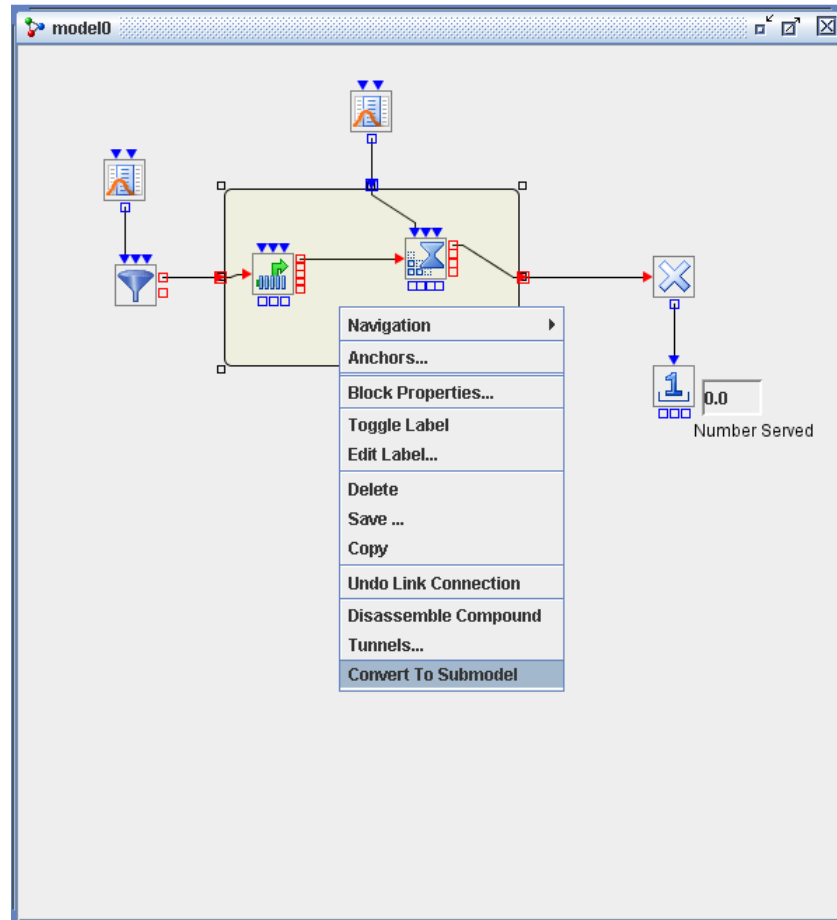
- You can drag the submodel block icon from the Advanced template and drop it into a Model window. Then you right-click on the submodel block to open its Block Properties dialog box and specify the path of a previously saved compound block (*.cblk* file), as shown in Figure 7.8.

In the submodel Block Properties dialog box, you select the **Visual** tab to change the submodel block label, to make the label visible in the model, to change the position of the label around the block icon, and to select a custom image for the submodel block icon. As with basic blocks, you can also right-click on a submodel block and select **Edit Label** to edit the submodel label and position or select **Toggle Label** to toggle between displaying and hiding the label.

Figure 7.8 Submodel Block Properties Dialog Box
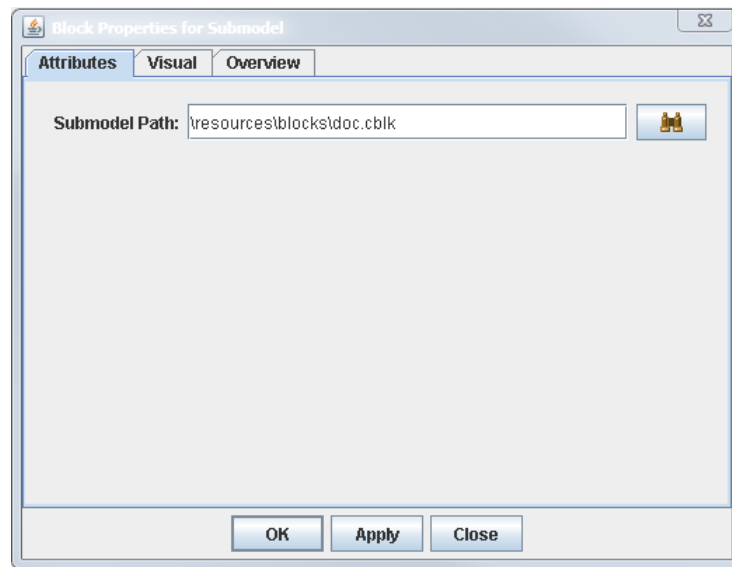
Viewing and Editing a Submodel

You double-click a submodel block to view its contents in a separate submodel window. Figure 7.9 shows a model with one submodel block and the submodel window open.

Figure 7.9 A Submodel Block and the Submodel Window
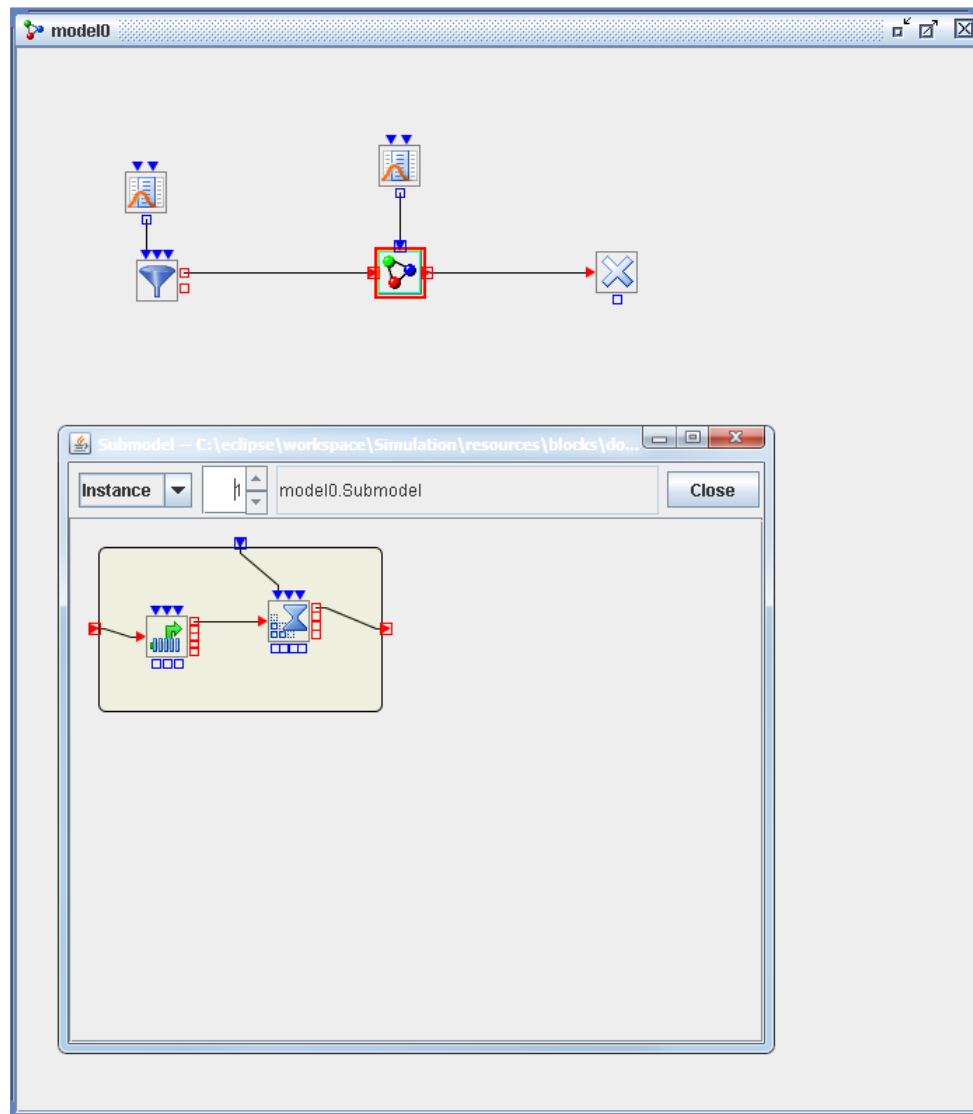
The Submodel window opens with the **Instance** option selected in the Submodel window combo box. The **Instance** option in the submodel window enables you to view (but not edit) the contents of a particular submodel instance. The **Instance** option is especially useful for viewing the animation of a submodel instance as the model runs, and it alleviates the need to open a separate window for each submodel instance. If you have a submodel window open and you double-click on another submodel block that is linked to the same submodel that is already open, then both instances are viewed in the same submodel window. Clicking the up and down arrows adjacent to the **Instance** option changes the submodel instance being viewed. The label that is associated with each submodel instance is displayed.

NOTE: The submodel instance number reflects the order in which the submodel instances were opened. If you close a submodel instance and then open it again, it could have a different instance number.

You open the **Definition** view in a Submodel window by clicking the down arrow adjacent to the **Instance** option and selecting **Definition**. Figure 7.10 shows a Submodel window in **Definition** mode. The **Definition** option in the submodel window enables you to edit the contents of a submodel. When you edit a submodel and save the changes, all instances of the submodel in the current model and in any other model reflect the changes. You can drag blocks into the submodel, connect blocks, delete blocks, and perform any other modeling action that you would perform in the regular Model window. You can drag blocks into a Submodel window from the template palette in the SAS Simulation Studio window. Alternatively, the **Definition** view in a Submodel window has a template palette that appears when you place the mouse pointer over the template image ▯. This template behaves in the same way as the template in the SAS Simulation Studio window. When you move the mouse pointer away from the ▯ icon, the Submodel window template disappears.

NOTE: In the Submodel window, new blocks can be dropped only within the submodel border, and this border might be smaller initially than the size of the Submodel window. In **Definition** mode, you can expand the border of the submodel by clicking anywhere in the submodel background and then dragging one of the submodel border corners to the desired size.

NOTE: If a submodel contains any blocks that use a random stream (such as the Numeric Source block), then every instance of that submodel uses the same random streams. If you want each submodel to use different random streams, then you could design your submodel so that the blocks that use a random stream are external to the submodel. Then any random values are input to the submodel through a tunnel. You also might consider using a compound block instead of a submodel block because each instance of a compound block in a particular model automatically contains different streams.

To save an edited submodel block, click **Save** in the top right of the Submodel window or right-click on the submodel and select **Save**. When you close the Submodel window, you will also be prompted to save any changes. The default filename extension for a saved submodel block is *.cblk*. When the edited submodel definition is saved, all instances of that submodel in the currently opened simulation model automatically refresh to reflect the new, updated definition. Furthermore, all instances of the submodel that are in other models (either in the same project or in different projects) are also updated.

To improve security, models and submodels that you create using version 14.3 and later are saved in a different format than models and submodels that you create using previous versions of Simulation Studio. To save a submodel created in version 14.3 so that it is compatible with previous versions of Simulation Studio, right-click on the submodel and select **Export As**. Furthermore, if you open a project or model in version 14.3 that contains a submodel that was created in a previous version of Simulation Studio, then you will see the message shown in Figure 7.11 (a separate warning message will appear for each submodel). After you save the old submodel in version 14.3, this warning message will no longer appear when you open the model.

Figure 7.10 Submodel Window in Definition Mode
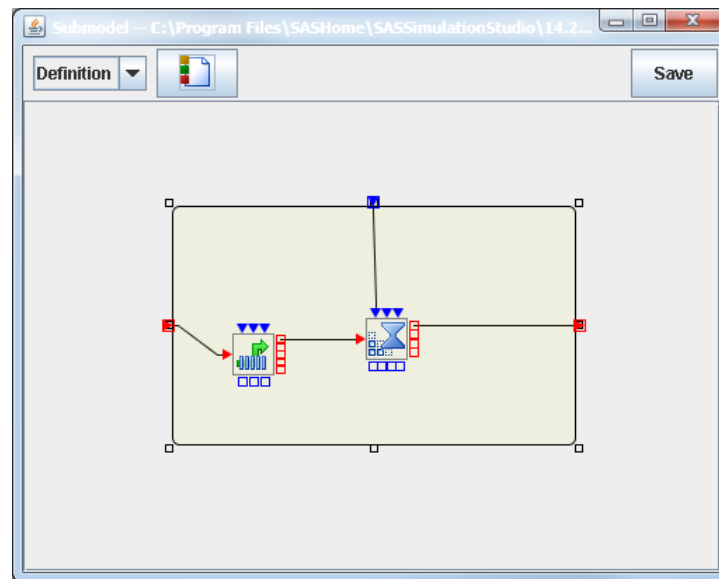
Figure 7.11 Warning Message for Old Submodel Format

Chapter 8
Entities

Contents

Overview of Entities

Entities are discrete objects that can traverse a simulation model or network. They can be used to represent physical or conceptual items in your model such as cars in a traffic model, telephone calls in a telecommunications system, customers in a retail environment, and so on. You can use various Simulation Studio blocks to assign attributes to entities as they flow through your simulation model and use other blocks to read entity attributes and act on them.

When you have the animation feature turned on in Simulation Studio, you can view the movement of entities through your simulation model while it is executing. If the Tracer is enabled, entity information is also displayed on the **Trace** tab in Simulation Studio.

Entity Types

All of the examples in this document so far have used the default regular entity type. If you do not specify an EntityType in the Entity Generator block, then the default regular entity type is used.

The Entity Types dialog box shown in Figure 8.1 enables you to add attributes to the default entity types, and it also enables you to create new entity types. Entity types can be defined at the model level. To open the Entity Types dialog box, right-click the model name in the Project Explorer and then select **Entity Types**.

Figure 8.1 Entity Types Dialog Box
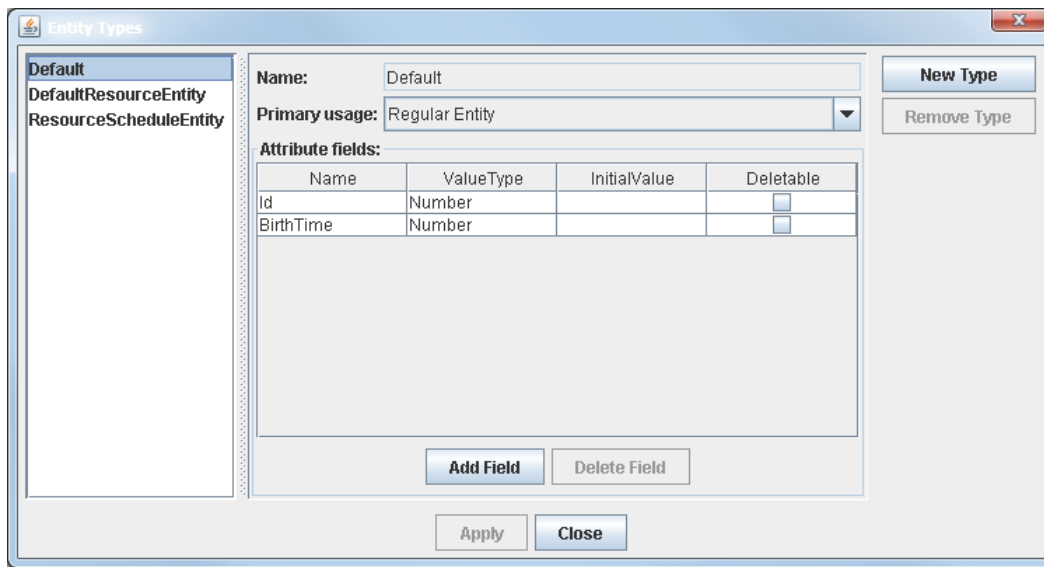
User-created entity types can be either *regular* entities, *resource* entities, or *resource schedule* entities. You can process all three types of entities by using any of the blocks that are provided in the basic modeling template. However, resource entities also have capabilities that are used by the blocks that are provided in the resource modeling template. A resource schedule entity is a special type of resource entity.

Unless otherwise specified, the term *entity* refers generically to either a regular entity, a resource entity, or a resource schedule entity. For more information about resource and resource schedule entities, see Chapter 10, "Resources."

Creating Entities

The default basic modeling template that Simulation Studio provides contains two blocks capable of generating entities: the ▼ Entity Generator block and the ⊡ Clone block. These blocks are described in Appendix A, "Templates." Entity Generator blocks are usually the primary source of entities in your simulation models. You can specify the quantity and type of entities that you want your Entity Generator blocks to generate. If more than one entity is being generated, you need to have a connection to the Entity Generator block's InterArrivalTime input port so the Entity Generator can determine when to schedule the next entity creation event. You can create multiple entities at each entity creation event by using the BatchSize input port on the Entity Generator block. (See the Entity Generator details in Appendix A, "Templates.") There is no limit on the number of Entity Generator blocks that your simulation model can contain, and the limit on the number of entities of a given type that you can generate is the Java programming language value Integer.MAX_VALUE.

Disposing of Entities

Although it is possible to generate an almost endless number of entities, each entity has associated memory costs. It is important to let Simulation Studio know when you are finished with an entity so that it can efficiently manage any memory issues related to entities. In a simulation model, you indicate you are finished using an entity by routing it to a Disposer block ✂. When an entity enters a Disposer block, it is marked as *free* and reduces memory allocation costs at run time in the application.

Entity Attributes

Entities can have attributes associated with them. Attribute names must be unique, are case sensitive, and cannot contain spaces or blanks. Attribute values can be strings, numbers, Booleans, or Java objects.

Each entity has two default attributes named Id and BirthTime. The Id value is a unique integer assigned (in sequence) when the entity is created, and the BirthTime value is the simulation clock time when the entity was created. You can add additional default attributes (along with their default values) to an entity type by using the Entity Types dialog box at design time. (See Figure 8.1.) Each new entity created of that type automatically has all the attributes defined on that entity type as shown in the Entity Types dialog box.

The ⬚ Modifier block in your simulation model can also be used to add or modify entity attributes at simulation time. Other blocks are provided in Simulation Studio to read entity attribute values and use this information in their processing. Individual blocks are discussed in Appendix A, "Templates."

As a simple example of using entity attributes, consider the scenario where you want to model an electronics repair shop. Your entities could represent customers coming into the repair shop. You could assign attributes to each of these customers to represent *(i)* what type of equipment the customer needs to have repaired; *(ii)* an indicator of the severity of the problem; and *(iii)* warranty information. You could then use these attributes to route customer entities to different technicians in the shop depending on the values of the attributes. You could use the severity attribute to calculate a time-to-repair value.

Attributes are intended to give your entities unique characteristics that you can use to make your simulation model more representative of the system that you are investigating.

Entity Groups

Simulation Studio implements an object named *entity group* that is a collection of *entity references*. An *entity reference* contains information that uniquely identifies a particular entity. Therefore, an entity group holds information about a collection of entities, but not the actual entities themselves. Entity groups add another level of modeling sophistication to your simulation modeling environment.

Most holding-type blocks, such as the Queue, Server, and Delay blocks, provide an OutHoldings output port that you can use to pull an entity group that contains references to the entities currently held by the block. Other Simulation Studio blocks can use this information to inspect the contents of a holding block and then act on that information, possibly preempting specific entities from the holding block. Some blocks (the

Queue and Server blocks, for example) have an InPreempt input port that accepts an entity group as input. These blocks compare the entity references in the incoming entity group to the entities currently being held by the block and preempt any matches.

The Entity Group Holder block can be used to create a new entity group (with characteristics specific to your model needs) that can then be used by other blocks in your model. The Gate and Seize blocks also make use of entity groups. See Appendix A, "Templates," for details about these and other blocks that use entity group objects.

Chapter 9
Queueing

Contents

Overview

The Queue block is one of the fundamental components of SAS Simulation Studio models. It is used for the transient storage of entities, and you have extensive options for controlling how many entities can be stored in the queue and the order in which entities are stored, as well as options for reneging, preemption, and nonblocking. Figure 9.1 shows the Block Properties dialog box with default settings. In this chapter, the various options available in the Queue block are described and examples are presented.

Figure 9.1 Block Properties Dialog Box with Default Settings
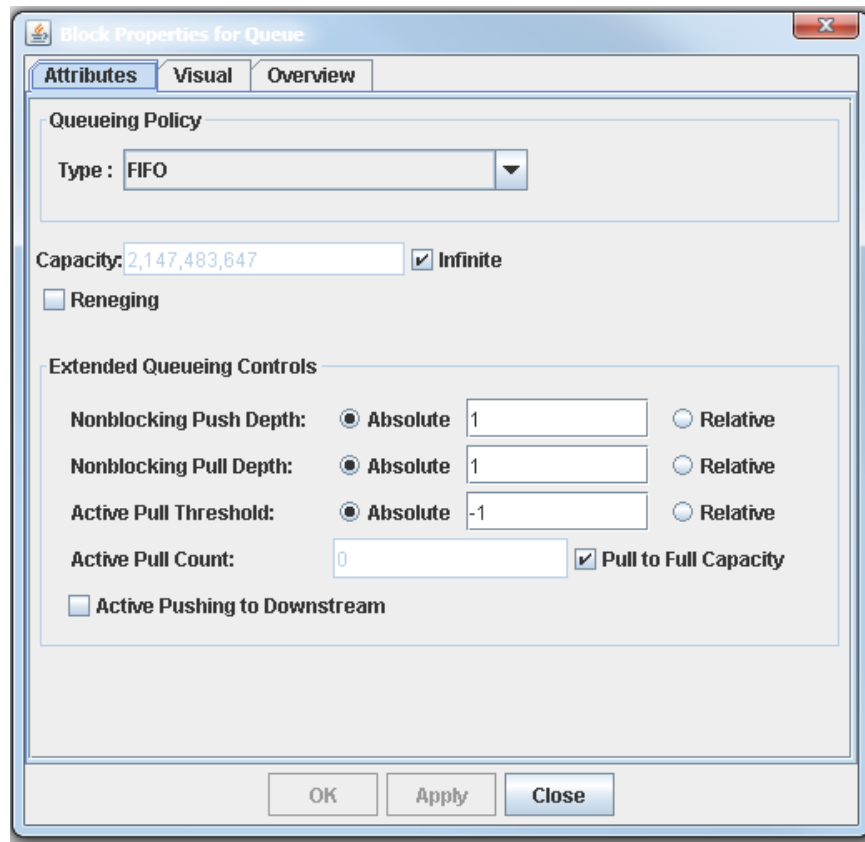
Queueing Policies

The **Queueing Policy** option in the Block Properties dialog box is used to define the order in which entities are stored in a Queue block. There are three queueing policies available in the Queue block: FIFO (first-in, first-out; the default), LIFO (last-in, first-out), and priority. When an entity arrives at a Queue block that is using a FIFO or LIFO queueing policy, the entity is stored at the appropriate end of the buffer. For Queue blocks that use a priority policy, the block extracts the priority value from an attribute defined for the entity and uses that value to determine where to place the entity in the buffer. The priority queueing policy also enables you to select an appropriate tie-breaking policy: FIFO, LIFO, or random.

In addition to setting the queueing policy by using the Block Properties dialog box, you can define a factor that is anchored to the QueueingPolicy anchor candidate for the Queue block and then set the queueing policy by using the Experiment window. For more information about the format of the Queue block QueueingPolicy factor candidate, see the section "Queue Block" on page 141. For more information about defining factors and anchors, see Chapter 5, "Experiments."

Capacity

By default, the capacity of a Queue block (the maximum number of entities that it is permitted to store in its buffer) is infinity. You can set the capacity of a Queue block by using the **Capacity** field in the Block Properties dialog box, or you can dynamically set the capacity by using the InCapacity port on the Queue block. Valid incoming values for the InCapacity port are integers greater than or equal to zero. If the capacity of a Queue block is reduced dynamically during the simulation run, any excess entities are removed from the block (according to the selected queueing policy) and sent out through the OutBalk port. If there are no connections to the OutBalk port, the entities are destroyed.

If you dynamically change the capacity of a Queue block by using the InCapacity port, then you need to reset the capacity to the initial starting value at the start of the next run. Otherwise, the next run of the model will use the final capacity from the previous run. When dynamically changing the Queue block capacity during a run, you can ensure that the capacity is set to the correct initial starting value for each run by defining a factor for the Queue block capacity. You can then specify the initial starting capacity in the Experiment window so that the initial Queue block capacity is set at the start of the run. For more information about defining factors and anchors, see Chapter 5, "Experiments."

Extended Queueing Controls

In addition to the queueing policy option, there are other controls that you can use to adjust how entities flow into and out of a Queue block. The **Extended Queueing Controls** section of the Block Properties dialog box has the options **Nonblocking Push Depth**, **Nonblocking Pull Depth**, **Active Pull Threshold**, **Active Pull Count**, and **Active Pushing to Downstream**. These extended controls are mostly used to prevent a Queue block from being blocked.

There are two ways that entities can flow out of a Queue block. First, the block can actively push entities downstream when possible, where by default the block will attempt to push only one entity at a time. Second, other blocks can pull entities out of the Queue block. By default, the specified queueing policy determines which entity in the buffer to push or pull out of a Queue block. However, you can use the extended queueing controls to indicate whether other entities in the buffer should be pushed or pulled out of the Queue block in the case where the first entity in the buffer is unable to flow downstream. The order in which additional entities are examined for pushing or pulling out of the Queue block is dictated by the specified queueing policy.

To demonstrate how these extended queueing controls are used to prevent a blocked queue, consider the following example. Suppose patients are assigned a hospital bed on a first-come, first-served basis. Each patient has certain bed types that he or she can be assigned, depending on which department the patient is associated with (for example, maternity, orthopedics, or cardiology). If you create a simulation model to study the effects of different bed assignment policies on patient bed-waiting times, it makes sense to use a Queue block to hold patient entities that are waiting for a bed. The following sections describe how changing the default **Nonblocking Push Depth** and **Nonblocking Pull Depth** options can prevent a blocked queue. For more examples on the extended queueing controls, see the section "Extended Queueing Options" on page 334.

The Nonblocking Push Depth Option

Suppose a patient is waiting at the front of the queue because none of the required bed types are available. When a new patient entity arrives at the queue, the Queue block attempts to push an entity downstream (the arrival of a new entity at a queue signals the Queue block to attempt to push an entity out). If the default absolute value of 1 is used for the **Nonblocking Push Depth** option, then only the first entity in the queue will be checked to see whether it can be pushed downstream. In the hospital bed assignment case, the first entity cannot be pushed out of the queue because its required bed is not available, so it continues to wait in the queue. Furthermore, the new patient entity that just arrived waits behind the first entity, regardless of whether one of its bed types is available. That is, the newly arriving patient is blocked by the patient waiting at the front of the queue.

If you set the **Nonblocking Push Depth** option in the Queue block to a relative value of 1, then up to 100% of the capacity is checked when the Queue block attempts to push an entity out. In the hospital bed assignment case, this means that when a new patient entity arrives at the queue, the Queue block attempts to push the first entity out, as before. Because the first entity's required bed is not available, it will continue to wait at the front of the queue. The Queue block then attempts to push out the second entity (the newly arriving patient). If the second entity is able to flow downstream (that is, if its bed requirement can be fulfilled), then the Queue block pushes it out. In this case, the newly arriving patient entity is not blocked by the patient entity that is waiting for its bed.

The Nonblocking Pull Depth Option

In the hospital bed example, a waiting patient can also exit the Queue block when a bed becomes available and another downstream block (such as a Seize block) attempts to pull from the Queue block. If the **Nonblocking Pull Depth** option is set to the default absolute value of 1, then only the first entity in the Queue block is checked when a block attempts to pull an entity from the Queue block. If the newly available bed matches the requirement of the first entity waiting in the queue, then that patient entity will flow out. If it does not match the requirement, then no entities are pulled from the Queue block, even if the second patient entity waiting is a match for the newly available bed. In this case, the queue is blocked.

If you set the **Nonblocking Pull Depth** option in the Queue block to a relative value of 1, then up to 100% of the queue capacity is checked when another block attempts to pull an entity out of the Queue block. In the hospital bed assignment case, this means that when another block attempts to pull an entity from the Queue block, each waiting patient entity is checked to see whether it can flow downstream. The order in which waiting patient entities are checked is dictated by the queueing policy.

The Active Pushing and Active Pulling Options

By default, the Queue block pushes only one entity out at a time, even if it is possible for other waiting entities to flow downstream. If you select the **Active Pushing to Downstream** option in the Block Properties dialog box, then when the Queue block is able to push out the first waiting entity, it pushes out the first entity and then attempts to push out as many entities as it can after that.

The **Active Pull Threshold**, **Active Pull Count**, and **Pull to Full Capacity** options control when a Queue block attempts to pull entities from upstream and how many entities to pull. It would be inefficient to enable

the Queue block to continuously check to see whether it can pull entities from upstream, so you can determine at what point the Queue block should pull from upstream. By default, the Queue block attempts to pull when its length falls to Capacity – 1 (that is, a Queue block with fixed capacity pulls as soon as it has one space available). For example, if you set the **Active Pull Threshold** option to an absolute value of 3, then the Queue block will not pull from upstream until it reaches a length of three entities. Furthermore, if you set the **Active Pull Threshold** option to an absolute value of –5, then the Queue block will not pull from upstream until it reaches a length of (Capacity – 5).

You use the **Active Pull Count** box to specify the maximum number of entities that the Queue block will pull. If the **Pull to Full Capacity** option is selected, then the Queue block pulls as many entities as it can until the queue reaches its maximum capacity.

Reneging

In some applications, you might want to model a scenario where an entity enters a queue but is willing to wait there for only a certain amount of time. If that time is reached and the entity is still in the queue, then it leaves the queue. For example, a customer arriving at a bank on his lunch hour might be willing to wait in line for only 30 minutes. After that, he will leave the queue and exit the bank. The term *reneging* is used for this type of behavior.

In Simulation Studio, you can enable reneging in a Queue block by selecting the **Reneging** option in the Block Properties dialog box. When an entity enters the Queue block and the **Reneging** option is selected, the Queue block attempts to pull a numeric value from its InRenegeWait port. If the Queue block pulls a nonnegative number from the port, it will schedule a time for the entity to exit the block via the OutRenege port. When the renege time is reached, the Queue block sends the entity out the OutRenege port if the entity is still in the block and if the OutRenege port has a connection to it. If there is no connection to the OutRenege port, then no reneging occurs.

Preempting

Entities in Simulation Studio are hierarchical. That is, entities can hold other entities. The term *controlling entity* denotes an entity that holds other entities, and the term *root entity* denotes an entity that is not held by another entity. Each entity that is held by another entity has one root entity associated with it. The root entity for any held entity is found by moving up the entity hierarchy from the held entity.

Entities that are being held by a Queue block can be preempted either by input to the block's InPreempt port or by a scheduled resource entity event. In order for a root entity that is held by a Queue block to be preempted, the OutPreempt port (or OutBalk port) must have at least one link attached to it. Similarly, for a resource entity that is held by a controlling entity that is in turn held by the Queue block to be preempted, the OutResource port (or OutBalk port) must have at least one link connected to it.

The Queue block's InPreempt port accepts an Entity Group object as input. (An Entity Group is a collection of entity references.) When an Entity Group object is pushed to a Queue block's InPreempt port, the Queue block iterates through the Entity Group collection, looking for matches to root entities held by the Queue block. For any matched entity, the Queue block first tries to push that entity out its OutPreempt port. If this

push is not successful, the block attempts to push the entity out the OutBalk port. If this also fails, the entity continues to be held by the Queue block until it either exits the OutEntity port or is preempted again.

The Queue block, like all entity holding blocks, detects potential preemptive changes (such as those scheduled by a Resource Scheduler block) to resource entities that it holds (either directly or indirectly through a controlling entity). If the number of units that are associated with a held resource entity decreases or the state of a held resource entity becomes nonfunctional, the Queue block attempts to preempt that resource entity. If the resource entity that is identified for preemption is a root entity, then the Queue block follows the same protocol for pushing an entity out its OutPreempt port that the InPreempt port uses. If the resource entity is part of a controlling entity, the Queue block removes the resource entity from the controlling entity and attempts to push the associated root entity out the OutPreempt port. The Queue block then attempts to push the preempted resource entity out its OutResource port, or if that fails, out its OutBalk port. If there is a connection to the Queue block's OutResource port and the block cannot push the resource entity out either the OutResource or OutBalk port, the resource entity is destroyed.

The Queue block also provides an OutHoldings port that other blocks can use to pull an Entity Group object that contains a collection of references to entities held by the Queue block.

Chapter 10
Resources

Contents

Overview of Resources

In Simulation Studio, resources are special objects that provide services or materials to entities. In some models, the available resources are unlimited; in other models, the number of units of a resource is limited. You can vary the number of available resource units during a simulation run by using a resource schedule. The availability of resources can affect the flow of entities during a simulation run.

A system that is modeled in Simulation Studio can use two types of resource objects: *stationary resources* and *mobile resources*. Stationary resources include the entity holding blocks (the Queue, Server, Delay, and Resource Pool blocks), which delay entities for some period of time during the simulation. The stationary resources are static, and they are created at model building time. *Mobile resources*, which are resource objects that flow in the model, are dynamic and are created during the simulation execution. Mobile resources are defined as a special type of entity. They have all of the capabilities of regular entities, but they also have additional functionality that is managed by the blocks on the Resource template. Mobile resources (in particular, resource entities and their management) are the main focus of this chapter.

Like regular entities, resource entities are objects that can carry attributes. All resource entities in Simulation Studio have a predefined entity attribute named ResourceUnits, which represents the capacity (the number of units) of the resource. Although the ResourceUnits attribute has special uses for resource entities, it can

also be used as an ordinary numeric entity attribute for modeling purposes. In addition to the ResourceUnits attribute, each resource entity also has run-time state information (such as resource state and seize status) that is used by the simulation system to perform resource management during the run. The resource state can be Functional, Failed, Maintenance, or Offlined.

Modeling the Banking System Using Mobile Resources

In Chapter 2, "Overview of SAS Simulation Studio," a model of a simple banking system with one teller is used to illustrate some of the basic concepts that are involved in building models in Simulation Studio. In that example, the bank teller is modeled by using a stationary resource (in particular, a Server block). In this section, the same system is modeled by modeling the bank teller as a mobile resource (a resource entity). These two examples demonstrate the conceptual differences between stationary resources and mobile resources. All blocks that are used in these models can be found in the Standard and Resource templates.

To summarize the modeling requirements for this banking system, these models assume that customers arrive at the bank at a rate of 10 per hour (so that the interarrival time between customers is a sample from the exponential distribution with a mean of 6 minutes). Customers wait in a single line on a first-come, first-served basis. The models also assume that the teller has a service rate of 12 customers per hour (so that the service time for each customer is a sample from the exponential distribution with a mean of 5 minutes).

Figure 10.1 shows the original version of the model from Chapter 2, "Overview of SAS Simulation Studio." The bank teller (represented by the Server block) is a stationary resource in the original model and is created during the model building phase. As a stationary resource, the bank teller never flows or moves through the model. A customer arrives at the bank teller, the bank teller services the customer, and the customer exits the system.

Figure 10.1 The Banking System Model Using Stationary Resources
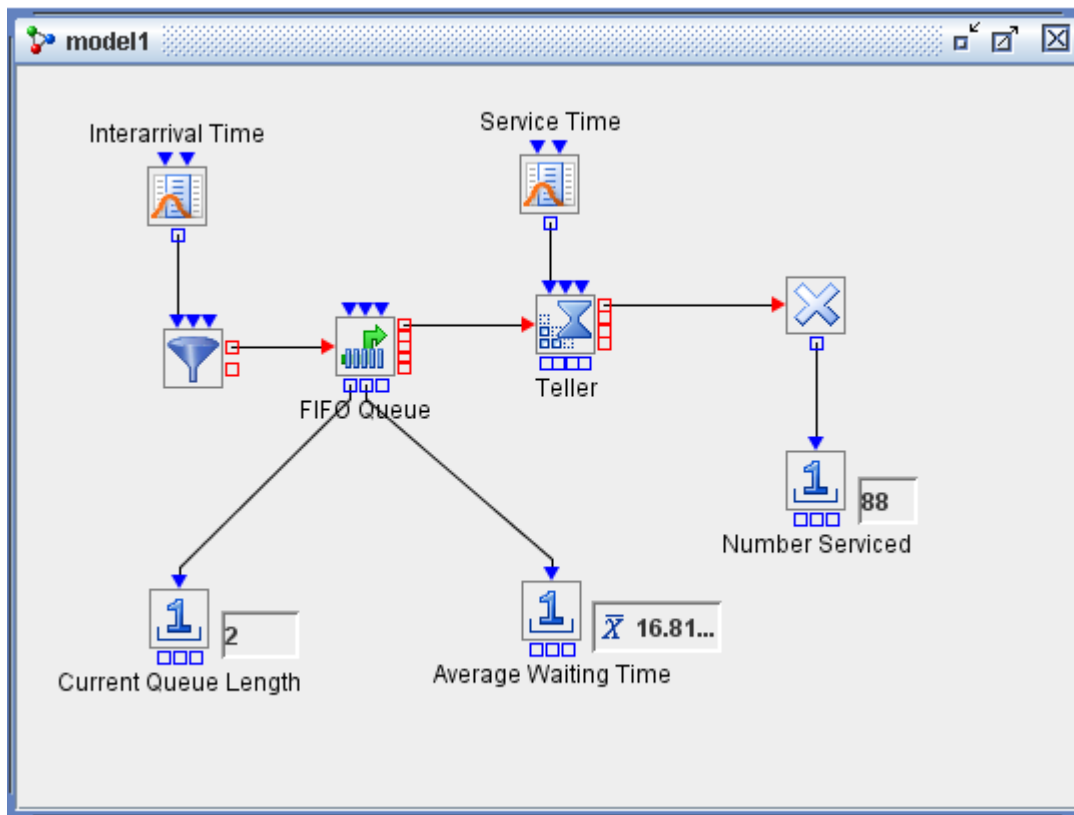
Figure 10.2 shows an alternative model of the same system. In this model, the bank teller is modeled as a resource entity. You can find the project files for this example (named `docMM1Resources`) in the `projects\examples` directory under your Simulation Studio installation directory. The following description of the model is a high-level overview. For more information about the individual blocks used in the model, see Appendix A, "Templates."

Figure 10.2 The Banking System Model Using Mobile Resources
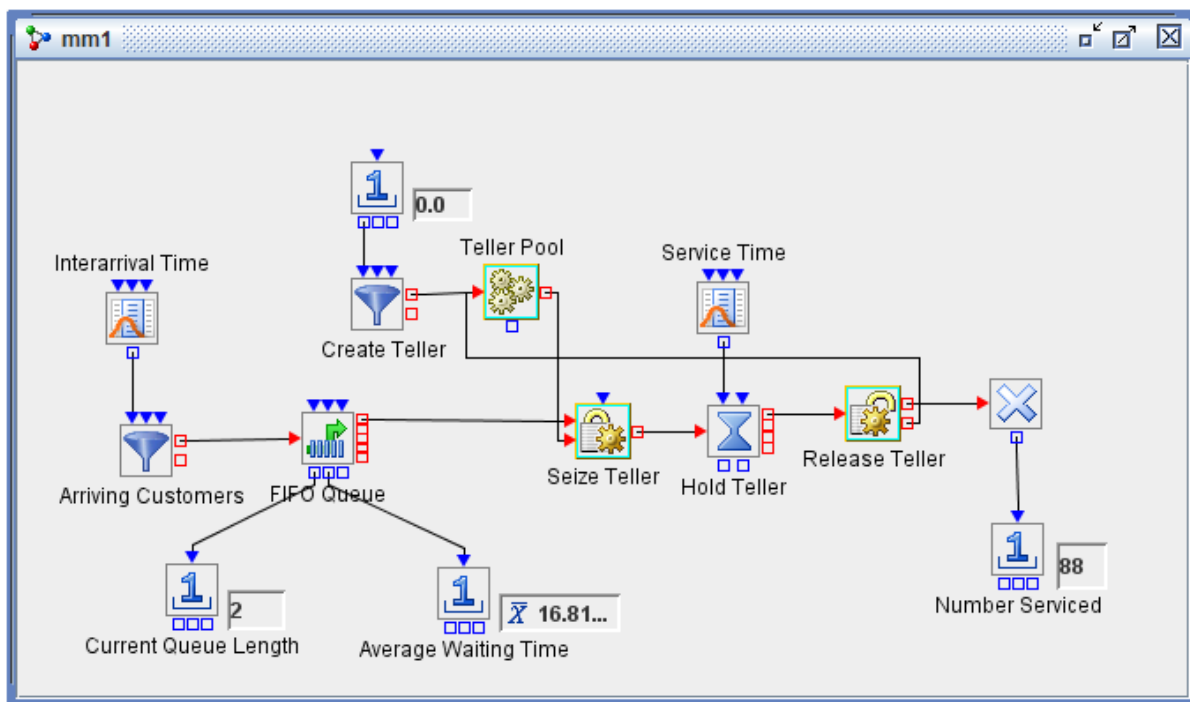
The customer arrival process is the same as in the original model—an Entity Generator block creates customers and sends them to a Queue block to wait for service. To model the bank teller as a resource entity, first a new resource entity type named BankTellers is created by using the Entity Types dialog box, available by right-clicking on the model name in the project tree and selecting **Entity Types**. For more information about the Entity Types dialog box, see Chapter 8, "Entities." Figure 10.3 shows the attributes that are associated with the BankTellers resource entity type. The ResourceUnits attribute has value 1 because there is only one teller.

Figure 10.3 BankTellers Entity Type

In the Block Properties dialog box for the Entity Generator block labeled "Create Teller," the BankTellers entity type is selected from the list in the Name field of the **EntityTypes** tab, as shown in Figure 10.4.

Figure 10.4 EntityTypes Tab in the Create Teller Block Properties Dialog Box

Because the model requires only one bank teller, you set the **Maximum Number of Entities** field value to 1 on the **Attributes** tab of the Create Teller Block Properties dialog box, as shown in Figure 10.5. The bank teller resource entity must be created before the simulation clock begins to advance. Therefore, you need to set the **Start Time** property to 0 and also select the **At Start Time** option in the **First Entity Creation** area. As soon as the bank teller resource entity is created, it is sent to a Resource Pool block (labeled Teller Pool) to wait until it is needed by a customer entity.

Figure 10.5 Attributes Tab in the Create Teller Block Properties Dialog Box

In this example, a Seize block (labeled Seize Teller), a Resource Pool block (labeled Teller Pool), a Delay block (labeled Hold Teller), and a Release block (labeled Release Teller) work together to mimic the functionality of the Server block (labeled Teller) in the original model. When a customer entity arrives at the FIFO Queue block, the FIFO Queue block notifies the Seize Teller block that a customer is waiting. The Seize Teller block then checks whether the bank teller resource entity is available in the Teller Pool block. If it is not available, then the customer entity stays in the queue. If the bank teller resource entity is available, then the Seize Teller block accepts the customer entity from the Queue block, pulls the bank teller resource entity from the Teller Pool block, and attaches it to the customer entity (forming a hierarchy of entities). The customer entity is then sent to the Hold Teller block where the customer entity (along with the bank teller resource entity) is held until its service is completed. It is then routed to the Release Teller block. The Release Teller block extracts the bank teller resource entity from the customer entity and sends the customer entity to the Disposer block to exit the model. The bank teller resource entity is routed back to the Teller Pool block.

A quick inspection of the values in the Number Holder blocks in both models at the end of the simulation run shows that both the original model and the new resource entity model produce the same results.

For this simple banking system, it is not necessary to use a resource entity to model the bank teller. However, in a more realistic model of a banking system, it might be necessary to seize multiple resources simultaneously, to vary the number of available resources according to a schedule, or to preempt a resource either because of a failure or because a higher priority entity arrives. It is not possible to model these scenarios using only stationary resources. In general, mobile resources offer more modeling flexibility and options.

A Common Resource Usage Pattern

A common usage pattern for resource entities in Simulation Studio consists of the following fundamental steps: create, store, locate, allocate, use, deallocate, and dispose. Each of these steps might use one or more Simulation Studio blocks. In addition to all the regular modeling blocks, six resource-specific blocks are available in Simulation Studio: the Seize, Release, Resource Pool, Resource Scheduler, Resource Agenda, and Resource Stats Collector blocks. These six blocks, together with the other regular blocks, provide all the powerful resource management capabilities in Simulation Studio.

The following sections provide a high level overview of each step in the common usage pattern for resource entities. For additional information about the individual blocks that are mentioned in each of these sections, see Appendix A, "Templates."

Creating Resource Entities

Like regular entities, resource entities can be generated by an Entity Generator block. The **Name** field in the **EntityType** tab in the Block Properties dialog box for an Entity Generator block includes the Default (for regular entities) and DefaultResourceEntity entity types that are defined by Simulation Studio, in addition to entity types that are defined by the user for the current simulation model. The resource state of all new resource entities is Functional. The default ResourceUnits value is 1.0, but it can be changed to any nonnegative value.

You can define new types of resource entities by using the Entity Types dialog box that is associated with each model. All user-created entity types appear in the **Name** list on the **EntityType** tab in the Block Properties dialog box for the Entity Generator block. For more information about entity types, see Chapter 8, "Entities."

Resource entities can also be created by the Resource Pool and Release blocks as a result of splitting other resource entities. Sometimes it is convenient to create a small number of resource entities, each with a large ResourceUnits attribute value, and store these resource entities in a Resource Pool block with the **Merge/split resource units among resource entities of same types** option selected. When a request for resource units is made, the Resource Pool block can split resource units that are held in a resource entity into smaller units and assign them to a new resource entity of the same type.

Similarly, a Release block can split units from a resource entity and allocate the split units to a newly created resource entity (of the same type) that is then released. For more information about merging and splitting of resources, see the section "Merging and Splitting Resource Entities" on page 106.

Resource entities can also be cloned by using the Clone block. The Clone block clones the attributes from the original resource entity, but it ignores other run-time information, such as the resource state and seize or unseize status that is set by the resource management blocks.

Note that using the Entity Generator and Clone blocks to create resource entities affects the total number of resource units, but using the Resource Pool and Release block to do so does not.

Storing Resource Entities

After a resource entity is created, it must be sent to a Resource Pool block for storage before it can be seized and allocated to meet resource demands. The Resource Pool block performs resource management tasks for resource entities. These tasks include maintaining the seize or unseize status, processing resource requests, and merging or splitting resource units.

A resource entity is considered *unseized* if it resides in a Resource Pool block; it is considered *seized* if it leaves the pool and is not directly held by any other Resource Pool block. A newly created resource entity is also considered unseized before it enters a Resource Pool block.

Occasionally, a common Queue block might be used to hold resource entities if the resource management tasks performed by a Resource Pool block are not needed. However, this approach should be used carefully because resource management capabilities are not provided by Queue blocks.

Locating Resource Entities

Resource entities are usually stored in the Resource Pool block. Resource entities need to be located, requested, and allocated to fulfill the resource requirements of other entities. Locating resources is also essential for other resource operations, including scheduling, statistics collection, and preemption. For example, the resource entities of interest need to be identified so their statistics can be collected during the simulation.

Simulation Studio primarily uses attribute-based rules to locate resource entities. An *attribute rule* is a Boolean expression that the attributes of targeted resource entities must satisfy. Run-time resource information, such as resource state and seize or unseize status, is also used to locate and identify resource entities.

The resource requirements of an entity that enters a Seize block (referred to as a *controlling entity*) can be specified using entity type or attribute rules in the Seize block. Multiple types of resource entities can be seized simultaneously, and resource entities can seize other resource entities. A Seize block dynamically creates an input resource entity port for each defined resource requirement. The input resource ports of a Seize block are connected to a Resource Pool block. During the simulation run, the Seize block uses the links to its input resource ports to locate and request resource entities from a Resource Pool block when a controlling entity enters the Seize block.

It is also possible to locate resource entities by their object references. Resource entities can flow through an Entity Group Holder block to form a *resource entity group*, which holds a group of references to these resource entities. The entity group and its subgroups can be queried later for locating and requesting the corresponding resource entity objects.

Allocating Resource Entities

After locating the required resource entities in a Resource Pool block, the Seize block requests the resources. The Resource Pool block processes the request and allocates the resource entities to the Seize block. In Resource Pool blocks, only resource entities in the Functional state can be seized.

The Resource Pool block delivers resource entities to a Seize block to fulfill a resource demand. If the Resource Pool block has its merging/splitting units option disabled, the requested resource entities are

delivered without alteration, even when the delivered resources have more units than requested. If the merging/splitting option is enabled, the Resource Pool block delivers the new resource entities after the splitting process. The delivered resource entities contain the exact amount of units requested. For more information about the merging and splitting feature of the Resource Pool block, see the section "Merging and Splitting Resource Entities" on page 106.

To decrease the likelihood of a resource deadlock, the Seize block in Simulation Studio does not support partial allocation of resources. All specified resource requests must be satisfied before resource entities are allocated to a controlling entity. Otherwise, the Seize block does not accept the request to take the controlling entity, and the controlling entity must wait (usually in a queue) for all required resources to become available.

The Batch block can also be used to seize resources. For example, if all resource entities have the same ResourceUnits value and are of the same type, then the Batch block can be used to allocate resources to a carrier entity as the resource entities become available. In this approach, the Batch block can hold a carrier entity with a partially completed resource allocation when there is a resource shortage until the additional resource entities arrive at the Batch block. For this kind of usage, the batch carrier entity used by the Batch block performs the same role as the controlling entity does for a Seize block, and the entities batched with the carrier entity are resource entities. When a Batch block is used, required resources do not need to be available all at the same time. The resource entities can be allocated or batched as they become available, one after another. For more information about the Batch block, see Appendix A, "Templates."

The modeling constraints and requirements of the particular system being simulated determine whether a Seize block or Batch block is appropriate for allocating resource entities. The Batch block offers a more simplistic approach, but it also provides fewer options than the Seize block provides.

Using Resource Entities

After the resource entities have been allocated to a controlling entity (or in the case of a Batch block, a carrier entity), the controlling entity usually continues to flow through the model. In the banking system example, the controlling entity might move on to a Delay block for a period of time and then be routed to a Release block to have the resource entity deallocated. However, if you are modeling a more complicated system, it is possible that resource entities stay with a controlling entity as it flows through various parts of the model. For example, in an emergency room simulation, a patient entity entering the emergency room might require a nurse, a doctor, and a surgery room simultaneously. After the required resources are seized, then the patient entity goes to a Delay block that represents the surgery. After surgery, the doctor and surgery room resource entities might be released from the patient. However, the nurse entity might stay with the patient entity and a recovery room resource entity might be added.

In a manufacturing example, parts could be modeled as resource entities, and they could be continually added to the controlling entity as it progresses down a virtual assembly line. In this case, the controlling entity never releases the part entities because they are essentially consumed to build the final product.

Deallocating Resource Entities

Resource entities that are seized by controlling entities can be released (deallocated) by using the Release block. Resource constraints can be defined on the Release block to locate targeted resource entities within the controlling entity to be released. The Release block provides an output resource entity port for each

constraint defined. For each controlling entity that enters the Release block during a simulation run, the user-defined resource constraints are used to locate and deallocate the targeted resources among the resources held by the controlling entity. The deallocated resources flow out the appropriate output resource ports.

Analogous to the seize process, releasing resource entities is treated as a special entity unbatching operation. Therefore, the Unbatch block can also be used to release resource entities if the deallocation process does not require partial resource entities to be released from the controlling entity (no manipulation of resource units) or if the model logic does not require different types of resource entities to flow to different locations. The released resource entities from an Unbatch block flow out of the same output port, one after another.

Released resource entities can be routed to any block in the model as dictated by the system logic. In the emergency room example, the doctor resource entity might be required to complete paperwork after being released by a patient entity and before being seized by the next patient entity. In this case, the released doctor resource entity could be routed to a Delay block (representing the paperwork completion time) and then back to a Resource Pool block, signifying that it is available to be seized by another patient entity.

Disposing Resource Entities

Like regular entities, resource entities can be disposed by the Disposer block. Resource entities that are attached to a controlling entity (or carrier entity) that enters a Disposer block are disposed along with the controlling entity.

The Resource Pool block can also dispose resource entities. With the merging/splitting units option enabled, the Resource Pool block disposes an incoming resource entity if the Resource Pool block can merge the units of the new resource entity with a compatible resource entity in the pool. For more information about merging and splitting in a Resource Pool block, see the section "Merging and Splitting Resource Entities" on page 106.

Using a Disposer block to dispose resource entities affects the total number of resource units (capacity) in the model. However, the automatic disposal of resource entities by the Resource Pool and Release blocks (when the merging/splitting option is used) does not.

Extending the Banking System Model

The resource facilities in Simulation Studio provide more advanced functionality than demonstrated in the previous simple banking system example. To illustrate some of this additional functionality, the previous banking system model has been extended. The basic premise remains the same—customers arrive at the bank and wait to be served by a bank teller. However, in this new model there are three bank tellers, not all of whom are available during the entire simulation run. This model also collects utilization statistics for the bank teller resource entities.

Figure 10.6 shows the new model. You can find the project files for this example (named `projects\docSchedule`) in the `examples` directory under your Simulation Studio installation directory. The most obvious difference between this resource model and the previous one is the addition of the Resource Agenda, Resource Scheduler, and Resource Stats Collector blocks. A less obvious difference is the creation, storage, and allocation approach used here for the three bank teller resources.

Figure 10.6 Banking System Model That Uses Resource Scheduling

There are two options for creating the three bank teller resources. Either you can create three BankTeller resource entity objects, each of which has a ResourceUnits value of `1`, or you can create one BankTeller resource entity object that has a ResourceUnits value of `3`. To demonstrate the merging/splitting feature in Simulation Studio, this model uses the latter approach.

The Create Teller block generates one BankTeller resource entity object and passes it to the Teller Pool block, just as in the previous model. This time, the resource entity attribute of the resource entity object is equal to `3` instead of `1`. To make efficient use of the ResourceUnits in the BankTeller resource entity object, it is necessary to use the Resource Pool block's merging and splitting resource entities capabilities. Selecting the **Merge/Split resource units among resource entities of same type** check box in the Block Properties for Teller Pool dialog box (see Figure 10.7) enables the block to look at the ResourceUnits attribute of its held resource entities and possibly subdivide a resource entity into two resource entities, one of which matches the needs of an incoming resource request. In this example, the Seize Teller block requests a BankTeller resource entity that has one ResourceUnit. With the merge/split option selected, the Teller Pool block can take a BankTeller resource entity that has a ResourceUnits value of 3, create a new BankTeller resource entity object that has a ResourceUnits value of 1, and decrease the ResourceUnits value of the existing BankTeller resource entity (already in the pool) to 2. The new BankTeller resource entity object (which has a ResourceUnits value of 1) is sent to the Seize Teller block to satisfy its request.

Similarly, when the BankTeller resource entity object returns to the Teller Pool block, its ResourceUnits can be merged with a BankTeller resource entity already in the pool, and the incoming BankTeller resource entity object is disposed.

Figure 10.7 Teller Pool Block Properties Dialog Box

For this model, all three bank tellers might not be available during the entire simulation—maybe they take a staggered lunch break. The previous model used a total simulation time of 9 hours (540 minutes). Assume in this model that for the first 4 hours (240 minutes) of the work day, all three bank tellers are available. For the next hour, two of the tellers go on lunch break and when they return, the third teller takes an hour lunch break. When the third teller returns from lunch break, all three tellers are available for the remainder of the work day.

A Resource Agenda block and a Resource Scheduler block are used together to implement the scheduling functionality in this model. A Resource Agenda block is used to create a list of resource adjustment actions (collectively known as a *resource agenda*) to be performed during the simulation run. The resource agenda information is passed to a Resource Scheduler block to arrange and perform the resource adjustment actions on specific resource entities. Figure 10.8 shows the Resource Agenda Block Properties dialog box for this model. Each row or entry in the agenda represents a resource adjustment action and consists of three pieces of information: Duration, Value, and Value Type. For a complete description of each of these fields, see the Resource Agenda block description in Appendix A, "Templates." For this model, these entries represent the changes in the number of bank tellers throughout the simulation period.

Figure 10.8 Resource Agenda Block Properties Dialog Box

The Resource Scheduler block receives the resource agenda at the beginning of the simulation run through its InAgenda input value port, and the scheduler performs the sequence of resource adjustments on a specified group of targeted resource entities. The block properties dialog box that is associated with the Resource Scheduler block in this model is shown in Figure 10.9. Each row in the **Appointments** table is called an *appointment*. For more information about using the Resource Scheduler block, see Appendix A, "Templates." Only the **Start Time**, **Agenda**, and **Search Targets By** fields are discussed here. The **Start Time** field specifies the simulation time to activate the associated agenda. The **Agenda** field supplies the name of the incoming agenda to use for this appointment. (Multiple Resource Agenda blocks can be linked to the same Resource Scheduler block, each sending a named agenda to the scheduler.) The **Entity Type** field under **Search Targets By** indicates which resource entities the associated agenda in the appointment applies to. For this model, the agenda that is created in the Resource Agenda block is activated at a Start Time of 0 (when the simulation run begins) and is applied to the BankTellers resource entity objects in the model. The **Immediate Actions** options selected here indicate that resource entities that are in a seized state are not preempted. For more information about preemption, see the section "Preempting Resource Entities" on page 111.

Figure 10.9 Resource Scheduler Block Properties Dialog Box
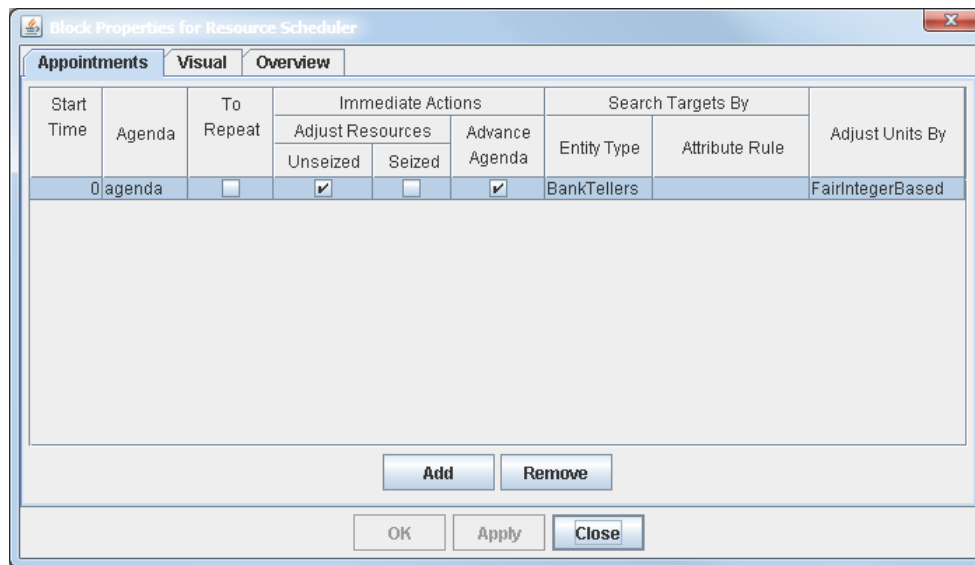
The final new block that is added to this resource model is the Resource Stats Collector block. This block collects statistics about resource entities during the simulation run. The Resource Stats Collector requires a minimum of two pieces of information: the resource entities that you want to collect statistics about and the statistics that you want to compute. The Resource Stats Collector Block Properties dialog box has separate tabs for you to enter this information. The **Groups** tab, as shown in Figure 10.10, identifies the targeted resource entities for statistics collection. In this example, resource entities of the BankTellers Entity Type have been targeted. The **Statistics** tab, as shown in Figure 10.11, specifies the statistics that you want to compute. For more information about the columns in this table, see the Resource Stats Collector block overview in Appendix A, "Templates."

Figure 10.10 Groups Tab in Resource Stats Collector Block Properties Dialog Box
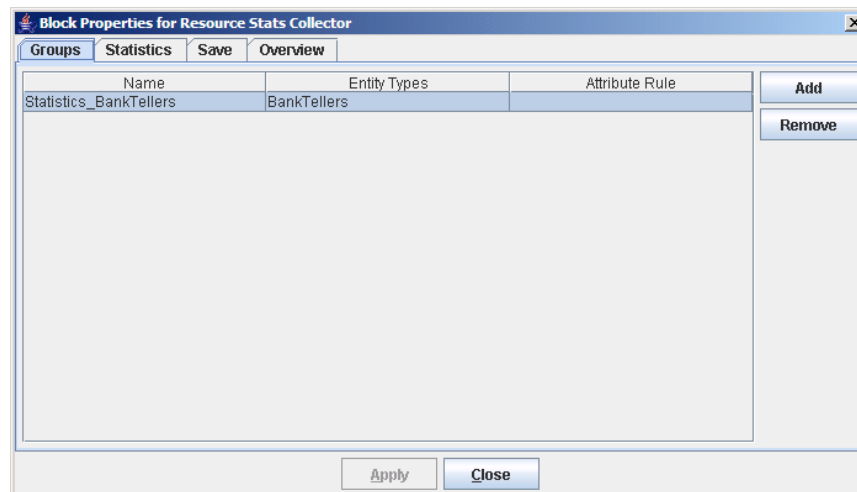
Figure 10.11 Statistics Tab in Resource Stats Collector Block Properties Dialog Box

In this example, two statistics are defined: Utilization and Idle. Both are computed using the **%AvgAvail** statistic type. For the statistic named Utilization, the **Seized** option is set to true. In this case, the result is the percentage of time that resource entities of type BankTellers are seized, relative to the time-average number of available units of resource entities of type BankTellers during the simulation run. For the statistics named Idle, the **Seized** option is set to false. In this case, the result is the percentage of time that resource entities of type BankTellers are not seized, relative to the time-average number of available units of resource entities of type BankTellers during the simulation run.

The Resource Stats Collector block provides an option to save the defined statistics to a file at the end of each run (that option was selected for this example). You can also attach a plot block to the OutData outport of the Resource Stats Collector block to display the statistics during the simulation run. In this example, a Table block displays the defined statistics. At the end of the run, the Utilization statistic equals 0.361853, and the Idle statistic equals 0.638147 (as shown in Figure 10.6).

This example provides a glimpse into the modeling capabilities and potential applications of the Resource Agenda, Resource Scheduler and Resource Stats Collector blocks. Although these blocks are more complex than many of the previously demonstrated blocks, they also provide powerful and flexible modeling functionality.

Additional Resource Functionality

The following sections describe in more detail some of the advanced features related to resource entities, including merging and splitting, statistics collection, and scheduling resource entity state and capacity adjustments. The concept of resource preemption is also discussed.

Merging and Splitting Resource Entities

The Resource Pool and Release blocks provide a unique capability, which is called resource entity merging and splitting. All resource entities have a numeric ResourceUnits attribute, which can be assigned a nonnegative value. The value contained in this attribute represents the capacity or available units that are associated with the individual resource entity object. The default value for the ResourceUnits attribute is 1. Therefore by default, each new resource entity that is created represents one unit of that particular resource.

The Teller Pool block in the previous example used the merging/splitting feature of the Resource Pool block. When the merging/splitting option is selected in a Resource Pool block, mini-pools of entities based on a user-defined criteria are kept. A common criterion is to merge and split the resource entities based on resource entity type. Merging and splitting helps reduce the number of resource entity objects that are needed in a model during a simulation run. Figure 10.7 shows the Block Properties for Teller Pool dialog box.

If you want to use the merging/splitting feature of a Resource Pool block, you must specify the criteria to use for grouping the resource entities in the pool. The simplest approach is to select the **Merge/Split resource units among resource entities of same type** check box in the Resource Pool block properties dialog box. This uses the resource entity type as the grouping criteria. You can also specify more definitive grouping criteria by using the **Key Entity Attribute Fields** section of the dialog box.

When the merging/splitting option is enabled on a Resource Pool block, the first resource entity for any defined group that enters the block remains in the block for the duration of the simulation run. It effectively becomes the mini-pool for that group. Any other resource entities that enter the Resource Pool block and are a match for the criteria for that group are merged with the first resource entity of that group. That is, the value of the ResourceUnits attribute of the resource entity just coming into the Resource Pool is added to the value of the ResourceUnits attribute of the existing, matching resource entity, and the incoming resource entity is disposed.

When a resource request comes into the Resource Pool block, the block looks for a resource entity in its possession that has matching criteria. If it finds a matching resource entity, it then looks at the ResourceUnits attribute on the matching resource to see whether sufficient units are available to fill the request. If enough units are available, the Resource Pool block creates a new resource entity from the matching resource entity, populates its ResourceUnits value with the requested number of units, and decreases the value of the ResourceUnits in the original resource entity accordingly. (The original resource entity might end up with a ResourceUnits value of 0.)

Although the Release block does not have (or need) any resource entity merging capabilities, it does provide an option for splitting resource entities. When the **Splittable** option is selected on a Release block, you can deallocate or release some of the ResourceUnits that are associated with an incoming resource entity. Fields are available in the Release Block Properties dialog box for defining the criteria to use for releasing and splitting resource entities. If the **Splittable** option is selected and an incoming resource entity meets the **Splittable** criteria that you have defined on the Release block, the Release block looks for the ResourceUnits

attribute value on the incoming resource entity and compares it to the value you specified to be deallocated from the resource entity. If the value to be deallocated is less than the value of units available in the incoming resource entity, the Release block creates a new resource entity from the incoming resource entity, populates its ResourceUnits value with the specified number of units, and decreases the value of the ResourceUnits in the incoming resource entity accordingly. The newly created resource entity then flows out the appropriate Release block output port. The original resource entity, with its ResourceUnits now decreased, flows out of the Release block with the controlling entity.

When the simulation model does not have the merging and splitting options enabled in its Resource Pool and Release blocks, the model is sometimes referred to as an *object-based* resource model. When the merging or splitting options in either of these types of blocks are enabled, the model is called a *unit-based* resource model. The Resource Pool and Release blocks provide special options that take advantage of the ResourceUnits attribute in a resource entities for unit-based models.

For more information about the Resource Pool and Release blocks, see Appendix A, "Templates."

Collecting Resource Entity Statistics

Resource entities are usually scattered throughout the model during a simulation run. For example, some resource entities might be stored in a Resource Pool block, some might be allocated to controlling entities (that is, they are seized by other entities), and some might be held in a Server or Delay block. The Resource Stats Collector block on the Data and Display template organizes resource entities of interest into *resource groups*, and it calculates and reports user-defined statistics for a group. For each resource group, resource constraints can be defined in the Resource Stats Collector block to locate and monitor the group's targeted resource entities during a simulation run. Each defined statistic is computed for all groups, and each statistic can also be defined with its own resource constraints to further limit the computation to a subset of a resource group. For example, suppose you create a resource entity type called DoctorEntity to represent doctors in a medical simulation, and one of the attributes you define on this DoctorEntity entity type is named *specialty*. Valid values for this *specialty* attribute might be cardiologist, neurologist, and ENT. You can use the Resource Stats Collector block to calculate statistics for all instances of the DoctorEntity resource entity that have a *specialty* attribute value of "neurologist" by defining a Group named Neurologists such that the group members have the value DoctorEntity for the **Entity Types** field and **Attribute Rule** specialty=="neurologist". The Resource Stats Collector block reports its results as a data table, with each group as a data row and each column containing statistics.

One particular statistic that simulation practitioners are often interested in is utilization, which is usually defined to be the percentage of time that a resource is "busy" relative to the time-averaged number of resources available. The definition of a busy resource can vary depending on the system being modeled. For example, "busy" could mean not idle (seized) and in the Functional state. Alternatively, "busy" could just mean seized, as in the banking system model in the section "Extending the Banking System Model" on page 100. The Resource Stats Collector block has a statistic type called **%AvgAvail**, which can be used to compute utilization statistics. Furthermore, the Resource Stats Collector block is designed so that you can specify your own definition of a "busy" resource when computing utilization statistics. You use the **Seized**, **State**, and **Attribute Rule** fields on the **Statistics** tab to create your own definition of "busy."

Scheduling Resource Entity State and Capacity Adjustments

Resources often undergo routine adjustments or changes, and the effects of such adjustments often last for a limited period of time. Examples are a truck having routine maintenance, a worker taking a lunch break, and salespeople being added for a holiday shopping season.

The scheduling of a resource adjustment often needs to address the following issues:

- what type of adjustment to make—either capacity or state change

- what resources to adjust—locate the targeted resources

- when to adjust

- how long to adjust

- whether the adjustment is preemptive (disruptive)

- where and how the adjustment takes place—in resource pools (unseized) or in other entity holding blocks (seized)

- how to proceed to the next related adjustment, if any—temporally constrained or not

- whether to repeat this adjustment in the future—repeatable or not

In Simulation Studio, resource scheduling is supported by the Resource Agenda and Resource Scheduler blocks together. The Resource Agenda block addresses the first issue in the preceding list, and the Resource Scheduler block addresses the other issues.

Resource adjustments are often related and happen in an orderly fashion. In Simulation Studio, related adjustment actions can be grouped together. A special type of value object called a Resource Agenda defines a sequence of related resource-adjustment actions based on a relative simulation time that starts at time zero. Each resource adjustment action includes a change to either the resource capacity value or the resource state value in targeted resource entities over a certain time period; it is defined as a resource agenda entry. The Resource Agenda block provides a resource agenda to describe what type of resource adjustments to make during a simulation run.

The Resource Scheduler block accepts and stores resource agenda objects during a simulation run. This block also accepts scheduling requests through an *appointment* to perform resource adjustments. The resource agendas are later activated and processed by the Resource Scheduler block as specified by the appointments. An appointment tells the Resource Scheduler which resource agenda to use for resource adjustments and how to deal with the second through last issues in the preceding list. After a resource agenda is activated, its entries are activated and processed sequentially. An appointment is fulfilled when all entries in the specified resource agenda are processed and all associated resource adjustments are completed.

There are two ways to specify an appointment in a Resource Scheduler block: statically or dynamically. You can enter a *static appointment* at the design or modeling time by using the **Appointments** tab of the Resource Scheduler block properties dialog box (see Figure 10.9). At simulation time, appointments are processed as the initial requests by the Resource Scheduler block. After an unrepeatable request is processed and all associated adjustments are completed, the request is discarded by the Resource Scheduler block.

In addition to the appointments that you schedule through the properties dialog box controls, you can also use *resource schedule entities* to request a Resource Scheduler block to process resource agendas *dynamically* through the InRequest port during a simulation run. If a request is repeatable and the OutRequest port is connected, the Resource Scheduler block sends a resource schedule entity out through the OutRequest port to represent the request that can be used again later. You can route these resource schedule entities to a Resource Scheduler block through its InRequest port as *dynamic requests*. Before being routed to a Resource Scheduler block, these resource schedule entities can be processed (delayed, modified, counted, stored, and so on) to simulate complicated scheduling situations. Resource schedule entities have attributes for each field that is required in order to define an appointment in a Resource Scheduler block. You can also create resource schedule entities by using an Entity Generator block, and the values of the resource schedule entity attributes can be set before routing it to a Resource Scheduler block. Note that if an appointment is repeatable but the OutRequest port is not connected, then the Resource Scheduler block automatically reschedules the appointment, using the current simulation time as the new start time.

Sometimes the contents of the resource agenda entries in a resource agenda are fixed and can be specified completely at the design time. But at other times, the duration or capacity change of an adjustment action is not fixed or cannot be specified in the corresponding resource agenda entries at the design time. For example, the downtime or duration of a machine failure is not fixed but follows a certain statistical distribution (such as a normal distribution). In Simulation Studio, the numeric contents (such as duration, units, and units offset) of a resource agenda entry can be left unspecified, or blank. Such an agenda entry is called a *dynamic agenda entry*. The Resource Agenda block populates the unspecified values by pulling these values dynamically though the InDuration or InValue input ports during the simulation. The desired values are created through submodels that are connected to these input ports.

When a resource agenda is active, the Resource Scheduler block performs the appropriate actions as specified by the **Immediate Actions**, **Search Targets By**, and **Adjust Units By** rules. For each adjustment action, certain groups of resource entities are identified as follows:

- *Potential targets* are resource entities in the functional state that satisfy the requirements of the **Search Targets By** rule, which contains the **Entity Type** and **Attribute Rule** fields. If no rule is specified in these fields (that is, they are left blank), then all resource entities are potential targets.

- *Targets* are resource entities from the set of potential targets that the Resource Scheduler block selects for adjustment so that the set of targeted entities is a subset of the set of potential targets. The resource entities that are selected to be targets depend on several factors, including the **Adjust Units By** rule and an "unseized first" heuristic, which means that the Resource Scheduler block always attempts to adjust unseized resource entities before seized entities.

Figure 10.12 and Figure 10.13 depict the relationship between the sets of resource entities that are potential targets and those that are actual targets.

Figure 10.12 Resource Entity Groups That Are Formed for an Adjustment Action
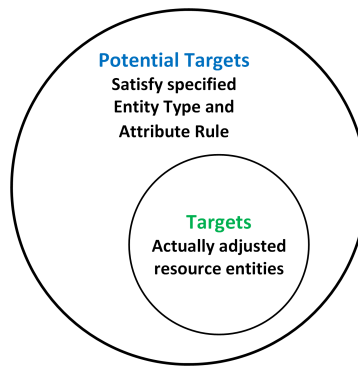
Figure 10.13 How Resource Entity Groups Are Formed for an Adjustment Action
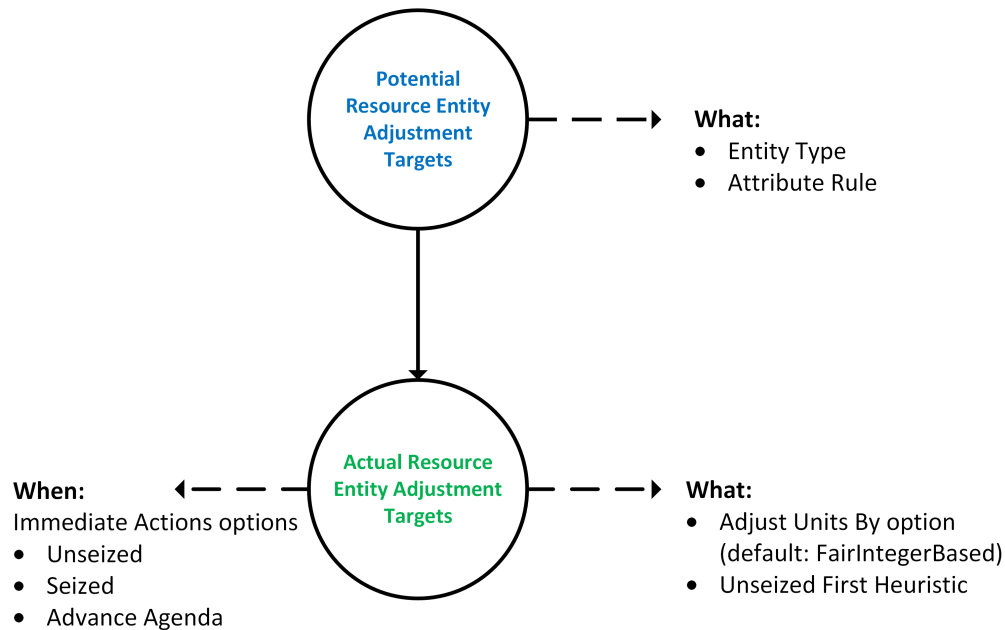
When the targeted resource entities are actually adjusted depends on the **Immediate Actions** rule, which contains the **Unseized**, **Seized**, and **Advance Agenda** options. The **Unseized** option specifies when targeted unseized resource entities are adjusted. When selected, any targeted, unseized resource entities are adjusted immediately. Examples of when selecting this option is useful include increasing inventory levels immediately at all inactive warehouses or putting all trucks currently in a garage through routine maintenance. If you do not select the **Unseized** option, then any targeted unseized resource entities are adjusted after all seized resource entities that are *potential* targets become unseized. This represents the situation when all resource entities in the potential target group need to be gathered (unseized in a Resource Pool block) before any targeted entities are adjusted.

The **Seized** option specifies when seized resource entities (if any) are adjusted. When selected, any *targeted* resource entities that are currently seized are adjusted immediately if there are not enough unseized potential targets available to adjust. Whether the adjustment is a capacity change or a state change, this could trigger a *preemption* in a holding block where a seized entity is located at the time of the adjustment. If either the

OutPreempt or OutResource port on the holding block is connected, then preemption will occur. If you do not select the **Seized** option, then any targeted, seized resource entities are adjusted after all seized resource entities that are potential targets return to a Resource Pool block and become unseized. Selecting the **Seized** option does not guarantee that any seized resource entities are adjusted. Seized entities are identified as targets for adjustment only if there are not enough unseized entities to cover the scheduled adjustment.

The **Advance Agenda** option is the third part of the **Immediate Actions** rule. When selected, the Resource Scheduler block moves on to process the next agenda entry without waiting for all seized and unseized targets to be adjusted for the current entry. This could result in a shorter time between actual resource adjustments than what the agenda specifies. If you do not select the **Advance Agenda** option, the Resource Scheduler block waits for all seized and unseized targets for the current entry to be adjusted before moving on to process the next agenda entry. As a result, the time between resource adjustments could be longer than the duration time specified in the resource agenda.

The Resource Scheduler block uses several heuristics to adjust the units of targeted resource entities when a request is made. In general, the Resource Scheduler block always attempts to adjust unseized entities before seized entities. This "unseized first" heuristic avoids unnecessary preemptive adjustments. The **Adjust Units By** field in the Resource Scheduler block dialog box specifies the policy that is used to allocate or deallocate resource units among targeted resource entities when there is a scheduled resource adjustment. The default **FairIntegerBased** policy requires that any adjustment to the units of a resource entity be made on an integer basis (integrally) and that any fractional resource units be ignored. If there is an increase in resource units, then those units are distributed integrally and as evenly as possible among all unseized resource entities that are *potential* targets (according to the "unseized first" heuristic). If there are not enough unseized potential targets available to adjust, then seized entities are adjusted. If there is a decrease in resource units, then the capacities of any unseized resource entities that are potential targets are integrally and evenly reduced. If there are not enough unseized resource units to cover the scheduled decrease, then the capacities of seized resource entities that are potential targets are integrally and evenly reduced either immediately (if you select the **Seized** option) or when all seized entities that are potential targets return to a Resource Pool block and become unseized (if you do not select the **Seized** option). For more information about the functionality of the Resource Agenda and Resource Scheduler blocks, see Appendix A, "Templates."

Preempting Resource Entities

In Simulation Studio, the stationary resources (including most of the entity holding blocks, such as the Queue, Server, and Delay blocks) support two common types of resource preemption: priority-based and scheduled.

Priority-based preemption is primarily for preempting stationary resources, which are the entity holding blocks such as the Queue, Server, and Delay blocks. The entity that wants to enter such a holding block is considered a consumer of the static resource that is represented by that block. Allocation of static resources is usually about accepting entering entities into a holding block to take up space in the block. Preemption of static resources forces out some entities currently in a holding block to make space for some other entities. The selected holding blocks (Queue, Server, and Delay) provide an InPreempt input port that accepts an EntityGroup object as input. These blocks compare the entity references in the EntityGroup to the entities that are currently held by the block and preempt any matches. This type of preemption is often triggered by the higher priority of the new entities attempting to enter these blocks.

Scheduled preemption is primarily for preempting mobile resources, which are resource entities, and is based on a resource adjustment agenda. Sometimes the allocated and seized resource entities need to be preempted from their current controlling entities so that these resource entities can be re-allocated to other controlling

entities. This type of preemption can be triggered by the preemptive resource adjustments that are arranged and processed by a Resource Scheduler block. Most entity holding blocks, including the Queue, Server, and Delay blocks, provide OutPreempt and OutResource output ports. If a resource entity that is allocated to a controlling entity that is currently held in a holding block is adjusted preemptively, the holding block attempts to force the controlling entity out of the block's OutPreempt port and the resource entity out of the OutResource port. If the OutPreempt port is not connected, the controlling entity remains in the entity holding block. If the OutResource port is not connected, the adjusted resource entity remains allocated to its controlling entity.

The post-preemption processing of preempted entities and resources is often highly specific to the application. For example, when a job is preempted from a service, some applications might resume the job to finish its remaining service time, some might restart the job from the beginning, some might simply scrap the job, and so on. The modeling facilities that Simulation Studio provides make it possible to construct suitable solutions to handle these situations.

Additional examples that demonstrate preemption and other resource modeling techniques are provided in Appendix E, "Examples of Simulation Studio Models."

Chapter 11
Model Debugging and Verification

Contents

Overview of Debugging and Verification Tools

The **Log**, **Trace**, and **Animation** tabs (which can be expanded at the bottom of the Project window) provide feedback and debugging capabilities during the execution of a simulation model. Both the application and individual blocks can post model execution state, event, and error information to the **Log** and **Trace** tabs while the model is running. The **Log** tab contains messages of varying severity levels about potential configuration and execution state anomalies. The **Trace** tab (if the Tracer is enabled) displays simulation clock timestamps and state information for individual blocks as execution progresses. You can customize the content of the **Trace** tab to filter out unwanted trace messages. The **Animation** tab enables you to select which regions of the model to animate during model execution. You can also specify the animation speed, start time, and end time for each selected region.

When parallel mode is selected for model execution, the **Trace** and **Animation** tabs are disabled. For more information about running a model in parallel mode, see the section "Running a Model in Parallel Mode" on page 38 in Chapter 4, "Simulation Models."
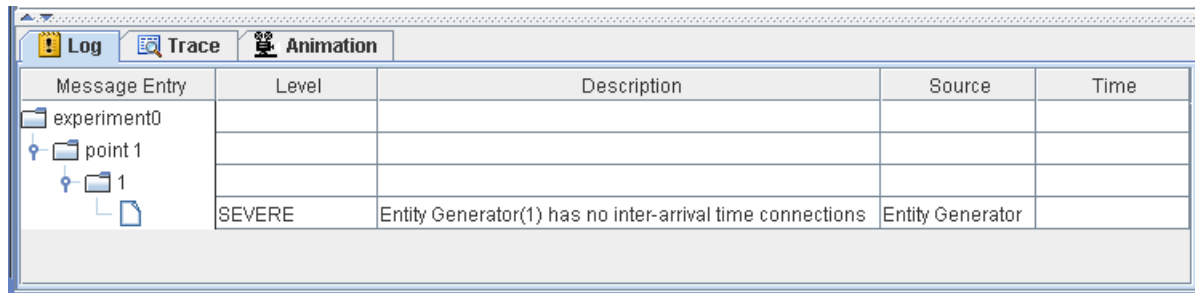
Log Tab

Messages can be posted to the **Log** tab by both the application framework and the individual blocks. Each message consists of five components: message entry, level, description, source, and time.

The Message Entry column uses a tree structure to display the active experiment along with the design points and replicates that are selected in that experiment for execution. Each message that is posted to the log is associated with a particular replicate. The value in the Level column represents the severity of the log message; possible values are SEVERE, WARNING, and INFO. A SEVERE log message indicates that a major problem has been encountered with the simulation model, and execution is terminated.

Figure 11.1 displays a SEVERE log message from an Entity Generator block for the first replicate of the design point named point 1. To function properly, an Entity Generator block requires a connection to its InterArrivalTime input port from which it can pull numeric values. That connection appears to be missing in this example.

Figure 11.1 Sample Log Tab

A WARNING message usually suggests that a condition has occurred that warrants further investigation. An example of such a condition might be a block receiving a negative number when it was expecting a nonnegative value. An INFO message simply contains information but does not indicate a potential problem.

The Time column on the **Log** tab displays the (simulation clock) time when the message was logged. Some messages, such as the one in the SEVERE log message example in Figure 11.1, can be logged before the model execution actually begins; therefore the Time value is empty for these messages.

The Description column contains the message text, and the Source column displays the label of the block that generated the message. Clicking an entry on the **Log** tab causes the associated block to be highlighted in the Model window. The **Log** tab pop-up menu contains an option called **Auto Sync with Model**, which is turned on by default so that the Model window will scroll (if necessary) to display the highlighted block that is associated with an entry in the **Source** column.

Trace Tab

Trace messages provide details about state changes, events, and execution flow within individual blocks; they are useful for debugging and verifying your simulation models. The Tracer must be enabled before any trace messages are generated. You can enable the Tracer by using the pop-up menu available on the **Trace** tab background.

The following types of entries are displayed on the **Trace** tab:

- simulation clock timestamps (displayed in black)

- entity information (displayed in red)

- value information (displayed in blue)

When the Tracer is enabled, a timestamp is posted to the **Trace** tab every time the simulation clock advances. All other trace messages are generated by the individual blocks. Each block is responsible for the content of its trace messages and for determining when to generate trace messages. Clicking on a message in the Trace window causes the associated block to be highlighted in the Model window. The **Trace** tab pop-up menu contains an option called **Auto Sync with Model**, which when enabled causes the model window to scroll (if necessary) to display the highlighted block that is associated with an entry in the Trace window.

Although it is possible (and likely) for any simulation model execution to generate a considerable volume of trace messages, the **Trace** tab has a limited size buffer associated with it to store the messages. Therefore, only the most recent trace messages are retained in the trace buffer when the buffer is full. To change the maximum number of trace records to display, select **Records** from the **Trace** tab pop-up menu to open the Trace Records dialog box, as shown in Figure 11.2. You can also use the Trace Records dialog box to save the trace records to a file. Figure 11.3 shows a sample **Trace** tab.

Figure 11.2 Trace Records Dialog Box
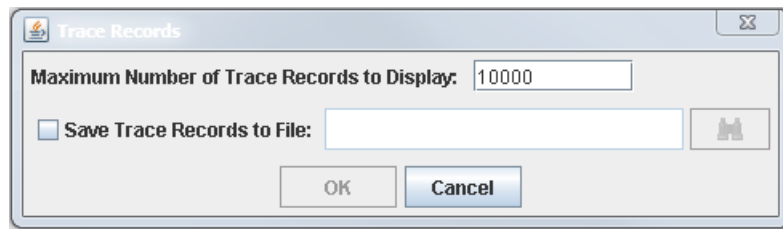
Figure 11.3 Sample Trace Tab
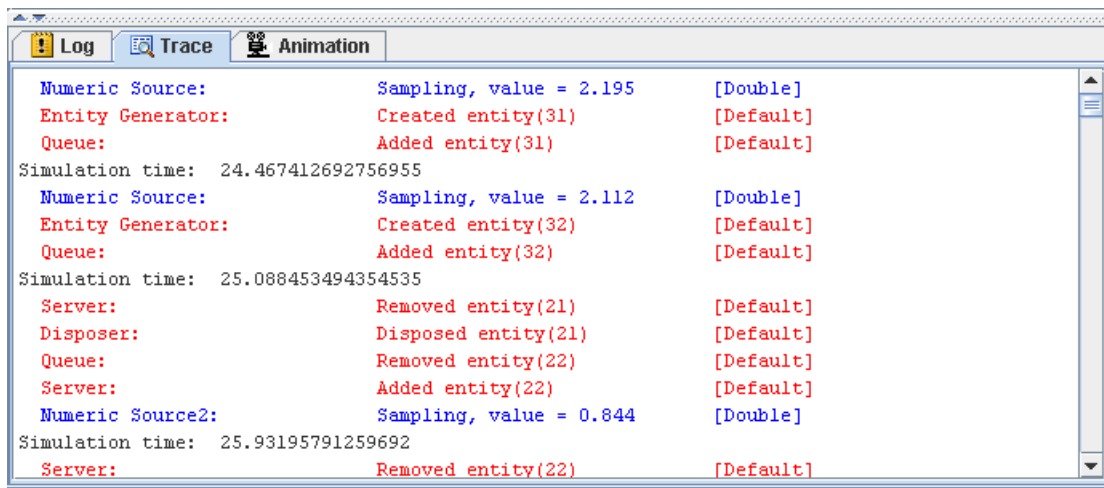
Tracing Configuration

You can control the amount of information that is displayed on the **Trace** tab by using the Tracing Configuration dialog box. To open the Tracing Configuration dialog box, right-click on the **Trace** tab background and select **Tracer Configuration**.

You can use the Tracing Configuration dialog box to filter trace messages according to various criteria: the blocks that generate trace messages, the entities that are mentioned in trace messages, and the simulation clock time of trace messages. You can combine more than one of these criteria to further refine the number of trace messages that appear on the **Trace** tab. Changes made in the Tracing Configuration dialog box apply only to the model that is active when you open the Tracing Configuration dialog box. These settings last only while Simulation Studio is open. They are not saved when you close Simulation Studio.

Figure 11.4 shows a sample Tracing Configuration dialog box.

Figure 11.4 Tracing Configuration Dialog Box
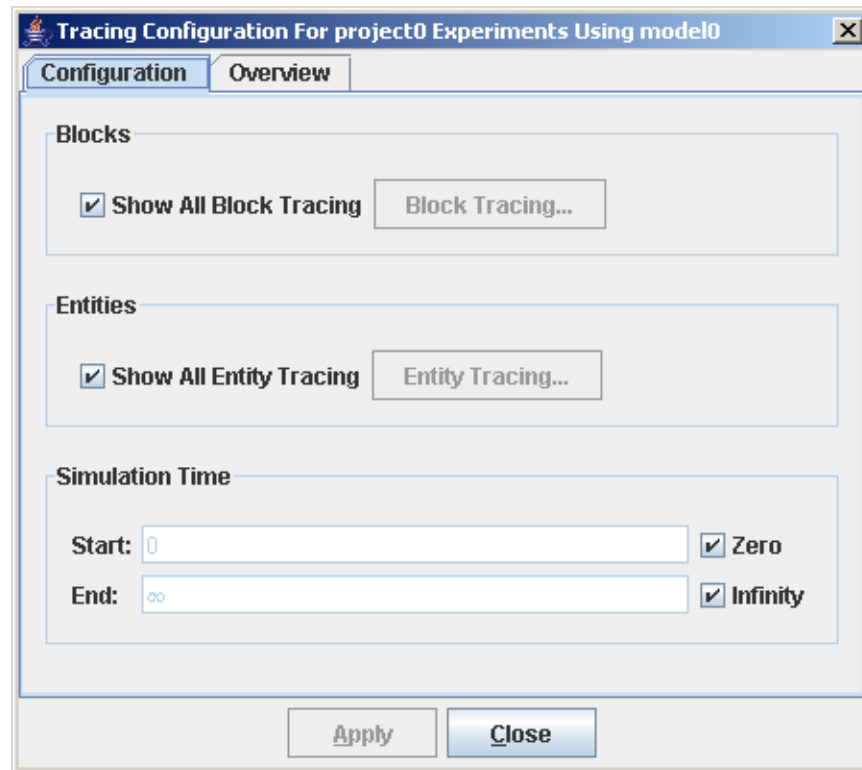
The following sections appear in the Tracing Configuration dialog box:

- **Blocks**

- **Entities**

- **Simulation Time**

By default, all trace messages that are generated during simulation execution appear on the **Trace** tab. Therefore, when you open the Tracing Configuration dialog box for the first time, the default options are selected to show all block tracing, show all entity tracing, and show tracing for all simulation times.

If you clear the **Show All Block Tracing** check box, the **Block Tracing** button becomes enabled. Click
Block Tracing to open the Block Tracing dialog box. Figure 11.5 shows a sample Block Tracing dialog box.

Figure 11.5 Block Tracing Dialog Box
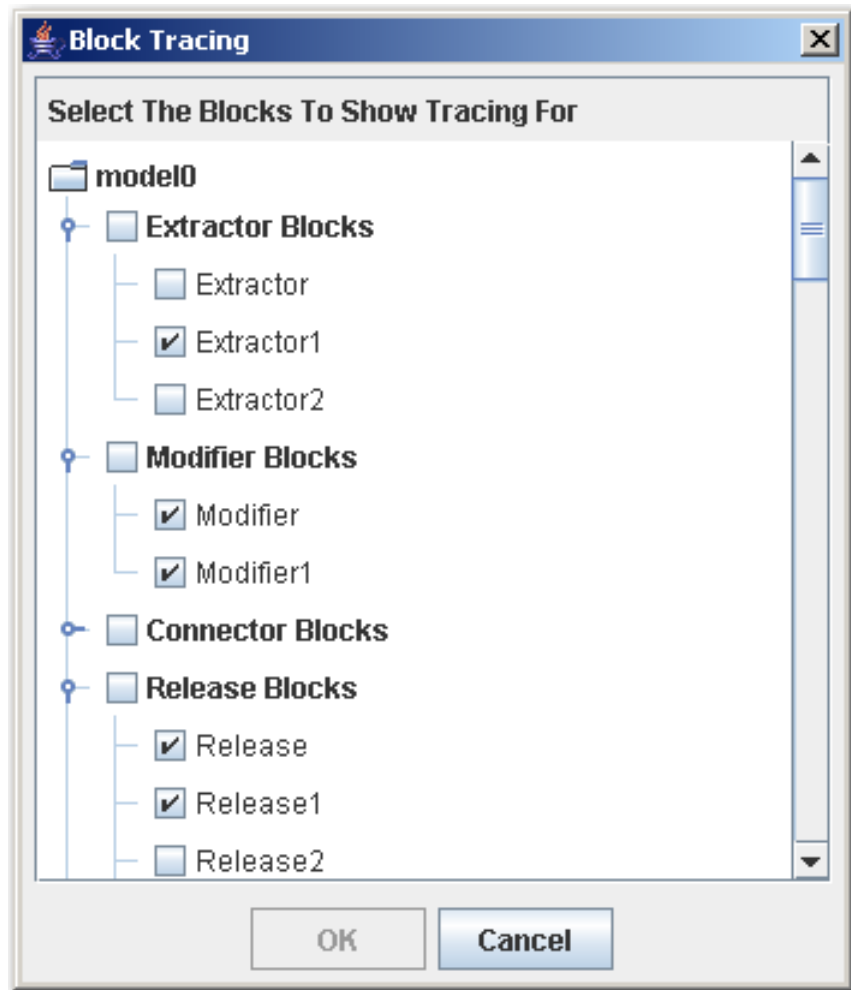
In this dialog box, select the blocks for which you want to see trace messages and click **OK**. For trace
messages that are generated by blocks, only trace messages generated by the selected blocks are displayed on
the **Trace** tab. If you want to display trace messages generated by all blocks, simply check the **Show All
Block Tracing** check box in the Tracing Configuration dialog box.

If you clear the **Show All Entity Tracing** check box, the **Entity Tracing** button becomes enabled. Click
Entity Tracing to open the Entity Tracing dialog box. Figure 11.6 shows a sample Entity Tracing dialog
box.

Figure 11.6 Entity Tracing Dialog Box
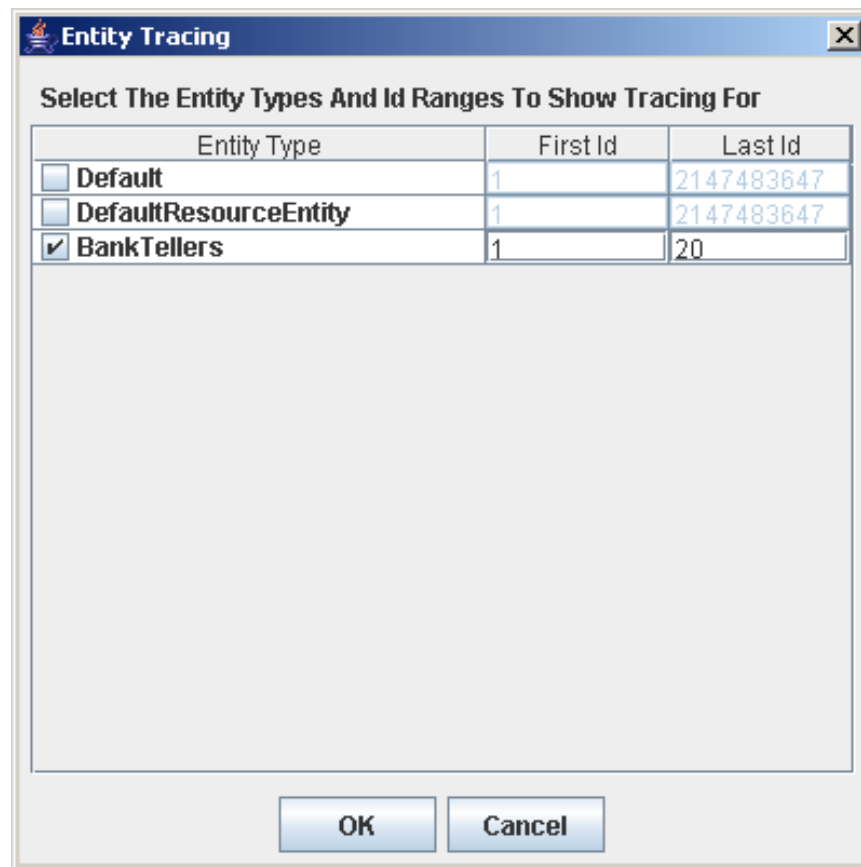
In this dialog box, select the entity types for which you want to see trace messages. Also, you can specify a range of Id numbers for each selected entity type, which causes only trace messages for entities with Id numbers in the specified range to be displayed on the **Trace** tab. Click **OK** to apply the settings and close the Entity Tracing dialog box. For trace messages regarding entities, only trace messages regarding the selected entity types and Id ranges are displayed on the **Trace** tab. If you want to display trace messages regarding all entities, simply check the **Show All Entity Tracing** check box in the Tracing Configuration dialog box.

To show trace messages only for a specified range of simulation clock time, you can specify a start time, an end time, or both. To specify a start time, clear the **Zero** check box in the Tracing Configuration dialog box and enter a positive integer value in the **Start** field. Trace messages that are generated before the specified start time are not displayed on the **Trace** tab. To specify an end time, clear the **Infinity** check box and enter a positive integer value in the **End** field. Trace messages that are generated after the specified end time are not displayed on the **Trace** tab.

The settings in the **Blocks**, **Entities**, and **Simulation Time** sections of the Tracing Configuration dialog box combine to determine what is ultimately displayed on the **Trace** tab. For example, if you select to show tracing for only the Entity Generator A block, select to show tracing for only the Default entity type, and select to show tracing for only time 10 to 20 on the simulation clock, then the **Trace** tab displays only messages that are generated by block Entity Generator A. Furthermore, if this block generates messages for entity types other than Default, they are not displayed. Lastly, no trace messages are displayed before time 10 or after time 20, regardless of the block that generated the trace message or the entity type that is referred to by the trace message.

Animation Tab

Animation can be a useful tool for verifying that your model is operating as intended. The **Animation** tab provides options for controlling the animation as a model runs; a sample is shown in Figure 11.7. The **Region** column in the **Animation** tab displays a hierarchical tree of all the compound and submodel blocks in the current active model. You can use the **Region** and **Enabled** columns together to select which parts of the model to animate during model execution. For a nested group of compound or submodel blocks, you can select the block name at the highest level to turn on or off the animation for all blocks in the nested group. Similarly, you can select the model name at the topmost level in the **Region** column to either turn on or off the animation for all compound and submodel blocks in the model. If there are no compound or submodel blocks in the model, then the **Region** column displays only the model name. Before you run a model and view the animation, you must enable animation either by clicking the animation icon 🎬 on the Simulation Studio toolbar or by selecting **Animate** from the **Run** menu.

Figure 11.7 Sample Animation Tab
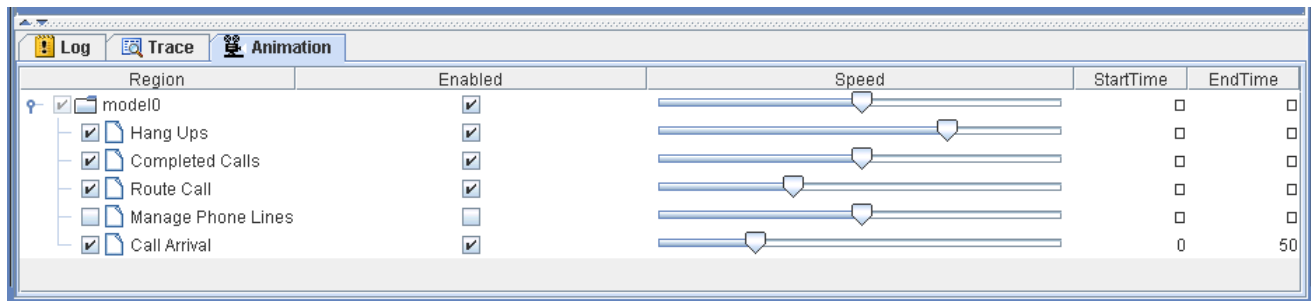
You can move the sliders in the **Speed** column to set the animation speed for each region of the model. To reset all sliders to the default speed, right-click on the **Speed** column and select **Reset All: Speed**. The default animation speed is set by using the slider on the Simulation Studio toolbar.

You specify animation time intervals for each region of the model in the **StartTime** and **EndTime** columns. Figure 11.7 indicates that the compound block labeled Call Arrival will be animated from time 0 to time 50. You can reset all start times to the default value by right-clicking the **StartTime** column header and selecting **Reset All: StartTime**. Similarly, you can reset all animation end times to the default value by right-clicking the **EndTime** column header and selecting **Reset All: EndTime**.

Clicking an entry on the **Animation** tab causes the associated block to be highlighted in the Model window. The **Animation** tab pop-up menu contains an option called **Auto Sync with Model**, which when enabled causes the model window to scroll (if necessary) to display the highlighted block that is associated with an entry on the **Animation** tab.

Chapter 12
Block Templates

Contents

Overview of Block Templates

SAS Simulation Studio templates provide a facility for managing the blocks that you use to build your simulation models, and the Simulation Studio template palette offers a visual representation of the content of templates. A Simulation Studio template contains information about a collection of blocks. This information is stored as an XML document. There is no limit on the number of templates that you can load into Simulation Studio. You can view the content of any loaded template in the template palette area of the application. As discussed in Chapter 4, "Simulation Models," you drag an item from a Simulation Studio template into a Model window to create an instance of the associated block in your simulation model.

When Simulation Studio starts, it automatically loads a series of default templates named Standard, Advanced, Data and Display, Resource, and Output Analysis. The blocks that these templates contain are described in detail in Appendix A, "Templates." You can also create a custom template and save it to a file, as described in the section "Using the Template Pop-Up Menu" on page 122. For a custom template, you can create a collection of blocks or compound blocks that have a common theme. For example, you might create a template with blocks that simulate a manufacturing environment, or you might create a template with blocks that are specifically designed to model health care systems.

Using the Template Menu

You use the **Template** menu to create a new (empty) template, to open an existing template, to close a template that is currently displayed in the template palette, or to save a modified template. To load an existing template into Simulation Studio, select **Open** from the menu. This opens a file selection dialog box where you choose the template filename and then click **Open** to open the template. The chosen template is then loaded into the application, and the template name is added to the Templates list box. Selecting a template name in the Templates list box populates the template palette area of Simulation Studio with the blocks that the associated template contains. Only one template is active at any time.

Using the Template Pop-Up Menu

A pop-up menu that contains options for displaying palettes opens when you right-click in the template palette background. (See Figure 12.1.)

Figure 12.1 Template Palette Menu
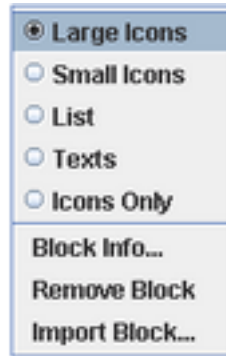
In addition to palette formatting options, this menu contains three other items: **Block Info**, **Remove Block**, and **Import Block**. Select **Block Info** to view template information for the highlighted block in the palette. Editing the values in the Block Info dialog box changes those values for all current and future instances of the associated block in your models. You cannot change the values in the Block Info dialog box for blocks in the default Simulation Studio templates.

Select **Remove Block** to delete the highlighted block from the template. Select **Import Block** to add a new block to the template that is currently displayed in the palette. Selecting **Import Block** opens a dialog box that contains fields where you enter the same information that appears in the **Block Info** dialog box. NOTE: You cannot modify any of the default Simulation Studio templates. As a result, the options **Remove Block** and **Import Block** are not available for the default templates.

Chapter 13
Data Input, Collection, and Analysis

Contents

Overview

The subject matter of a simulation investigation or the sophistication of a model often dictates what type of data need to be collected from each simulation run and the amount of data required to perform an appropriate analysis. Furthermore, the accuracy of the simulation results depends on the suitability of the distributions that are used as inputs to the model, making input analysis and distribution fitting one of the critical considerations in the design and construction of a simulation model. Simulation Studio is integrated well with both SAS and JMP to take advantage of the rich and powerful data processing and analysis capabilities that are available in each package. This section provides an overview of the data management capabilities in Simulation Studio and describes how Simulation Studio interacts with both SAS and JMP. Before interacting with SAS or JMP from Simulation Studio, make sure the appropriate server has been launched. See the section "Launching Local SAS and JMP Servers" on page 21 in Chapter 3, "Introduction to SAS Simulation Studio," for details.

Data Value Types

As described in the section "Entities and Values" on page 34 in Chapter 4, "Simulation Models," data values in Simulation Studio can be numbers, character strings, Boolean values, or objects. Frequently used object types include data model objects and observation objects. A *data model object* can be viewed as an in-memory representation of a SAS data set or JMP table during a simulation run. It contains information or values specified in rows, columns, and cells. An *observation object* represents one row of a data model object. It can be viewed as the simulation-time representation of a data observation from a SAS data set or a data row of a JMP table. The data model and observation objects are used in Simulation Studio blocks to

represent data for various access and collection tasks. For example, you can use the Dataset Holder block from the Data and Display template as a holding facility for a data model object, making it useful both for matrix computations and also for modeling scenarios that require repeated access to a data set (look-up table) to perform a particular computation. During a simulation run, you can access through user-defined output ports the contents of a data model object (such as individual data cell values and observation objects) that is stored in a Dataset Holder block and you can pass the queried results to other blocks in the model.

Figure 13.1 is a model of a machining center where parts are processed at four different stations in a particular sequence that is based on part type. In this example, a Dataset Holder block with one user-defined output port (located at the bottom right of the Dataset Holder block) is used to hold the machining sequence data set, which is displayed by using a Table block (located at the bottom left of Figure 13.1). The data set value that is pulled from the bottom right output port is a particular cell value based on part type; it indicates the next station in the processing sequence. In this example, the Dataset Holder block holds a data set that is used repeatedly by all entities. An alternative is to store the information in the machining sequence data set as entity attributes, but that would result in the same data being stored multiple times. A complete description of this Machining Center example can be found in the section "Machining Center Model" on page 323 in Appendix E, "Examples of Simulation Studio Models."

Figure 13.1 Machining Center Model
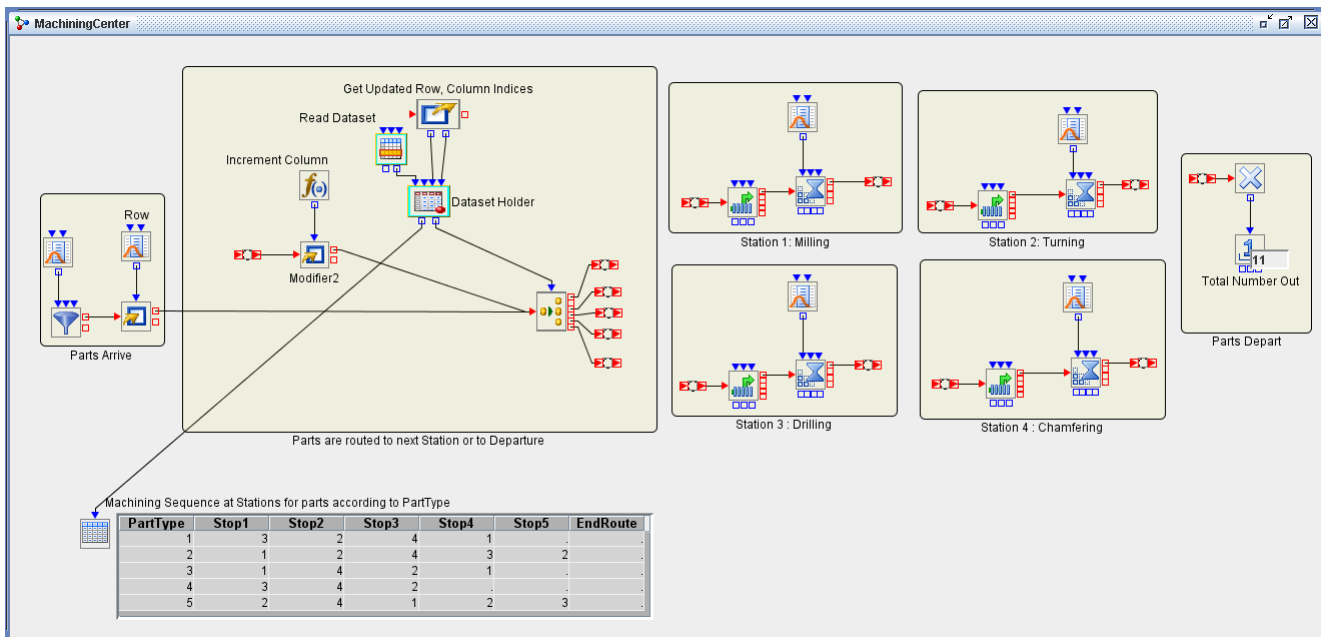
The functionality of a Dataset Holder block can be viewed as analogous to that of a Number Holder block with the **To Downstream** and **From Upstream** propagation options turned off. Whereas the Number Holder block holds a single numeric value, the Dataset Holder block can hold a collection of related numeric or character data values in the form of a data set.

Data Input

The Simulation Studio blocks that can be used to input data to a model are the Numeric Source, Text Source, and Observation Source blocks. In general, three levels of data can be retrieved: a single value, a single row, and an entire data set. The Numeric Source and Text Source blocks provide a stream of single data values by reading a column of data from a SAS data set or JMP table, and the Observation Source block provides a stream of data observation (row) objects from a SAS data set or JMP table. For example, you can use the Observation Source block to read a row of data from a data set and either assign the entire row as entity attributes or assign a subset of the data cell values in the row as attributes. This is demonstrated in the example "Using the Observation Source Block to Set Entity Attributes" on page 325 in Appendix E, "Examples of Simulation Studio Models." Entity objects in Simulation Studio are tightly integrated with data management schemes, and the Observation Source block provides a straightforward method for creating entities with specific attributes based on input data.

The Observation Source block can also be used to read in an entire SAS data set or JMP table, as shown in the machining example in Figure 13.1. An Observation Source block (labeled Read Dataset) is used to read in the entire machining sequence data set. The data model that represents that data set is passed from the OutData port of the Observation Source block to the InData port of the Dataset Holder block, where it is held until needed.

NOTE: SAS data sets or JMP tables that will be loaded into Simulation Studio cannot contain missing values.

Data Collection and Output

Data that are generated by a Simulation Studio model can be collected and saved either through the Experiment window or through a dedicated data collection block. The following sections describe how to use these data collection methods.

Block Data Storage

Eight blocks can accumulate data and store it as a data model object: the Bucket, Dataset Writer, Number Holder, Probe, Queue Stats Collector, Resource Stats Collector, Server Stats Collector, and Stats Collector blocks. For more information about the functionality of these blocks and the types of information they can collect, see Appendix A, "Templates." A data model from a particular data collection block can be accessed by other blocks via the OutData or OutCollected port. For example, you can connect a plot or table block to the OutData port of a Queue Stats Collector block to visually display the queue statistics (such as average waiting time) while the simulation model is running.

At the end of a run, you can save the contents of a data model from any data collection block as a SAS data set or as a JMP table. Furthermore, dedicated Boolean ports enable you to collect event-driven data. For example, you can save the contents of a data model object at any point during a simulation run by connecting to the InData port on a Dataset Writer block. The data saving operation is triggered by a Boolean signal that is sent to the InSaveNow port on a Dataset Writer block from another block in the model. See the section "Using the Dataset Writer Block to Save Data during a Run" on page 326 in

Appendix E, "Examples of Simulation Studio Models," for an example that uses the Dataset Writer block to save data during a run. Some of the data collection blocks, such as the Number Holder, Bucket, Probe, and Stats Collector blocks, also have a data clearing port. When a `'true'` Boolean signal is received at the data clearing port, all data collected by the block up to that time during the simulation run are cleared. The data clearing port facilitates collecting and saving data that correspond to specific periods during the simulation run.

Simulation Studio saves the data that are collected by blocks on a project basis in a hierarchical directory structure that is based on the hierarchical structure of the model. This structure reflects any nesting of blocks that results from the use of compound blocks. By default, a folder called `results` is created in the project folder and serves as the root directory for any data collection results. The data that are saved by individual blocks are stored in folders by block name in the `results` folder.

NOTE: At the start of each simulation run, the `results` folder is cleared of its contents. If you want to save data permanently from a simulation run, you should move them to another location that is not in the `results` folder.

For each project, you can change the root directory name for data collection results by selecting **Results** from the project pop-up menu in the Project Explorer window. Selecting **Results** opens the Results dialog box, where you enter the name of the root directory to be used for storing model execution results. (See Figure 13.2.)

Figure 13.2 Results Dialog Box
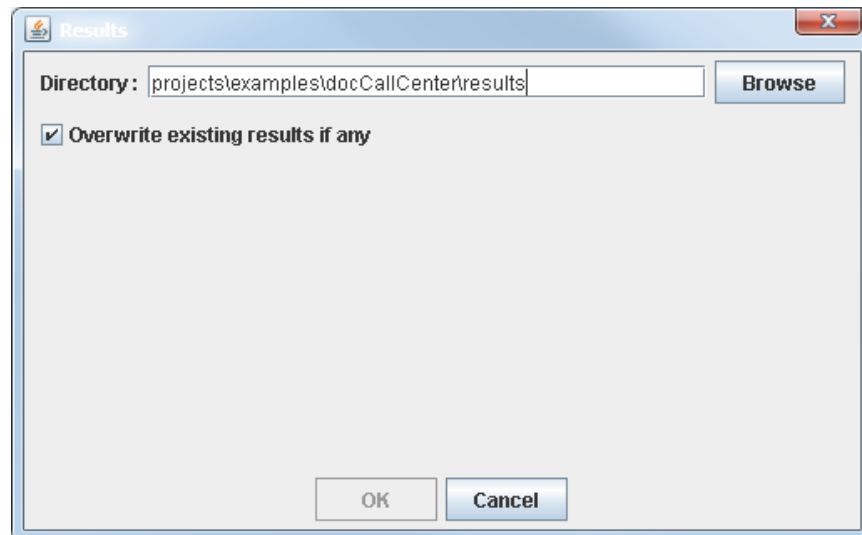
You can supply the filename associated with an individual block's data storage by using the individual block's Block Properties dialog box. If you do not provide a filename, a default filename is generated automatically.

You can also collect data by selecting **Auto Save Results** in the model pop-up menu in the Project Explorer window. (See Figure 13.3.)

Figure 13.3 Model Menu
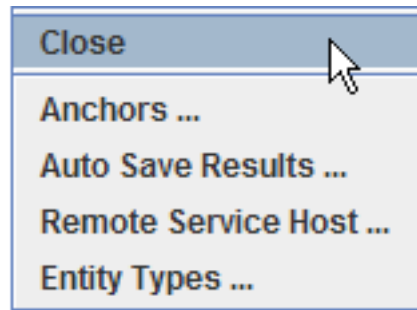
All blocks that are capable of collecting and saving data provide an option to automatically save any collected data at the end of each simulation run. This option is usually set in the Block Properties dialog box for each individual block. If you have many blocks that are collecting data in your model or the collection blocks are nested in compound blocks, it might take a considerable amount of effort to open all the individual dialog boxes and make the appropriate selections. Using a hierarchical format, the Auto Save Results dialog box displays all the blocks in the model that have data collection capabilities. (See Figure 13.4.) In the Auto Save Results dialog box, you can set the automatic save option for any of these blocks by selecting its corresponding check box without having to open the individual block dialog boxes.

Figure 13.4 Auto Save Results Dialog Box
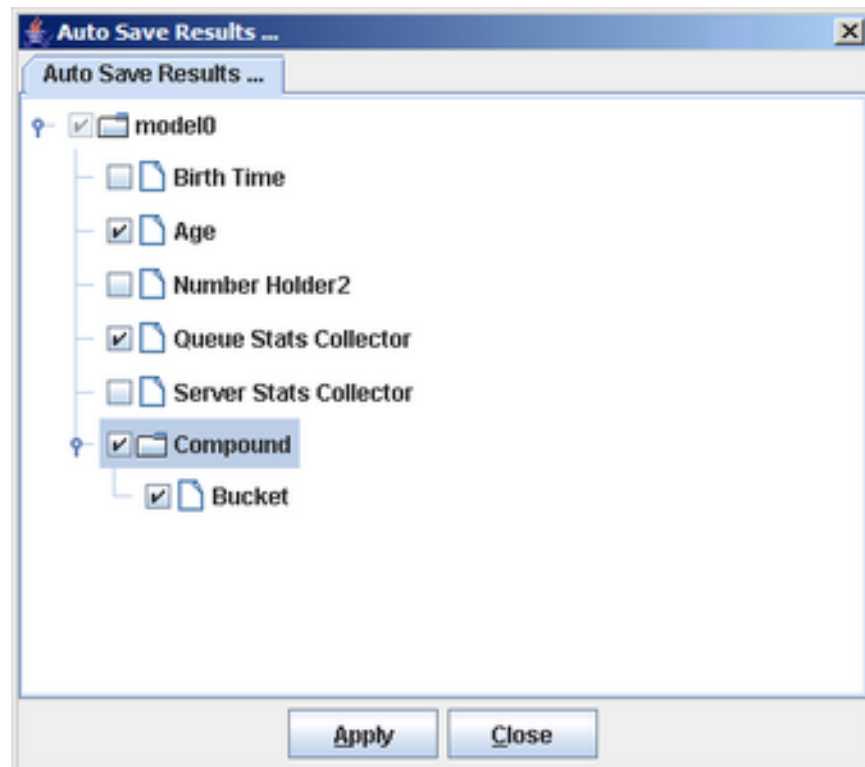
Experiment Window Data Storage

The Simulation Studio Experiment window provides another option for collecting data on simulation runs. Recall from Chapter 5, "Experiments," that experiments are composed of factors and responses; factors are set before running an experiment, and responses are values extracted from the model at the end of a simulation run. Another means of collecting and saving simulation data is to select **Save Design** from the Experiment window pop-up menu to save the experiment to a file. You can save Experiment window data as either a SAS data set or a JMP table for analysis purposes.

To systematically study the effect of specific input parameters on the simulation model output, you can create an experimental design in JMP, run the experiment in Simulation Studio, and select **Analyze Results** from the Experiment window pop-up menu to pass the entire contents of the Experiment window directly back to JMP for analysis. For example, you can use the simulated results to estimate a statistical model, which in turn you can use to determine optimal levels of the factors so that a particular response is maximized or minimized. See Appendix C, "Design of Experiments," for details.

Data Analysis

As discussed in the following sections, Simulation Studio can compute basic summary statistics for data that are collected during a simulation run. In addition, there is a state-of-the-art block for computing statistically valid confidence intervals for responses that are generated from a steady-state simulation. Simulation Studio is also integrated with both SAS and JMP so that you can customize your input and output data analysis needs.

Output Analysis

Each block that collects data provides an output port (labeled OutData or OutCollected for the Number Holder block) that other blocks can use to access its data model object. The plot blocks (Histogram and Scatterplot, for example) are the usual recipients of these data; these blocks can provide real-time data analysis while the simulation is running. Typically, however, you also want to use SAS or JMP software to analyze the data that you have saved to data sets during a simulation run. The following sections provide an overview of output analysis options that are available within Simulation Studio, including the process for computing statistics for time-dependent and time-independent data, analysis techniques for terminating and steady-state simulations, and the use of the SAS block for data analysis and report generation.

Classification of Statistics

Data that are collected during a simulation run can be used to estimate the parameters (such as the mean and variance) of the underlying population from which the data are sampled. For example, consider the population of waiting times for a particular queue. The waiting time data that are collected during a simulation run represent a *sample* from the population that consists of all possible waiting times. When you estimate population parameters from sample data, you need to consider two classifications of statistics:

- For *observation-based* statistics, the data collected and used to estimate parameters are time-independent so that there is interest only in the observed value and not in when it was collected. Waiting-time data are an example of a time-independent sample, and the average waiting time is an example of an observation-based statistic. The following formula can be used to compute the average waiting time: $\bar{x} = \sum_{i=1}^{n} x_i / n$, where $x_1, x_2, \ldots, x_n$ are the set of n observed waiting times.

- For *time-persistent* statistics, the data collected are time-dependent and it is necessary to record both the values and the time periods for which each value persisted. Queue length data are an example of a time-dependent sample, and the average queue length is an example of a time-persistent statistic. To compute the average queue length $\bar{x}_T$, the following formula is used: $\bar{x}_T = \int_0^T x(t) dt / T$, where T is the total time period observed and $x(t)$ is the number in the queue at time t. The average queue length is a weighted average of the possible queue lengths, where the weights are the amount of time that a particular queue length value is observed. Another example of a time-dependent sample is the number of busy cashiers in a store, and the average utilization of the cashiers is an example of a time-persistent statistic.

You can use the data collection blocks described in the section "Block Data Storage" on page 125 to collect both time-dependent and time-independent data and, in some cases, to compute statistics. The Bucket block collects data—namely, the specified attributes of all entities that pass through the block along with the time that the attributes were recorded. However, the Bucket block does not compute statistics for those data. The Queue Stats and Server Stats Collector blocks collect data that are related to specific Queue or Server blocks in the model and automatically compute summary statistics such as average queue length, average waiting time, and average utilization. The Resource Stats Collector block can compute user-defined, time-persistent statistics for specific groups of resource entities. The Stats Collector block can compute statistics for any time-dependent or time-independent data that are generated by a model. For example, as shown in the simple bank system example in Figure 13.5, you can use a Number Holder block (labeled NumberInSystem) connected to a Stats Collector block to record data about the total number of customers in a bank. Each time the value in the Number Holder block is updated, that value and the current time are stored by the Stats Collector block. At the end of the run, the Stats Collector block computes the user-defined time-average number in the system, as displayed in the table labeled NumberInSystem Statistics.

Although you can use the Number Holder block to collect time-independent data and compute observation-based statistics (such as mean, minimum, and maximum), you should not use the Number Holder block to compute statistics for time-dependent data. In Figure 13.5, a Number Holder block labeled Non-WeightedAvgQueueLength is connected to the OutLength port of the FIFO Queue block and the Display option in the Number Holder block is set to `Mean`. The average queue length computed by the Number Holder block is not time-weighted. The correct time-weighted average queue length is computed by connecting a Stats Collector block to the OutLength port of the FIFO Queue block. This mean value, in the table labeled Time-Weighted Average Queue Length in Figure 13.5, matches the value AvgQLength that is computed by the Queue Stats Collector block, as displayed in the table labeled Queue Stats Collector Results.

Figure 13.5 Stats Collector Block Example
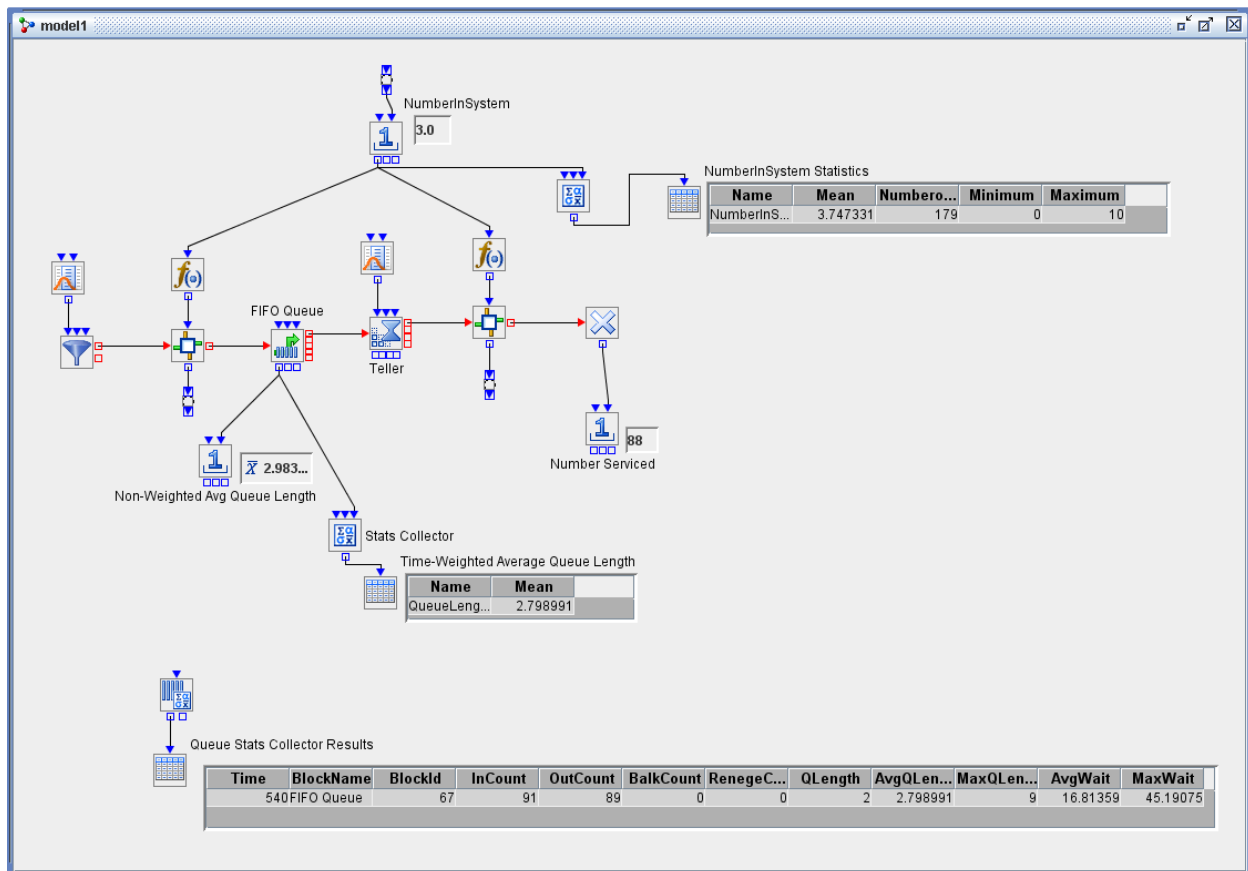
Terminating and Steady-State Simulations

For some systems, a clear and logical time determines the duration of the simulation run. For example, a doctor's office might be open from 8:00 a.m. until 5:00 p.m. Monday through Friday. If you are interested in estimating the average time that a patient waits to see a doctor, then you can run the simulation for nine hours, which corresponds to the length of one day. This type of simulation is called a *terminating* simulation. On the other hand, some systems have no clear end time. For example, you might be interested in estimating the long-run average throughput for a manufacturing facility that operates 12 hours a day, with the work in process carried over to the next day. This type of simulation is called a nonterminating (*steady-state*) simulation. In general, you are interested in the long-run behavior of the system while it is operating normally. Let $X_i : i = 1, 2, \ldots$ denote a stochastic process that represents the sequence of outputs that are generated by a single run of a steady-state simulation. For example, the random variable X_i might represent the time in the system (cycle time) for the ith piece of work to complete all its processing in a simulation of a production facility. If the simulation is in steady-state operation, then the random variables X_i have the same steady-state cumulative distribution function $F_X(x) = \Pr(X_i \leq x)$ for all real x and for $i = 1, 2, \ldots$.

Typically the data for a particular output process that are collected during a single terminating or steady-state simulation run are neither independent nor identically distributed (iid); therefore, classic statistical methods (such as those for computing point and confidence interval estimators) are not applicable. For example, in the bank system model in Figure 13.5, the individual observed waiting times for customers $x_1, x_2, \ldots$ that are collected during a single simulation run are nonstationary and autocorrelated. Since the requirement of iid

observations is violated, you should not use the data $x_1, x_2, \ldots$ from a single simulation run to compute, for example, a confidence interval for the average waiting time at the bank.

For a terminating simulation, suppose that you run k independent replications of the same simulation model so that each replication uses the same initial conditions and a different set of random numbers. Furthermore, you define a response or performance measure, such as the average waiting time, so that for each replication a single value is collected. Then the observed k responses are iid and classic statistical methods can be applied to analyze the data. For example, suppose k replications of the bank system model are run and the average customer waiting time $\bar{x}_j$ is computed for each replication j. A statistically valid point estimate of the mean waiting time in the bank system can then be computed as $\sum_{j=1}^{k} \bar{x}_j / k$.

For terminating simulations, the Experiment window provides the most straightforward method for collecting data and computing basic statistics for defined responses. By default, the Experiment window reports the average response over all replications for each design point. But you can also display the standard deviation, minimum value, or maximum value by right-clicking the column heading for a particular response and selecting the **Summary** menu item. Furthermore, by default each random stream in the model advances to its next substream at the start of a new replication. This guarantees that different random numbers are used for each replication. Alternatively, you can run multiple replications of a model, use the various data collection blocks to collect data for each replication, and then use SAS or JMP to analyze the data.

As in the case of a terminating simulation, the observations $x_1, x_2, \ldots$ from a single long run of a nonterminating simulation are usually correlated. Furthermore, it is usually impossible to start a simulation in steady-state operation. Thus, it is necessary to decide how long the warm-up period should be so that for each simulation output that is generated after the end of the warm-up period, the corresponding expected value is sufficiently close to the steady-state mean. If observations that are generated before the end of the warm-up period are included in the analysis, then any computed point estimator (such as for the steady-state mean) might be biased. You could use a replication/deletion approach to analyze data from a steady-state simulation, similar to the replication method used in the terminating simulation case. You would first make k replications of the simulation, each of length n, and delete the first l observations from each replication. You would then compute the truncated sample mean for each replication as $\sum_{i=l+1}^{n} x_{j,i} / n - l$ for $j = 1, 2, \ldots, k$. By deleting those observations at the beginning of the simulation runs, you eliminate the initial bias due to the model's initial conditions. The replication/deletion method is simple to understand and implement. However, it is computationally inefficient because it requires the deletion of a total of $l \times k$ observations. In addition, it can be difficult to determine how large the warm-up period l should be.

An alternate method to replication/deletion for analyzing data from a steady-state simulation is to make one long simulation run of length n and apply a batch means approach. The Steady State block in Simulation Studio provides an automated batch means method for producing a statistically valid confidence interval estimator for a steady-state mean response in a nonterminating simulation model where the delivered confidence interval satisfies a user-specified precision requirement. The procedure is based on the method of spaced batch means and has an algorithm built in to automatically detect the end of the warm-up period and to address the correlation that exists between observations. For more information about the specific batch means method used by the Steady State block, see Lada, Steiger, and Wilson (2008). For more information about using the Steady State block specifically, see Appendix A, "Templates."

Using the SAS Program Block

The SAS Program block enables you to execute a SAS program or JMP script at any point during a simulation run. This enables you to analyze simulation-generated data automatically either at the end of a run (see the example "Using the SAS Program Block to Analyze Simulation Results" on page 318) or during a run by

sending a signal to the InSubmitCode port of the SAS Program block. For example, in a simulation model of an inventory system it might be necessary, based on the current state of the system, to update a production plan data set. If the number of backlogged orders exceeds a certain level, a SAS Program block can be signaled to execute a program that generates a new production plan data set that is used to set production levels downstream in the model.

Input Analysis

The accuracy of the analysis of any output generated by a simulation model is highly dependent on the appropriateness of the inputs that are used to drive the simulation model. Often, data are available and you want to use those data to estimate the parameters of a theoretical distribution and then sample from that fitted distribution to generate inputs to your model. In this case, you can use the **Fitted** option in the Numeric Source block to access the JMP automatic distribution-fitting procedure. After you specify a value for `File Path` and click the **Fit Distribution** button in a Numeric Source block, JMP automatically fits a series of distributions to the specified data and ranks the results. Either you can select the best fit that is suggested by JMP or you can investigate other distributions and use the analysis options available in JMP to make a distribution choice. After you select a distribution, the parameters for that distribution are automatically passed back to the Numeric Source block. See Appendix D, "Input Analysis," for details.

In addition to selecting a theoretical distribution to generate inputs to a model, the Numeric Source block also enables you to generate samples from discrete and continuous empirical distributions, which can be especially useful when it is difficult to find an appropriate theoretical distribution that accurately represents the data. Finally, you can also use the Numeric Source block to specify a nonhomogeneous Poisson process that is based on either count data or rate data. Simulation Studio uses the count data or rate data to generate a time-dependent arrival process for a specified time interval that can be used as an input to a model. For more information about using empirical distributions and nonhomogeneous Poisson processes, see Appendix B, "Random Variation in a Model."

References

Lada, E. K., Steiger, N. M., and Wilson, J. R. (2008). "SBatch: A Spaced Batch Means Procedure for Steady-State Simulation Analysis." *Journal of Simulation* 2:170–185.

Chapter 14
Batch Execution

Contents

Overview of Batch Execution

Everything in this document so far has focused on using the SAS Simulation Studio GUI to construct and run simulation models. You build your simulation model in the Model window, create your experiment, and then save and run your simulation model. To rerun a saved model, you reload it into a project (along with an associated experiment) and start the model execution process.

Simulation Studio provides an alternative method of running saved models and experiments that does not involve using the GUI: a command line executable program named *simstudio_batch* that enables you to run models in batch mode.

Command Line Interface

You can use the *simstudio_batch* program to run simulation models from a command prompt. The *simstudio_batch* program accepts five command line arguments: **-m**, **-e**, **-d**, **-r**, and **-nonparallel**. The **-m** argument specifies the pathname of a Simulation Studio model that you want to execute, and the **-e** argument specifies the experiment pathname; both the **-m** and **-e** arguments are required. The optional **-d** argument specifies the pathname of the location where you want to save the contents of the Simulation Studio Experiment window.

The optional **-r** argument specifies the location of the simulation output data. When data from Simulation Studio are saved (either in batch mode or interactively), the current contents of the specified output results folder are deleted when model execution begins. If you use the **-r** option, *you should specify a location that is dedicated to storing simulation output data in order to prevent accidental deletion of files*. If you do not use the **-r** option, then any data collected during the simulation execution are saved in a hierarchical folder named `results` that is created in the directory where the *simstudio_batch* program was launched.

If you want to load SAS data, save SAS data, or submit SAS code locally during your batch run, you need to have a SAS Workspace Server running locally on your computer. The default local workspace server on the Windows operating system does not require configuration or credentials and is available when you are running in batch mode. If you are using the Linux operating system or you are using the SAS Simulation Studio Object Spawner for SAS services instead of the default local SAS Workspace Server, then you

must first launch Simulation Studio, provide the local port number and your local credentials, and leave Simulation Studio open before running your model in batch mode. Similarly, if you are using a remote SAS Workspace Server, then you must first launch Simulation Studio, provide your remote host credentials, and leave Simulation Studio open before running your model in batch mode. If you want to save data as a JMP table, load a JMP table, or submit JMP script during your batch run, then you must manually launch the JMP server before running your model in batch mode. For more information about launching a local SAS or JMP server, see the section "Launching Local SAS and JMP Servers" on page 21.

In batch mode, different design points and replications are distributed on different cores simultaneously by default if you are running the model on a machine that has a multicore processor. This is equivalent to selecting the **Parallel Mode** icon on the toolbar in the GUI. If you include the optional argument **-nonparallel** in your batch run command, then one single core is used to run the model. In parallel mode, SAS should be used to read or write data during a simulation run and to execute code by using the SAS Program block. If you are using JMP to load data, write data, or execute a JSL script, then you should use the **-nonparallel** option when running your model in batch mode.

To invoke Simulation Studio in batch mode on the Windows operating system, open a Microsoft Windows command prompt window and navigate to the location of the executable *simstudio_batch*. The current default installation location is \Program Files\SASHome\SASSimulationStudio\<release_number>. A sample command line for executing a model-experiment pair looks like this:

```
simstudio_batch –m projects\MyProject\MyModel.simmdl –e projects\
MyProject\MyExperiment.simexp –d projects\MyProject\experiment.sas7bdat
–r projects\MyProject\MyResults
```

For the Linux operating system, you invoke Simulation Studio in batch mode from the installation directory in a terminal window by using the command ./simulation_batch. The default installation location is /usr/local/SASHome/SASSimulationStudio/<release_number>. A sample command line for executing a model-experiment pair on the Linux operating system looks like this:

```
./simstudio_batch –m /usr/local/SASHome/SASSimulationStudio/14.3/projects/
examples/docMM1/model1.simmdl –e /usr/local/SASHome/SASSimulationStudio/14.3/
projects/examples/docMM1/experiment0.simexp
```

Log Messages

Any log messages that are generated during the execution of a model are directed to the command prompt window.

Appendix A
Templates

Contents

Overview of Templates

Simulation Studio templates provide collections of blocks you can use to build simulation models. The following sections are overviews of the blocks provided in the various Simulation Studio templates. Each block description includes a brief summary of what the block does along with a description of the fixed ports for the block and the controls in the block's properties dialog box. Also included are the Factor and Response candidates associated with each block for use with the design-of-experiment features in Simulation Studio.

Overview of the Standard Template

The Simulation Studio Standard template provides a fundamental collection of the blocks that are most commonly used to build simulation models.

Entity Generator Block

Description

The Entity Generator block generates entities. You can control when the entities are created, the total number of entities created, and how many entities are created simultaneously.

After an entity is created, the Entity Generator block attempts to send the new entity out the OutEntity port. If this fails, it then tries to push the entity out the OutBalk port. If this also fails, the entity is destroyed and a message is sent to the Tracer.

Multiple entities can be generated every time an entity creation event occurs in an Entity Generator block. The number of entities to create at an entity creation event is referred to as the *batch size*. When the Entity Generator block is preparing to schedule an entity creation event, it attempts to pull a value from its BatchSize port and associate this value with the entity creation event. (If nothing is connected to the BatchSize port, it uses a default batch size of 1.) When the entity creation event occurs in the Entity Generator block, the Entity Generator block creates the number of entities specified by the associated batch size value (within the constraints of the maximum limits described in the next paragraph). All entities are sent out individually either through the OutEntity port or the OutBalk port.

You can specify the maximum number of entities that the Entity Generator block can generate in addition to the maximum number of batches. The Entity Generator block stops creating entities whenever either of these limits is reached. Fields are also provided to set the start and end time (in terms of the simulation clock) for controlling the duration of operation of the block.

The Boolean Signal port can be used to initiate entity creation as well. When a true value arrives at the Signal port, the Entity Generator block pulls values from its InterArrivalTime and BatchSize ports and schedules an entity creation event.

You can use the Entity Types dialog box to specify the types of entities the Entity Generator block can create. To open the Entity Types dialog box, right-click in the Project Explorer window and select **Entity Types**. You can enter default values for any of the editable entity attribute fields (indicated by a check in the **Editable** column) in the Entity Types dialog box.

Fixed Ports

InterArrivalTime   Input numeric port for how long to wait before the next entity creation event.

BatchSize   Input integer port for how many entities to create at the next entity creation event.

Signal   Input Boolean port that schedules an entity creation event (when true is passed in).

OutEntity   Output entity port for entities that can be accepted by a downstream block.

OutBalk   Output entity port for entities that cannot leave using the OutEntity port.

Attributes Dialog Box Controls

Limits	The **Maximum Number of Entities** field specifies the maximum number of entities this Entity Generator block is permitted to generate. Selecting the **Infinite** check box supersedes the value of the **Maximum Number of Entities** field. Similarly, the **Maximum Number of Batches** field specifies the maximum number of batches of entities this Entity Generator block is permitted to generate. Selecting the **Infinite** check box supersedes the value of the **Maximum Number of Batches** field. By default, both **Infinite** check boxes are checked. If both the **Maximum Number of Entities** field and the **Maximum Number of Batches** field contain valid values, the Entity Generator block stops creating new entities as soon as either of the criteria has been met.
Timing	The **Start Time** field designates the simulation time at which the first entity is generated by this Entity Generator block. This value must be greater than or equal to 0. The default **Start Time** is 0. Similarly, the **End Time** field specifies the simulation time when no more entities can be generated by the Entity Generator block. The **End Time** must be greater than or equal to the **Start Time**. Selecting the **Infinite** check box supersedes the value of the **End Time** field.
First Entity Creation	Specifies when the first entity is created by the Entity Generator block. Select **At Start Time** to cause the first entity to be created at the time specified in the **Start Time** field. This is the default selection. If you select **At First Interarrival Time**, then at **Start Time** the Entity Generator block pulls the first interarrival time value from the InterArrivalTime port and schedules the first entity to be created at that time. The pulled value determines how long the Entity Generator block waits before generating the first entity. (Whenever the interarrival time value is not a number, the simulation terminates. If the value is a number less than 0, the Entity Generator block logs a warning and uses a value of 0.) If you select **After Signal Arrival**, the Entity Generator block waits until a true value arrives at the Signal port before scheduling the first entity creation.
To Schedule the Creation of Next Entity	If you check this check box, after the Entity Generator block has created a new entity and pushed it downstream it automatically pulls a value from its InterArrivalTime port and uses this value to schedule the generation of the next entity creation. If you clear this check box, future entity creation events can be scheduled only by using the Signal port. By default, this check box is checked.

EntityType Dialog Box Controls

Name	Specifies the name of the EntityType used for entity creation.
Fields	Displays the default attributes associated with the selected EntityType. You can set the default value for editable entity attributes directly in the table.

Candidates for Design of Experiments

Factors	StartTime (double), EndTime (double), MaxEntities (integer), MaxBatches (integer), RankValue (double)
Responses	None

Value Generator Block

InterValueTime ┐ ┌ InValue
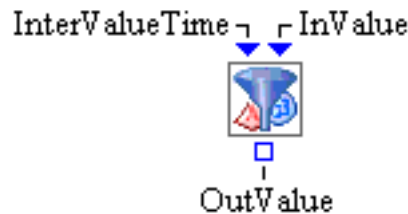
OutValue

Description

The Value Generator block generates numeric, text, or Boolean values. The Value Generator block pulls a value from its InterValueTime port to determine how long it waits before generating the next value. (If the intervalue time value is not a number, the simulation terminates. If the value is less than 0, the Value Generator block logs a warning and uses a value of 0.) After the Value Generator block has a valid intervalue time value, it pulls a value from its InValue port and passes it out the OutValue port. If there are no connections to the InValue port, the value specified in the **Default Value** field is passed out the OutValue port.

You can specify the maximum number of values the Value Generator block can generate, the default value generated, and the start and end times (in terms of the simulation clock) for controlling the operation of the block. You can also specify when the first value is created.

Fixed Ports

InterValueTime	Input numeric port for how long to wait before the next value creation event.
InValue	Input value port for the next value to create.
OutValue	Output value port for the created values.

Properties Dialog Box Controls

Values	The **Maximum Number of Values** field specifies the maximum number of values the Value Generator block is permitted to generate. Selecting the **Infinite** check box supersedes the value in the **Maximum Number of Values** field. The **Value Type** field specifies the type of value that the Value Generator block generates. The **Default Value** field specifies the value to use when the InValue port has no connections.

Timing	The **Start Time** field designates the simulation time at which the first value is generated by the Value Generator block. This value must be greater than 0. Similarly, the **End Time** field specifies the simulation time when no more values can be generated by the Value Generator block. This must be greater than or equal to the **Start Time**. Selecting the **Infinite** check box supersedes the value in the **End Time** field.
First Value	Determines when the first value is created by the Value Generator block. Select **Start Time** to cause the first value to be created at the time specified in the **Start Time** field. If you select **First Intervalue Time**, then at the **Start Time** the Value Generator block pulls the first intervalue time value from the InterValueTime port and schedules the first value to be created at that time.

Candidates for Design of Experiments

Factors	StartTime (double), EndTime (double), MaxValues (integer), RankValue(double)
Responses	None

Disposer Block
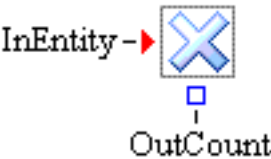
Description

The Disposer block disposes of entities after they are no longer needed in the model, reducing memory usage. For optimal memory management, you should route entities that are no longer needed to a Disposer block. It is possible to have more than one Disposer block in a model. The disposer block keeps a count of the number of entities that enter the block. If there are connections to the block's OutCount port, the count is pushed out the port every time its value changes.

Fixed Ports

InEntity	Input entity port for entities to be disposed.
OutCount	Output integer port for the number of entities that have been disposed.

Candidates for Design of Experiments

Factors	None
Responses	OutCount (integer)

Queue Block
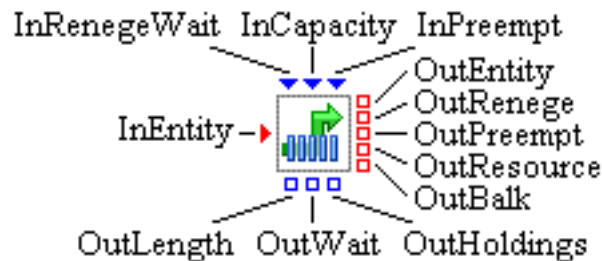
Description

The Queue block is used for transient storage of entities. Three types of queueing policies are available for a Queue block: FIFO, LIFO, and Priority.

When a request to send (or push) an entity arrives at a Queue block, the Queue block determines whether it has room to store the entity. If its buffer is full, the Queue block rejects the request to have the entity sent to it. If space is available in its buffer, the Queue block responds that it can accept the entity.

When an entity arrives at a Queue block that uses a FIFO or LIFO queueing policy, the entity is stored at the appropriate end of the buffer. For Queue blocks that use a Priority policy, the Queue block extracts the priority value from an attribute defined for the entity and uses that value to determine where to place the entity in the buffer. The Queue block then sequentially notifies each block connected to its OutEntity port to ask whether it is ready to receive an entity. The Queue block selects an entity (based on the queueing policy—FIFO, LIFO, or Priority) to send out through the OutEntity port to the first downstream block that responds affirmatively. (Entities can also be pulled out through the Queue block's OutEntity port by a downstream block. In this case the Queue block also selects the entity to release according to the queueing policy.)

When a Queue block's buffer is no longer full (due to an entity leaving the Queue block or the Queue block's capacity being increased), the Queue block attempts to pull entities from upstream through the InEntity port until it is at capacity or no entities are available to pull.

If the Queue block's reneging option is activated (by selecting the **Reneging** option in the properties dialog box), then after an entity enters the Queue block, the Queue block attempts to pull a numeric value from its InRenegeWait port. If the Queue block pulls a nonnegative number from the port, it schedules a time for the entity to exit the Queue block via the OutRenege port if the entity is still in the Queue block's buffer at that time. Otherwise no time for reneging is scheduled. If there is no connection to the OutRenege port, no reneging occurs.

Any time an entity enters or exits the Queue block, the Queue block pushes the value of its buffer's length (the number of entities being held by the Queue block) to the OutLength port. Any time an entity exits the Queue block via its OutEntity port, the Queue block pushes a value that represents how long that entity waited in the buffer to the OutWait port.

An integer value can be pushed through the InCapacity port to set the *capacity* for the Queue block (the size of its buffer). Valid incoming values for this port are integers in the range of 0 to 2,147,483,647. If the capacity of a Queue block is reduced dynamically during the simulation run, any excess entities are removed from the Queue block (according to the queueing policy being used) and are sent out through the OutBalk port. If there are no connections to the OutBalk port, the entities are destroyed.

Holding Block Preemption

Entities in Simulation Studio are hierarchical. That is, entities can hold other entities. The term *controlling entity* denotes an entity that holds other entities, and the term *root entity* denotes an entity that is not held by another entity. Each entity held by another entity has one root entity associated with it. The root entity for any held entity is found by traversing up the entity hierarchy from the held entity.

Entities being held by a Queue block can be preempted either by input to the block's InPreempt port or by a scheduled resource entity event. In order for a root entity that is held by a Queue block to be preempted, the OutPreempt port (or OutBalk port) must have at least one link attached to it. Similarly, for a resource entity that is held by a controlling entity that is in turn held by the Queue block to be preempted, the OutResource port (or OutBalk port) must have at least one link connected to it.

The Queue block's InPreempt port accepts an Entity Group object as input. (An Entity Group is a collection of references to entities.) When an Entity Group object is pushed to a Queue block's InPreempt port, the Queue block iterates through the Entity Group collection looking for matches to root entities held by the Queue block. For any matched entity, the Queue block first tries to push that entity out its OutPreempt port. If this push is not successful, the block attempts to push the entity out the OutBalk port. If this also fails, the entity continues to be held by the Queue block until either it exits out the OutEntity port or it is preempted again.

The Queue block, like all entity holding blocks, detects potential preemptive changes (such as those scheduled by a Resource Scheduler block) to resource entities it holds (either directly or indirectly through a controlling entity).

If the number of units associated with a held resource entity decreases or the state of a held resource entity becomes nonfunctional, the Queue block attempts to preempt that resource entity. If the resource entity identified for preemption is a root entity, then the Queue block follows the same protocol for pushing an entity out its OutPreempt port that the InPreempt port uses. If the resource entity is part of a controlling entity, the Queue block removes the resource entity from the controlling entity and attempts to push the associated root entity out the OutPreempt port. The Queue block then attempts to push the preempted resource entity out its OutResource port, or if that fails, out its OutBalk port. If there is a connection to the Queue block's OutResource port and the Queue block cannot push the resource entity out either the OutResource or OutBalk port, the resource entity is disposed.

The Queue block also provides an OutHoldings port that other blocks can use to pull an Entity Group object that contains a collection of references to entities held by the Queue block.

Fixed Ports

InEntity   Input entity port for entities to be added to the Queue block.

OutEntity   Output entity port for entities that can be accepted by a downstream block.

OutRenege   Output entity port for entities that are reneged and can be accepted by a downstream block.

OutPreempt	Output entity port for root entities that are preempted and can be accepted by a downstream block.
OutResource	Output entity port for resource entities held by controlling entities that are preempted and can be accepted by a downstream block.
OutBalk	Output entity port for entities that cannot leave using the other output entity ports.
InRenegeWait	Numeric input value port that sets the amount of time to wait before an entering entity is reneged.
InCapacity	Numeric input value port that dynamically sets the capacity of the Queue block.
InPreempt	Entity Group input port that causes the Queue block to preempt any root entities it is holding that match entities in the incoming Entity Group.
OutLength	Numeric output value port for the number of root entities held in the Queue block's buffer.
OutWait	Numeric output value port for the amount of time an exiting entity waited in the Queue block's buffer.
OutHoldings	Entity Group output port from which a group of entity references can be pulled, representing the entities in the Queue block's buffer.

Properties Dialog Box Controls

Capacity Specifies the maximum number of entities the Queue block is permitted to store in its buffer. Valid values for this field are integers in the range from 0 to 2,147,483,647. Selecting the **Infinite** check box supersedes any **Capacity** value.

Reneging Selecting this check box activates the Queue block's automatic reneging functionality.

Queueing Policy Selecting a policy type in the **Type** list box specifies the queueing policy that is used by this Queue block in determining the order in which entities leave the Queue block. Some policies have additional parameters that can be specified when the policy is selected from the list box. The **FIFO** policy has no parameters and uses a first-in-first-out policy for determining the order of entities leaving the Queue block. The **LIFO** policy has no parameters and uses a last-in-first-out policy for determining the order of entities leaving the Queue block. The **Priority** policy allows entities to exit the Queue block based on entity priority. It has the following parameters:

Entity Attribute Type Specifies the type of the attribute (Number or String) used to extract the priority value from an entity.

Default Priority Number If Entity Attribute Type is Number, this field specifies the numeric priority value to use for an entity when the Queue block cannot extract a valid priority value from the specified Entity Attribute.

Default Priority String If Entity Attribute Type is String, this field specifies the string priority value to use for an entity when the Queue block cannot extract a valid priority value from the specified Entity Attribute.

Entity Attribute Specifies the name of the attribute to use when extracting the priority value from an entity.

Priority Order Specifies whether higher values or lower values are interpreted to have a higher priority.

Tie Breaking Policy   Specifies the algorithm to use for placing entities in the Queue block's buffer when entities have the same priority value. Algorithm options include FIFO, LIFO, and Random (for random placement).

Random Stream Seed   If the value of **Tie Breaking Policy** is Random, this field specifies the random number generator seed.

Extended Queueing Controls   Adjust how entities flow into and out of a Queue block by using the following parameters and options. For a depth or threshold value in general, an **Absolute** value specifies an absolute entity count and a **Relative** value specifies a value between 0.0 and 1.0, representing a proportion of a Queue block's length relative to its full queue capacity, as specified in the **Capacity** option.

Nonblocking Push Depth   Specifies how deep the Queue block will search to identify an entity to push downstream when a new entity enters the Queue block from upstream, starting at the beginning of the entity holding list as defined by the queueing policy. This value must be ≥ 0.0; the default is 1.0.

Nonblocking Pull Depth   Specifies how deep the Queue block will search to identify an entity to send (or push) downstream when there is a new pull request from another block, starting at the beginning of its entity holding list as defined by the queueing policy. This value must be ≥ 0.0; the default is 1.0.

Active Pull Threshold   Specifies a queue length threshold so that when the queue length falls to this threshold value, the Queue block will attempt to pull more entities from upstream to increase the holding level. If this value is ≥ 0.0, then the threshold is measured from the back, or end, of the queue when the queue is empty. If the threshold value is < 0.0, then the threshold is measured from the front, or beginning, of the queue when the queue is at full capacity. The default is -1.

Active Pull Count   Specifies the maximum number of active pull attempts that the Queue block can conduct each time the active pull threshold is reached. Selecting **Pull to Full Capacity** supersedes any fixed count value.

Active Pushing to Downstream   Activates the Queue block to automatically push the next entity out after the entity currently at the front of the queue successfully leaves the Queue block.

Candidates for Design of Experiments

Factors   Capacity (integer), RankValue (double), QueueingPolicy (string)
The format for specifying the value of the QueueingPolicy factor is as follows:
Type==*policyType*;Entity Attribute Type==*attributeType*;Default Priority Number== *priorityNumber*;Default Priority String==*priorityString*;Entity Attribute== *attributeName*;Priority Order==*priorityOrder*;Tie Breaking Policy==*tieBreakingPolicy*; Random Stream Seed==*seed*
where:

policyType	is FIFO, LIFO, Priority, or the fully-qualified Java class name of a queueing policy class.
attributeType	is Number or String.
priorityNumber	is a decimal number.
priorityString	is a string value.
attributeName	is the name of an entity attribute.
priorityOrder	is one of the following: Highest Value Has Highest Priority, Lowest Value Has Highest Priority.
tieBreakingPolicy	is one of the following: FIFO, LIFO, Random.
seed	is an integer number.

Each name==value parameter within the factor value is optional. If it is not specified, it is assigned the value specified in the properties dialog box if possible; otherwise it is assigned a default value.

Responses   AverageWait (double), MaximumWait (double), AverageLength (double), Maximum-Length (integer), BalkCount (integer), RenegeCount (integer)

Delay Block
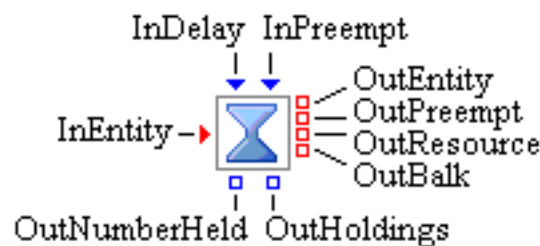
Description

The Delay block delays the progression of an entity through a simulation model. When an entity enters a Delay block via its InEntity port, the Delay block pulls a value (the delay time) from its InDelay port. If the delay time value is not a number, the simulation terminates. If the value is less than 0, the Delay block logs a warning and uses a value of 0. The Delay block holds the entity for the duration of the delay time and then releases it through its OutEntity port. If the push through the OutEntity port fails, the Delay block attempts to push the entity out the OutBalk port. If this is not successful, the entity is destroyed and a message is posted to the Tracer.

Any time an entity enters or exits the Delay block, the Delay block pushes the number of entities held by the block to the OutNumberHeld port.

Holding Block Preemption

Entities in Simulation Studio are hierarchical. That is, entities can hold other entities. The term *controlling entity* denotes an entity that holds other entities, and the term *root entity* denotes an entity that is not held by another entity. Each entity held by another entity has one root entity associated with it. The root entity for any held entity is found by traversing up the entity hierarchy from the held entity.

Entities being held by a Delay block can be preempted either by input to the block's InPreempt port or by a scheduled resource entity event. In order for a root entity that is held by a Delay block to be preempted, the OutPreempt port (or OutBalk port) must have at least one link attached to it. Similarly, for a resource entity that is held by a controlling entity that is in turn held by the Delay block to be preempted, the OutResource port (or OutBalk port) must have at least one link connected to it.

The Delay block's InPreempt port accepts an Entity Group object as input. (An Entity Group is a collection of references to entities.) When an Entity Group object is pushed to a Delay block's InPreempt port, the Delay block iterates through the Entity Group collection looking for matches to root entities held by the Delay block. For any matched entity, the Delay block first tries to push that entity out its OutPreempt port. If this push is not successful, the block attempts to push the entity out the OutBalk port. If this also fails, the entity continues to be held by the Delay block until either it exits out the OutEntity port or it is preempted again.

The Delay block, like all entity holding blocks, detects potential preemptive changes (such as those scheduled by a Resource Scheduler block) to resource entities it holds (either directly or indirectly through a controlling entity).

If the number of units associated with a held resource entity decreases or the state of a held resource entity becomes nonfunctional, the Delay block attempts to preempt that resource entity. If the resource entity identified for preemption is a root entity, then the Delay block follows the same protocol for pushing an entity out its OutPreempt port that the InPreempt port uses. If the resource entity is part of a controlling entity, the Delay block removes the resource entity from the controlling entity and attempts to push the associated root entity out the OutPreempt port. The Delay block then attempts to push the preempted resource entity out its OutResource port, or if that fails, out its OutBalk port. If there is a connection to the Delay block's OutResource port and the Delay block cannot push the resource entity out either the OutResource or OutBalk port, the resource entity is disposed.

The Delay block also provides an OutHoldings port that other blocks can use to pull an Entity Group object that contains a collection of references to entities held by the Delay block.

Fixed Ports

InEntity	Input entity port for entities to be added to the Delay block.
OutEntity	Output entity port for entities that can be accepted by a downstream block.
OutPreempt	Output entity port for controlling (parent) entities that are preempted and can be accepted by a downstream block.
OutResource	Output entity port for resource entities held by controlling entities that are preempted and can be accepted by a downstream block.
OutBalk	Output entity port for entities that cannot leave using the other output entity ports.
InDelay	Input numeric port for how long the Delay block should delay the next entity.

InPreempt   Entity Group input port that causes the Delay block to preempt any controlling entities it is holding that match entities in the incoming Entity Group.

OutHoldings   Entity Group output port from which a group of entity references can be pulled, representing the entities held by the Delay block.

Candidates for Design of Experiments

Factors   RankValue (double)

Responses   None

Server Block
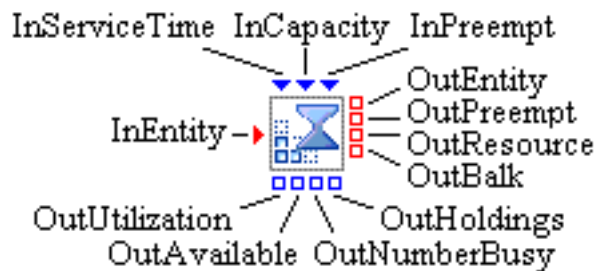
Description

The Server block models a resource used by an entity for a specified period of time. An entity can enter a Server block only when the Server block is not busy. A Server block is deemed *busy* if all of its capacity is being used to service entities. After an entity enters the Server block, the Server block pulls a value from its InServiceTime port. If the service time value is not a number, the simulation terminates. If the value is less than 0, the Server block logs a warning and uses a value of 0. The entity is held for the duration of the service time and then released through the OutEntity port.

The InCapacity port can be used to set the *capacity* for a Server block. This value represents the number of entities the Server can service simultaneously. Valid incoming values for this port are integers in the range of 0 to 2,147,483,647. During simulation execution, an integer value can be pushed through the InCapacity port to dynamically change the capacity. If the value from the port is less than the currently busy capacity, the capacity reduction request will be deferred and accommodated as entities finish service and leave the Server block. Entities will not be balked or preempted in this case.

Holding Block Preemption

Entities in Simulation Studio are hierarchical. That is, entities can hold other entities. The term *controlling entity* denotes an entity that holds other entities, and the term *root entity* denotes an entity that is not held by another entity. Each entity held by another entity has one root entity associated with it. The root entity for any held entity is found by traversing up the entity hierarchy from the held entity.

Entities being held by a Server block can be preempted either by input to the block's InPreempt port or by a scheduled resource entity event. In order for a root entity that is held by a Server block to be preempted, the OutPreempt port (or OutBalk port) must have at least one link attached to it. Similarly, for a resource entity

that is held by a controlling entity that is in turn held by the Server block to be preempted, the OutResource port (or OutBalk port) must have at least one link connected to it.

The Server block's InPreempt port accepts an Entity Group object as input. (An Entity Group is a collection of references to entities.) When an Entity Group object is pushed to a Server block's InPreempt port, the Server block iterates through the Entity Group collection looking for matches to root entities held by the Server block. For any matched entity, the Server block first tries to push that entity out its OutPreempt port. If this push is not successful, the block attempts to push the entity out the OutBalk port. If this also fails, the entity continues to be held by the Server block until either it exits out the OutEntity port or it is preempted again.

The Server block, like all entity holding blocks, detects potential preemptive changes (such as those scheduled by a Resource Scheduler block) to resource entities it holds (either directly or indirectly through a controlling entity).

If the number of units associated with a held resource entity decreases or the state of a held resource entity becomes nonfunctional, the Server block attempts to preempt that resource entity. If the resource entity identified for preemption is a root entity, then the Server block follows the same protocol for pushing an entity out its OutPreempt port that the InPreempt port uses. If the resource entity is part of a controlling entity, the Server block removes the resource entity from the controlling entity and attempts to push the associated root entity out the OutPreempt port. The Server block then attempts to push the preempted resource entity out its OutResource port, or if that fails, out its OutBalk port. If there is a connection to the Server block's OutResource port and the Server block cannot push the resource entity out either the OutResource or OutBalk port, the resource entity is disposed.

The Server block also provides an OutHoldings port that other blocks can use to pull an Entity Group object that contains a collection of references to entities held by the Server block.

Fixed Ports

InEntity	Input entity port for entities to enter the Server block.
OutEntity	Output entity port for entities that can be accepted by a downstream block.
OutPreempt	Output entity port for controlling entities that are preempted and can be accepted by a downstream block.
OutResource	Output entity port for resource entities held by controlling entities that are preempted and can be accepted by a downstream block.
OutBalk	Output entity port for entities that cannot leave using the other output entity ports.
InServiceTime	Input numeric port for how long the next entity should remain in the Server block.
InCapacity	Input integer port for the number of entities the Server block can service simultaneously.
InPreempt	Entity Group input port that causes the Server block to preempt any controlling entities it is holding that match entities in the incoming Entity Group.
OutUtilization	Output numeric port for the current utilization of the Server block's capacity.
OutAvailable	Output integer port for the Server block's capacity that is not currently busy.
OutNumberBusy	Output integer port for the Server block's capacity that is currently busy.
OutHoldings	Entity Group output port from which a group of entity references can be pulled, representing the entities held by the Server block.

Properties Dialog Box Controls

Values   The **Capacity** field specifies the capacity of the Server block.

Candidates for Design of Experiments

Factors   RankValue (double), Capacity (integer)

Responses   AvgUtilization (double), MaxUtilization (double)

Modifier Block

Description

The Modifier block assigns attributes to an entity as it passes through the block. Each attribute has an input value port associated with it. When an entity enters the block, values are pulled from the input value ports and assigned to the associated attributes in the entity. If there is no connection to an input value port, then the Modifier block assigns the default value that is specified for the attribute.

You identify the attributes to be assigned to the entity using one of the assignment options described in the "Attributes Dialog Box Controls" section that follows.

Fixed Ports

InEntity   Input entity port for entities to enter the Modifier block.

OutEntity   Output entity port for entities that can be accepted by a downstream block.

OutBalk   Output entity port for entities that cannot leave by using the OutEntity port.

Attributes Dialog Box Controls

Specify the attributes to be assigned to the incoming entity.

The Modifier block provides two options for assigning entity attributes. The first assignment option is based on independent value inputs. Each attribute has an input value port associated with it. Different attributes are associated with different input value ports for assigning values.

The second assignment option is based on an Observation value input. All attributes that are assigned to an incoming entity are the variable values from the same Observation input value. One Observation input value port is provided for this option.

When you select the **Assigned from: Independent Value Inputs** option, the associated **Attributes** table and editing buttons are enabled. You define a new attribute by clicking the **Add** button beside the **Attributes** table. This results in a new attribute (with a default name and value) being added to the **Attributes** table. You can edit the name, type, and default value of the attribute directly in the table. The attribute names that a Modifier block uses must be unique. You can change the attribute type by using a drop-down box on the type table cell. You can delete attributes from the **Attributes** table by selecting the attribute row in the table and then clicking the **Remove** button. Clicking the **Apply** button pushes all entries in the **Attributes** table to the Modifier block, and input ports are created or deleted as needed. You use the **Move Up** and **Move Down** buttons to reorder the attributes in the **Attributes** table. The order of the input value ports on the Modifier block is adjusted to be the same as the order of the attributes in the **Attributes** table.

Add	Adds a new attribute with a default name, type, and default value to the **Attributes** table. You can change the name, type, and default value of the attribute directly in the table. The attribute names in the Modifier block's **Attributes** table must be unique. You can change the attribute type through a drop-down box on the cell in the table. (You cannot change an attribute **Type** in the table after you click the **Apply** button. To change an attribute type after clicking **Apply**, you must remove the attribute, add it again, and then modify the type of the newly added attribute before clicking **Apply** again.)
Remove	Deletes the selected attribute from the **Attributes** table.
Move Up	Moves the row for a selected attribute up in the **Attributes** table. The value port that is associated with the selected attribute is moved to the left.
Move Down	Moves the row for a selected attribute down in the **Attributes** table. The value port that is associated with the selected attribute is moved to the right.
Apply	Updates all attributes in the Modifier block as specified in the **Attributes** table, and creates or deletes value ports as needed.

When you select the **Assigned from: An Observation Value Input** option, the following assignment options are available:

Full Assignment	Indicates that all variable values from an input observation are assigned as different entity attributes. The attribute names are the same as the corresponding variable names.
Selected Assignment	Indicates that only the attributes identified by the user will be assigned to an incoming entity by using the variable values from the input observation. An **Assignment** table and editing buttons are provided to specify the entity attributes and the associated observation variables as assignment pairs. Each row of the **Assignment** table corresponds to such a pair. For each pair, you must specify the attribute name. If the observation variable name is left blank or not specified, the attribute name is used as the variable name during the simulation.
Excluded Assignment	Indicates that specific observation variables identified by the user are to be excluded. All other variable values from an input observation are assigned as different entity attributes. An **Excluded Observation Variable** table and editing buttons are provided to specify the names of the variables to exclude.

Candidates for Design of Experiments

Factors	None
Responses	None

Extractor Block

Description

The Extractor block extracts attribute values from an entity as it passes through the block. Each attribute has an output value port associated with it. When an entity enters the block, entity attribute values are retrieved from the entity and pushed to their respective output value ports.

You can also connect an Extractor block's output value port to an input value port of another block without any connections to the Extractor block's InEntity port. For example, you can connect an output value port of an Extractor block to the InServiceTime port of a Server block. After an entity enters the Server block, it is passed to the Extractor block (via the InServiceTime port) to extract the appropriate entity attribute value to be used for the InServiceTime value.

You identify attributes to be extracted from the entity by clicking the **Add** button on the Properties dialog box **Attributes** page. This results in a new attribute entry (with a default name) being added to the **Attributes** table. You can edit the name and type of the attribute directly in the table. The attribute names that are listed in the Extractor block's **Attributes** table must be unique. You can change the attribute type through a drop-down box on the type table cell. You can delete attributes from the **Attributes** table by selecting the attribute row in the table and then clicking the **Remove** button. Clicking the **Apply** button pushes all entries in the **Attributes** table to the Extractor block, and output ports are created or deleted as needed. (Note: You cannot change an attribute's type after you close the Properties dialog box.) You use the **Move Up** and **Move Down** buttons to reorder the attributes in the **Attributes** table. The order of the output value ports on the Extractor block is adjusted to be the same as the order of the attributes in the **Attributes** table.

Fixed Ports

InEntity	Input entity port for entities to enter the Extractor block.
OutEntity	Output entity port for entities to exit the Extractor block.

Properties Dialog Box Controls

Add	Adds a new attribute with a default name and type to the **Attributes** table. You can change the name and type of the attribute directly in the table. The attribute names in the Extractor block's **Attributes** table must be unique. You can change the attribute type through a drop-down box on the cell in the table. (You cannot change an attribute **Type** in the table after you click the **Close** button. To change an attribute type after you click **Close**, you must remove the attribute, add it again, and then modify the type of the newly added attribute.)
Remove	Deletes the selected attribute from the **Attributes** table.
Move Up	Moves the row for a selected attribute up in the **Attributes** table. The output value port that is associated with the selected attribute is moved to the left.
Move Down	Moves the row for a selected attribute down in the **Attributes** table. The output value port that is associated with the selected attribute is moved to the right.
Apply	Updates all attributes in the Extractor block as specified in the **Attributes** table, and creates or deletes value ports as needed.

Candidates for Design of Experiments

Factors	None
Responses	None

Switch Block
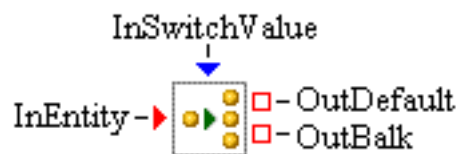
Description

The Switch block directs the flow of an entity through a simulation model. You define switch cases on the Switch block. The case names must be unique, and each switch case must have an integer value (called the *switch value*) associated with it. When an entity enters a Switch block, the block calculates the switch value to be used for the entity. Depending on the Switch block configuration, the block either attempts to extract the switch value from an attribute in the entity or to pull it from the InSwitchValue port. After the switch value is acquired, the Switch block searches the cases in its **Cases** table until it finds a case with the same switch value. The entity is then pushed out through the entity out port associated with the matching case. If a match is not found, the entity is sent out through the OutDefault port.

Fixed Ports

InEntity   Input entity port for entities to enter the Switch block.

OutDefault   Output entity port for entities that do not match a switch case defined for the Switch block.

OutBalk   Output entity port for entities that cannot leave using the other output entity ports.

InSwitchValue Input integer port for the switch value to be used for the next entity, if the Switch block is configured to obtain the switch value from this port rather than from an entity attribute.

Properties Dialog Box Controls

Add   Adds a new switch case to the **Cases** table with a default name and value. You can edit the name and value of the switch case directly in the table. The case names and values used by a Switch block must be unique.

Remove   Deletes the selected switch case from the **Cases** table.

Switch Value If you select the **Port** option, the Switch block pulls the switch value from the InSwitch-Value port. If you select the **Entity** option, you must also supply the name of an attribute in the **Entity Attribute** field. The Switch block attempts to extract the switch value from the appropriate attribute on the entity.

Apply   All values in the **Cases** table are saved to the Switch block and any entity output ports are created or deleted as needed.

Candidates for Design of Experiments

Factors   None

Responses   None

Selector Block
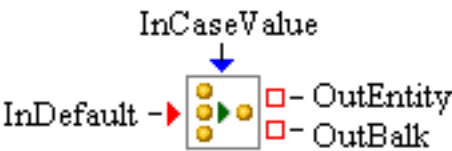
Description

The Selector block selects and outputs entities from one of its input entity ports based on its case values. Each input entity port is associated with a unique case value. Every time the block receives a request from downstream to output an entity, the InCaseValue input value port checks the current case value to determine which input entity port to use to fetch an entity. Similarly, when an entity from upstream attempts to enter the

block through one of the input entity ports, the InCaseValue input value port checks the current case value to verify that its value matches with the input entity port.

By default, the Selector block provides one input entity port named InDefault. You can create additional input entity ports in the properties dialog box by adding new cases to the **Cases** table. Each entry in the table specifies the case's **Name** and **Value**. At experiment run time, the **Value** is compared to the current InCaseValue. If the two match, the corresponding input entity port is active. If the current case value from the InCaseValue port does not match any case value in the **Cases** table, an entity is allowed only to enter or be pulled through the InDefault input entity port.

The case names and values must be unique within each Selector block.

Fixed Ports

InDefault	Input entity port that allows entities to enter the block if the value pulled from the InCaseValue input value port does not match any cases in the Selector block.
OutEntity	Output entity port for entities to leave.
OutBalk	Output entity port for entities to leave that cannot leave using the OutEntity port.
InCaseValue	Numeric input value port used to determine which of the input entity ports allows entities to enter.

Properties Dialog Box Controls

Add	Adds a new case to the **Cases** table with a default name and value. The name and value can be edited directly in the table.
Remove	Deletes the selected case from the **Cases** table.
Apply	All values in the **Cases** table are saved to the Selector block, and any input entity ports are created or deleted as needed.

Candidates for Design of Experiments

Factors	None
Responses	None

Number Holder Block
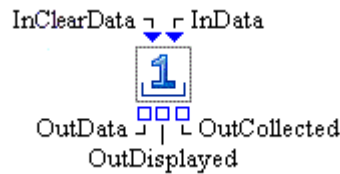
Description

The Number Holder block is used to display a number that represents some user-defined state information. Values can be pushed to or pulled from a Number Holder block via its InData and OutData ports. A Number Holder block will automatically attempt to push any value received at its InData port out its OutData port. Similarly, when a request comes in to pull a value from the OutData port, a Number Holder block will, by default, attempt to pull a value from upstream using its InData port. The user can modify this default behavior using the **Propagation** controls on a Number Holder block.

By default a Number Holder block will display the last value to enter through its InData port, however options are available to display the minimum value, maximum value, mean value, sum of all values, or a count of how many values have entered the block.

The Number Holder block provides the capability of storing values that enter it using its data collection facility. This data can be saved to a SAS data set or JMP table. Values (along with time stamps) are stored in a data model object and the data model object can be accessed through the block's OutCollected port. Display blocks, such as the Histogram block, are often connected to a Number Holder block's OutCollected port to visualize the data. Any block using a data model object is automatically notified when the data in the data model object is modified.

When a true Boolean value arrives at the InClearData port, it will be used as a signal to clear all the data collected by the Number Holder block up to that time during the simulation execution (the displayed value will also be cleared). If the InClearData port receives a false value, the signal will be ignored and data will not be cleared.

Fixed Ports

InData	Input numeric port for the value to be held by the Number Holder block.
InClearData	Input Boolean port that is used to clear all data collected by the Number Holder block.
OutData	Output numeric port for the value currently held by the Number Holder block.
OutDisplayed	Output numeric port for the value currently being displayed by the NumberHolder block.
OutCollected	Output port for the data model object representing the data that have been collected by the Number Holder block.

Attributes Dialog Box Controls

Current	The last value to enter the Number Holder block. The value entered here will be displayed in the block (if the **Display** option is set to **Value**) until a new value enters the block.
Default	The Default Value is used to initialize the Current Value in the Number Holder block when it is created or when the block is reset.
Display	The drop-down box associated with the **Display** option controls what value is displayed on the Number Holder block. Options include the current **Value**, **Minimum**, **Maximum**, or **Mean** value, the **Sum** of all values, or the **Count** of how many values have entered the block.
Propagation	The **To Downstream** check box is used to control propagation of values sent to the InData port of a Number Holder block. If this check box is selected any values entering the InData port will be automatically sent out the OutData port. Otherwise the value propagation will stop at the Number Holder block. If the **From Upstream** check box is selected any attempt to pull a value from the OutData port of a Number Holder block will result in the Number Holder block attempting to pull a value from block(s) connected to its InData port.
Data Collection	The **Collect Data** check box is used to turn data collection on or off. The value entered in the **Capacity** field determines how many values are saved in the data model object. If the capacity is exceeded, a warning message is logged and values are overwritten in the data model object.

Save Dialog Box Controls

Automatic Save	Turns on or off automatic saving of any collected data at the end of each design point replication run. If automatic saving is turned on, data are saved to a file with the base file name specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder containing your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, then any collected data are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server using the **Default File Path** specified in the Simulation Studio Configuration dialog box.
Save Now	Forces the Number Holder block to attempt an immediate save of any collected data. Data are saved to the same location as when automatic saving is turned on.
Location	Displays the pathname of the folder for the file in which to save any collected data.
Base File Name	Specifies the base file name for the SAS data set or JMP table that is used to save any collected data. This name will be the prefix of the actual file name. The zero-based index of the design point and the zero-based index of the replication number will be added as suffixes to the file name, separated by underscore characters. For example, the data for the first replication of the first design point will be saved in a file named *BaseFileName_0_0*, and the data for the second replication of the first design point will be saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors	DefaultValue (double)
Responses	Value (double), MeanValue (double), SumOfValues (double), MinimumValue (double), MaximumValue (double), Count (integer)

String Holder Block

InClearData ┐ ┌ InData

A

OutData ┘ └ OutCollected
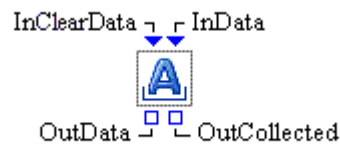
Description

The String Holder block displays a string that represents state information that you define. Values can be pushed to or pulled from a String Holder block via its InData and OutData ports. The String Holder block displays the last value to enter through its InData port and automatically attempts to push any value that is received at its InData port out its OutData port. Similarly, when a request comes in to pull a value from the OutData port, a String Holder block, by default, attempts to pull a value from upstream using its InData port. You can use the **Propagation** controls on a String Holder block to modify this default behavior.

The data collection facility of the String Holder block enables you to store values that enter the block. Data can be saved to a SAS data set or JMP table. Values (along with timestamps) are stored in a data model object, and the data model object can be accessed through the block's OutCollected port. Any block that uses a data model object is automatically notified when the data in the data model object are modified.

When a true Boolean value arrives at the InClearData port, it is used as a signal to clear all the data collected by the String Holder block up to that time during the simulation execution. If the InClearData port receives a false value, the signal is ignored and data are not cleared.

Fixed Ports

InData	Input string port for the value to be held by the String Holder block.
InClearData	Input Boolean port that is used to clear all data collected by the String Holder block.
OutData	Output string port for the value currently held by the String Holder block.
OutCollected	Output port for the data model object representing the data that have been collected by the String Holder block.

Properties Dialog Box Controls

Current	Displays the last value to enter the String Holder block. The value entered here is displayed in the block until a new value enters the block.

Default	Specifies the current value for the String Holder block when the block is initialized.
Propagation	The **To Downstream** check box controls propagation of values sent to the InData port of a String Holder block. If this check box is selected, any values that enter the InData port are automatically sent out the OutData port. Otherwise, the value propagation stops at the String Holder block. If the **From Upstream** check box is selected, any attempt to pull a value from the OutData port of a String Holder block results in the String Holder block attempting to pull a value from one or more blocks that are connected to its InData port.
Data Collection	The **Collect Data** check box turns data collection on or off. The value entered in the **Capacity** field determines how many values are saved in the data model object. If the capacity is exceeded, a warning message is logged and values are overwritten in the data model object.

Save Dialog Box Controls

Automatic Save	Turns on or off automatic saving of any collected data at the end of each design point replication run. If automatic saving is turned on, data are saved to a file with the base filename that is specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder that contains your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, then any collected data are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.
Save Now	Forces the String Holder block to attempt an immediate save of any collected data. Data are saved to the same location as when automatic saving is turned on.
Location	Displays the pathname of the folder for the file in which to save any collected data.
Base File Name	Specifies the base filename for the SAS data set or JMP table that is used to save any collected data. This name will be the prefix of the actual filename. The zero-based index of the design point and the zero-based index of the replication number will be added as suffixes to the file name, separated by underscore characters. For example, the data for the first replication of the first design point will be saved in a file named *BaseFileName_0_0*, and the data for the second replication of the first design point will be saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors	DefaultValue (string)
Responses	Value (string)

Numeric Source Block

Description

The Numeric Source block provides a source of random variation using pseudo-random number generators. This block can also read numbers from a SAS data set or JMP data table.

A collection of discrete and continuous distributions are available, or you can provide the file path for the SAS data set or JMP data table along with the numeric variable column name.

The data provided by the Numeric Source block can be viewed as a stream of numbers, and the numbers are pulled from the stream one after another during a simulation. For example, each time a value is pulled from the OutValue port of a Numeric Source block, the block outputs a number from its current data stream by generating a sample from its related distribution or by reading a value from the data set, whichever is appropriate. If the last observation is reached when reading from a data set, the process resets to the beginning of the column.

The Numeric Source block provides three types of data streams: **Theoretical**, **Fitted**, and **Data Driven**. The **Theoretical** data streams include a collection of theoretical discrete and continuous distributions. The **Fitted** data streams are these same distributions that can be fitted using an input data variable (column) in a SAS data set or JMP data table. After the parameters of the distributions are estimated, the input data set is not needed during a simulation run. The **Data Driven** data streams require an input data set and include options such as empirical distributions and nonhomogeneous Poisson processes, in addition to the numeric SAS data column.

Fixed Ports

InUpdate   Input Boolean value port that signals an update of input data and stream parameter specifications. The new specifications are pulled from the InStreamPolicy or InDataPolicy ports (or both) if these ports are connected. A false Boolean value is ignored.

InStreamPolicy   Input string value port from which the new stream parameter specifications are pulled after an update signal is received. The format for specifying the value for the **Theoretical** data streams is as follows:

class==*distributionName*;*attribute1*==*attribute1Value*;...;*attributeN*==*attributeNValue*; Random Stream Seed==*seedValue*

where:

distributionName	is the distribution name as specified by the **Type** option.
attribute1	is the name of the first parameter associated with the specified data stream type.
attribute1Value	is the value of the first parameter associated with the specified data stream type.
attributeN	is the name of the last parameter associated with the specified data stream type.
attributeNValue	is the value of the last parameter associated with the specified data stream type.
seedValue	is the value of the random stream seed, with integer values in the range of 0 to the Java Long.MAX_VALUE.

The following examples show (case-sensitive) string values for the InStreamPolicy port. Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated. The distribution and parameter names are specified exactly as they are in the **Theoretical** option in the Numeric Source Block Properties dialog box, including any spaces or hyphens. For a list of Simulation Studio distribution and parameter names, see Appendix B, "Random Variation in a Model."

> class==Exponential;Mean==10;Random Stream Seed==100
>
> class==Gamma;Scale==5;Shape==3.2
>
> class==Chi-Square;degrees of freedom==2

If you want to keep the same distribution but alter one or more parameter settings for the distribution, you can supply only the information that is required for the parameter that you want to change. For example, if you are using a normal distribution and you want to change the mean value that is associated with the distribution, the following values are possible for the string input value:

> Mean==1.2
>
> Mean==10

The format for specifying the InStreamPolicy value for the **Data Driven** data stream is as follows:

Lazy Loading==*BooleanValue*;Random Stream Seed==*seedValue*

where:

BooleanValue	is either true or false.
seedValue	is the value of the random stream seed, with integer values in the range of 0 to the Java.Long.MAX_VALUE. The SAS Data Column type does not have the Random Stream Seed option.

OutValue   Output number port for the numeric values to be pulled.

Dynamic Ports

InDataPolicy Input string value port from which the new input data specifications are pulled after an update signal is received. This port is available only for the **Data Driven** data streams. The format for specifying the value is as follows:

attribute1==attribute1Value;attribute2==attribute2Value;...;attributeN==attributeNValue

where:

attribute1	is the name of the first parameter associated with the specified Data Driven Type.
attribute1Value	is the value of the first parameter associated with the specified Data Driven Type.
attributeN	is the name of the last parameter associated with the specified Data Driven Type.
attributeNValue	is the value of the last parameter associated with the specified Data Driven Type.

Each **Type** value for the **Data Driven** option has different required **Input Data** parameters. Parameter names must be specified in the case-sensitive string value exactly as they appear in the Numeric Source Block Properties dialog box, including any spaces. Quotation marks are not required around the string value. For example, the following **InDataPolicy** port value for the Discrete Empirical **Data Driven** type is possible:

> File Path==C:\SimStudio\projects\discemp1.sas7bdat;X==age;Y=pmf

You can also specify only the parameters that need to be updated. In the following example, the File Path parameter remains the same, but the X and Y parameters change:

> X=group2;Y=prob2

Properties Dialog Box Controls

Specify the type of data stream to be provided by this Numeric Source block.

When the **Theoretical** type is selected, the following dialog box controls are available:

Type From the list, select the distribution to sample from. The **Parameters** section will be updated to reflect the distribution.

Stream Parameters This area provides fields for modifying the parameter values associated with the selected distribution. Each distribution has a **Random Stream Seed** entry field. Although Simulation Studio automatically assigns a different seed for each source of randomness, you can use this field to specify the seed value. Valid values for this field are integers in the range of 0 to the Java Long.MAX_VALUE.

Reset Sampling at Update This option can be used to reset the random substream for the current replication during a simulation run. If selected, this option resets the random substream back to the beginning when a true Boolean value is received at the **InUpdate** port. By default, this option is not selected.

When the **Fitted** option is selected, the following dialog box controls are available:

Input Data This area provides fields to specify the file path of input SAS data set or JMP data table and the variable (column) name for distribution fitting. Click **Fit Distribution** to send the fitting request to the JMP program to perform the fitting. If you do not provide the file path or variable name, you can provide these directly using the JMP user interface before the fitting operation proceeds. The fitting results can be sent back from the JMP program to update the rest of the dialog box contents, including the associated distribution **Type** and **Stream Parameters** controls described below. You can also use these controls to edit the parameters.

Type From the list, select the distribution to sample from. The **Parameters** section will be updated to reflect the distribution.

Stream Parameters This area provides fields for displaying and modifying the parameter values associated with the selected distribution. Each distribution has a **Random Stream Seed** entry field. Although Simulation Studio automatically assigns a different seed for each source of randomness, you can use this field to specify the seed value. Valid values for this field are integers in the range of 0 to the Java Long.MAX_VALUE.

Reset Sampling at Update This option can be used to reset the random substream for the current replication during a simulation run. If selected, this option resets the random substream back to the beginning when a true Boolean value is received at the **InUpdate** port. By default, this option is not selected.

When the **Data Driven** option is selected, the following dialog box controls are available:

Type From the list, select the distribution or data stream to sample from. The following Input Data and Stream Parameters section will be updated to reflect the selection.

Input Data This area provides fields to specify the file path of an input SAS data set or JMP data table and the variable or column names. The **Load from Remote SAS Workspace Server** check box indicates that the input SAS data set file is to be loaded from the remote SAS Workspace Server host specified in the Configuration dialog box.

Stream Parameters This area provides fields for modifying the parameter values that are associated with the selected distribution or data stream. The Discrete Empirical, Empirical, NHPP Count, and NHPP Rate type have a **Random Stream Seed** entry field. Although Simulation Studio automatically assigns a different seed for each source of randomness, you can use this field to specify the seed value. Valid values for this field are integers in the range of 0 to the Java Long.MAX_VALUE.

All **Data Driven Type** values have the **Lazy Loading** Boolean field. If the **Lazy Loading** field is false, the input data set file has to be loaded at the start of simulation. Otherwise, the data file is loaded only when its contents are needed during simulation.

Reset Sampling at Update For the SAS Data Column type value, this option resets the current data set back to the first observation when a true Boolean value is received at the **InUpdate** port. If the **InDataPolicy** port is also connected and a new data set is specified, then the new data set starts at the beginning with the first observation if the **Reset Sampling at Update** option is selected. For the Discrete Empirical, Empirical, NHPP Count, and NHPP Rate type values, this option, if selected, resets the random substream for the current replication back to the beginning when a true Boolean value is received at the **InUpdate** port.

Candidates for Design of Experiments

Factors   DataStreamDescription (string), InputDataPolicy (string), ResetStreamAtUpdate (Boolean)

When the **Theoretical** option is selected, the format for specifying the value of the DataStreamDescription factor is the same as for the InStreamPolicy port:

class==*distributionName*;attribute1==*attribute1Value*;...;attributeN==*attributeNValue*; Random Stream Seed==*seedValue*

where:

distributionName	is the distribution name as specified by the **Type** option.
attribute1	is the name of the first parameter associated with the specified data stream type.
attribute1Value	is the value of the first parameter associated with the specified data stream type.
attributeN	is the name of the last parameter associated with the specified data stream type.
attributeNValue	is the value of the last parameter associated with the specified data stream type.
seedValue	is the value of the random stream seed, with valid integer values in the range of 0 to the Java Long.MAX_VALUE.

The following examples show (case-sensitive) string values for the DataStreamDescription factor. Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated. The distribution and parameter names are specified exactly as they are in the **Theoretical** option in the Numeric Source Block Properties dialog box, including any spaces or hyphens. For a list of Simulation Studio distribution and parameter names, see Chapter B, "Random Variation in a Model."

> class==Discrete Uniform;i==10;j==20
>
> class==Normal;Mean==1;Std Dev==2

If you want to keep the same distribution but alter one or more parameter settings for the distribution, you only need to supply the information that is required for the parameter that you want to change. For example, if you are using an exponential distribution and you want to change the mean value associated with the distribution, possible values for the DataStreamDescription factor might be as follows:

> Mean==4
>
> Mean== 4.5

When the **Data Driven** option is selected, the format for specifying the value of the DataStreamDescription factor is as follows:
Lazy Loading==*BooleanValue*;Random Stream Seed==*seedValue*
where:

BooleanValue   is either true or false.

seedValue   is the value of the random stream seed, with integer values in the range of 0 to the Java.Long.MAX_VALUE. The SAS Data Column type does not have the Random Stream Seed option.

When the **Data Driven** option is selected, the format for specifying the value of the InputDataPolicy factor is as follows:
attribute1==attribute1Value;attribute2==attribute2Value;...;attributeN==attributeNValue
where:

attribute1   is the name of the first parameter associated with the specified Data Driven Type.

attribute1Value   is the value of the first parameter associated with the specified Data Driven Type.

attributeN   is the name of the last parameter associated with the specified Data Driven Type.

attributeNValue   is the value of the last parameter associated with the specified Data Driven Type.

Each **Type** value for the **Data Driven** option has different required **Input Data** parameters. Parameter names must be specified in the case-sensitive string value exactly as they appear in the Numeric Source Block Properties dialog box, including any spaces. Quotation marks are not required around the string value. For example, the following values are possible for the InputDataPolicy factor for the SAS Data Column **Data Driven** type:

    File Path==C:\SimStudio\projects\data1.sas7bdat;Variable Name==age

You can specify only the parameters that need to be updated. In the following example, the File Path parameter remains the same, but the Variable Name parameter is changed.

    Variable Name==cost

Responses   None

Text Source Block

InStreamPolicy
InUpdate ⌐ | ⌐ InDataPolicy

OutValue

Description

The Text Source block reads strings from a SAS data set. You supply the file path for the SAS data set or JMP data table along with the text variable column name. Each time a value is pulled from the OutValue port of a Text Source block, the block reads a value from the data set. If the last observation in the data set is reached, the process resets to the beginning of the column.

Fixed Ports

InUpdate   Input Boolean value port that signals an update of input data and stream parameter specifications. The new specifications are pulled from the InStreamPolicy or InDataPolicy ports (or both) if these ports are connected. A false Boolean value is ignored.

InStreamPolicy   Input string value port that allows new stream parameter specifications to come in when an update signal is received. The format for specifying the value is as follows:
Lazy Loading==*BooleanValue*
where *BooleanValue* is either true or false.

InDataPolicy   Input string value port from which the new input data specifications are pulled when an update signal is received. The format for specifying the value is as follows:
File Path==*filePathValue*;Variable Name==*variableNameValue* where:

     filePathValue   is the pathname for the SAS data set or JMP table.

     variableNameValue   is the column name from the SAS data set or JMP table.

     It is not necessary to specify all arguments for the input string value. For example, if you want to continue sampling from the same data set but change the column from the data set, then you can specify a string for the InDataPolicy port that contains only the Variable Name==*variableNameValue* option.

OutValue   Output string value port for text values to be pulled.

Properties Dialog Box Controls

Input Data   This area provides fields for modifying the input data specifications associated with the Text Source block. The **File Path** field specifies the absolute or relative file path for the input SAS data set or JMP data table file. The **Variable Name** field identifies the variable or column name in input data file. The **Load from Remote SAS Workspace Server**

check box indicates whether the input SAS data set file is to be loaded from the remote SAS Workspace Server host.

Stream Parameters   This area provides fields for modifying the stream parameter specifications that control how the stream of text values from the Text Source block is prepared. The field **Lazy Loading** is Boolean. If the **Lazy Loading** field is false, the input data set file has to be loaded at the start of simulation. Otherwise, the data file is loaded only when its contents are needed during simulation.

Reset Sampling at Update   This option resets the current data set back to the first observation when a true Boolean value is received at the InUpdate port. If the InDataPolicy port is also connected and a new data set is specified, then the new data set starts at the beginning with the first observation if the **Reset Sampling at Update** option is selected.

Candidates for Design of Experiments

Factors   DataStreamDescription (string), InputDataPolicy (string), ResetStreamAtUpdate (Boolean)

The format for specifying the value of the InputDataPolicy factor is as follows:
File Path==*filePathValue*;Variable Name==*variableNameValue*
where:

filePathValue   is the pathname for the SAS data set or JMP table.

variableNameValue   is the column name from the SAS data set or JMP table.

The format for specifying the value of the DataStreamDescription factor is as follows:
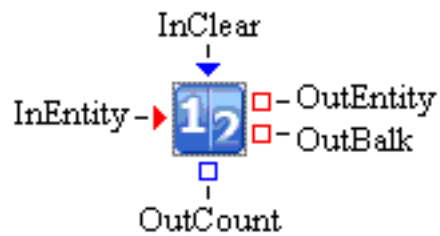Lazy Loading==*BooleanValue*
where *BooleanValue* is either true or false.

Responses   None

Counter Block

Description

The Counter block counts the number of entities that pass through it. If the OutCount value port has any connections to it, the Counter block pushes its count value to the port every time it changes.

After an entity enters the Counter block, the block determines whether any block downstream of the Counter block's OutEntity port can accept the entity before pushing the entity out the OutEntity port. If this acceptance fails, the entity is either pushed out the OutBalk port or destroyed if there are no connections to the OutBalk port.

When a true Boolean value arrives at the InClear port, it will be used as a signal to reset the current count value to zero. If the InClear port receives a false value, the signal will be ignored and the count will not be cleared.

Fixed Ports

InEntity	Input entity port for entities to enter the Counter block.
InClear	Input Boolean port that is used to reset the current count value to zero.
OutEntity	Output entity port for entities that can be accepted by a downstream block.
OutBalk	Output entity port for entities that cannot leave using the OutEntity port.
OutCount	Output integer port for the number of entities that have passed through the Counter block.

Candidates for Design of Experiments

Factors	None
Responses	Count (integer)

Time Now Block

OutValue ⌐ | ⌐ OutReplicationIndex
OutPointIndex

Description

The Time Now block can be used to access the current simulation time while the model is running. This is accomplished by pulling a value from the OutValue port on the Time Now block. The indices of the current design point and replication for the current simulation experiment are also provided.

Fixed Ports

OutValue Output number port that can be pulled for the current simulation clock time. (**NOTE:** The clock value is not pushed out from the port automatically.)

PointIndex Output number port for the 1-based index of the current design point during the run of a simulation experiment.

ReplicateIndex Output number port for the 1-based index of the current replication within the current design point during the run of a simulation experiment.

Candidates for Design of Experiments

Factors None

Responses None

Overview of the Advanced Template

The Simulation Studio Advanced template provides a collection of blocks used to build more complex simulation models.

Batch Block

Description

The Batch block groups entities so that they flow together through a simulation model. Entities arrive at a Batch block individually through its InEntity input entity port. The Batch block holds the entities until the number of entities it is holding reaches the value that is specified in the Batch block's batch size parameter. At this point, the Batch block attaches the held entities to a carrier entity and attempts to send the carrier entity out through the OutCarrier output entity port. The carrier entity carries the batched group of entities through the simulation model, but only the attributes of the carrier entity are accessible by other blocks. That is, the attributes of the entities that are included in the batch are not accessible by other blocks. Downstream in the simulation model, an Unbatch block can be used to separate the individual entities from the batch carrier entity. At that point all unbatched entities have their original attributes.

If nothing is attached to the InCarrier input entity port, a Default entity is created and used as the carrier entity whenever a batch of entities is ready to be sent out through the OutCarrier port. If there is a connection to the InCarrier port, the entities that arrive at that port are used as the carriers. If there is a connection to the InCarrier port, the Batch block waits until it has both a carrier entity and a complete batch of entities before it attempts to send the carrier entity out through the OutCarrier port. The Batch block can hold only a single carrier and a single batch of entities at any given time. Therefore, if there is a connection to the Batch block's InCarrier port and the Batch block is holding a carrier entity but is not holding a complete batch of entities, the Batch block does not accept another carrier entity through its InCarrier port. Similarly, if there is a connection to the Batch block's InCarrier port and the Batch block is holding a complete batch of entities but is not holding a carrier entity, the Batch block does not accept another entity through its InEntity port.

The InSignal input value port is used to force the Batch block to send out a carrier regardless of the number of entities it is holding. If a true value arrives at the InSignal port, the Batch block attempts to attach any entities it is holding to a carrier entity and send the carrier entity out through the OutCarrier port. In this case the carrier might not hold any entities or it might hold a smaller number of entities than the batch size. If there is a connection to the InCarrier port and a true value arrives at the InSignal port but the Batch block is not holding a carrier, the signal is ignored. A false signal arriving at the InSignal port is always ignored since it signifies that no action needs to be taken.

An integer value can be pushed through the InBatchSize port to set the batch size for the Batch block. If the batch size is decreased while the Batch block is holding more entities than the new batch size, the existing entities are batched together according to the new batch size, and the Batch block attempts to send a carrier entity out through its OutCarrier port for each smaller batch of entities. If there is a connection to the Batch block's InCarrier port, the Batch block waits for a carrier to arrive before sending out each smaller batch of entities.

Fixed Ports

InEntity   Input entity port for entities that are batched together.

InCarrier   Input entity port for entities used as carriers for entity batches.

OutCarrier   Output entity port for carriers (containing batches of entities) that can be accepted by a downstream block.

OutBalk   Output entity port for carriers (containing batches of entities) that cannot leave using the OutCarrier port.

InSignal   Boolean input port that forces the Batch block to send out a carrier (if one is available) that contains any entities being held by the Batch block. The carrier can be empty (containing no entities).

InBatchSize   Numeric input port that dynamically sets the batch size of the Batch block.

Properties Dialog Box Controls

Batch Size   Specifies the number of entities the Batch block stores in its buffer before attempting to send out those entities on a carrier. Valid values for this field are integers in the range from 0 to 2,147,483,647. Selecting the **Infinite** check box supersedes any value entered. If the **Infinite** check box is selected, a connection should be made to the InSignal port of the Batch block to determine when the Batch block should attempt to release a carrier containing a batch of entities.

Queueing Policy   Specifies the queueing policy for the queue used internally by the Batch block. See the **Queueing Policy** control in the section "Queue Block" on page 141 for details.

Candidates for Design of Experiments

Factors   QueueingPolicy (string)
                      See the Queueing Policy design-of-experiment factor in the section "Queue Block" on page 141 for details.

Responses   None

Unbatch Block

Description

The Unbatch block is used to separate individual entities from a batch carrier entity. Carrier entities (populated with a group of zero or more entities by a Batch block) arrive at an Unbatch block through its InCarrier input entity port. The Unbatch block first separates individual entities from the carrier entity. Then it attempts to send the carrier entity (which might or might not be empty depending on whether the Unbatch block separated all of the individual entities from the carrier) out its OutCarrier output entity port. It also attempts to send each of the separated individual entities (if any) out its OutEntity port.

Fixed Ports

InCarrier   Input entity port for carrier entities that contain a batch of zero or more entities.

OutCarrier   Output entity port for a carrier to leave that can be accepted by a downstream block.

OutEntity   Output entity port for each of the individual entities separated from the carrier to leave that can be accepted by a downstream block.

OutCarrierBalk   Output entity port for a carrier to leave that cannot leave using the OutCarrier port.

OutEntityBalk   Output entity port for each of the individual entities separated from the carrier to leave that cannot leave using the OutEntity port.

Properties Dialog Box Controls

Identify Candidate Entities   Use these fields to define the criteria for selecting possible entities to be separated from the carrier. The Unbatch block considers separating from the carrier only entities that satisfy all of the criteria specified in this section.

For **Primary Usage**, you can select to have either regular entities or resource entities separated from the carrier.

For **Entity Type** (optional), you can specify the name of a particular entity type for entities to be separated from the carrier.

For **Attribute Rule** (optional), you can specify a Boolean expression that includes attribute values for entities to be separated from the carrier. You can type the expression in the Attribute Rule field, or you can right-click on the Attribute Rule field and select the **Edit** option to open the Edit Expression window. For more information about how to write the Boolean expression, see Appendix F, "Expressions."

If you select **Resource Entity** for **Primary Usage**, then you can specify a **Resource State** (optional) for entities to be separated from the carrier. Valid values are Functional, Failed, Maintenance, and Offlined.

Unbatch Entities among Candidates   Use these fields to specify which entities meeting the criteria speci-
fied in the **Identify Candidate Entities** section should be separated from the carrier.
Begin At specifies where in the buffer of entities to begin separating them from the carrier
and attempting to send them out through the OutEntity port. **First** means to start with the
first entity in the buffer and then proceed with the following entities in order, stopping
if the end of the buffer is reached. **Last** means to start with the last entity in the buffer
and then proceed backwards through the entities, stopping if the beginning of the buffer
is reached. **Middle** means to start with the entity at the index specified in the entry field
and then proceed with the following entities in order, stopping if the end of the buffer is
reached.

Count specifies a maximum number of entities to separate from the carrier. Checking **All**
causes any **Count** value to be ignored. If **All** is checked and **Begin At** is set to **First** or
Last, all of the entities that meet the criteria in the **Identify Candidate Entities** section
are separated from the carrier. If **All** is checked and **Begin At** is set to **Middle**, all of the
entities except the ones before the specified middle index are separated from the carrier.

Candidates for Design of Experiments

Factors   None

Responses   None

Clone Block
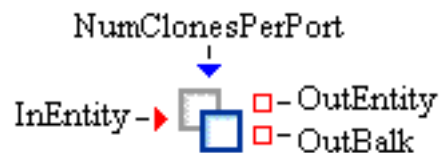
Description

The Clone block creates clones of entities that pass through it. A *clone* is a new entity with the same type
and all of the same attributes as the original entity. When an entity enters a Clone block, the block first
determines whether anything is connected to its NumClonesPerPort port. If it finds a connection, the Clone
block attempts to pull a value from the NumClonesPerPort port. This value represents the number of clones
of the original entity that the Clone block generates for each clone output port. If this value is greater than 1,
multiple clones flow sequentially out of each clone output port.

If there are no connections to the NumClonesPerPort port, the Clone block uses the value specified in its
ClonesPerPort properties dialog box field for the number of clones to generate per clone output port.

You can set the number of clone output ports in the properties dialog box. If no clone output ports exist, the
new clone entities are pushed through the OutEntity port. The original entity is always the first entity to

exit the Clone block, and it exits via the OutEntity port. If the original entity or a clone cannot be accepted downstream, it flows out through the OutBalk port.

Fixed Ports

InEntity	Input entity port for entities to enter the Clone block.
OutEntity	Output entity port for the input entity if it can be accepted by a downstream block. If there are no clone output ports, clone entities that can be accepted by a downstream block also go out through this port.
OutBalk	Output entity port for any entities that cannot be accepted by a downstream block.
NumClonesPerPort	Input integer port for the number of clone entities for the clone block to generate for each clone output port.

Properties Dialog Box Controls

Clones Per Port	Specifies how many clones are generated for each clone output port. This value is used only if there are no connections to the NumClonesPerPort port.
Cloning Ports	Specifies how many clone output ports are available.

Candidates for Design of Experiments

Factors	ClonesPerPort (integer)
Responses	None

Gate Block

Description

The Gate block provides a facility to pull and push multiple values every time an entity passes through the block. For each action that is defined on a Gate block, an input value port and an output value port are created on the block. When an entity enters a Gate block, the block steps through its list of actions, first pulling from the input value port associated with the action and then pushing the retrieved value to the output value port associated with the action. If there is no connection to an input value port, the Gate block uses the default value associated with that action.

You specify a new action by clicking the **Add** button on the Properties dialog box **Actions** tab. This results in a new action (with a default name, type, and value) being added to the **Actions** table. You can edit the name, type, and value of the action directly in the table. The action names that a Gate block uses must be unique. You can change the action type through a drop-down box on the type table cell. You can delete actions from the **Actions** table by selecting the action row in the table and then clicking the **Remove** button. Clicking the **Apply** button pushes all entries in the **Actions** table to the Gate block, and any input and output value ports are created or deleted as needed. (Note: You cannot change an action's type after the action is pushed to the Gate block.) You use the **Move Up** and **Move Down** buttons to reorder the actions in the **Actions** table. The order of the input and output value ports is also adjusted to be the same as the order of the actions in the **Actions** table.

Fixed Ports

InEntity	Input entity port for entities to enter the Gate block.
OutEntity	Output entity port for entities to exit the Gate block.

Properties Dialog Box Controls

Add	Adds a new action with a default name, type, and value to the **Actions** table. You can change the name, type, and default value of the action directly in the table. The action names in the Gate block's **Actions** table must be unique. You can change the action type through a drop-down box on the cell in the table. (You cannot change an action type in the table after you click the **Apply** button. To change an action type after you click **Apply**, you must remove the action, add it again, and then modify the type of the newly added action before clicking **Apply** again.)
Remove	Deletes the selected action from the **Actions** table.
Move Up	Moves the row for the selected action up in the **Actions** table. The value ports that are associated with the selected action are moved to the left.
Move Down	Moves the row for the selected action down in the **Actions** table. The value ports that are associated with the selected action are moved to the right.
Apply	Updates all actions in the Gate block as specified in the **Actions** table, and creates or deletes input and output value ports as needed.

Candidates for Design of Experiments

Factors	None
Responses	None

Valve Block
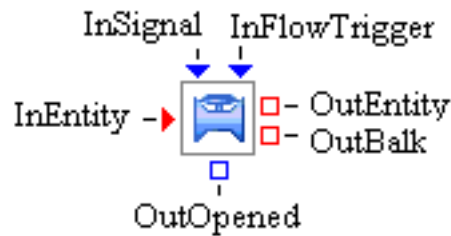
Description

The Valve block controls the flow of entities through a simulation model. If the Valve block is closed, an entity cannot flow through the Valve block. If the Valve block is opened, its behavior depends on which flow directions are enabled:

- If the Push To Downstream option is enabled, a block connected to the InEntity port can push entities through the Valve block to a block connected to the OutEntity port. If disabled, pushing is not allowed through the Valve block.

- If the Pull From Upstream option is enabled, a block connected to the OutEntity port can pull entities through the Valve block from a block connected to the InEntity port. If disabled, pulling through the Valve block is not allowed.

Fixed Ports

InEntity	Input entity port for entering entities.
OutEntity	Output entity port for entities to leave that can be accepted by a downstream block.
OutBalk	Output entity port for entities to leave that cannot leave using the OutEntity port.
InSignal	Boolean input port that allows the Valve block to be dynamically opened (true) or closed (false).
InFlowTrigger	Boolean input port that (by passing in true) can trigger the flow of entities through the Valve block. The flow of entities is still subject to the settings specified in the **Flow Directions** section of the Valve block's properties dialog box and whether the Valve block is opened or closed. An input value of false is ignored.
OutOpened	Boolean output port that pushes out whether the Valve block is opened (true) or closed (false) each time the Valve block changes state between opened and closed.

Properties Dialog Box Controls

Initial State	Specifies whether the Valve block is **Opened** or **Closed** when the model starts executing.
Flow Directions	Specifies the flow directions supported by the Valve block: **Push To Downstream** and **Pull From Upstream** can be enabled or disabled independently.

Candidates for Design of Experiments

Factors   None

Responses   None

Formula Block

OutValue

Description

The Formula block can evaluate an expression based on state or model information. You create variables (called *input variables*) to be used in the expression, and you formulate them into an expression that is evaluated every time a value is pulled from the Formula block's OutValue port. The expression is also evaluated and pushed out the Formula block's OutValue port every time input is pushed into one of the Formula block's input value ports. The value that is associated with an input variable can come either from an entity attribute or from an input value port. If the source for an input variable is designated to be a port, an input value port is created on the block and is associated with the appropriate input variable. Whenever a value is requested from a Formula block or new input arrives at a Formula block, the Formula block first determines which values that are associated with its input variables need to be acquired (based on the setting for the **To Acquire Port Values Only When Needed** properties dialog box option) and then attempts to evaluate its expression. The result of this evaluation leaves through the OutValue port.

You use the **Expression** text box to enter the expression to be evaluated in the Formula block. Appendix F, "Expressions." contains a complete list of available expressions. You define variables that are used in the expression in the **Input Variables Table**. The operators available for building expressions include Boolean (&&, ||, !), arithmetic (+, -, *, /, %), and comparison (<, >, ≤, ≥, ==, !=) operators. Available functions include arithmetic (*abs*), trigonometric (*sin*, *cos*, *tan*, *asin*, *acos*, *atan*, *sinh*, *cosh*, *tanh*), and logarithmic (*exp*, *log*, *ln*) functions, along with *floor, ceil, round, min, max,* and *power* functions. The *min* and *max* functions can be called with any number of numeric arguments. In addition, you can use the conversion functions *degrees* and *radians* to convert between degrees and radians. You can use the *concat* function to concatenate strings. Two logical functions, *cond* and *switch*, are also provided. The format for the *cond* function is *cond(Boolean expression, true return value, false return value).* The value returned by this function is determined by evaluating its Boolean expression. The format for the *switch* function is *switch(Boolean expression1, value1, Boolean expression2, value2, ..., default value).* The value returned by the switch function is the value that follows the first Boolean expression that evaluates to true. The default value is returned if none of the Boolean expressions is true. You can also use the *minindex* and *maxindex* functions, which return the zero-based index of the minimum (or maximum) value in the list of values passed into the function.

The Formula block also supports a dot (.) operator. When you use an input variable of type Observation, you can use the dot operator to access the values of the observation's member variables. For example, suppose an observation input variable named Record has member variables Name and GPA (so that the observation Record is a row from a data set with columns Name and GPA). The expression Record.Name returns the value of Name for the current observation. Similarly, the following expression returns a string that depends on the value of GPA: cond(Record.GPA < 60.0, "Fail", "Pass").

Fixed Ports

OutValue   Output value port for the result of the Formula block's expression.

Properties Dialog Box Controls

Add   Adds a new input variable with a default **Name**, **Type**, and **Source** to the **Input Variables** table. You can edit the **Name**, **Type**, and **Source** of the variable directly in the table. The variable names listed in the Formula block's **Input Variables** table must be unique. You can change the **Type** through a drop-down box on the cell in the table. A variable **Type** cannot be changed in the table after the **Apply** button is clicked. If you want to change a variable **Type** after **Apply** has been clicked, you must remove the variable, add it again, and then modify the **Type** of the newly added variable before clicking **Apply** again.

Remove   Deletes the selected variable from the **Input Variables** table.

Move Up   Moves the row for the selected input variable up in the **Input Variables** table. The value port associated with the selected input variable is moved to the left.

Move Down   Moves the row for the selected input variable down in the **Input Variables** table. The value port associated with the selected input variable is moved to the right.

To Acquire Port Values Only When Needed   If this option is turned off, the Formula block always acquires values for all of its input variables. If this option is turned on, the Formula block acquires only the values for its input variables that are required in order to determine the result of the expression.

Expression   Contains the expression to be evaluated in the Formula block. Any variables used in the expression must be defined in the **Input Variables** table. Appendix F, "Expressions." contains a complete list of available expressions.

Result Type   Identifies the value type that results from evaluating the expression. The selected option specifies the output port type for the Formula block.

Result Precision Specifies the decimal precision for rounding a numeric result value, if the Result Type is Number. A positive precision value rounds the numeric result to the specified number of digits to the left of the decimal point. A negative precision value rounds the numeric result to the specified number of digits to the right of the decimal point. A precision value of zero rounds the numeric result to the nearest integer.

Apply   Validates the expression and saves the input variables, the expression, and the expression result. Input value ports are created or deleted and the type of the output value port is set as needed.

Candidates for Design of Experiments

Factors   None

Responses   None

Connector

Entity Connector   Value Connector
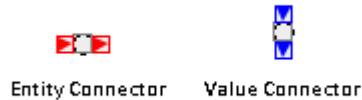
Description

Connecting blocks that are far apart in a model usually results in a link that crosses over other blocks and links. This can be visually distracting and confusing. You can use Connectors to create invisible links between blocks. To create an inter-Connector link:

1. Drag a separate Connector block into the model window and place it adjacent to each of the blocks in the model that you want to directly link. When you drag a Connector into a model, the Connector Type dialog box appears. In this dialog, you select the type of Connector (Entity, Number, String, Boolean, Observation, Data Model, or Entity Group) based on the type of the block ports between which you want to create a hidden link. Only Connectors of the same type can be connected. The ports on an entity-type Connector are red, and the ports on a value-type Connector are blue.

2. Create a link from the appropriate port on each block to the adjacent Connector block.

3. Create a link between the two Connector blocks. When the link is created, it briefly flashes and then disappears.

To show all inter-Connector links in the model, select **Navigation►Show Connector Links** from the pop-up menu either on an individual block or on a Model window. Links between entity-type Connectors are displayed red when they are visible, and links between value-type Connectors are displayed blue when visible. To hide all inter-Connector links, select **Navigation►Show Connector Links** again.

You can view a list of all input (or output) links by right-clicking on the input (or output) port of a Connector to open the Port Connections dialog box. To display a Connector link in the model, click on a row in the Port Connections dialog box (where each row represents a link with another Connector). You can also remove links or change the precedence order of the links in the Port Connections dialog box.

Caster Block

Description

The Caster block casts a value from a generic object value back to its original or concrete value type.

Fixed Port

InValue   Input value port for a generic object value.

Dynamic Port

OutValue   Output value port for the user-specified output value type.

Properties Dialog Box Controls

Output Type   From the list, select the output value type to cast the input object value to.

Propagation   The **To Downstream** option controls propagation of values sent to the InValue port of a Caster block. If this option is selected, any values that enter the InValue port are automatically sent out the OutValue port. Otherwise, the value propagation stops at the Caster block. If the **From Upstream** option is selected, any attempt to pull a value from the OutValue port of a Caster block results in the Caster block attempting to pull a value from block(s) connected to its InValue port.

Candidates for Design of Experiments

Factors   None

Responses   None

Submodel Block

Description

The Submodel block creates hierarchical models and facilitates component reuse. It is similar to a compound block except for one key difference: the contents of a compound block are embedded in a model when it is created, whereas a Submodel block provides a linkage from the simulation model to its content definition. Once the content definition of a Submodel block is changed, all instances of the submodel, whether in the same simulation model or in different simulation models, reflect those changes.

You double-click a Submodel block to view and edit its content in a separate submodel window. The **Instance** option in the submodel window enables you to view (but not edit) the contents of a submodel. If you have a submodel window open and you double-click on another submodel block that is linked to the same submodel that is already open, then both instances are displayed in the same submodel window. Clicking the up and down arrows adjacent to the **Instance** option changes the submodel instance being viewed. The label that is associated with each submodel instance is displayed. You can click the **Close** button to close the current submodel instance. The **Instance** option is especially useful for viewing the animation of a submodel instance as the model runs.

The **Definition** option in the submodel window enables you to edit the contents of a submodel. You can drag blocks into the submodel, connect blocks, delete blocks, and perform any other modeling action that you would perform in the regular Model window. To save an edited submodel block, right-click on the submodel and select **Save**. The default filename extension for a saved submodel block is *.cblk*. When the edited submodel definition is saved, all instances of that submodel in the currently open simulation model automatically refresh to reflect the new, updated definition.

To open the Submodel Block Properties dialog box, right-click on the Submodel block and select **Block Properties**.

Properties Dialog Box Controls

Submodel Path   Specifies the filename for the compound block (*.cblk* file) that is associated with this Submodel block.

SAS Program Block

InSubmitCode

Description

The SAS Program block can be used to execute a SAS program or a JMP script.

Fixed Ports

InSubmitCode Input Boolean data port that starts the execution of the SAS program or JMP script if the value true is passed in. For example, a Value Generator block that produces Boolean data can have its OutValue port connected to a SAS Program block's InSubmitCode port.

Properties Dialog Box Controls

SAS Code Path Specifies the pathname of the SAS program or JMP script to be executed.

Auto Submit If selected, the SAS program or JMP script will automatically execute at the end of a run (after all replications for all selected design points have finished executing). If the **Submit to Remote SAS Workspace Server** option is selected, then the SAS program or JMP script will execute on the remote SAS workspace server host specified in the Simulation Studio Configuration dialog box.

Candidates for Design of Experiments

Factors None

Responses None

Entity Filter Block

InEntity — ▶ OutPassed
OutFailed

Description

The Entity Filter block routes incoming entities to one of two output paths: one for entities that pass the filter criteria and another for entities that do not.

When an entity arrives at the InEntity port of an Entity Filter block, the Entity Filter block tests the entity against a set of filter criteria including primary usage, entity type, attribute rule, and (if the primary usage is resource entity) resource state. If any filter criterion does not have a value, that criterion is ignored. If the entity matches all of the specified criteria, the entity is sent out the OutPassed output entity port. Otherwise, the entity is sent out the OutFailed output entity port.

Fixed Ports

InEntity	Input entity port for entering entities.
OutPassed	Output entity port for entities that meet the Entity Filter block's criteria.
OutFailed	Output entity port for entities that do not meet the Entity Filter block's criteria.

Properties Dialog Box Controls

Primary Usage	Selects whether an entity must be a **Regular Entity** or a **Resource Entity** in order to meet the filter criteria.
Entity Type	(optional) Specifies the name of a particular entity type that an entity must have in order to meet the filter criteria.
Attribute Rule	(optional) Specifies a Boolean expression that includes attribute values of an entity that must evaluate to true in order to meet the filter criteria. You can type the expression in the Attribute Rule field, or you can right-click on the Attribute Rule field and select the **Edit** option to open the Edit Expression window. For more information about how to write the Boolean expression, see Appendix F, "Expressions."
Resource State	If you select **Resource Entity** for **Primary Usage**, then you can specify a **Resource State** (optional) that a resource entity must have in order to meet the filter criteria. Valid values are Functional, Failed, Maintenance, and Offlined.

Candidates for Design of Experiments

Factors	None
Responses	None

Entity Group Holder Block
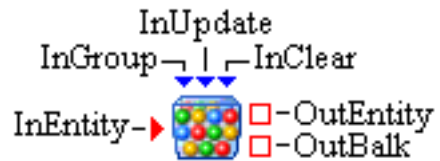
Description

The Entity Group Holder block serves as a holding facility for an *entity group*, which is a collection of entity references. Rather than holding each actual entity in an entity group, an *entity reference* (which is information that uniquely identifies a particular entity) is held.

The Entity Group Holder block stores only references to entities that pass a set of filter criteria defined in the properties dialog box. When a single entity enters the block through the InEntity port and passes the input filter criteria, a single entity reference for the entity is added to the Entity Group Holder block. When a group of entity references arrives through the InGroup port, those entity references that pass the input filter criteria can either replace any existing entity references being held by the block or be *merged* with the existing group of entity references (combining the new and existing entity references but not storing duplicate entity references), depending on how the Entity Group Holder block is configured in the properties dialog box.

OutSubgroup ports can be configured for the Entity Group Holder block that allow a group of entity references to be pulled from the block, based on a set of output filter criteria. When a pull request arrives at an OutSubgroup port, the Entity Group Holder block applies the output filter criteria associated with that port to the group of entities it is currently holding. The resulting entity group is then passed out through the OutSubgroup port.

Fixed Ports

InEntity	Input entity port for entering entities. An entity reference for the entity is added to the Entity Group Holder block's set of entity references if the entity passes the input filter criteria.
OutEntity	Output entity port for entities to leave that can be accepted by a downstream block.
OutBalk	Output entity port for entities to leave that cannot leave using the OutEntity port.
InGroup	Input data port for an incoming entity group. For entities in the incoming entity group that pass the input filter criteria, entity references for those entities either replace any existing entity references being held by the block or are merged with the existing group of entity references, depending on the setting of **Handling of Input Entity Group** in the properties dialog box.
InUpdate	Input Boolean port that, if true is passed in, forces the EntityGroup block to pull an entity group from the first (zero ordered) link connected to its InGroup port.
InClear	Input Boolean port that, if true is passed in, clears the Entity Group Holder block's set of entity references.

Properties Dialog Box Controls

Input Filter
These fields define the criteria for selecting the incoming entity references to be stored by the Entity Group Holder block.

For **Primary Usage**, select to have references to either entity type **Regular Entity** or **Resource Entity** stored by the Entity Group Holder block.

For **Entity Type** (optional), you can restrict the Entity Group Holder block to store only references to entities of a particular entity type.

For **Attribute Rule** (optional), you can restrict the Entity Group Holder block to store only references to entities that satisfy a Boolean expression that includes entity attribute values. You can type the expression in the Attribute Rule field, or you can right-click on the Attribute Rule field and select the **Edit** option to open the Edit Expression window. For more information about how to write the Boolean expression, see Appendix F, "Expressions."

If you select **Resource Entity** for **Primary Usage**, then you can specify a **Resource State** (optional) that entities must have in order to be stored by the Entity Group Holder block. Valid values are Functional, Failed, Maintenance, and Offlined.

Handling of Input Entity Group
Select **Override the current group** to make the Entity Group Holder block replace its current entity group with a new entity group whenever an entity group arrives through the InGroup port.

Select **Merge with the current group** to make the Entity Group Holder block add any nonduplicate entity references to its current entity group whenever an entity group arrives through the InGroup port.

For entities that arrive through the InEntity port, an entity reference is always merged with the current group if the entity passes the input filter criteria.

Query Outputs
The **Query Outputs** table defines the entity group output ports for the Entity Group Holder block. Each output port has an associated name and set of filter criteria that determine which entity references are included in the entity group to be pulled from the port. Click **Add Query** to add a new row to the **Query Outputs** table that represents a new entity group output port. The column values for the new row can be edited directly in the table:

- **Port Name** uniquely names the entity group output port for the Entity Group Holder block.

- **Key Attribute** (optional) sets the name of the key attribute in the group of entity references. In order for an entity reference in the group to match this filter criterion, the referenced entity must have an attribute by this name, and the attribute value must be equal to the value of an attribute by the same name that is defined on an entity that enters an input entity port of another block that the entity group output port is connected to. In other words, an entity attribute name/value pair for an entity that enters some other block is used as a key to search the holdings of the Entity Group Holder block in order to determine which entity references can be included in the entity group to be pulled from the port by the other block.

- **Entity Type** (optional) restricts the port to allow only references to entities of a particular entity type to be included in the entity group to be pulled from the port.

- **Attribute Rule** (optional) restricts the port to allow only references to entities that satisfy a Boolean expression for entity attribute values to be included in the entity group to be pulled from the port. For more information about how to write the Boolean expression, see Appendix F, "Expressions."

- **Offset** (optional) specifies an index into the group of references being held by the Entity Group Holder block. The entity references that are selected to exit through the output port begin at this index, and any entity references that occur prior to this index are not included in the output entity group. A value of 0 or 1 is equivalent to a blank value, causing the first entity reference to be selected first. A negative value is an index that starts at the end of the entity references. For example, an index of –1 causes the last entity reference to be selected first. A negative value causes entity references to be selected by traversing backwards through the list of entity references. If the absolute value of the offset is larger than the number of entity references that satisfy the criteria, an empty entity group exits the port. The ordering of the entity references within the Entity Group Holder block reflects the ordering of the holding block that originally generated the entity references.

- **Maximum Count** (optional) specifies a maximum number of entity references that can be included in the entity group that exits the port.

Click **Remove Query** to remove the selected row from the **Query Outputs** table, indicating that the corresponding entity group output ports should be removed.

Apply Saves all information to the Entity Group Holder block, creating or removing entity group output ports as needed.

Candidates for Design of Experiments

Factors None

Responses None

Stopper Block
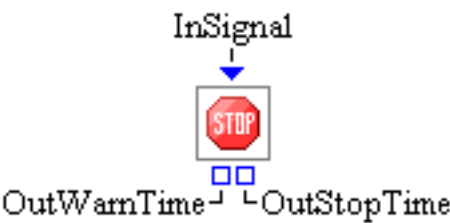
Description

The Stopper block outputs the simulation stop time when a simulation replication ends normally. It can also be used to stop the current simulation replication with an input stopping signal.

In many simulation applications, the actual end time of the simulation replication is often not known ahead of time. However, certain processing steps (such as data collection and special-purposed computation) might need to occur right before the end of the simulation. To facilitate the modeling of this type of simulation task, the Stopper block also warns about the upcoming end of a simulation replication. When the end of the simulation replication is near, the number output port OutWarnTime delivers the current simulation time of the warning. This value can then be converted to a Boolean signal (for example) to trigger data collection before the actual end of the simulation.

If the simulation ends abnormally, either because of errors or user interference, the Stopper block does not provide the stop or warning time values.

Fixed Ports

InSignal   Input Boolean port that stops the current simulation replication with a true input value. A false input value is ignored.

OutWarnTime   Output number port for the simulation time when a warning of simulation ending occurs.

OutStopTime   Output number port for the simulation time when the current replication ends.

Candidates for Design of Experiments

Factors   RankValue (double)

Responses   None

Overview of the Data and Display Template

The Simulation Studio Data and Display template provides a collection of blocks used to collect and display data in simulation models.

Bucket Block
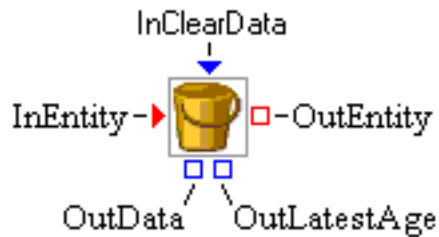
Description

The Bucket block is used to extract and store entity attribute values from entities that enter the block. Attributes to be extracted from the entity are identified in the Bucket block's **Attributes** Table. When an entity enters the block, entity attribute values are retrieved from the entity and stored in a data model object during the simulation run. The age of the entity is calculated and pushed to the bucket's OutLatestAge port.

The data model object can be accessed through the block's OutData port. Display blocks, such as the Histogram block, are often connected to a Bucket block's OutData port to visualize the data. Any block using a data model is automatically notified when the data in the data model is modified.

The user identifies attributes to be extracted from the entity using one of the attribute collecting options described in the **Dialog Controls** section below.

The user can limit the number of observations stored in a Bucket block through its **Data Collection** controls. If the capacity is exceeded, a warning will be logged and observations will be overwritten.

The Bucket block provides the capability of saving values it extracts to a SAS data set or JMP table. Saving options are available on the Save Dialog.

When a true Boolean value arrives at the InClearData port, it will be used as a signal to clear all the data collected by the Bucket block up to that time during the simulation execution. If the InClearData port receives a false value, the signal will be ignored and data will not be cleared.

Properties Dialog Box Controls

Specify the attributes to be extracted from the incoming entity.

The Bucket block provides two options for collecting entity attributes. With the **Collect Selected Entity Attributes** option selected, the associated attribute table and editing buttons are enabled. The user identifies attributes to be extracted from the entity by selecting the **Add** button beside the attribute table. This results in a new attribute entry (with a default name) being added to the Attributes Table. The **Name** and **Type** of the

attribute can be edited directly in the table. The attribute names listed in the Bucket block's Attributes Table must be unique. The attribute type can be changed through a drop-down box on the **Type** table cell. Attributes can be deleted from the Attributes Table by selecting the attribute row in the table and then selecting the **Remove** button. Selecting the **Apply** button causes all entries in the Attributes Table to be pushed to the Bucket block.

When the **Collect All Entity Attributes** option is selected, all attributes (except the internal entity ID) will be extracted from the incoming entities.

Add	adds a new attribute to the **Attributes** table.
Remove	deletes an attribute that has been selected from the **Attributes** table.
Data Collection	The **Collect Data** check box is used to turn data collection on or off. The value entered in the **Capacity** field determines how many values are saved in the data model object. If the capacity is exceeded, a warning message is logged and values are overwritten in the data model object.

Save Dialog Box Controls

|	
Automatic Save	Turns on or off automatic saving of any collected data at the end of each design point replication run. If automatic saving is turned on, data are saved to a file with the base file name specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder containing your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, then any collected data are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.
Save Now	Forces the Bucket block to attempt an immediate save of any collected data. Data are saved to the same location as when automatic saving is turned on.
Location	Displays the pathname of the folder for the file in which to save any collected data.
Base File Name	Specifies the base file name for the SAS data set or JMP table that is used to save any collected data. This name will be the prefix of the actual file name. The zero-based index of the design point and the zero-based index of the replication number will be added as suffixes to the file name, separated by underscore characters. For example, the data for the first replication of the first design point will be saved in a file named *BaseFileName_0_0*, and the data for the second replication of the first design point will be saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors	Capacity (integer)
Responses	None

Probe Block
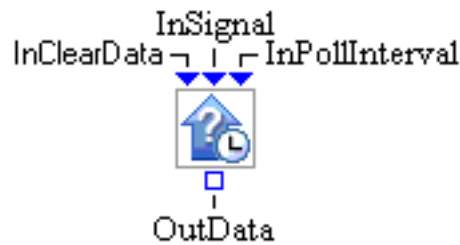
Description

The Probe block pulls and stores other block state information values at specified time intervals. Attributes to be pulled from other blocks are named in the Probe block's Attributes Table. The names are arbitrary but must be unique within each Probe block. When values are pulled from other block ports, they are stored in a data model object. The data model object can be accessed through the block's OutData port. Display blocks, such as the Histogram block, are often connected to a Probe block's OutData port to visualize the data. Any block that also uses a data model object is automatically notified when the data in the data model object is modified.

You create attribute entries in the Probe block's Attributes Table by clicking the **Add** button on the properties dialog box **Attributes** page. This results in a new attribute entry (with a default name and type) being added to the Attributes Table. You can edit the **Name** and **Type** of the attribute directly in the table. The attribute names listed in the Probe block's Attributes Table must be unique. You can change the attribute type through a drop-down list in the **Type** table cell. You can delete attributes from the Attributes Table by selecting the attribute row in the table and then clicking the **Remove** button. Clicking the **Apply** button causes all entries in the Attributes Table to be pushed to the Probe block and any input value ports to be created or deleted as needed.

Each attribute in the Probe block's Attributes Table has an input value port associated with it. At the specified time interval, the Probe block attempts to pull values from each of its attribute input ports and store those values in its data model instance. The frequency with which the Probe block pulls values is controlled through its **Poll Interval** options. You can decide to pull at a constant interval by selecting the **Constant** option and entering a valid value in the **Interval** field. You can also have the Probe block pull a value from its InPollInterval port to determine when the next attributes pull should be scheduled. In this case, you must select the **Port** option and attach a valid numeric source (such as a Numeric Source block) to the InPollInterval port.

You can limit the number of observations stored in a Probe block through its **Capacity** control. If the capacity is exceeded, a warning is logged and observations are overwritten.

The Probe block can save values that it extracts to a SAS data set or JMP table. Saving options are available on the Save dialog box.

When a true Boolean value arrives at the InClearData port, it is used as a signal to clear all the data that have been collected by the Probe block up to that time during the simulation execution. If the InClearData port receives a false value, the signal is ignored and data are not cleared. When a true Boolean value arrives at the InSignal port, the Probe block attempts to pull values from each of its attribute input ports and store those values in its data model instance. If the InSignal port receives a false value, the signal is ignored.

Fixed Ports

InClearData	Input Boolean port that is used to clear all collected data.
InSignal	Input Boolean port that is used to signal the Probe block to pull a value from each attribute input port and store those values in its data model.
InPollInterval	Input numeric port for how long to wait before the next time the Probe block pulls and stores data from its input attribute value ports. This port is used only if the Probe block is configured to use the **Port** option for its **Poll Interval** in the properties dialog box.
OutData	Output port for the latest updated data model object that contains the data held by the Probe block.

Properties Dialog Box Controls

Add	Adds a new attribute to the Attributes Table.
Remove	Deletes an attribute that has been selected from the Attributes Table.
Capacity	Controls how many observations the Probe block stores.
Poll Interval	Selecting the **Constant** option and entering a valid value in the **Interval** field causes the Probe block to pull at a constant time interval. Selecting the **Port** option causes the Probe block to pull a value from the InPollInterval port to determine its next sampling time.

Save Dialog Box Controls

Automatic Save	Turns on or off automatic saving of any collected data at the end of each design point replication run. If automatic saving is turned on, data are saved to a file with the base filename specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder that contains your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, then any collected data are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.
Save Now	Forces the Probe block to attempt an immediate save of any collected data. Data are saved to the same location as when automatic saving is turned on.
Location	Displays the pathname of the folder for the file in which to save any collected data.
Base File Name	Specifies the base filename for the SAS data set or JMP table that is used to save any collected data. This name will be the prefix of the actual filename. The zero-based index of the design point and the zero-based index of the replication number will be added as suffixes to the filename, separated by underscore characters. For example, the data for the first replication of the first design point will be saved in a file named *BaseFileName_0_0*, and the data for the second replication of the first design point will be saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors	Capacity (integer), PollingInterval (double)
Responses	None

Observation Source Block
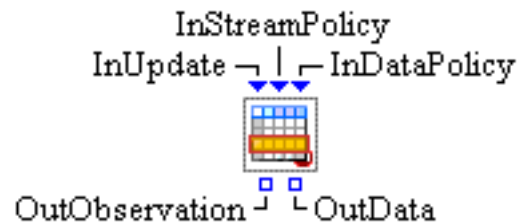
Description

The Observation Source block provides a stream of data observation (row) objects from a SAS data set or JMP table. Each time an observation object is pulled from the OutObservation port, the block pulls a new observation of the data set. If you reach the last observation from the data set, the process resets to the beginning of the data set.

Fixed Ports

InUpdate	Input Boolean value port that signals an update of input data and stream parameter specifications. The new specifications are pulled from the InStreamPolicy or InDataPolicy ports (or both) if these ports are connected. A false Boolean value is ignored.
InStreamPolicy	Input string value port from which the new stream parameter specifications are pulled when an update signal is received. The format for specifying the value is as follows: Lazy Loading == *BooleanValue* where *BooleanValue* is true or false.
InDataPolicy	Input string value port from which the new input data specifications are pulled when an update signal is received. The format for specifying the value is as follows: File Path == *filePathValue* where *filePathValue* is the path name plus file name for the SAS data set or JMP table. Quotation marks are not required around the *filePathValue* string value.
OutObservation	Output observation object port for observations to be pulled.
OutData	Output data model port for accessing the data model object that holds the contents of the input data set.

Properties Dialog Box Controls

Input Data This area provides fields for modifying the input data specifications associated with the Observation Source block. The **File Path** field specifies the absolute or relative file path for the input SAS data set or JMP data table file. The **Load from Remote SAS Workspace Server** check box indicates that the input SAS data set file is to be loaded from the remote SAS Workspace Server host.

Stream Parameters This area provides fields for modifying the stream parameter specifications that control how the stream of observations from the Observation Source block is prepared. The field **Lazy Loading** is Boolean. If the **Lazy Loading** field is false, then the input data set file has to be loaded at the start of simulation. Otherwise, the data file is loaded only when its contents are needed during simulation.

Reset Sampling at Update This option resets the current data set back to the first observation when a true Boolean value is received at the InUpdate port. If the InDataPolicy port is also connected and a new data set is specified, then the new data set starts at the beginning with the first observation if the **Reset Sampling at Update** option is selected.

Candidates for Design of Experiments

Factors DataStreamDescription (string), InputDataPolicy (string), ResetStreamAtUpdate (Boolean)

The format for specifying the value of the InputDataPolicy factor is as follows:

File Path==*filePathValue*

where *filePathValue* is the path name plus file name for the SAS data set or JMP table. Quotation marks are not required around the *filePathValue* string value. The format for specifying the value of the DataStreamDescription factor is as follows:
Lazy Loading==*BooleanValue*
where *BooleanValue* is either true or false.

Responses None

Stats Collector Block
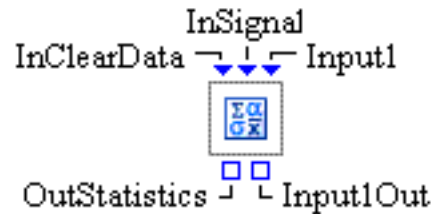
Description

The Stats Collector block is used to calculate simple statistics on its incoming input data. You can define as many data inputs on the block as needed using the **Inputs** table in the dialog box. A new input port is created for each row in the **Inputs** table. You select which statistics you want to compute using the **Statistics Selection** hierarchy provided on the **Properties** tab. The Stats Collector block creates an output data model object to hold the computed statistics with one row for each input and one column for each selected statistic. This data model object can be accessed through the block's OutStatistics port.

In the **Inputs** table you can choose to have the statistics associated with an input entry computed as time-weighted statistics by checking the **Time Persistent** check box in the corresponding table entry. You can store the individual data values associated with a specific input port in a data model object for later access by selecting the DataModel Port option in an **Inputs** table entry. An individual output port is created for each DataModel Port selection.

You choose the statistics you want to be computed from the **Statistics Selection** hierarchy. By default, the Stats Collector block calculates the statistics at the end of each design point replication run. You can use the InSignal port to force a statistics calculation during a simulation run. You can also use the InClearData port to clear all accumulated data at any point during the simulation run. Both of these ports expect Boolean values as inputs.

The Stats Collector block can store the statistics it calculates using its data collection facility. This data can be saved to a SAS data set or JMP table and can also be passed to other Simulation Studio blocks. Values are stored in a data model object, and you can access the data model object through the block's OutStatistics port.

Properties Dialog Box Controls

Add	Adds a new input entry to the **Inputs** table.
Remove	Deletes an input entry that has been selected from the **Inputs** table.
Apply	All entries in the **Inputs** table are pushed to the Stats Collector block, and input ports (and output ports) are created or deleted as needed. Statistic selections are pushed to the Stats Collector block.

Save Dialog Box Controls

Automatic Save Turns on or off automatic saving of any collected data at the end of each design point replication run. If automatic saving is turned on, data are saved to a file with the base file name specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder that contains your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, any collected statistics are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.

Save Now Forces the Stats Collector block to attempt an immediate save of any collected data. Data are saved to the same location as when automatic saving is turned on.

Location Displays the pathname of the folder for the file in which to save any collected data.

Base File Name Specifies the base file name for the SAS data set or JMP table that is used to save any collected data. This name will be the prefix of the actual file name. The zero-based index of the design point and the zero-based index of the replication number will be added as suffixes to the file name, separated by underscore characters. For example, the data for the first replication of the first design point will be saved in a file named *BaseFileName_0_0*, and the data for the second replication of the first design point will be saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors None

Responses None

Queue Stats Collector Block

Description

The Queue Stats Collector block accumulates statistics generated by Queue blocks in your model. In the properties dialog box associated with the Queue Stats Collector block, you can select from a list of available Queue blocks from which you want to collect statistics.

By default, statistics are gathered from the selected blocks at the end of each design point replication run. Options are provided to collect statistics on a continuous basis (whenever the statistics change) or to force an instantaneous update of the statistics.

The Queue Stats Collector block uses its data collection facility to store values it collects. The statistics can be saved to a SAS data set or JMP table. The statistics are stored in a data model object, which can be accessed through the block's OutData port. To visualize the statistics, you can connect a display block (such as the Bar Chart block) to the OutData port. Any block connected to the OutData port is automatically notified when the statistics in the data model object are modified.

When a true Boolean value arrives at the InClearData port, it will be used as a signal to clear all the data collected up to that time during the simulation execution. If the InClearData port receives a false value, the signal will be ignored and data will not be cleared.

Fixed Ports

OutData   Output port for the latest updated data model object that contains the statistics held by the Queue Stats Collector block.

InClearData   Input Boolean port that is used to clear all collected data.

Attributes Dialog Box Controls

You use the hierarchical check-box tree to choose Queue blocks in the simulation model from which you want to collect statistics. The leaf nodes in the tree represent Queue blocks in the model, whereas the intermediate tree nodes represent compound blocks or submodels, which can contain one or more Queue blocks. Selecting the check box for a Queue block designates that block as a target for statistic collection.

Continuous Collection   Turns on or off statistics collection during a simulation run whenever a monitored block changes state. If Continuous Collection is not selected, then statistic collection will only occur at the end of the run.

Now   Forces the Queue Stats Collector block to attempt an immediate collection of any statistics.

Save Dialog Box Controls

Automatic Save   Turns on or off automatic saving of any collected statistics at the end of each design point replication run. If automatic saving is turned on, statistics are saved to a file with the base filename specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder that contains your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, any collected statistics are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.

Save Now   Forces the Queue Stats Collector block to attempt an immediate save of any collected statistics. Statistics are saved to the same location as when automatic saving is turned on.

Location   Displays the pathname of the folder for the file in which to save any collected statistics.

Base File Name   Specifies the base filename for the SAS data set or JMP table that is used to save any collected statistics. This name is the prefix of the actual filename. The

zero-based index of the design point and the zero-based index of the replication number are added as suffixes to the filename, separated by underscore characters. For example, the statistics for the first replication of the first design point are saved in a file named *BaseFileName_0_0*, and the statistics for the second replication of the first design point are saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors	None
Responses	None

Server Stats Collector Block

Description

The Server Stats Collector block accumulates statistics generated by Server blocks in your model. In the properties dialog box associated with the Server Stats Collector block, you can select from a list of available Server blocks from which you want to collect statistics.

By default, statistics are gathered from the selected blocks at the end of each design point replication run. Options are provided to collect statistics on a continuous basis (whenever the statistics change) or to force an instantaneous update of the statistics.

The Server Stats Collector block uses its data collection facility to store statistics it collects. The statistics can be saved to a SAS data set or JMP table. The statistics are stored in a data model object, which can be accessed through the block's OutData port. To visualize the statistics, you can connect a display block (such as the Bar Chart block) to the OutData port. Any block connected to the OutData port is automatically notified when the statistics in the data model object are modified.

When a true Boolean value arrives at the InClearData port, it will be used as a signal to clear all the data collected up to that time during the simulation execution. If the InClearData port receives a false value, the signal will be ignored and data will not be cleared.

Fixed Ports

OutData	Output port for the latest updated data model object that contains the statistics held by the Server Stats Collector block.
InClearData	Input Boolean port that is used to clear all collected data.

Attributes Dialog Box Controls

You use the hierarchical check-box tree to choose Server blocks in the simulation model from which you want to collect statistics. The leaf nodes in the tree represent Server blocks in the model, whereas the intermediate tree nodes represent compound blocks or submodels, which can contain one or more Server blocks. Selecting the check box for a Server block designates that block as a target for statistic collection.

Continuous Collection   Turns on or off statistics collection during a simulation run whenever a monitored block changes state. If Continuous Collection is not selected, then statistic collection will only occur at the end of the run.

Now   Forces the Server Stats Collector block to attempt an immediate collection of any statistics.

Save Dialog Box Controls

Automatic Save   Turns on or off automatic saving of any collected statistics at the end of each design point replication run. If automatic saving is turned on, statistics are saved to a file with the base filename specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder that contains your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, any collected statistics are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.

Save Now   Forces the Server Stats Collector block to attempt an immediate save of any collected statistics. Statistics are saved to the same location as when automatic saving is turned on.

Location   Displays the pathname of the folder for the file in which to save any collected statistics.

Base File Name   Specifies the base filename for the SAS data set or JMP table that is used to save any collected statistics. This name is the prefix of the actual filename. The zero-based index of the design point and the zero-based index of the replication number are added as suffixes to the filename, separated by underscore characters. For example, the statistics for the first replication of the first design point are saved in a file named *BaseFileName_0_0*, and the statistics for the second replication of the first design point are saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors   None

Responses   None

Resource Stats Collector Block

Description

The Resource Stats Collector block accumulates statistics for resource entities in your model. In the Block Properties dialog box that is associated with the Resource Stats Collector block, you specify the resources for which you want to collect statistics and the types of statistics you want to collect. Statistics are gathered on a continuous basis (whenever the statistic changes). The Resource Stats Collector block is very flexible and enables you to base definitions of groups of resource entities on both entity type and Boolean rules for attribute values. You define statistics that are based on certain criteria for each group of resource entities. Possible criteria include whether the resource entities in the group are seized, whether the resource entities in the group are in a particular resource state, and whether the attribute values of the resource entities in the group satisfy a particular Boolean expression. For example, you could define a statistic named "utilization" that computes a statistic of type **%AvgAvail** that has the criterion **Seized**=TRUE. The result would be the percentage of time that the resource entities in the defined group are seized ("busy"), relative to the time-averaged number of available resource units in the group during the simulation run.

The Resource Stats Collector block uses its data collection facility to store values it collects. The statistics can be saved to a SAS data set or JMP table. The statistics are stored in a data model object, which can be accessed through the block's OutData port. To visualize the statistics, you can connect a display block (such as the Bar Chart or Table block) to the OutData port. Any block connected to the OutData port is automatically notified when the statistics in the data model object are modified.

When a true Boolean value arrives at the InClearData port, it will be used as a signal to clear all the data collected up to that time during the simulation execution. If the InClearData port receives a false value, the signal will be ignored and data will not be cleared.

Fixed Ports

OutData   Output port for the latest updated data model object that contains the statistics held by the Resource Stats Collector block.

InClearData   Input Boolean port that is used to clear all collected data.

Groups Dialog Box Controls

Use the **Groups** table to define the groups (collections of resource entities) for which you want to collect statistics. Each group is represented by a single row in the **Groups** table. For each group, you can set criteria (columns in the **Groups** table) to restrict the resources included in the group.

Add   Defines a new group for which to collect statistics. Each group represents one observation (row) in the collected data. A group has the following properties: **Name**, **Entity Types**, and **Attribute Rule**. You can edit each property directly in the **Groups** table.

- **Name** specifies a name for the group of resource entities. This name appears as the first variable in the observation, with the title GroupName.

- **Entity Types** (optional) restricts the group to include only those resource entities that have a particular entity type.

- **Attribute Rule** (optional) restricts the group to include only those resource entities that satisfy the specified Boolean expression for their attributes. You can type the expression in the Attribute Rule field, or you can right-click on the Attribute Rule field and select the **Edit** option to open the Edit Expression window. For more information about how to write the Boolean expression, see Appendix F, "Expressions."

  If **Entity Types** and **Attribute Rule** are left blank, all resource entities in your model are included in the group.

Remove   Removes the selected groups from the collected data.

Statistics Dialog Box Controls

You use the **Statistics** table to name and define the statistics to be gathered for the defined groups. Each statistic is represented by a single row in the **Statistics** table. For each statistic, you set properties (columns in the **Statistics** table) that define the rules for how the statistic is calculated. Specifically, the **Seized**, **State**, and **Attribute Rule** fields define the resource entity criteria that are used in calculating the statistic.

Resource entities contain resource units, as specified in the ResourceUnits attribute field. A resource entity can represent multiple resource units. The total number of available units of a resource in a group is the sum of the ResourceUnits attribute values for all resource entities in the group. A resource entity must meet all of the specified criteria in order for its associated resource units to be included in the computation of the statistic.

Add   Defines a new statistic in the collected data model. Each statistic represents one variable (column) in the collected data model. A statistic has the following properties: **Name**, **Statistics**, **Seized**, **State**, and **Attribute Rule**. You can edit each property directly in the **Statistics** table.

- **Name** specifies the name of the statistic in the data model object.

- **Statistics** specifies how to calculate the statistic for each defined resource entity group. The following definitions are used in the statistics descriptions: $N(t)$ is the total number of resource units in the group at time t, $S(t)$ is the number of resource units in the group whose associated resource entity satisfies the criteria at time t, and T is the simulation run length.

  - **%AvgAvail** is the percentage of time that the resource units in a group meet the specified criteria relative to the time-averaged number of available resource units in the group during the simulation run. The statistic is computed as a ratio of two time-averaged statistics. The denominator of this ratio is computed as $\int N(t)/T$ and the numerator of the ratio is computed as $\int S(t)/T$ so that the statistic **%AvgAvail**$=\int S(t)/\int N(t)$.

- **TimeAvg** is the time-weighted average of the total number of resource units in the group that satisfy the specified criteria: **TimeAvg** = $\int S(t)/T$.

- **Current** is the proportion of resource units in the group that meet the criteria for the statistic, relative to the number of available resource units at time *t*: **Current**=$S(t)/N(t)$. At the end of a design point replication, **Current** equals the proportion of resource units in the group that meet the criteria for the statistic when the design point replication ends. This statistic is particularly useful as an indication of how the proportion of resource units that meet the criteria changes with time.

- **Min** is the minimum value of the **Current** statistic.

- **Max** is the maximum value of the **Current** statistic.

- **Count** is the current number of resource units in the group whose associated resource entity meets the criteria for the statistic. At the end of a design point replication, it holds the number of resource units in the group that meet the criteria for the statistic when the design point replication ends.

Seized, **State**, and **Attribute Rule** are the resource criteria used in calculating the statistic. A resource entity must meet all of the specified criteria in order for its resource units to be included in the statistic.

- For the **Seized** criterion, **false** means a resource entity is available in a resource pool, **true** means a resource entity is not available in a resource pool, and an empty value means a resource entity can be either seized or unseized.

- For the **State** criterion, you can specify that a resource entity must have a particular state. Valid values are Functional, Failed, Maintenance, and Offlined. An empty value means a resource entity can be in any state.

- For the **Attribute Rule** criterion, you can specify that a resource entity's attribute values satisfy a Boolean expression. For more information about how to write the Boolean expression, see Appendix F, "Expressions." An empty value means a resource entity's attributes can have any value.

Remove   Removes the selected statistics from the collected data.

Save Dialog Box Controls

Automatic Save   Turns on or off automatic saving of any collected statistics at the end of each design point replication run. If automatic saving is turned on, statistics are saved to a file with the base filename specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder that contains your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, any collected statistics are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.

|	
Save Now	Forces the Resource Stats Collector block to attempt an immediate save of any collected statistics. Statistics are saved to the same location as when automatic saving is turned on.
Location	Displays the pathname of the folder for the file in which to save any collected statistics.
Base File Name	Specifies the base filename for the SAS data set or JMP table that is used to save any collected statistics. This name is the prefix of the actual filename. The zero-based index of the design point and the zero-based index of the replication number are added as suffixes to the filename, separated by underscore characters. For example, the statistics for the first replication of the first design point are saved in a file named *BaseFileName_0_0*, and the statistics for the second replication of the first design point are saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors	None
Responses	None

Dataset Holder Block
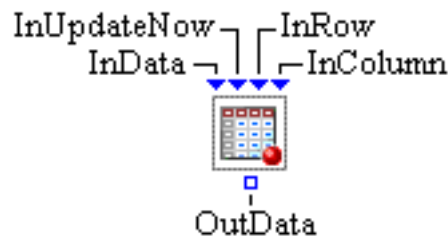
Description

The Dataset Holder block serves as a holding facility for a data model object, whose contents resemble those of a SAS data set or JMP data table. During a simulation run, the data contents (including data cell values and observation objects) can be pulled through query output ports.

Custom query output ports can be configured for the Dataset Holder block to allow either the data cell values or the observation objects to be pulled from the block, based on a set of output query criteria. When a pull request arrives at an output port, the Dataset Holder block applies the output query criteria associated with that port to the data contents it is currently holding. The resulting value or observation object is then passed out the output port.

The Dataset Holder block also supports *dynamic located queries* with dynamic row or column index values (or both) that are not prespecified. When a dynamic query output is pulled and its output processing is activated at simulation time, the needed index values are pulled dynamically through the InRow or InColumn port (or both).

Fixed Ports

InData   Input data port for the entering data model object.

OutData   Output data port for accessing the data model object held in the Dataset Holder block.

InUpdateNow   Input Boolean port that forces the Dataset Holder block to pull a data model object from the first (zero-ordered) link connected to its InData port when a value of true is passed in.

InRow   Input number port for dynamically providing the row index, if needed, of a leaving query output object.

InColumn   Input number port for dynamically providing the column index, if needed, of a leaving query output object.

Dialog Box Controls

Query Outputs   The **Query Outputs** table defines the value or observation query output ports for the Dataset Holder block. Each output port has an associated name and set of query criteria that determine the result of the query to be pulled from the port. Click **Add Query** to add a new row to the **Query Outputs** table that represents a new output port. The column values for the new row can be edited directly in the table.

- **Port Name** uniquely names the query output port for the Dataset Holder block.
- **Target Type** specifies the value type for the query outputs to be pulled from the port.
- **Row Index** optionally specifies the observation (row) index of the query output to be pulled from the port. If the observation index is not specified, the index can be pulled from the InRow port dynamically every time the output port is pulled.
- **Column Index** specifies the variable (column) index of the query output to be pulled from the port. If the variable index is not specified, its actual value can be pulled from the InColumn port dynamically every time the output port is pulled.

Click **Remove Query** to remove the selected rows from the **Query Outputs** table, indicating that the corresponding query output ports should be removed.

Apply   Saves all information to the Dataset Holder block, creating or removing query output ports as needed.

Candidates for Design of Experiments

Factors   None

Responses   None

Dataset Writer Block
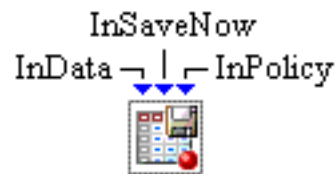
Description

The Dataset Writer block saves the contents of a data model object as either a SAS data set or JMP data table during a simulation run.

The data saving operation is triggered by a saving signal to the InSaveNow port. The data model to output can be pushed into the Dataset Writer block through the InData port before the saving signal arrives.

When a true Boolean value arrives at the InSaveNow port, it is used as a signal to save the contents of the data model object that is currently provided to the Dataset Writer block. If the data model has never been provided before the saving signal arrives, the Dataset Writer block attempts to pull a data model to save after receiving the saving signal.

If the InSaveNow port receives a false value, the signal is ignored and data are not saved.

The output file location can be specified statically before simulation using the Save dialog box. It can also be dynamically generated and pushed into the Dataset Writer block during simulation through the InPolicy port.

Fixed Ports

InData   Input data port for an entering data model object.

InSaveNow   Input Boolean port for saving the current data model to an output file. If the data model is not available yet, the Dataset Writer block pulls a data model object from the first link that connects to the InData port before saving.

InPolicy   Input string port for dynamically providing the output file location, if needed, for the current data model object.

Save Dialog Box Controls

Automatic Save   Turns on or off the automatic saving of any data in the existing data model at the end of each design point replication run. If automatic saving is turned on, data are saved to a file with the base filename specified in the **Base File Name** field. Simulation Studio automatically determines the pathname of the folder for this file based on the pathname of the folder that contains your saved project. If the **Submit to Remote SAS Workspace Server** option is selected, any collected data are saved to a file on a remote SAS server. Simulation Studio automatically determines the pathname of the folder for this file on the remote SAS server by using the **Default File Path** specified in the Simulation Studio Configuration dialog box.

Save Now	Forces the Dataset Writer block to immediately attempt to save any data in the existing data model. Data are saved to the same location as when automatic saving is turned on.
Location	Displays the pathname for the file in which to save any data in the existing data model.
Base File Name	Specifies the base filename for the SAS data set or JMP table that is used to save any data in the existing data model.

If the saving operation is triggered by a saving signal during a simulation replication run, this name or the filename from InPolicy port is the actual filename. At the end of the replication, if the **Automatic Save** option is enabled, this name is the prefix of the actual filename. The zero-based index of the design point and the zero-based index of the replication number are added as suffixes to the filename, separated by underscore characters. For example, the data for the first replication of the first design point is saved in a file named *BaseFileName_0_0*, and the data for the second replication of the first design point is saved in a file named *BaseFileName_0_1*.

Candidates for Design of Experiments

Factors	None
Responses	None

Data Trimmer Block
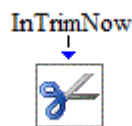
Description

You can use the Data Trimmer block to notify data collection blocks (such as the Bucket, Probe, Stats Collector, and Number Holder blocks) to clear or reset their accumulated data during a simulation run. The accumulated data for a particular block can be in the form of individual data values or computed statistics. For some blocks (such as the Number Holder block), the reset also causes the block's **Current Value** to reset to the block's **Default Value**. You select which data collection blocks you want to notify for trimming (clearing) by using a hierarchical check-box tree that displays potential block targets.

When a true Boolean value arrives at the InTrimNow port of the Data Trimmer block, it is used as a signal to notify the targeted data collection blocks to clear or trim their data. If the InTrimNow port receives a false value, then the signal is ignored.

Fixed Ports

InTrimNow   Input Boolean port that triggers the data trimming operation in the targeted data collection blocks.

Properties Dialog Box Controls

Targets   Provides a hierarchical check-box tree to display and choose data collection blocks in the simulation model to be targeted for trimming during simulation execution. The leaf nodes in the tree represent specific data collection blocks in the model, whereas the intermediate tree nodes represent compound blocks or submodels, which can contain one or more data collection blocks. Selecting the check box for a data collection block designates that block as a target for trimming by the Data Trimmer block during simulation execution.

Candidates for Design of Experiments

Factors   None

Responses   None

Histogram Block

Description

The Histogram block creates a visual estimate of the distribution of data from a discrete or continuous variable. The range of the variable is divided into a certain number of subintervals (bins). The height of the bar in each bin is proportional to the number of data points that have values in that bin. The Histogram block expects a data model object as input via its InData port. Some examples of blocks that can produce a data model object as output are the Bucket, Number Holder, and Stats Collector blocks. You must supply the name of the variable from the data model object to be used to construct the histogram bins or bars. Context-sensitive pop-up menus are available on the plot for manipulating various aspects of the plot such as axis scaling; right-click on the histogram display to access these menus.

Fixed Ports

InData   Input data port for a data model object to enter the block.

Properties Dialog Box Controls

Name   Specifies the name of the variable from the data model object to use to create the histogram.

Candidates for Design of Experiments

Factors   None

Responses   None

Bar Chart Block

InData
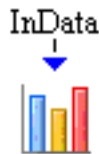
Description

The Bar Chart block graphically depicts the distribution of data from a discrete variable. The height of each bar represents the frequency, which is either the number of data points in each category or the sum of the attribute values of a particular attribute in each category. The Bar Chart block expects a data model object as input via its InData port. Some examples of blocks that can produce a data model object as output are the Bucket, Number Holder, and the Stats Collector blocks. You must supply the names of the X and Frequency (optional) variables from the data model object to be used to construct the bar chart display. Context-sensitive pop-up menus are available on the plot for manipulating various aspects of the plot such as axis scaling; right-click on the bar chart display to access these menus.

Fixed Ports

InData   Input data port for a data model object to enter the block.

Properties Dialog Box Controls

Variables   **X** specifies the name of the variable from the data model object used to categorize data on the X axis. **Frequency** specifies the value to show on the Y axis for each category. You can choose **By Count** to calculate frequency as the total number of items in each category. Alternatively, you can choose **By Variable** to calculate frequency as the sum of a particular **Variable** for all items in each category.

Candidates for Design of Experiments

Factors	None
Responses	None

Scatter Plot Block

Description

A Scatter Plot block displays a graphical representation of the relationship between two variables. The Scatter Plot block expects a data model object as input via its InData port. Some examples of blocks that can produce a data model object as output are the Bucket, Number Holder, and the Stats Collector blocks. You must supply the names of the X and Y variables from the data model object to be used to construct the scatter plot display. Context-sensitive pop-up menus are available on the plot for manipulating various aspects of the plot such as axis scaling; right-click on the scatter plot display to access these menus.

Fixed Ports

InData	Input data port for a data model object to enter the block.

Properties Dialog Box Controls

Variables	Specifies the names of the **X** and **Y** variables from the data model object to use to create the scatter plot.

Candidates for Design of Experiments

Factors	None
Responses	None

Box Plot Block

Description

The Box Plot block is a schematic summary of the distribution of data from a continuous numeric variable. The central line in a box plot indicates the median of the data, and the bottom and top of the box indicate the first and third quartiles (that is, the 25th and 75th percentiles). Extending from the box are whiskers that represent data that are a certain distance from the median. Beyond the whiskers are outliers—observations that are relatively far from the median.

The Box Plot block expects a data model object as input via its InData port. Some examples of blocks that can produce a data model object as output are the Bucket, Number Holder, and the Stats Collector blocks. You must supply the name of the Y variable from the data model object to be used to construct the box plot display. Optionally you can also provide the name of a variable to be used as a group variable for producing individual box plots for each unique category found in the group variable. Context-sensitive pop-up menus are available on the plot for manipulating various aspects of the plot such as axis scaling; right-click on the scatter plot display to access these menus.

Fixed Ports

InData   Input data port for a data model object to enter the block.

Properties Dialog Box Controls

Variables   Specifies the name of the **Y** variable from the data model object to use to create the box plot. Optionally you can select the **Use Groups** option and provide the name of the **Group** variable from the data model object to use to create multiple box plots.

Candidates for Design of Experiments

Factors   None

Responses   None

Table Block

InData

Description

The Table block displays a tabular representation of a data model. The Table block expects a data model object as input via its InData port. Some examples of blocks that can produce a data model object as output are the Bucket, Number Holder, and the various Stats Collector blocks.

Fixed Ports

InData   Input data port for a data model object to enter the block.

Candidates for Design of Experiments

Factors   None

Responses   None

Comment Block

Description

The comment block is used to hold text comments that describe the model or a portion of the model. This block is for visual purposes only and has no effect on running the model.

To enter or edit the comment, first click on the icon in the upper left corner in order to activate the comment block. Then click in the editor area to activate the editor.

The comment can contain multiple lines of text, and it can be resized by clicking and dragging any edge or corner while the comment is active for editing.

Candidates for Design of Experiments

Factors   None

Responses   None

Overview of the Resource Template

The Simulation Studio Resource template provides a collection of blocks used to manipulate resources in a simulation model.

Seize Block

InFocusedResources

InEntity –▸ [icon] □– OutEntity
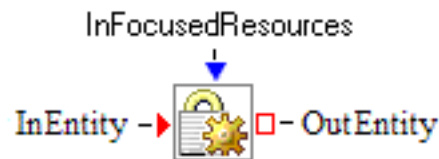
Description

The Seize block obtains resource entities from resource holding blocks (for example, Resource Pool blocks) and allocates them to a controlling entity. The controlling entity must have acquired all the required resources before it can pass through the Seize block.

The required resources are specified by one or more resource constraints. Each resource constraint is defined in the Seize block's **Constraints** properties dialog box table and is associated with an input resource entity port on the block. The input resource ports are meant to be connected to resource holding blocks. When a controlling entity attempts to enter a Seize block, the resource constraints associated with all resource ports are checked for availability. If all needed resources are available, they are pulled from the resource input ports and allocated to the controlling entity. If any of the resource constraints cannot be satisfied, the controlling entity is not allowed to enter the block.

The **Input Variables** table can be used to define variables that can be used in the **Constraints** table.

The InFocusedResources port, if connected, can be used to provide a reference group of resource entities as the initial set of the resources to be examined and seized from resource input ports. If no qualified resources are found among these focused resources to satisfy a resource constraint, the Seize block attempts to look for other resources to seize from the resource holder blocks connected to the input resource entity ports.

Fixed Ports

InEntity	Input entity port for entering controlling entities.
OutEntity	Output entity port for exiting controlling entities.
InFocusedResources	Input entity group port for pulling the references of a group of resource entities to be used as initial resource entity candidates to seize from the resource input ports.

Input Variables Dialog Box Controls

The Input Variables dialog box enables you to specify the table entries for input variables. Variables defined in the **Input Variables** table can be used to define constraints in the **Constraints** table.

Add   Adds a new input variable (with a default name) to the **Input Variables** table and creates a new input value port on the Seize block. Each field of an input variable can be edited directly in the table.

- The **Name** field of the entry specifies the name of the variable and the name of its associated input port. You can use the input port during the simulation execution to dynamically assign a value to the input variable. For example, an Extractor block could be connected to the input port to extract an attribute from the controlling entity to be used as the value of the input variable.

- The **ValueType** field specifies the input variable value type. The type can be number, string, or Boolean.

- The optional **Default Value** field specifies the value that will be used in the case where there is no connection to the input port associated with the variable.

Remove   Deletes an input variable that has been selected from the Input Variables table.

Constraints Dialog Box Controls

You can specify the table entries for input resource entity ports and associated resource constraints in the Constraints dialog box.

Add   Adds a new resource port (with default values for its fields) to the **Constraints** table. Each field can be edited directly in the table.

- The **PortName** field of the entry is the name of the input resource entity port to use when attempting to seize a resource entity for use by a controlling entity.

- The **Units** field specifies the desired amount of resource units in the resource to be seized using the port. Its default value is 1. If the **Units** field is left blank, a numeric port is created, from which the Seize block dynamically pulls the units value for the current controlling entity during simulation.

- The **Separable** flag indicates whether the needed resource units can be provided by two or more resource entities jointly.

- The optional **Attributes** field can be used to specify a Boolean expression that includes attribute values for the targeted resource entities. You can type the expression in the Attributes field, or you can right-click on the Attributes field and select the **Edit** option to open the Edit Expression window. For more information about how to write the Boolean expression, see Appendix F, "Expressions."

- The optional **Entity Type** field specifies the type of the resource entity to be seized.

The optional fields are available by clicking the down arrow next to the **Constraints** table.

Remove   Deletes the selected resource ports from the **Constraints** table.

Apply   All entries in the **Constraints** table are saved to the Seize block. Input value and resource entity ports are created or deleted as needed.

Candidates for Design of Experiments

Factors	None
Responses	None

Release Block

InEntity –▶ 🔧 □– OutEntity

Description

The Release block releases resource entities from a controlling entity as the controlling entity passes through the block.

The released resources are specified by one or more resource constraints. Each resource constraint is associated with an output resource port defined in the **ResourcePorts** properties dialog box table. When a controlling entity enters the block, the resource constraints associated with all resource ports are checked for matches. If matched resources are found, they are released from the controlling entity and pushed through the corresponding resource output ports. If a matched resource cannot flow out through the corresponding output resource port, the resource remains with the controlling entity.

Fixed Ports

InEntity	Input entity port for entering controlling entities.
OutEntity	Output entity port for exiting controlling entities.

Properties Dialog Box Controls

Add port	Adds a new resource port (with default values for its fields) to the **ResourcePorts** table. Each field can be edited directly in the table.

- The **PortName** field of the entry is the name of the output resource entity port to use to release a resource entity from a controlling entity.

- The **Units** field specifies the desired amount of resource units in the resource to be released from the controlling entity. Its default value is blank, which allows any resource that satisfies the other constraints to be released.

- The **Splittable** flag indicates whether the desired resource units can be obtained by splitting a resource with more units than desired.

- The **Separable** flag indicates whether the desired resource units can be provided by two or more resource entities jointly.

- The optional **Attributes** field can be used to specify a Boolean expression that includes attribute values for the targeted resource entities. You can type the expression in the Attributes field, or you can right-click on the Attributes field and select the **Edit** option to open the Edit Expression window. For more information about how to write the Boolean expression, see Appendix F, "Expressions."

- The optional **Entity Type** field specifies the type of the resource entity to be released.

The optional fields are available by clicking the down arrow next to the **ResourcePorts** table.

Remove port	Deletes the selected resource ports from the **ResourcePorts** table.
Apply	All entries in the **ResourcePorts** table are saved to the Release block, and output resource entity ports are created or deleted as needed.

Candidates for Design of Experiments

Factors	None
Responses	None

Resource Pool Block
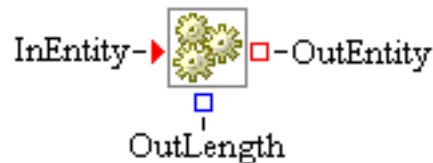
Description

The Resource Pool block accepts and maintains unseized resource entities. These resource entities can be seized later by other blocks. The Resource Pool block also processes resource requests from other blocks (a Seize block, for example) that can result in a distribution of unseized resource entities from the Resource Pool block.

A resource entity is considered *unseized* if it resides in a resource pool. Once it leaves a resource pool (and is not directly held by any other resource pool), it is treated as *seized*. Any newly created resource entity (generated outside a resource pool) is also considered unseized, even if it has not yet entered a Resource Pool block.

The Resource Pool block manages resource entities as objects. More than one type of resource entity can be maintained by an individual resource pool. The number of resource entity objects in a Resource Pool block can be monitored using its OutLength port. The Resource Pool block can also manage compatible resource entities with its merging/splitting units option.

To process a resource request from another block, the Resource Pool block chooses resource entities (from those it is maintaining) with enough units to satisfy the capacity needs of the request. Sometimes, resource entities with more units than are requested are chosen and distributed.

When the merging/splitting units option is enabled, the Resource Pool block can split resource units held in a resource entity currently in the resource pool into smaller units to satisfy a request and assign the smaller units to new resource entities. The new resource entities created by the Resource Pool block have the same entity type and attribute values as the existing resource entity, but are assigned the smaller resource units. The new resource entities are then distributed out of the resource pool. The original resource entity remains in the pool, even if its capacity reaches zero units after splitting. Similarly, when a resource entity enters the pool, its units can be merged into a compatible resource entity currently in the pool. The newly arrived resource entity is disposed and ceases to exist as an object in the simulation run. To be considered *compatible* resource entities, the resource entities must first be of the same entity type; in addition, you can specify optional key attribute fields to use for merging resources. Units merging between resource entities can take place only if their entity types and all of their key attribute fields, if any, match.

Fixed Ports

InEntity   Input entity port for entering resource entities.

OutEntity   Output entity port for exiting resource entities

OutLength   Output integer port for the number of resource entity objects currently in the pool.

Units Dialog Box Controls

Merge / split resource units among resource entities of same types   Turns on or off optional merging and splitting of resource units.

Key Entity Attribute Fields for Merging Units   Specifies optional key attribute fields to use during the units merging process when the merge/split units check box is checked.
There are two ways to specify key attribute fields for a resource entity type. For specific entity types, the key field table can be used to list key fields for any specific resource entity type. Each entry in the table contains a **Resource Entity Type** value and the corresponding **Key Attribute Field**. The latter lists either one attribute name or multiple comma-separated attribute names. You can create a new entry by clicking **Add** beside the table. Entries can be deleted from the table by selecting the entry rows in the table and then clicking **Remove**.
For any unspecified resource entity types, you can either choose the key attribute fields to be **All adjustable fields** or use **No key fields**. In the latter case, different resource entities are considered compatible for merging if and only if they are of the same resource entity type. In the former case, all adjustable attribute fields must match as well.

ResourceQueue Dialog Box Controls

Queueing Policy   Specifies the queueing policy for the queue used internally by the Resource Pool block. For more information, see the **Queueing Policy** control in the section "Queue Block" on page 141.

Candidates for Design of Experiments

Factors   QueueingPolicy (string)
For more information, see the Queueing Policy design-of-experiment factor in the section "Queue Block" on page 141.

Responses   None

Resource Scheduler Block

Description

The Resource Scheduler block arranges and performs sequences of resource adjustments on targeted resource entities. The description of an adjustment sequence is specified in a resource agenda object received through the InAgenda port from an agenda provider (for example, a Resource Agenda block). Using the properties dialog box controls, you can schedule an *appointment* with various scheduling options to request that the Resource Scheduler block process a resource agenda. The Resource Scheduler block activates its appointments and conducts the resource adjustment sequences in the corresponding resource agendas based on the scheduling options during a simulation run.

When all the adjustment actions within the sequence of an activated appointment are performed and the resulting changes pass their respective duration periods, the current processing of the appointment by the Resource Scheduler block is considered finished.

If needed, the Resource Scheduler block can activate multiple appointments and process their respective sequences of resource adjustments at the same time.

In addition to the appointments scheduled through the properties dialog box controls, you can use resource scheduling entities to request that a Resource Scheduler block process resource agendas dynamically through the InRequest port during a simulation run. The resource scheduling entity is a special type of resource entity that is defined and generated by either a Resource Scheduler block or an Entity Generator block.

Fixed Ports

InAgenda	Input object port to receive a resource agenda object.
InRequest	Input entity port to receive resource scheduling entities to dynamically schedule resource adjustments.
OutRequest	Output entity port for resource scheduling entities to leave the block.
OutBalk	Output entity port for resource scheduling entities that cannot leave through the OutRequest port.

Appointments Dialog Box Controls

Add   Adds a new appointment entry (with default values) to the **Appointments** table. The appointments in the table are used as the initial set of appointments to be processed by the Resource Scheduler block during a simulation run. Each appointment entry has the following fields:

- **Start Time** specifies the time to activate the adjustment sequence listed in the specified agenda.

- **Agenda** specifies the identifier of the agenda to use for the appointment.

- **To Repeat** specifies whether to repeat the appointment at a later time. When the Resource Scheduler finishes an appointment and the **To Repeat** option is selected and the OutRequest port is not connected, the block automatically reschedules the appointment with the current simulation time as the new start time. Otherwise, the block sends a resource scheduling entity out through its OutRequest port for that repeating appointment. This entity can be sent to the InRequest port of any Resource Scheduler block to repeat the appointment at a different time. The resource scheduling entity has a numeric **StartTime** attribute field, an object **Agenda** attribute field, and a Boolean **ToRepeat** attribute field that you can adjust dynamically for complicated scheduling needs. If the current simulation time exceeds the start time value in a newly arrived scheduling entity, then the repeating appointment is activated immediately with the current simulation time as the start time value.

- **Immediate Actions** contains three check boxes that specify the immediate actions taken by the Resource Scheduler block when it processes a resource agenda entry. The **Adjust Resources** check boxes specify when a resource entity adjustment (either capacity or state change) should occur, depending on whether that entity is unseized or seized at the time of the scheduled adjustment. In the description that follows, specific terminology is used to distinguish certain groups of resource entities for a particular resource unit adjustment action. *Potential targets* are resource entities in the functional state that satisfy the requirements specified in the **Entity Type** and **Attribute Rule** fields. Not all potential targets are necessarily adjusted. *Targets* are resource entities from the set of potential targets that have been selected for adjustment. The selection of a resource entity to be a target depends on several factors, including the **Adjust Units By** rule and an "unseized first" heuristic, which means the scheduler always attempts to adjust unseized resource entities before seized resource entities.

– **Unseized** specifies when targeted unseized resource entities are adjusted. When selected, any targeted unseized resource entities are adjusted immediately. When not selected, any targeted unseized resource entities are adjusted after all seized resource entities that are potential targets (that satisfy the **Entity Type** and **Attribute Rule** specifications) return to a Resource Pool block and become unseized.

– **Seized** specifies when targeted seized resource entities are adjusted. When selected, any targeted resource entities that are currently seized are adjusted immediately (seized resource entities are targeted only if there are remaining units to adjust after targeting all available unseized entities). Whether the adjustment is a capacity change or a state change, this could result in *preemptive changes* that could trigger a preemption in a holding block where a seized resource entity is located at the time of the adjustment. Preemption occurs if either the OutPreempt or OutResource port on the holding block where the adjusted seized entity is located is connected. Selecting the **Seized** option does not guarantee that any seized resource entities will be adjusted. Seized entities are identified to be adjusted only if there are not enough unseized entity units to cover the scheduled adjustment. If the **Seized** option is not selected, then any targeted, seized entities are adjusted after all seized resource entities that are potential targets return to a Resource Pool block and become unseized.

– **Advance Agenda** causes the Resource Scheduler block to move on to process the next agenda entry without waiting for all seized and unseized targets to be adjusted for the current agenda entry. This could result in a shorter time between actual resource adjustments than what is specified in the agenda. If the **Advance Agenda** option is not selected, the Resource Scheduler block waits for all seized and unseized targets to be adjusted before moving on to process the next agenda entry. As a result, the time between resource adjustments could be longer than the duration time specified in the resource agenda and could delay other succeeding adjustments.

If a resource scheduling entity is used to dynamically process a resource agenda, then you use the Boolean attribute fields named **AdjustUnseized**, **AdjustSeized**, and **AdvanceAgenda** on the resource scheduling entity to specify the immediate actions that the Resource Scheduler block should take.

• **Search Targets By** specifies the criteria used to identify a collection of resource entities as the potential adjustment targets of the appointment. Not all resource entities that satisfy the **Entity Type** and **Attribute Rule** specifications are necessarily selected for adjustment. The resource entities from the group of potential targets that are actually adjusted depends on several factors, including the **Adjust Units By** rule and an "unseized first" heuristic, which means the scheduler always attempts to adjust unseized entities before seized entities.

– **Entity Type** identifies the type of resource entities to adjust.

– **Attribute Rule** specifies a filtering rule that the resource entities must satisfy. The rule is a Boolean expression that includes attribute values of a resource entity that must evaluate to true for the entity to be considered as a potential adjustment target. You can type the expression in the **Attribute Rule** field, or you can right-click on the **Attribute Rule** field and select the **Edit** option to

open the Edit Expression window. For more information about how to write the Boolean expression, see Appendix F, "Expressions."

If a resource scheduling entity is used to dynamically process a resource agenda, then you use the **TargetSearchRule** attribute field to specify the criteria to identify a collection of resource entities as the potential adjustment targets. The **Target-SearchRule** attribute is of the type object; an example of the format follows:

Entity Type== Doctor; Attribute Rule== "Specialty==1 || Specialty==2"

If necessary, you can specify only the Entity Type or only the Attribute Rule part of the **TargetSearchRule** object. Double quotation marks are optional around the **Entity Type** and **Attribute Rule** values.

- **Adjust Units By** specifies the policy used to allocate or deallocate resource units among targeted resource entities when there is a scheduled resource adjustment. In the following description, a *potential* target is a resource entity that satisfies the **Entity Type** and **Attribute Rule** specifications. There are three policies to select from:

  – **FairIntegerBased** indicates that any adjustment to the units of a resource entity is made on an integer basis (integrally) and that any fractional resource units are ignored. If there is an increase in resource units, then those units are distributed integrally and as evenly as possible among all unseized resource entities that are potential targets (the "unseized first" heuristic). If the set of potential targets includes only seized resource entities and the **Seized** option in the Resource Scheduler is selected, then the units are distributed integrally and evenly among all targeted seized resource entities. If the set of potential targets includes only seized resource entities and the **Seized** option is not selected, then the units are assigned integrally and evenly among the targeted seized resource entities, but the actual capacity increase for each resource entity does not occur until all seized entities that are potential targets return to a Resource Pool block and become unseized.

  If there is a decrease in resource units, then the capacities of any unseized resource entities that are potential targets are integrally and evenly reduced. If there are not enough unseized resource units to cover the scheduled decrease (that is, there is a balance after targeting unseized entities), then the capacities of seized resource entities that are potential targets are integrally and evenly reduced either immediately (if the **Seized** option is selected) or when all seized resource entities that are potential targets return to a Resource Pool block and become unseized (if the **Seized** option is not selected). The FairIntegerBased policy is the default allocation policy.

  – **IntegerBased** indicates that any adjustment to the units of a resource entity is made on an integer basis (integrally) and that any fractional resource units are ignored. If there is an increase in resource units, then those units are distributed integrally and as evenly as possible among all unseized resource entities that are potential targets (the "unseized first" heuristic). If the set of potential targets includes only seized resource entities and the **Seized** option in the Resource Scheduler is selected, then the units are distributed integrally and evenly among all targeted seized resource entities. If the set of potential targets includes only seized resource entities and the **Seized** option is not selected, then the units are assigned integrally and evenly among the targeted seized resource entities,

but the actual capacity increase for each resource entity does not occur until all seized resource entities that are potential targets return to a Resource Pool block and become unseized. Note that in the case of an increase in units, the **IntegerBased** option is the same as the **FairIntegerBased** option.

If there is a decrease in resource units, then the capacity of the first unseized resource entity that is a potential target is integrally reduced. If the first targeted unseized resource entity does not have enough resource units to cover the capacity change, then a second unseized resource entity is targeted and its resource units are integrally reduced, and so forth. If there are not enough unseized resource units to cover the scheduled decrease, then the capacity of the first seized resource entity that is a potential target is integrally reduced either immediately (if the **Seized** option is selected) or when all seized resource entities that are potential targets return to a Resource Pool block and become unseized (if the **Seized** option is not selected). If the first targeted seized resource entity does not have enough units to cover the capacity change, then additional seized resource entities are targeted, one by one.

– **FractionBased** indicates that any adjustment to the units of a resource entity could result in fractional units. If there is an increase in resource units, then those units are distributed evenly among all unseized resource entities that are potential targets. This could result in fractional resource units for some targeted resource entities. If the set of potential targets includes only seized resource entities and the **Seized** option in the Resource Scheduler is selected, then the units are distributed evenly among all targeted seized resource entities, possibly resulting in fractional units for some resource entities. If the set of potential targets includes only seized resource entities and the **Seized** option is not selected, then the units are assigned evenly among the targeted seized resource entities, but the actual capacity increase for each resource entity does not occur until all seized resource entities that are potential targets return to a Resource Pool block and become unseized.

If there is a decrease in resource units, then the capacity of the first unseized resource entity that is a potential target will be reduced, including any fractional units. If the first targeted unseized resource entity does not have enough resource units to cover the capacity change, then a second unseized resource entity is targeted and its resource units are reduced, and so forth. If there are not enough unseized resource units to cover the scheduled decrease, then the capacity of the first seized resource entity that is a potential target is reduced either immediately (if the **Seized** option is selected) or when all seized resource entities that are potential targets return to a Resource Pool block and become unseized (if the **Seized** option is not selected). If the first targeted seized resource entity does not have enough units to cover the capacity change, then additional seized resource entities are targeted, one by one. Note that the **FractionBased** policy and the **IntegerBased** policy are the same except for the use of fractional resource units. The **FractionBased** allocation policy is the default in versions before Simulation Studio 14.1.

Remove   Deletes the selected appointment entries from the **Appointments** table.

Candidates for Design of Experiments

Factors	RankValue (double)
Responses	None

Resource Agenda Block
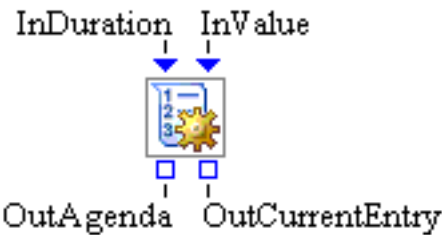
Description

The Resource Agenda block holds a resource agenda that describes and organizes a series of resource adjustment actions sequentially.

An *adjustment action* is a change to either the resource units or the resource state of one or more targeted resource entities. Each action is specified as a *resource agenda entry*, which lists the change type and value, in a resource agenda. The entry also lists a duration value to indicate how long the new value is expected to be effective starting from the change time. An agenda organizes its entries based on a relative starting time of 0. The agenda can be used and activated by a resource scheduling facility, such as a Resource Scheduler block, to schedule resource adjustments with an absolute starting time during a simulation run. The targeted resource entities are identified by the scheduling facility at simulation time.

The Resource Agenda block also supports *dynamic entries* with dynamic durations or adjustment values or both that are not prespecified. When a dynamic entry is activated by a resource scheduling facility to become the current entry at simulation time, the dynamic values are pulled through the InDuration port or InValue port or both.

Fixed Ports

InDuration	Input numeric port to pull the dynamic duration value, if needed, of the agenda entry being activated.
InValue	Input numeric port to pull the dynamic adjustment value, if needed, of the agenda entry being activated.
OutAgenda	Output object port to provide an instance of the resource agenda held in this block.
OutCurrentEntry	Output integer port for the zero-based index of the entry being activated.

Properties Dialog Box Controls

ID   Specifies a textual identifier for the agenda.

Entries   Specifies the table of agenda entries.
You can create a new resource agenda entry by clicking **Add** beside the **Entries** table. This results in a new agenda entry (with default values) being added to the **Entries** table. You can edit the field values directly in the table:

- **Duration** specifies how long the result of the resource adjustment is expected to last.

- **Value** specifies the adjustment value.

- **Value Type** specifies the adjustment type, which is one of the following:

  - **UNITS** indicates the adjustment of total resource units for targeted resource entities. The adjustment value is the new units count, which is a nonnegative number.

  - **UNITS_OFFSET** indicates the adjustment of total resource units for targeted resource entities by offsetting the current units. The adjustment value is the units offset amount. A positive offset increases the units count, and a negative offset decreases it. Because resource units should never be negative, the maximum amount of units to decrease is the existing units count.

  - **STATE** indicates the adjustment of resource state for targeted resource entities. The adjustment value is the new resource state, which can be one of Functional, Failed, Maintenance, and Offlined.

To create a dynamic entry with a dynamic numeric value for duration or adjustment value, erase the current contents of the **Duration** or **Value** field, leaving it blank. If an agenda contains dynamic entries, it is recommended to limit its use to only one resource scheduling facility to ease the modeling task of providing the needed dynamic values. For the same reason, when your model uses the transient entry index from the OutCurrentEntry port of a Resource Agenda block, the modeling process might be easier if the agenda block provides its agenda to only one resource scheduling facility.
You can delete agenda entries from the **Entries** table by selecting the entry rows in the table and then clicking **Remove**.

Candidates for Design of Experiments

Factors   None

Responses   None

Overview of the Output Analysis Template

The Simulation Studio Output Analysis template provides a collection of blocks used to analyze the output of a simulation model.

Steady State Block
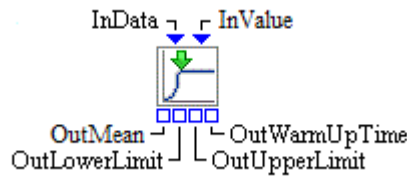
Description

The Steady State block provides an automated procedure for producing a confidence interval estimator for a steady-state mean response in a nonterminating simulation model. The procedure is based on spaced batch means (see Lada, Steiger, and Wilson 2008). You specify the precision and coverage-probability requirements for the desired confidence interval. You can also specify whether the precision is relative or absolute. You can use the Steady State block for both time-dependent and time-independent data.

The Steady State block requires a link to either its InData port or InValue port. If there are no connections to either port, or if the connection is to its InData port and the Steady State block cannot pull a data model from its InData port, then the block does not start. If you are using the InData port, then you use the properties dialog box to specify the name of the data model variable that contains the numeric values to use to construct a confidence interval. You also use controls in the properties dialog box to set the desired precision of the half-width of the confidence interval along with the coverage-probability parameter.

You can specify whether the value that you are measuring is time-persistent in nature by using the **Time Persistent** check box. The **Time Interval** field is used with time-persistent values to indicate how often the accumulated values are collapsed into a single, time-weighted average data point. For example, if the value in the **Time Interval** field is 1.0, then the computation of the time-averaged statistic occurs every 1.0 time units during the simulation run.

The Steady State block controls the length of the simulation run within the limits of the EndTime system parameter specified in the current Experiment window. If the Steady State block fails to acquire sufficient data to calculate the desired confidence interval before reaching the EndTime value of the experimental design point, the simulation terminates and no confidence interval is output. If the block is successful in calculating the desired confidence interval, it pushes the lower and upper limits of the confidence interval out through its OutLowerLimit and OutUpperLimit ports, respectively, and pushes the sample mean out through its OutMean port. If the EndTime value is infinity, then the model runs until the Steady State block can deliver a confidence interval that satisfies the specified precision requirement.

The Steady State block pushes the simulation time value associated with the last data value of the estimated warm-up period out through its OutWarmUpTime port at the point when its algorithm detects this value.

If you use multiple Steady State blocks in the same model, then the simulation terminates when one of the Steady State blocks successfully calculates a confidence interval.

Fixed Ports

InData	Input port for a data model object.
InValue	Input port for individual numeric data values.
OutMean	Output numeric port for the sample mean.
OutLowerLimit	Output numeric port for the lower limit of the confidence interval.
OutUpperLimit	Output numeric port for the upper limit of the confidence interval.
OutWarmUpTime	Output numeric port for the simulation clock time associated with the last data value of the estimated warm-up period.

Properties Dialog Box Controls

Variable Name	Specifies the name of the variable in the data model object to be used for data values.
Desired Precision	Specifies the desired relative half-width of the calculated confidence interval. For example, a desired precision of 0.075 indicates that you want the final confidence interval half-width to be within +/– 7.5% of the estimated mean.
Relative Precision	Specifies whether the precision is relative or absolute.
Beta	Specifies the coverage probability for the confidence interval. For example, a Beta value of 0.05 indicates that you want a 95% (1.0 – 0.05) confidence interval.
Time Persistent	Specifies whether the data are time-persistent or not.
Time Interval	Specifies how often to compute the time-averaged statistic when the **Time Persistent** option is selected.

Candidates for Design of Experiments

Factors	DesiredPrecision (double), PrecisionRelative (Boolean), Beta (double)
Responses	LowerLimit (double), UpperLimit (double), Mean (double)

References

Lada, E. K., Steiger, N. M., and Wilson, J. R. (2008). "SBatch: A Spaced Batch Means Procedure for Steady-State Simulation Analysis." *Journal of Simulation* 2:170–185.

Appendix B
Random Variation in a Model

Contents

Overview of Random Variation

Random and exogenous sources of variation play a central role in discrete- event simulation. Blocks such as the Entity Generator, Value Generator, Server, and Delay blocks usually require a connection to a source of variation. The principal source of variation is the Numeric Source block. The functionality of the Numeric Source block is described in Appendix A, "Templates," but this section also provides a quick overview.

The Numeric Source block provides an OutValue output port to which other blocks can connect to pull numeric values. The values produced by this block are dependent on their parameter settings. Figure B.1 shows the Block Properties dialog box for a Numeric Source block with the default settings. The **Theoretical** option is selected and the **Type** list box provides a list of the statistical distributions available in Simulation Studio for sampling purposes. You select a distribution from the **Type** list and then supply the desired parameters in the approripate fields for the distribution that you have chosen. (The details about the distributions available in Simulation Studio are presented later in this appendix.) When a request for a sample comes into the Numeric Source block, the block generates a value based on its parameter settings.

Figure B.1 Sample Numeric Source Block Dialog Box
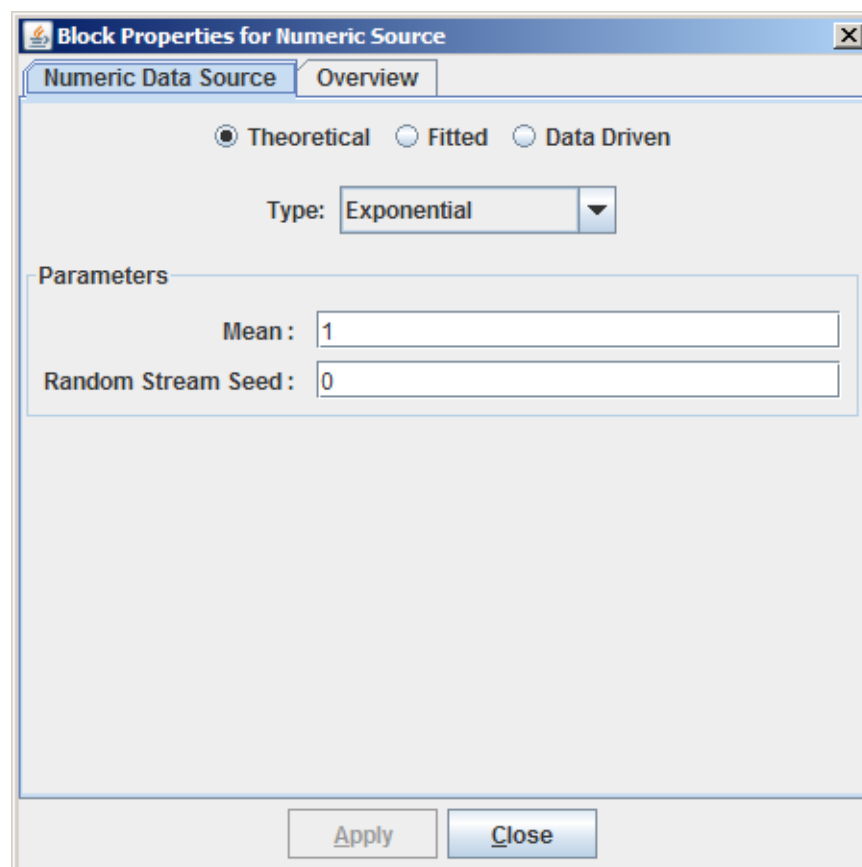
In the Numeric Source Block Properties dialog box, the **Fitted** option enables you to specify the location of a data set. Then JMP is used to automatically fit a theoretical distribution to the data. The **Type** and **Parameters** fields of the Block Properties for Numeric Source dialog box are populated with the appropriate information from the JMP distribution-fitting tool. As indicated in the distribution descriptions in this appendix, some

of the distributions available in Simulation Studio are not available in the JMP distribution-fitting tool. For more information about the **Fitted** option, see Appendix D, "Input Analysis."

The **Data Driven** option in the Numeric Source block properties has a **Type** list box that provides you with a variety of methods for generating samples that are based on a specific data set. For example, the **Discrete Empirical** and the **Empirical** options are especially useful when it is not possible to find a theoretical distribution that fits the data accurately. Furthermore, the nonhomogeneous Poisson process options **NHPP Count** and the **NHPP Rate** allow you to generate a time dependent arrival process that is based on either count or rate data. Finally, the **SAS Data Column** option can be used to read values from a SAS data set or a JMP data table that are then used directly as a source of input to a simulation model. When using this option in the Numeric Source block, you must supply the file pathname along with the column or variable name in the data set. (See Figure B.2.) Simulation Studio uses the filename extension to determine whether the file is a SAS data set or JMP data table. If a filename extension is not specified, Simulation Studio assumes the file is of the type (SAS data set or JMP table) specified in the Default Data Format section of the SAS Simulation Configuration dialog box.

Figure B.2 Sample Numeric Source Block That Uses a SAS Data Set
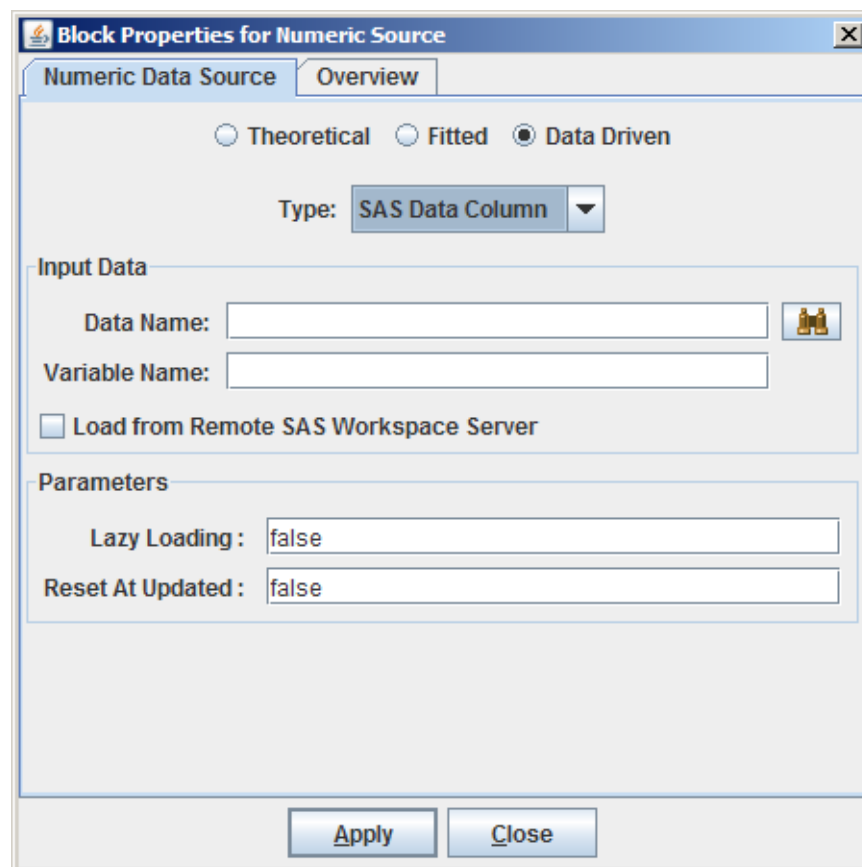
Discrete Distributions

Binomial

The probability mass function of the binomial distribution is

$$p(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}$$

for $x \in [0, 1, \ldots, n]$.

Parameters:

> n is a positive integer that represents the number of independent Bernoulli trials.
> $p \in [0, 1]$ is the probability of success on each trial.

Table B.1 shows how the binomial distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.1   Binomial Distribution Parameter Names

	Simulation Studio	JMP
n	N	n
p	Probability	p

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

```
class == Binomial;N == 5;Probability == 0.5

Probability == 0.1
```

Discrete Uniform

The probability mass function of the discrete uniform distribution is

$$p(x) = \frac{1}{j - i + 1}$$

for $x \in [i, i + 1, \ldots, j]$, where i and j are integers with $i \leq j$.

Parameters:

i is a location parameter.

$j - i$ is a scale parameter.

Table B.2 shows how the discrete uniform distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The discrete uniform distribution is not available with the Distribution option in JMP.

Table B.2   Discrete Uniform Distribution Parameter Names

	Simulation Studio	**JMP**
i	i	–
j	j	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

     class == Discrete Uniform;i == 5;j == 10

     j == 100

Geometric

The probability mass function of the geometric distribution is

$$p(x) = p(1 - p)^{x-1}$$

for $x \in \{1, 2, \ldots\}$.

Parameter:

$p \in (0, 1)$ is the probability of success on each trial.

Table B.3 shows how the geometric distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The geometric distribution is not available with the Distribution option in JMP.

Table B.3   Geometric Distribution Parameter Names

	Simulation Studio	JMP
p	Probability	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

class == Geometric;Probability == 0.5

Probability == 0.3

Negative Binomial

The probability mass function of the negative binomial distribution is

$$ p(x) = \frac{(n + x - 1)!}{x!(n - 1)!} p^n (1 - p)^x $$

for $x \in \{0, 1, \dots\}$.

Parameters:

n is a positive integer ≥ 1 which represents the number of successes in a series of independent Bernoulli trials.

$p \in (0, 1)$ is the probability of success on each trial.

Table B.4 shows how the negative binomial distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The negative binomial distribution is not available with the Distribution option in JMP.

Table B.4   Negative Binomial Distribution Parameter Names

	Simulation Studio	JMP
n	N	–
p	Probability	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

> class == Negative Binomial;N == 5;Probability == 0.5

> Probability == 0.3

Poisson

The probability mass function of the Poisson distribution is

$$p(x) = \frac{e^{-\lambda}\lambda^x}{x!}$$

for $x \in \{0, 1, \dots\}$.

Parameter:

λ is the mean, $\lambda > 0$.

Table B.5 shows how the poisson distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.5   Poisson Distribution Parameter Names

	Simulation Studio	**JMP**
λ	Mean	λ (Scale)

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

> class == Poisson;Mean == 2

> Mean == 6

Continuous Distributions

Beta

The density function of the beta distribution is

$$f(x) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)(b - a)^{\alpha+\beta-1}}(x - a)^{\alpha-1}(b - x)^{\beta-1}$$

for $a \le x \le b$. The gamma function $\Gamma(z)$ is defined for any real number $z > 0$ as

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\,dt$$

Parameters:

$\alpha > 0$ is a shape parameter.
$\beta > 0$ is a shape parameter.
a is the minimum value, $a < b$.
b is the maximum value.

Table B.6 shows how the beta distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.6 Beta Distribution Parameter Names

	Simulation Studio	**JMP**
α	Shape 1	α
β	Shape 2	β
a	Min	θ
b	Max	$\theta + \sigma$

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

class == Beta;Min == 0;Max == 1;Shape 2 = 5;Shape 1 == 1.5

Shape 1 == 5;Shape 2 == 1.5

Chi-Square

The chi-square distribution with k degrees of freedom is the same as the gamma distribution with $\alpha = \frac{k}{2}$ and $\lambda = 2$.

Parameter:

k is an integer ≥ 1 which represents the degrees of freedom.

Table B.7 shows how the chi-square distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The chi-square distribution is not available with the Distribution option in JMP.

Table B.7   Chi-Square Distribution Parameter Names

	Simulation Studio	**JMP**
k	degrees of freedom	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

    class == Chi-Square;degrees of freedom == 4

    degrees of freedom == 6

Erlang

The Erlang distribution is a special case of the gamma distribution. The density function of the Erlang distribution is

$$f(x) = \frac{1}{(k-1)!} \lambda^{-k} x^{k-1} e^{-\frac{x}{\lambda}}$$

where $x \geq 0$.

Parameters:

λ is a real number > 0.
k is an integer ≥ 1.

If $X_1, X_2, \ldots, X_k$ are independent exponential random variables with mean λ, then $X_1 + X_2 + \cdots + X_k$ has the k-Erlang distribution.

Table B.8 shows how the erlang distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The erlang distribution is not available with the Distribution option in JMP.

Table B.8   Erlang Distribution Parameter Names

	Simulation Studio	JMP
k	K	–
λ	Lambda	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

class == Erlang;K == 2;Lambda == 1.0

K == 3

Exponential

The density function of the exponential distribution is

$$f(x) = \frac{1}{\lambda} e^{-\frac{x}{\lambda}}$$

where $x \geq 0$.

Parameter:

λ is the mean, $\lambda > 0$.

Table B.9 shows how the exponential distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.9   Exponential Distribution Parameter Names

	Simulation Studio	JMP
λ	Mean	σ

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

> class == Exponential;Mean == 1.0

> Mean == 1.25

Gamma

The density function of the gamma distribution is

$$f(x) = \frac{\lambda^{-\alpha} x^{\alpha-1} e^{-\frac{x}{\lambda}}}{\Gamma(\alpha)}$$

where $x \geq 0$. The function $\Gamma(z)$ is defined in the section "Beta" on page 232.

Parameters:

α is the shape parameter, $\alpha > 0$.
λ is the scale parameter, $\lambda > 0$.

Table B.10 shows how the gamma distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.10   Gamma Distribution Parameter Names

	Simulation Studio	**JMP**
λ	Scale	σ
α	Shape	α

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

> class == Gamma;Shape == 2;Scale == 1

> Shape == 2

Johnson Bounded Distribution (JohnsonSB)

The density function of the Johnson bounded distribution (JohnsonSB) is

$$f(x) = \frac{\delta}{\lambda\sqrt{2\pi}} g'\left(\frac{x-\xi}{\lambda}\right) \exp\left(-\frac{1}{2}\left[\gamma + \delta g\left(\frac{x-\xi}{\lambda}\right)\right]^2\right)$$

where

$$g(y) = \ln\left(\frac{y}{1-y}\right)$$
$$g'(y) = \frac{1}{y(1-y)}$$

and $x \in [\xi, \xi + \lambda]$.

Parameters:

δ (delta)　　is a shape parameter, $\delta > 0$.
γ (gamma)　is a shape parameter.
ξ (xi)　　　　is the location parameter.
λ (lambda)　is the scale parameter, $\lambda > 0$.

Table B.11 shows how the Johnson bounded distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.11　Johnson Bounded Distribution Parameter Names

	Simulation Studio	**JMP**
γ	gamma	γ
δ	delta	δ
ξ	xi	θ
λ	lambda	σ

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

class == Johnson Bounded;gamma == 0;delta == 2;xi == 0;lambda == 1

gamma == 0.5;delta == 0.8

Johnson Lognormal Distribution (JohnsonSL)

The density function of the Johnson lognormal distribution (JohnsonSL) is

$$f(x) = \frac{\delta}{\lambda\sqrt{2\pi}} g'\left(\frac{x-\xi}{\lambda}\right) \exp\left(-\frac{1}{2}\left[\gamma + \delta g\left(\frac{x-\xi}{\lambda}\right)\right]^2\right)$$

where

$$g(y) = \ln(y)$$
$$g'(y) = \frac{1}{y}$$

and $x \in [\xi, \infty)$.

Parameters:

δ (delta)	is a shape parameter, $\delta > 0$.
γ (gamma)	is a shape parameter.
ξ (xi)	is the location parameter.
λ (lambda)	is the scale parameter, $\lambda = \pm 1$.

Table B.12 shows how the Johnson lognormal distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.12   Johnson Lognormal Distribution Parameter Names

	Simulation Studio	**JMP**
γ	gamma	γ
δ	delta	δ
ξ	xi	θ
λ	lambda	σ

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

    class == Johnson Lognormal;gamma == 3;delta == 2;xi == 5;lambda == 1

    lambda == -1

Johnson Unbounded Distribution (JohnsonSU)

The density function of the Johnson unbounded distribution (JohnsonSU) is

$$f(x) = \frac{\delta}{\lambda\sqrt{2\pi}} g'\left(\frac{x-\xi}{\lambda}\right) \exp\left(-\frac{1}{2}\left[\gamma + \delta g\left(\frac{x-\xi}{\lambda}\right)\right]^2\right)$$

where

$$g(y) = \ln\left[y + \sqrt{y^2 + 1}\right]$$
$$g'(y) = \frac{1}{\sqrt{y^2+1}}$$

and $x \in (-\infty, \infty)$.

Parameters:

δ (delta)   is a shape parameter, $\delta > 0$.
γ (gamma)   is a shape parameter.
ξ (xi)   is the location parameter.
λ (lambda)   is the scale parameter, $\lambda > 0$.

Table B.13 shows how the Johnson unbounded distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.13   Johnson Unbounded Distribution Parameter Names

	Simulation Studio	**JMP**
γ	gamma	γ
δ	delta	δ
ξ	xi	θ
λ	lambda	σ

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

     class == Johnson Unbounded;gamma == 0;delta == 2;xi == 0;lambda == 1

     gamma == 3;delta == 2

Lognormal

The density function of the lognormal distribution is

$$f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}}\exp\left(\frac{-(\ln(x) - \mu)^2}{2\sigma^2}\right)$$

where $x > 0$.

Parameters:

μ is the mean of $\ln(x) \sim \text{Normal}(\mu, \sigma^2)$.

σ is the standard deviation of $\ln(x) \sim \text{Normal}(\mu, \sigma^2)$, $\sigma > 0$.

Table B.14 shows how the lognormal distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.14   Lognormal Distribution Parameter Names

	Simulation Studio	JMP
μ	Mean	μ
σ	Std Dev	σ

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

class == Lognormal;Mean == 0;Std Dev == 0.5

Std Dev == 0.25

Normal

The density function of the normal distribution is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

for all real values of x.

Parameters:

μ is the mean, $\mu \in (-\infty, \infty)$.

σ is the standard deviation, $\sigma > 0$.

Table B.15 shows how the normal distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.15   Normal Distribution Parameter Names

	Simulation Studio	**JMP**
μ	Mean	μ
σ	Std Dev	σ

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

     class == Normal;Mean == 0;Std Dev == 1

     Std Dev == 0.5

Pearson Type V

The Pearson Type V distribution has the same density function as the gamma distribution with shape parameter α and scale parameter $\lambda = \frac{1}{\beta}$.

Parameters:

α is the shape parameter, $\alpha > 0$.

β is the scale parameter, $\beta > 0$.

Table B.16 shows how the Pearson Type V distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The Pearson Type V distribution is not available with the Distribution option in JMP.

Table B.16 Pearson Type V Distribution Parameter Names

	Simulation Studio	**JMP**
α	Shape	–
β	Scale	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

class == Pearson Type V;Shape == 4;Scale == 1

Shape == 2

Pearson Type VI

The density function of the Pearson Type VI distribution is

$$f(x) = \frac{\left(\frac{x}{\beta}\right)^{\alpha_1 - 1}}{\beta G(\alpha_1, \alpha_2)\left[1 + \left(\frac{x}{\beta}\right)\right]^{\alpha_1 + \alpha_2}}$$

where $x > 0$ and

$$G(\alpha_1, \alpha_2) = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)}{\Gamma(\alpha_1 + \alpha_2)}$$

The function $\Gamma(z)$ is defined in the section "Beta" on page 232.

Parameters:

α_1 is a shape parameter, $\alpha_1 > 0$.
α_2 is a shape parameter, $\alpha_2 > 0$.
β is a scale parameter, $\beta > 0$.

If X_1 and X_2 are independent random variables with $X_1 \sim \text{Gamma}(\alpha_1, \beta)$ and $X_2 \sim \text{Gamma}(\alpha_2, 1)$, then $Y = \frac{X_1}{X_2} \sim \text{PearsonTypeVI}(\alpha_1, \alpha_2, \beta)$.

Table B.17 shows how the Pearson Type VI distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The Pearson Type VI distribution is not available with the Distribution option in JMP.

Table B.17 Pearson Type VI Distribution Parameter Names

	Simulation Studio	JMP
α_1	Shape 1	–
α_2	Shape 2	–
β	Scale	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

class == Pearson Type VI;Shape 2 == 4;Shape 1 == 0.5;Scale == 1

Shape 2 == 2

Triangular

The density function of the triangular distribution is

$$f(x) = \begin{cases} \frac{2(x-a)}{(m-a)(b-a)} & a \leq x \leq m \\ \frac{2(b-x)}{(b-m)(b-a)} & m < x \leq b \end{cases}$$

where a, b, and m are real numbers with $a < m < b$.

Parameters:

a is the minimum.
b is the maximum.
m is the mode.

Table B.18 shows how the triangular distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The triangular distribution is not available with the Distribution option in JMP.

Table B.18 Triangular Distribution Parameter Names

	Simulation Studio	JMP
a	Min	–
b	Max	–
m	Mode	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

    class == Triangular;Min == 3;Max == 10;Mode == 5

    Mode == 5.5

Uniform

The density function of the uniform distribution is

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

where a and b are real numbers with $a < b$.

Parameters:

a is the minimum.
b is the maximum.

Table B.19 shows how the uniform distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block). The uniform distribution is not available with the Distribution option in JMP.

Table B.19   Uniform Distribution Parameter Names

	Simulation Studio	JMP
a	Min	–
b	Max	–

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

    class == Uniform;Min == 0;Max == 1

    Max == 10

Weibull

The density function of the Weibull distribution is

$$f(x) = \alpha \beta^{-\alpha} x^{\alpha-1} e^{-(\frac{x}{\beta})^{\alpha}}$$

for $x > 0$.

Parameters:

α is the shape parameter, $\alpha > 0$.
β is the scale parameter, $\beta > 0$.

Table B.20 shows how the weibull distribution parameter names are specified in Simulation Studio (specifically, in the Numeric Source block) and in JMP.

Table B.20   Weibull Distribution Parameter Names

	Simulation Studio	JMP
α	Shape	β
β	Scale	α

The following examples show (case-sensitive) string values that can be used as Numeric Source block DataStreamDescription factor values or InStreamPolicy port values. In these examples, the distribution and parameter names in the string value are the names that are used in the **Theoretical** option in the Numeric Source Block Properties dialog box (including any spaces or hyphens). Quotation marks are not required around the string value, and you can specify only the parameters that need to be updated (as demonstrated in the second example).

     class == Weibull;Shape == 4;Scale == 1

     Shape == 2

Empirical Distributions

Discrete

The **Discrete Empirical** option under the **Data Driven** option of a Numeric Source block requires the following input data, as shown in Figure B.3:

- **Data Name**: The location of the SAS data set or JMP table that contains the data to be used to specify a discrete empirical distribution.

- **X** : Name of the column in the data set that contains the set of n possible discrete values $x_1, x_2, ..., x_n$, ordered so that $x_1 < x_2 < ... < x_n$.

- **Y** : Name of the column in the data set that corresponds to the probability mass function values $p(x_i)$ for each $x_i : i = 1, 2, ..., n$ so that $y_i = p(x_i)$; $\sum_{k=1}^{n} p(x_k) = 1$; and $0 \leq p(x_i) \leq 1$ for $i = 1, 2, ..., n$.

Figure B.3 Discrete Empirical Option for the Numeric Source Block

Continuous

The **Empirical** option under the **Data Driven** option of a Numeric Source block requires the following input data:

- **Data Name**: The location of the SAS data set or JMP table that contains the data to be used to specify a continuous empirical distribution.

- **X** : Name of the column in the data set that corresponds to the set of n ordered values $x_1, x_2, ..., x_n$.

- **C** : Name of the column in the data set that corresponds to the cumulative probability values $c_1, c_2, ..., c_n$ so that $0 \leq c_j \leq 1$ for $j = 1, 2, ..., n$; $c_j < c_{j+1}$ for $j = 1, 2, ..., n-1$; and $c_n = 1$.

The probability density function is defined as

$$f(x) = \begin{cases} c_1 & \text{if } x = x_1 \\ c_j - c_{j-1} & \text{if } x_{j-1} \leq x < x_j, j = 2, 3, ..., n \\ 0 & \text{if } x < x_1 \text{ or } x \geq x_n \end{cases}$$

If $c_1 > 0$, then the result is a mixed continuous-discrete distribution that returns x_1 with probability c_1 and with probability $1 - c_1$ a continuous random variate on $(x_1, x_n]$ using linear interpolation. If a true continuous distribution on $[x_1, x_n]$ is desired, then specify $c_1 = 0$.

Nonhomogeneous Poisson Process

There are many systems in which the arrival rate of entities depends strongly on time (for example, arrivals of patients at an emergency room and the arrival of calls at a customer service center). In Simulation Studio, a nonhomogeneous Poisson process (NHPP) based on either count or rate data can be used to generate a time-dependent arrival process on the time interval $(0, S]$.

For either the count-based or rate-based case, a Numeric Source block with the NHPP option specified under the **Data Driven** option can be connected to the InterArrival time port of an Entity Generator block. In the Entity Generator block, the option **At First Interarrival Time** for **First Entity Creation** should be selected to ensure that the first arrival time is an actual event time for the NHPP. If either the **At Start Time** or **After Signal Arrival** option is selected, the first entity is generated at the start time specified in the Entity Generator block or the signal time and is not an arrival (event) time generated from the specified NHPP. However, all subsequent arrival times are generated from the NHPP.

Also, the start time of the NHPP might not correspond to the start time of the simulation. For example, suppose an NHPP is defined on the time interval from noon to 2:00 p.m., but the simulation start time is 8:00 a.m. In this case, if the time unit is in hours, an extra subinterval from 8:00 a.m. to noon with count (or rate) 0 could be added so that the NHPP is defined on the time interval $(0, 6]$. No events are generated for the time interval $(0, 4]$ and the process effectively begins at simulation time 4 (which corresponds to noon).

After the simulation clock reaches time S, the entity generator block associated with the NHPP shuts down and no more entities are generated. If necessary, a signal can be sent to the Entity Generator block to generate an entity at a specific time after S.

Count-Based

A method described in Leemis (2004) is used for generating arrival times from an estimated NHPP. This method uses event-time data that are given as counts that occur in disjoint subintervals (as opposed to the event times themselves). For the **NHPP Count** option under the **Data Driven** option of a Numeric Source block, the following inputs are required as shown in Figure B.4:

- **Data Name**: The location of the SAS data set or JMP table that contains the data to be used to estimate a cumulative intensity (rate) function.

- **X**: Name of the column in the data set that specifies the subinterval cutoff points $x_0, x_1, ..., x_m$ so that the NHPP has an intensity function that is piecewise constant on each subinterval $(x_0, x_1], (x_1, x_2], ..., (x_{m-1}, x_m]$. The subintervals do not necessarily have equal widths. The NHPP is defined on the time interval $(0, S]$ so that $x_0 = 0$, $x_m = S$, and m is the number of subintervals. The time units must be consistent with the data. For example, if the interval of interest is from 1:00 p.m. to 5:30 p.m., then the interval $(x_0, x_m]$ is $(0, 4.5]$ if the data are in hours or $(0, 270]$ if the data are in minutes.

- **Counts**: Name of the column in the data set where each value $c_1, c_2, ..., c_m$ is the total number of observed events in each subinterval over all replications. Specifically, c_1 is the total number of observed events in the subinterval $(x_0, x_1]$. The length of the **Counts** column should be one less than the length of the **X** column.

- **Number of Replications**: The number of realizations of the observed process.

Figure B.4 NHPP Count Option for the Numeric Source Block

Rate-Based

For the **NHPP Rate** option under the **Data Driven** option of a Numeric Source block, the following inputs are required as shown in Figure B.5:

- **Data Name**: The location of the SAS data set or JMP table that contains the data to be used to estimate a cumulative intensity (rate) function.

- **X**: Name of the column in the data set that specifies the subinterval cutoff points $x_0, x_1, ..., x_m$ so that the NHPP has an intensity function that is piecewise constant on each subinterval $(x_0, x_1], (x_1, x_2], ..., (x_{m-1}, x_m]$. The subintervals do not necessarily have equal widths. The NHPP is defined on the time interval $(0, S]$ so that $x_0 = 0$, $x_m = S$, and m is the number of subintervals. The time units must be consistent with the data. For example, if the interval of interest is from 1:00 p.m. to 5:30 p.m., then the interval $(x_0, x_m]$ is $(0, 4.5]$ if the data are in hours or $(0, 270]$ if the data are in minutes.

- **Rates**: Name of the column in the data set where each value $r_1, r_2, ..., r_m$ is the estimated rate of arrivals over each subinterval. Specifically, r_1 is the rate of arrivals over the subinterval $(x_0, x_1]$. The length of the **Rates** column should be one less than the length of the **X** column. Furthermore, the units of the rates $r_1, r_2, ..., r_m$ must be consistent with the rest of the simulation model. For example, if the units used in the model are minutes, then the rates $r_1, r_2, ..., r_m$ must be specified in number of arrivals per minute.

Figure B.5 NHPP Rate Option for the Numeric Source Block

References

Leemis, L. M. (2004). "Nonparametric Estimation and Variate Generation for a Nonhomogeneous Poisson Process from Event Count Data." *IIE Transactions* 36:1155–1160.

Appendix C
Design of Experiments

Contents

This appendix uses a modified version of the repair shop example in Chapter 2, "Overview of SAS Simulation Studio," to demonstrate how you can use SAS or JMP software to generate experimental designs for a SAS Simulation Studio model. You can find the project files for this example in the folder `\projects\ examples\docRepairshopDOE` in your Simulation Studio installation directory.

In this modified repair shop example, all parts that are fixed early at the service desk are now routed to quality control. There is also a Bucket block that records the time in system of a part before it leaves the system. Finally, there is extra modeling logic in the GenerateParts compound block to set the mean interarrival time at the beginning of each run from a factor value.

Suppose you have the option of adding more workers at the service desk and repair desk in the repair shop so that each station can have one, two, or three workers. Furthermore, suppose you have the option of adding a worker at the quality control station. These are the factors of your experiment. The responses that you could monitor are the total time in system of a part, the average utilization at all three stations (to determine whether workers are idle or overworked), the number of fixed parts, and the average waiting time at each of the three stations. The goal of this model is to determine the optimal number of workers to put at the service desk, repair station, and quality control station to minimize waiting times and the total time in system and to maximize utilization. Given these factors and responses, you might want to generate an experimental design (such as a full or fractional factorial design) to guide your simulation runs. This is a more efficient way to explore the effects of different parameters on your model responses than just randomly selecting combinations of your factor values to try. You can do this by using the Simulation Studio Experiment window and SAS or JMP software. Certain steps are required in Simulation Studio before you can generate a design by using either SAS or JMP. These steps include defining factors and responses, defining model anchors, and including factors and responses in the Experiment window.

Defining Factors and Responses

The first step in the experimental design process is to define the factors and responses in Simulation Studio. For more information about factors and responses, see Chapter 5, "Experiments." For the repair shop example, the factors NumService, NumRepair, and NumQC are defined to represent the number of workers at each of the three stations. An additional factor called MeanArrival represents the mean interarrival time between parts. The defined factor values for MeanArrival are 10, 15, and 20 minutes (corresponding to arrival rates of six, four, and three parts per hour, respectively). The purpose of including the factor for the mean interarrival time between parts is to see the effect of different part arrival rates on the optimal allocation of workers at each station (as demonstrated in the section "Analyzing Simulated Results Using JMP" on page 265).

To define the factors, first right-click the project name in the Projects panel and select **Factors** to open the Factors dialog box, as shown in Figure C.1. You specify a unique name for the factor in the **Name** field, and you use the optional **Label** field to specify a detailed description of the factor. The **Value Type** field specifies the data type of the factor in the simulation model (double, integer, text, or Boolean), and the **Default Value** field is used to set the initial value of an included factor when you add a new design point in the Experiment window. You can add values to the **Values** table to define the levels of the factor. If you use JMP to create a design, then the **Role Type**, **Value Type**, and **Values** fields are used in the design generation process. Even if you do not use JMP to generate the design, you can still use the **Role Type** and **Values** fields to define a suitable range of values for a factor, so the Factors dialog box essentially becomes a repository of information about the factors for a project.

Figure C.1 Factors Dialog Box
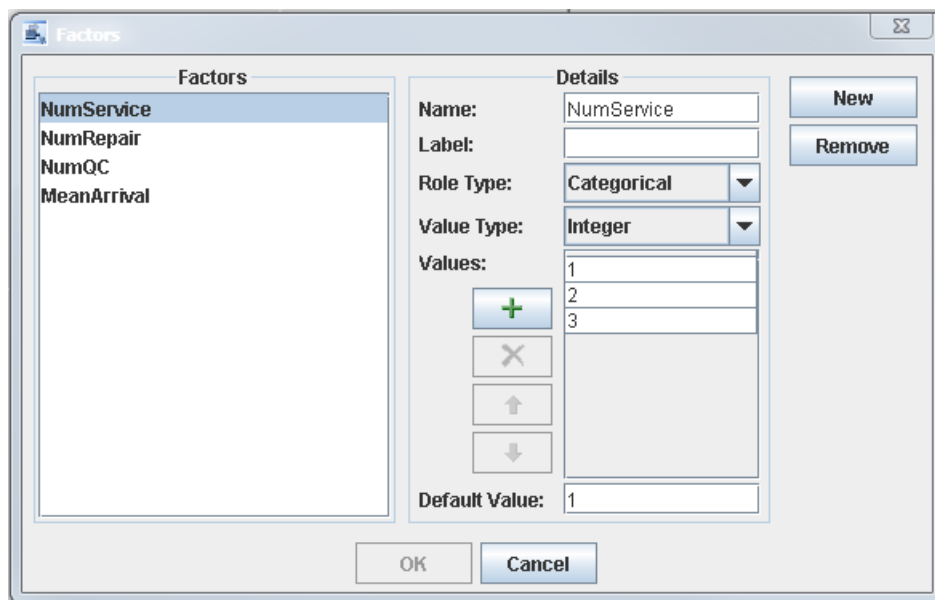
To define responses for the project, you right-click the project name in the Projects panel and select **Responses** to open the Responses dialog box, as shown in Figure C.2. You specify a unique name for the response in the **Name** field, and you use the optional **Label** field to specify a detailed description of the response. If you are not using JMP to generate your design, you can leave **Goal Type** as 'None' and **Values** as 'NaN'. If you are using JMP, then the **Goal Type** field tells JMP whether to maximize your response, minimize your response, or match a target or that you have no response goal.

JMP uses the **Goal Type** field to define a desirability function for each response, and the JMP Profiler and Contour Profiler use these desirability functions to find optimal factor settings. If you specify a goal type of 'None', then JMP does not construct a desirability function. For a goal type of 'Maximize' or 'Minimize', you can specify natural lower or upper bounds if necessary by using the **Values** field. For a goal type of 'Match Target', the default target value that JMP uses is assumed to be midway between the lower and upper limits, which you specify in the **Values** field in the Responses dialog box. For all goal types, if you do not specify a lower and upper limit in the **Values** field, then JMP uses the range of the simulated data for the response to define the limits for the desirability function. If you choose to analyze the simulated results from a design created using JMP, you can later change the goal type directly in JMP if necessary.

Figure C.2 Responses Dialog Box
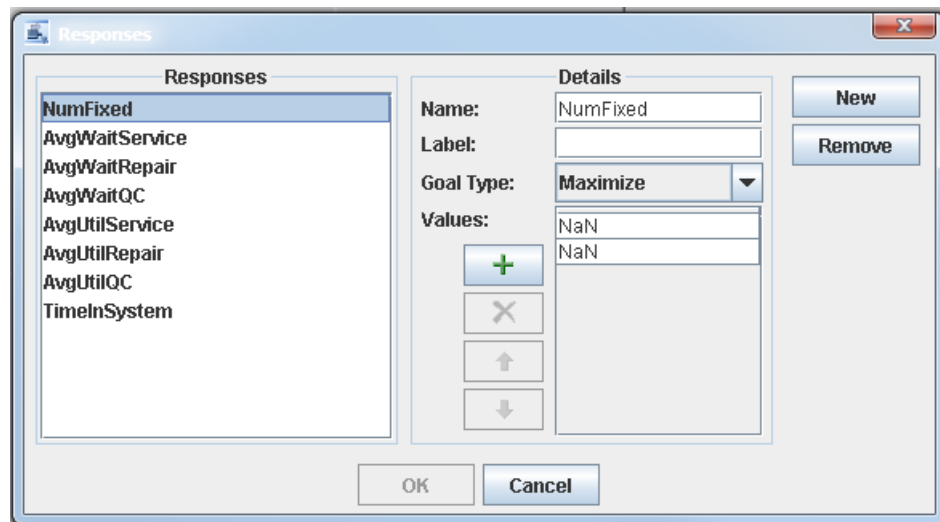
Creating Model Anchors

After you establish the repository of factors and responses for the project in Simulation Studio, you need to anchor or link each factor and response to a specific block in the model. To do this, right-click in the Model window and select **Anchors** to open the Anchors dialog box. Click **New** to open the New Anchor dialog box, where you can link a block in your model to your defined factor or response. For example, as shown in Figure C.3, the Capacity factor candidate for the Server block labeled Service Desk in the repair shop model is anchored to the factor NumService. You can also define new factors and responses for the project directly from the New Anchor window by using the **Add Factor** button. The responses are anchored to blocks in a similar fashion, as shown in Figure C.4.

Figure C.3 Factor Anchors
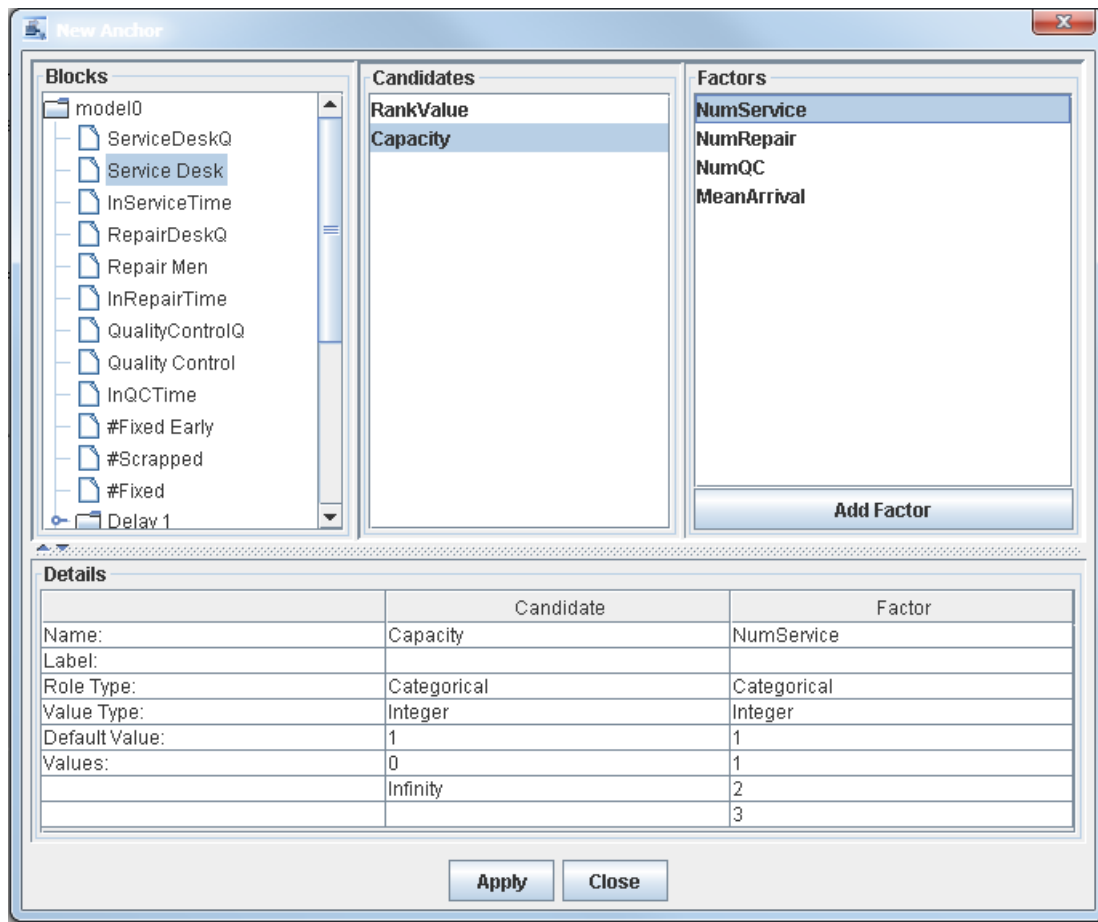
Figure C.4 Response Anchors
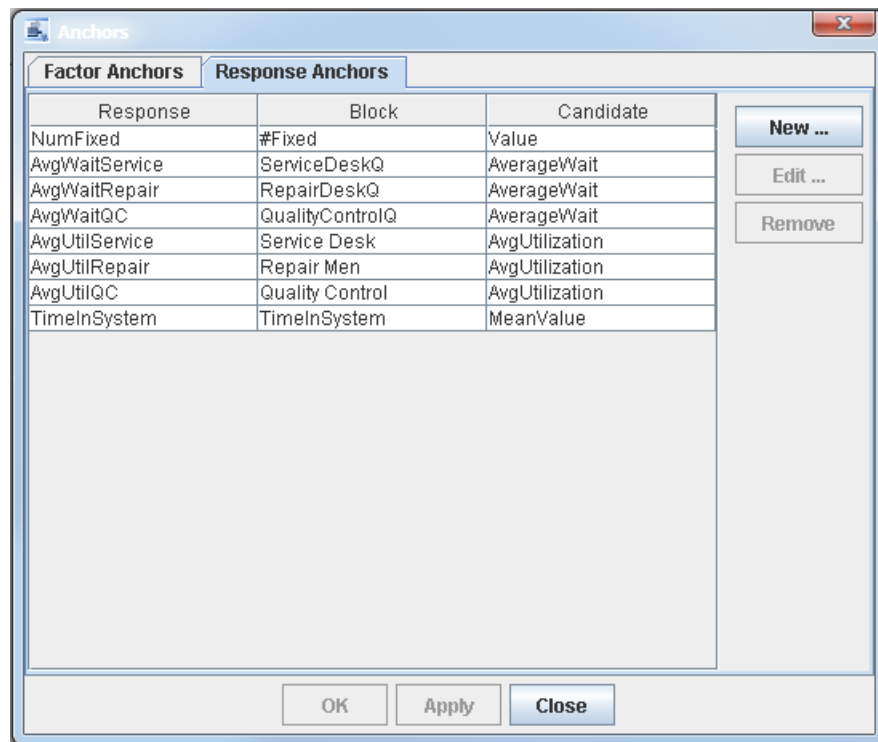
Setting Up the Experiment Window

After you anchor all factors and responses to blocks in the model, you need to include them in the experiment. To include factors, right-click in the Experiment window and select **Factor Inclusion**. In the Factor Inclusion dialog box, you can select the factors defined for your project that you want to include in the experiment. For now, the MeanArrival factor is not included in the experiment. (See Figure C.5.) Include responses in the experiment in a similar fashion by right-clicking in the Experiment window and selecting **Response Inclusion**. (See Figure C.6.)

The Experiment window for the repair shop model that includes factors and responses is shown in Figure C.7. The end time for all design points has been changed to 2,700 minutes, and the number of replications has been changed to 5 by right-clicking in the Experiment window and selecting **Properties**. The Properties dialog box for the Experiment window enables you to set default values for **StartTime**, **EndTime**, and **Replicates**. Each new design point has the default values for these parameters. Alternatively, to change the number of replications, the start time, or the end time for existing design points, you can right-click the appropriate column name and select **Set Value**.

Figure C.5 Including Factors in the Experiment
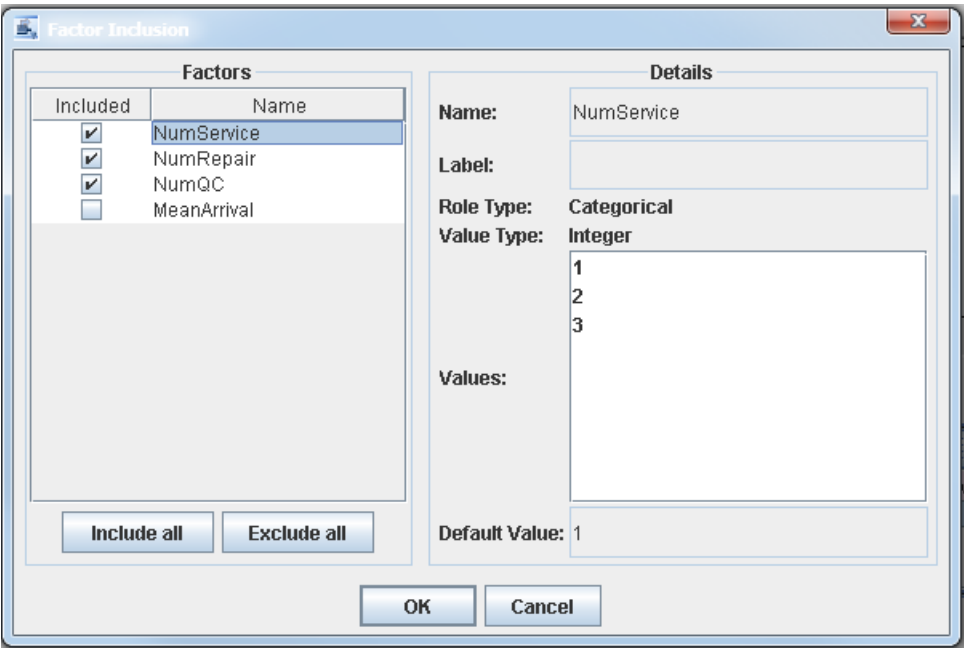
Figure C.6 Including Responses in the Experiment
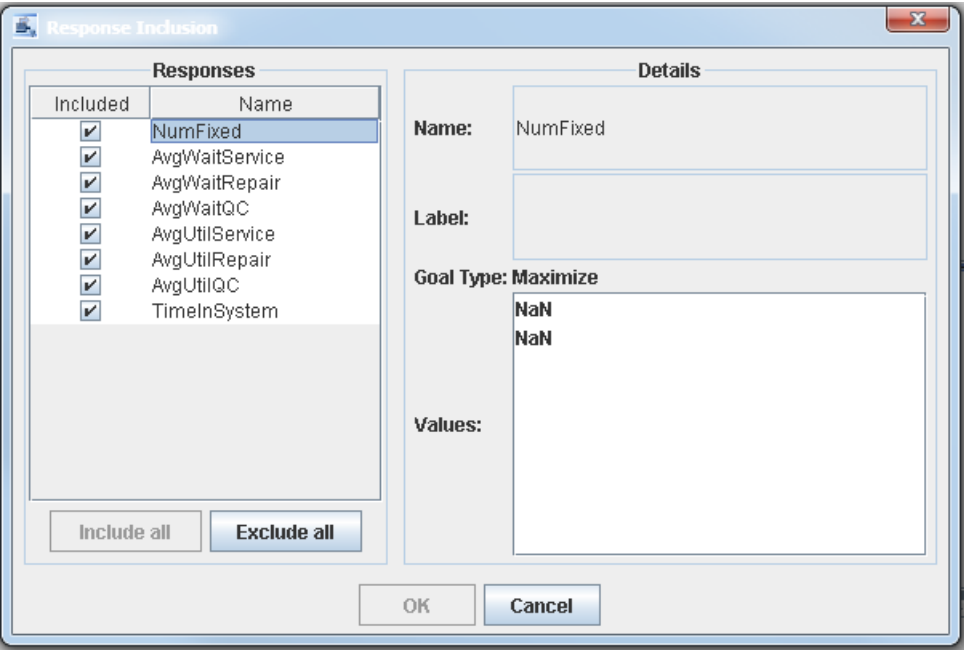
Figure C.7 Experiment Window with Factors and Responses Included

Experimental Design and Analysis Using JMP

After you define factors and responses, create anchors, and include the factors and responses in your Experiment window, you are ready to generate an experimental design. This section describes how you can use JMP to generate a design and to analyze the simulated results of that design. The instructions in this section use version 13 of JMP, but the steps are similar for other versions of JMP.

Using the JMP Custom Design Platform

Simulation Studio has options in the Experiment window that enable you to interact seamlessly with the JMP Custom Design platform. The default design that the Custom Design platform generates is automatically passed back to the Simulation Studio Experiment window. To generate a design by using the JMP Custom Design platform, first ensure that the JMP server has been launched. (See Chapter 3, "Introduction to SAS Simulation Studio.") Then, right-click in the Experiment window and select **Make Design**. For you to use the **Make Design** option, the Experiment window must include at least one factor and one response. Selecting **Make Design** invokes the JMP Custom Design platform, and a new JMP window called Simulation Studio DOE opens. Figure C.8 shows the JMP Simulation Studio DOE window, which displays the default design. Note that the results of a JMP custom design might differ from those shown here because of a random component in its algorithms. The **Responses** and **Factors** sections of the JMP Simulation Studio DOE window reflect the same information as the factor and response definitions in Simulation Studio. This default design is automatically passed back to the Experiment window in Simulation Studio, as shown in Figure C.9.

Figure C.8 JMP Simulation Studio DOE Window for the Repair Shop Model

Figure C.9 Simulation Studio Experiment Window with Default Design

The default design that the JMP Custom Design platform creates is a main-effects screening design (an optimal design). You can alter the JMP design by using the **Model** option in the JMP Simulation Studio DOE window. For example, suppose you want to generate a design that allows estimation of all possible interactions. To generate this design by using the JMP Custom Design platform, you first add second- and third-order interaction terms to the model by using the **Interactions** option in the **Model** section of the JMP Simulation Studio DOE window, as shown in Figure C.10. For **Number of Runs**, select **Minimum**, and click **Make Design**. Figure C.11 shows the resulting design in JMP. You can sort the order of the resulting design points in the JMP Simulation Studio DOE window by selecting the option **Sort Left to Right** for run order. After you click **Commit**, the design is passed to the Simulation Studio Experiment window, as shown in Figure C.12.

Note that the **Number of Replicate Runs** field in the JMP Simulation Studio DOE window does not have the same meaning as the **Replicates** column in the Simulation Studio Experiment window, and you should not use it to specify the number of replications to run for each design point. The **Number of Replicate Runs** field in the JMP Custom Design platform does not replicate the entire design but chooses the optimal design points to replicate.

Although it is possible to modify the model that JMP uses for the experimental design, you must create all factors and responses in Simulation Studio first, because they must be linked to specific model blocks. If you create new factors or responses in JMP, they will not be passed back to Simulation Studio.

Figure C.10 Adding Interaction Terms to the JMP Model
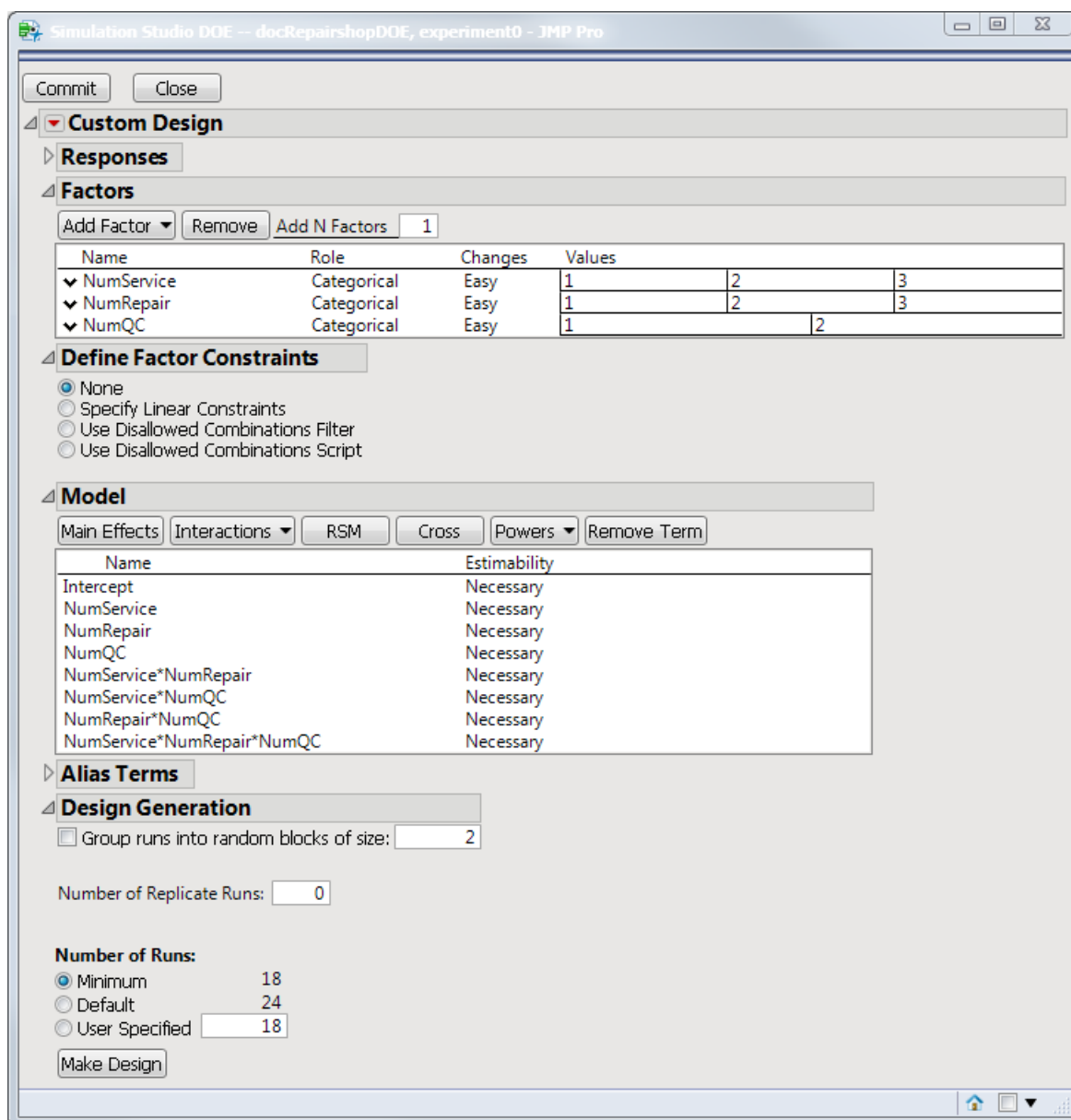
Figure C.11 Design Generated by JMP for Estimating Three-Way Interactions
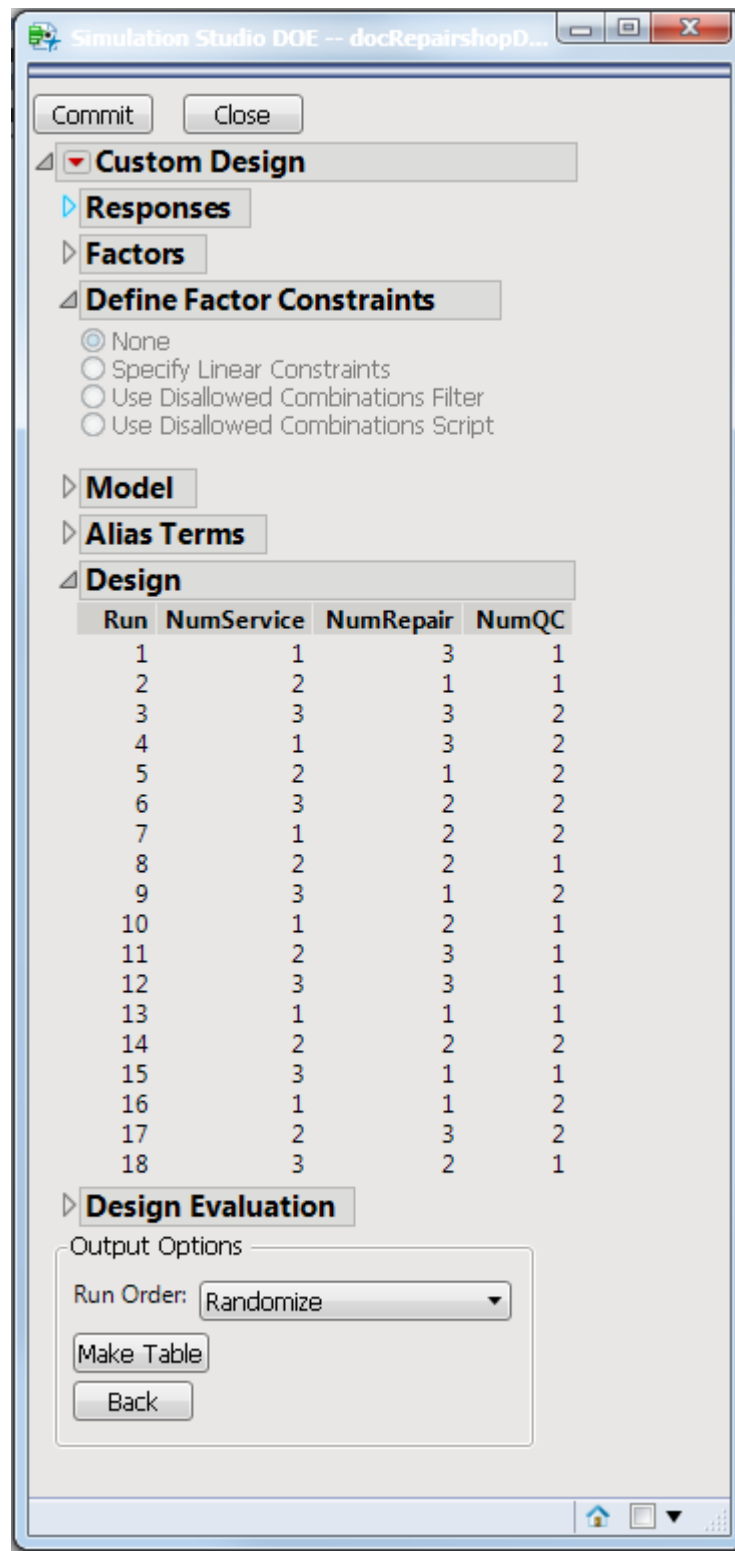
Figure C.12 Simulation Studio Experiment Window for Estimating Three-Way Interactions

Running the Experiment

After you generate a design in JMP and pass it back to Simulation Studio, the **Replicates** column in the Experiment window is set to 1 for all design points. You can set the number of replications easily for all design points by right-clicking the **Replicates** column name and selecting **Set Value**. For this example, the number of replications for each design point is set to 5. To run the experiment in Simulation Studio, highlight all the rows in the table by holding down the left mouse button on the first design point and dragging to highlight all the remaining design points. Now click the ▶ icon on the toolbar to run all replications of the design points.

Figure C.13 shows the Experiment window after you run all design points for the design for estimating three-way interactions. By default, the value that is reported for each response is the average of all five replications. To view the minimum or maximum value of all replications, right-click the column header for a response and select **Summary**. The dialog box that opens enables you to display the average, minimum, or maximum value for the selected response. You can view the individual response values for each of the five replications for each design point by clicking the blue arrow next to the number of replications within a design point. Figure C.14 shows the five replications for design point 1. To hide the replication results, click the blue arrow again.

Figure C.13 Experiment Window Showing Simulated Results for Estimating Three-Way Interactions

Figure C.14 Experiment Window Showing Results for All Five Replications of Design Point 1

Using Other JMP Design of Experiment Platforms

The **Make Design** option in the Simulation Studio Experiment window can be used only to automatically pass designs that are generated by the JMP Custom Design platform back to Simulation Studio. To use the other JMP design of experiment (DOE) platforms, you first create your factors and responses, anchor the factors and responses to the appropriate Simulation Studio blocks, and include them in an Experiment window, as described previously. Next, you right-click in the Experiment window and select **Save Design** to save the contents of the Experiment window as a JMP table. You need to have only one design point in the Experiment window before you save. This data set serves as a template for creating an experimental design by using JMP. It contains the factor and response information that Simulation Studio requires in order to run the experiment.

At this point, you are ready to use JMP to generate a design. From the JMP main window, you use the DOE menu to select a design platform. For example, you can use the Full Factorial Design platform for any number of categorical or continuous factors, or you can use the Response Surface design platform if your goal is prediction rather than factor screening. After you select a JMP design platform, you must manually enter all response and factor definitions to match your factor and response definitions in Simulation Studio. After you create the factors and responses in JMP, you can save them to a JMP table and import the table into most of the JMP DOE platforms for reuse. Figure C.15 shows the JMP Full Factorial Design window. For more information about design of experiments, see the JMP documentation.

Figure C.15 JMP Full Factorial Design Window
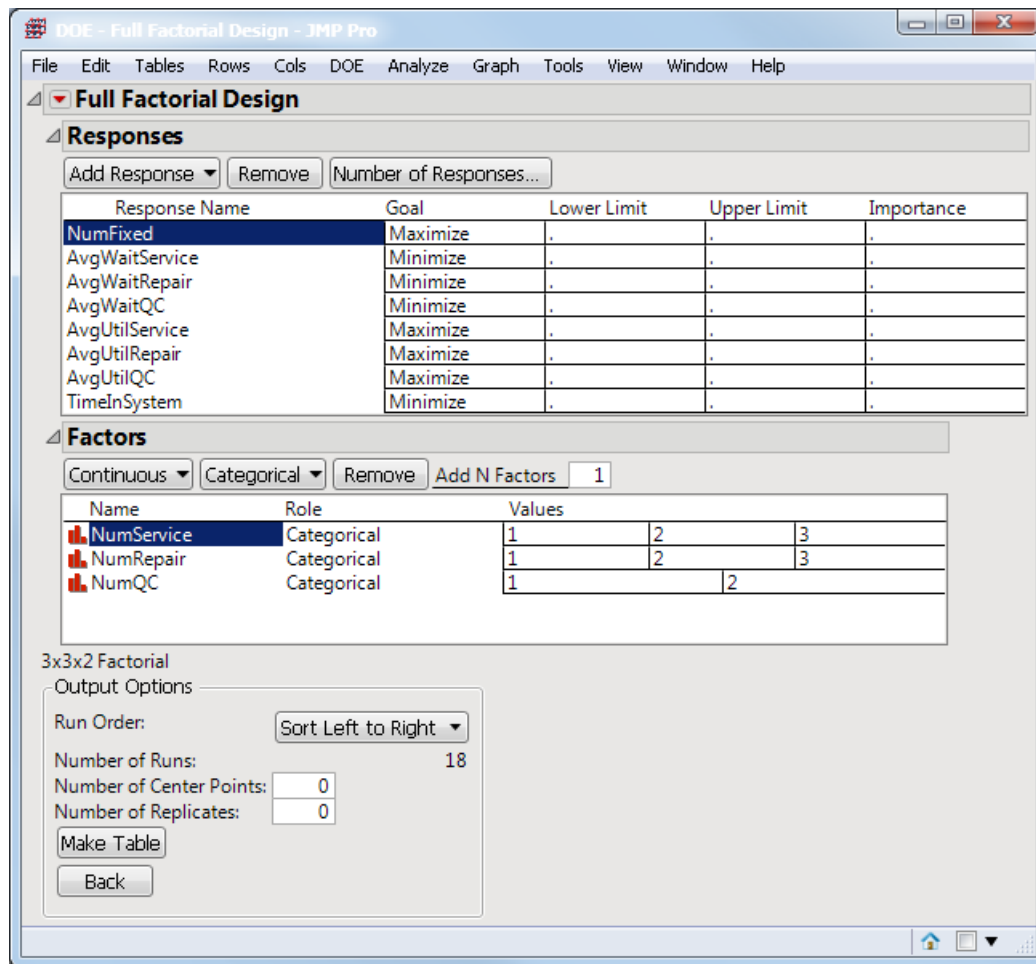
After you generate a design in any of the JMP DOE platforms, use the **Make Table** option in the JMP design window to generate a JMP table that contains the design points. Note that only the JMP Custom Design platform has the **Commit** button, which enables you to pass a design automatically back to Simulation Studio.

To load a design from other JMP DOE platforms into Simulation Studio, you first save the JMP table that contains the design points to the same location as the data table that you created from the Simulation Studio Experiment window (overwriting the contents). Before saving, you must check that the data type of each factor in JMP matches the data type of the corresponding factor in Simulation Studio. In particular, if you create a categorical factor in JMP, it might have a character type; in your Simulation Studio model, that factor might have a numeric type. If that is the case, you can right-click in the factor column in the JMP data table, select **Column Info**, and change the **Data Type** value.

After you save your design from JMP to the experiment data set, you import the experiment back into the Simulation Studio Experiment window by right-clicking in the Experiment window and selecting **Load Design**. It is important to save the JMP design to the same experiment file that you created from Simulation Studio (using the **Save Design** option), because Simulation Studio will be able to identify the appropriate use of each column (factor, response, replicates, point name, start time, or end time) when it is loaded into the Experiment window.
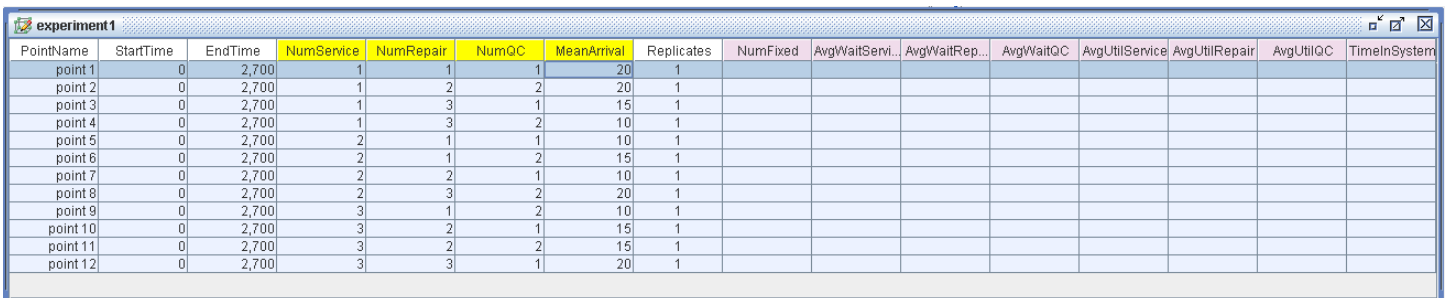
Analyzing Simulated Results Using JMP

You can use JMP to conduct statistical analyses of the simulated results after you run all replications of a design. For example, you can estimate a statistical model (or metamodel) that you can use for prediction purposes.

As mentioned previously, the goal of the modified repair shop model is to determine the optimal number of workers to put at the service desk, repair station, and quality control station to minimize waiting times and the total time in system and to maximize utilization. In this section, the factor MeanArrival (for the mean interarrival time in minutes between parts) is included in the experiment. The purpose of including this factor is to see the effect of different part arrival rates on the optimal allocation of workers at each station.

Before estimating a statistical model, you generate an experimental design in JMP. Figure C.16 shows the default design that the JMP Custom Design platform returns for this example after you select **Make Design** in the Experiment window. As mentioned previously, the JMP Custom Design platform incorporates a random component when it generates a design, so your design might differ slightly from the one shown in Figure C.16. To generate the design exactly as shown in Figure C.16, click **Back** in the JMP Simulation Studio DOE window that opens after you select the **Make Design** option in the Simulation Studio Experiment window. Then, from the Custom Design menu in the JMP Simulation Studio DOE window, select the option **Set Random Seed** and enter the value 876543. Click **Make Design** in the JMP Simulation Studio DOE window, change the **Run Order** to **Sort Left to Right**, and click **Commit**. The resulting design will be identical to the one shown in Figure C.16.

Figure C.16 Experiment Window Showing the JMP Default Design for the Modified Repair Shop

To generate a more interesting statistical model, you can modify the default design in Figure C.16. In the JMP Simulation Studio DOE window, click **Back** and modify the model by adding second-order interactions, and then select the default number of runs (30), as shown in Figure C.17. After you click **Make Design**, you can pass the new design back to the Simulation Studio Experiment window by clicking **Commit** in the resulting design window, as shown in Figure C.18. To generate the design exactly as shown in Figure C.18, set the random seed in JMP to 876566 by using the **Set Random Seed** option on the Custom Design menu before clicking **Make Design**, as described in the previous paragraph.

Figure C.17 JMP Simulation Studio DOE Window with Second-Order Interactions Added for the Modified Repair Shop

Figure C.18 Experiment Window Showing the New JMP Design for the Modified Repair Shop

PointName	StartTime	EndTime	NumService	NumRepair	NumQC	MeanArrival	Replicates	NumFixed	AvgWaitService	AvgWaitRepair	AvgWaitQC	AvgUtilService	AvgUtilRepair	AvgUtilQC	TimeInSystem
point 1	0	2,700	1	1	1	15	▶ 5								
point 2	0	2,700	1	1	2	10	▶ 5								
point 3	0	2,700	1	1	2	20	▶ 5								
point 4	0	2,700	1	2	1	20	▶ 5								
point 5	0	2,700	1	2	2	10	▶ 5								
point 6	0	2,700	1	2	2	15	▶ 5								
point 7	0	2,700	1	3	1	10	▶ 5								
point 8	0	2,700	1	3	1	15	▶ 5								
point 9	0	2,700	1	3	2	15	▶ 5								
point 10	0	2,700	1	3	2	20	▶ 5								
point 11	0	2,700	2	1	1	20	▶ 5								
point 12	0	2,700	2	1	2	10	▶ 5								
point 13	0	2,700	2	1	2	15	▶ 5								
point 14	0	2,700	2	2	1	10	▶ 5								
point 15	0	2,700	2	2	1	15	▶ 5								
point 16	0	2,700	2	2	2	15	▶ 5								
point 17	0	2,700	2	2	2	20	▶ 5								
point 18	0	2,700	2	3	1	15	▶ 5								
point 19	0	2,700	2	3	2	10	▶ 5								
point 20	0	2,700	2	3	2	20	▶ 5								
point 21	0	2,700	3	1	1	10	▶ 5								
point 22	0	2,700	3	1	1	15	▶ 5								
point 23	0	2,700	3	1	2	15	▶ 5								
point 24	0	2,700	3	1	2	20	▶ 5								
point 25	0	2,700	3	2	1	15	▶ 5								
point 26	0	2,700	3	2	2	10	▶ 5								
point 27	0	2,700	3	2	2	20	▶ 5								
point 28	0	2,700	3	3	1	20	▶ 5								
point 29	0	2,700	3	3	2	10	▶ 5								
point 30	0	2,700	3	3	2	15	▶ 5								

After you change the number of replicates to 5 and run all 30 design points, click the Reset icon ■ on the main toolbar, right-click in the Experiment window, and select **Analyze Results**. This opens the JMP Simulation Studio DOE Analyzer window, as shown in Figure C.19. Note that the JMP table for this experiment has 150 rows, one for each of the five replicates for each design point. You can now fit a model to your simulated results by clicking the green arrow next to **Model** in the Simulation Studio DOE Analyzer window to open the JMP Report: Fit Model window. In this window, change the **Emphasis** box to **Minimal Report**, as shown in Figure C.20, and click **Run**.

As an example of what you can do in JMP at this point, select **Profilers** and then **Profiler** from the Least Squares Fit menu in the JMP Simulation Studio DOE Analyzer window. (See Figure C.21.) The JMP Prediction Profiler is a series of interactive plots that show how the factor levels drive the response values. You can change one factor at a time and see the effect on the predicted response values, and you can assess the importance of a factor to some extent by the steepness of the prediction trace (the black line). For each response, a goal type was specified during the response definition process, and the resulting desirability function for each response is shown in the last column of the Prediction Profiler. The overall desirability (shown in the bottom row of the Prediction Profiler) is the geometric mean of the desirability for each response.

In the Prediction Profiler, lock the value of the factor MeanArrival at 20 by holding down the Alt key, clicking in a plot in the MeanArrival column, and selecting **Lock Factor Setting**. From the Prediction Profiler menu, select **Optimization and Desirability▶Maximize Desirability**. The optimal factor settings for a mean interarrival time of 20 minutes (or three parts per hour) is one worker at the service desk, one worker at the repair station, and one worker at the quality control station. If the mean interarrival time decreases to 10 minutes (that is, the arrival rate of parts doubles to six parts per hour), then both the time in system and the average wait at the repair station increase dramatically. You can see how the response predictions change as you slide the vertical red line in the MeanArrival column from 20 to 10. It appears that the optimal solution does not hold up if the arrival rate of parts increases from three to six parts per hour.

To determine the optimal factor levels for an arrival rate of six parts per hour (MeanArrival=10), keep the vertical red line for MeanArrival at 10 and select **Optimization and Desirability▸Maximize Desirability** again from the Prediction Profiler menu. For an arrival rate of six parts per hour, the optimal factor levels are one worker at the service desk, two workers at the repair station, and two workers at the quality control station. Figure C.21 shows the optimal factor settings for an arrival rate of six parts per hour (MeanArrival=10).

Figure C.19 Simulation Studio DOE Analyzer Window for the Modified Repair Shop Example

Figure C.20 JMP Report: Fit Model Window for the Modified Repair Shop Example
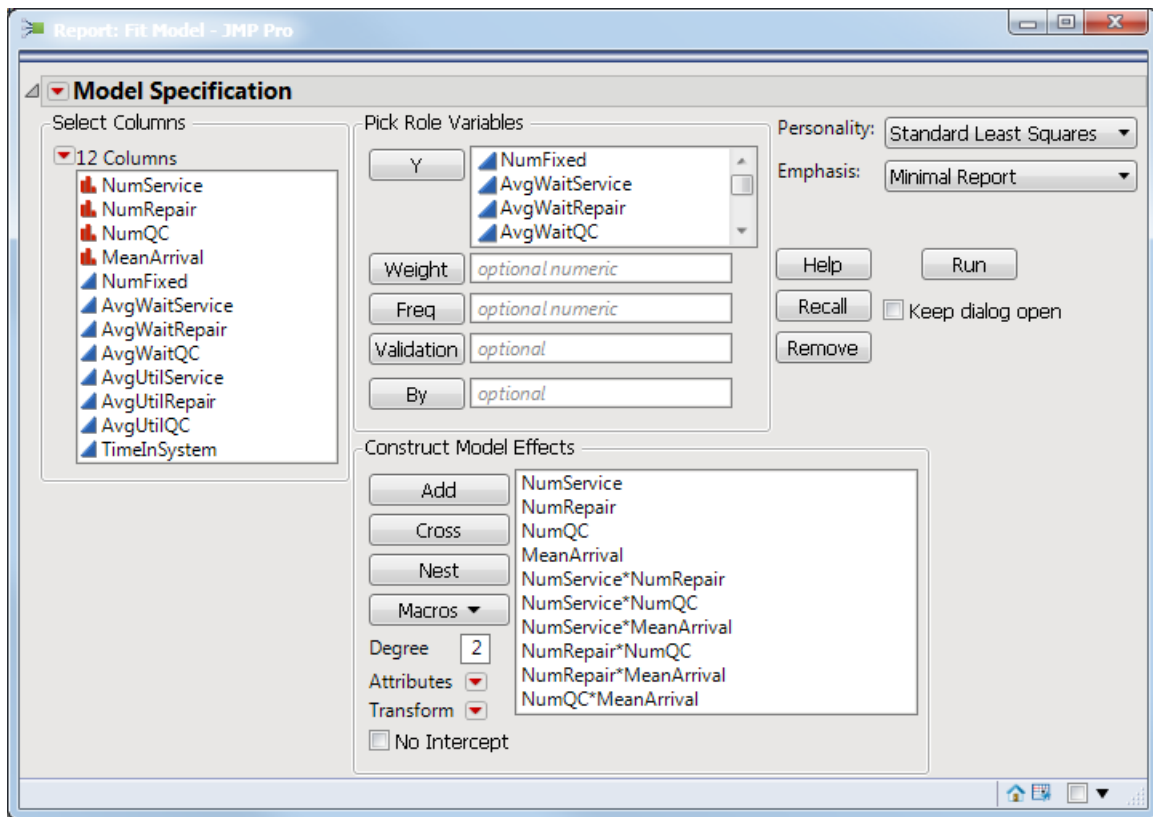
Figure C.21 JMP Prediction Profiler for the Modified Repair Shop Example
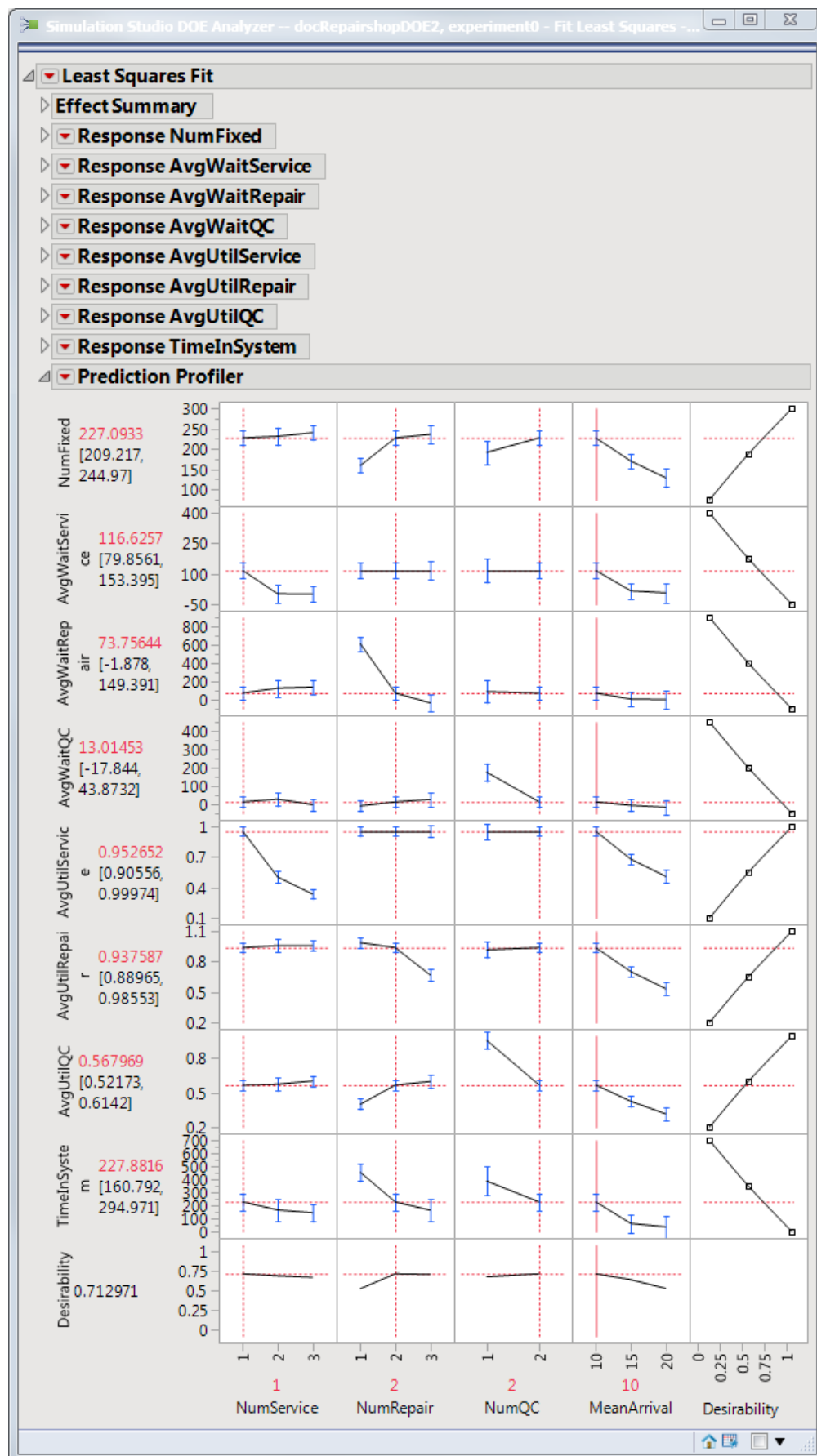
Augmenting a Design in JMP

Experimental design is often an iterative process. For example, screening designs are widely used with the goal of identifying those factors that have large effects on the response. The factors that are identified as important are then studied further in subsequent experiments. It is possible to combine the runs of a screening design with additional design points to estimate interactions of interest.

The Augment Design platform in JMP constructs additional runs for an existing design in a way that optimizes the augmentation process. From the JMP Simulation Studio DOE Analyzer window, you can choose to augment the current design. Using the modified repair shop example, click the green arrow next to the option **Augment This Design** (in the upper left corner of Figure C.19) to open the JMP Augment Design window, as shown in Figure C.22. For more information about using the JMP augment design feature, see the JMP documentation.

If you make any changes in the Augment Design window, you can click **Commit** to pass the augmented design back to Simulation Studio. Figure C.23 shows the Experiment window in Simulation Studio after you select **Augment** in the JMP Augment Design window for the default design in Figure C.16. Eight additional design points were added to the original design.

You can run the new design points in Simulation Studio by highlighting all rows in the Experiment window and then clicking **Augment** on the **Run** menu or the Augment icon on the toolbar. Only new points or current points that have additional replications are run. The results of all previously run replications are preserved. Furthermore, the simulated results from using augment run will be identical to the results that you would get if you ran the entire experiment again (old and new design points and replicates together). Notice that the design points in the Experiment window might now be in a different order than they were before you augmented the design. For example, design point 1 in Figure C.16 is now design point 7 in Figure C.23, but the results are the same.

Figure C.22 JMP Augment Design Window
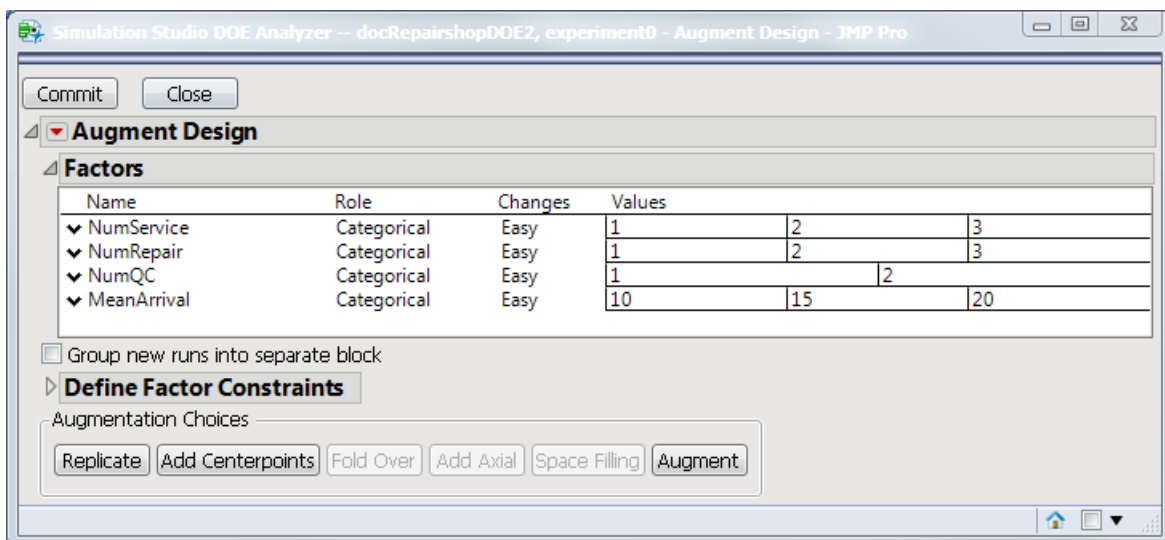
Figure C.23 Experiment Window after Using the JMP Augment Design Feature

PointName	StartTime	EndTime	NumService	NumRepair	NumQC	MeanArrival	Replicates	NumFixed	AvgWaitService	AvgWaitRepair	AvgWaitQC	AvgUtilService	AvgUtilRepair	AvgUtilQC	TimeInSystem
point 1	0	2,700	3	2	2	20	▶ 5	126	0.053820485...	5.965169167...	0.4553140...	0.170107275...	0.535126998...	0.31552...	51.693038695...
point 2	0	2,700	2	1	2	15	▶ 5	133.6	0.837410260...	399.5696828...	0.3829517...	0.341983689...	0.982473034...	0.33528...	288.71170459...
point 3	0	2,700	1	2	2	10	▶ 5	230.6	116.6256788...	70.59126047...	1.2022229...	0.952652193...	0.937021638...	0.57788...	213.96127632...
point 4	0	2,700	2	3	2	20	▶ 5	126	0.441428146...	0.677426507...	0.6079786...	0.255160913...	0.356751332...	0.31552...	48.087720120...
point 5	0	2,700	2	2	1	15	▶ 5	165.2	0.837410260...	13.63194443...	53.407844...	0.341983689...	0.698835716...	0.82909...	112.65832397...
point 6	0	2,700	3	3	2	15	▶ 5	169.2	0.114440806...	1.944917278...	1.3003907...	0.228188356...	0.471422514...	0.42377...	49.020942549...
point 7	0	2,700	1	1	1	20	▶ 5	114	6.361406433...	190.4423318...	4.5633479...	0.509916082...	0.937099357...	0.57057...	194.20750679...
point 8	0	2,700	1	3	1	15	▶ 5	164.8	16.76477298...	1.100613075...	47.338658...	0.679540191...	0.467979654...	0.82833...	113.50417019...
point 9	0	2,700	3	1	1	10	▶ 5	161.2	0.298386351...	677.1466983...	28.860643...	0.336100912...	0.991372522...	0.81389...	408.29446908...
point 10	0	2,700	2	3	1	10	▶ 5	194	2.041833446...	5.728547716...	285.22899...	0.504061044...	0.669297288...	0.97758...	332.11222857...
point 11	0	2,700	3	2	1	20	▶ 5	125	0.053820485...	5.567275653...	12.875238...	0.170107275...	0.534875698...	0.62703...	64.549017710...
point 12	0	2,700	1	1	2	10	▶ 5	158.4	116.6256788...	606.2070628...	0.3809645...	0.952652193...	0.990204728...	0.39928...	459.80807860...
point 13	0	2,700	3	1	2	15	1								
point 14	0	2,700	2	2	2	10	1								
point 15	0	2,700	1	2	1	15	1								
point 16	0	2,700	1	3	2	20	1								
point 17	0	2,700	1	3	1	20	1								
point 18	0	2,700	2	1	1	20	1								
point 19	0	2,700	3	3	1	10	1								
point 20	0	2,700	1	2	2	15	1								

Experimental Design Using SAS Software

You can also generate an experimental design for your simulation project by using SAS. As described previously, you define factors and responses, create anchors, and include factors and responses in your Simulation Studio Experiment window before you generate a design. The process of generating a design by using SAS is similar to the process of generating a design by using alternative JMP DOE platforms, as described in the section "Using Other JMP Design of Experiment Platforms" on page 263. You first right-click in the Experiment window and select **Save Design** to save the contents of your Experiment window (with factors included) as a SAS data set. You need to have only one design point in the Experiment window before you save. SAS has a variety of procedures that you can use to generate a design. The FACTEX procedure (in SAS/QC) constructs orthogonal factorial experimental designs, which can be either full or fractional factorial designs. The OPTEX procedure (in SAS/QC) searches for optimal experimental designs. The PLAN procedure (in SAS/STAT) constructs designs that include full factorial designs, certain balanced and partially balanced incomplete block designs, generalized cyclic incomplete block designs, and Latin square designs.

As an example, Figure C.24 shows SAS code that you can use to generate a full factorial design by using the PLAN procedure for the repair shop example. The output file `Project.Exp` is the data set that was saved from the Simulation Studio Experiment window (where `Project` is the name of the SAS library that points to the location of the saved experiment file `Exp`). After you run this code, the data set `Exp.sas7bdat` is overwritten by the full factorial design that the PLAN procedure generates. After you save your design from SAS, you import the design into the Simulation Studio Experiment window by right-clicking in the Experiment window and selecting **Load Design**. Figure C.25 shows the Experiment window after you load the full factorial design generated by SAS.

Figure C.24 Sample SAS Code for Generating a Full Factorial Experimental Design

```
proc plan seed=87654;
    factors NumService=3  NumRepair=3 NumQC=3
        / noprint;
    output out=Project.Exp
        NumService nvals=(1,2,3)
        NumRepair nvals=(1,2,3)
        NumQC nvals=(1,2,3);
    run;
quit;
```

Figure C.25 Experiment Window after Loading the Full Factorial Design Generated by SAS

PointName	StartTime	EndTime	NumService	NumRepair	NumQC	Replicates
point 1	0	2,700	3	1	1	5
point 2	0	2,700	3	1	2	5
point 3	0	2,700	3	1	3	5
point 4	0	2,700	3	2	1	5
point 5	0	2,700	3	2	3	5
point 6	0	2,700	3	2	2	5
point 7	0	2,700	3	3	2	5
point 8	0	2,700	3	3	1	5
point 9	0	2,700	3	3	3	5
point 10	0	2,700	1	3	1	5
point 11	0	2,700	1	3	3	5
point 12	0	2,700	1	3	2	5
point 13	0	2,700	1	2	1	5
point 14	0	2,700	1	2	3	5
point 15	0	2,700	1	2	2	5
point 16	0	2,700	1	1	2	5
point 17	0	2,700	1	1	3	5
point 18	0	2,700	1	1	1	5
point 19	0	2,700	2	3	2	5
point 20	0	2,700	2	3	3	5
point 21	0	2,700	2	3	1	5
point 22	0	2,700	2	1	2	5
point 23	0	2,700	2	1	1	5
point 24	0	2,700	2	1	3	5
point 25	0	2,700	2	2	2	5
point 26	0	2,700	2	2	1	5
point 27	0	2,700	2	2	3	5

Appendix D
Input Analysis

Contents

Overview of Input Analysis

When you build a simulation model of a system, part of the process is likely to include analyzing data in various formats so that they can be used as inputs to drive the simulation model. This data might be in the form of raw data sets that must be read directly by the simulation model or from which a statistical distribution must be estimated and then sampled in the model. In any case, extreme care must be taken to determine appropriate inputs for a simulation model because the accuracy of a model's output data is directly dependent on how accurately you estimate the inputs.

Use JMP Software for Automated Input Analysis

When data are available and you want to estimate a statistical distribution from them, you can use the JMP distribution-fitting tool. To access the JMP distribution-fitting tool:

1. Make sure the JMP server has been launched. (See the section "Launching Local SAS and JMP Servers" on page 21 in Chapter 3, "Introduction to SAS Simulation Studio," for details.)

2. In a Simulation Studio model, open the Block Properties dialog box for a Numeric Source block. (See Figure Figure D.1). Select the **Fitted** option on the **Numeric Data Source** tab.

3. In the **Data Name** field of the Input Data section, specify the pathname of the data set that you want to fit a theoretical distribution to. The data could be a specific column from a SAS data set or JMP table or it could be contained in a text file.

4. In the **Column Name** field, specify the name of the column in the data set that you want to fit. If the data are contained in a text file, the column name should be the first entry.

5. Click **Fit Distribution**.

Figure D.1 Numeric Source Block Fitted Option
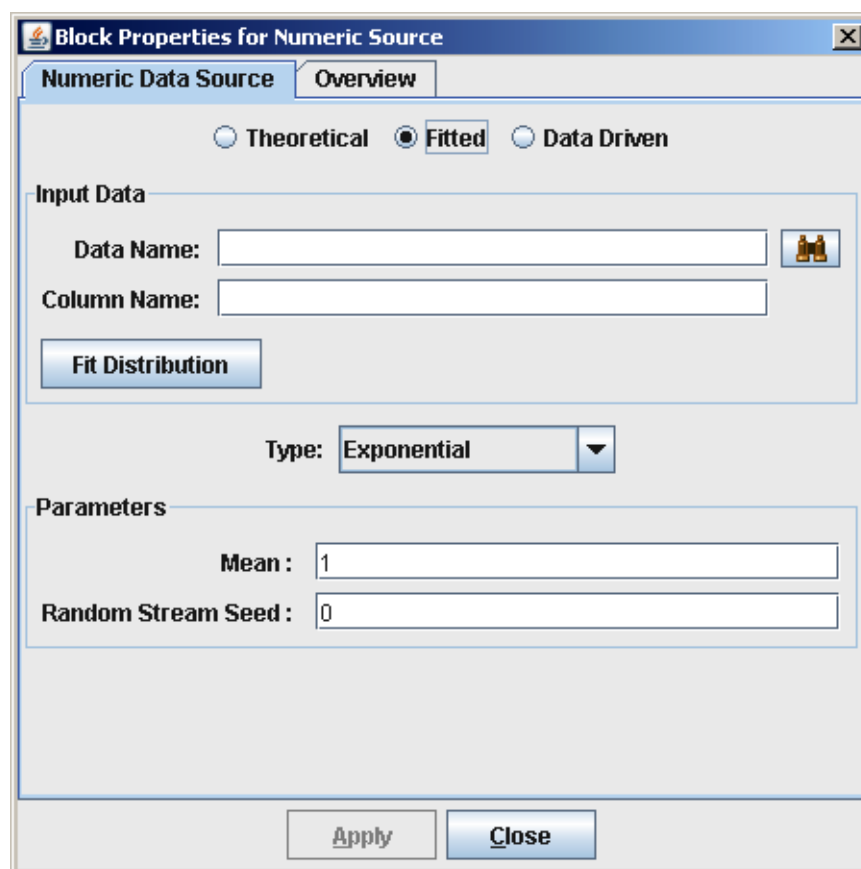
6. Simulation Studio displays a message to alert you that the JMP server is waiting for input. (See Figure D.2).

Figure D.2 JMP Request Message

7. Click **OK** in this message box.

8. Change focus to your JMP window to view the distribution-fitting results in the **Compare Distributions** section of the Distribution for Simulation Studio-JMP window. (See Figure D.3). See the JMP documentation for specific information about distribution fitting (specifically, the JMP **Fit All** option is used by Simulation Studio to generate the distribution fitting results).

Figure D.3 Distribution for Simulation Studio - JMP Window
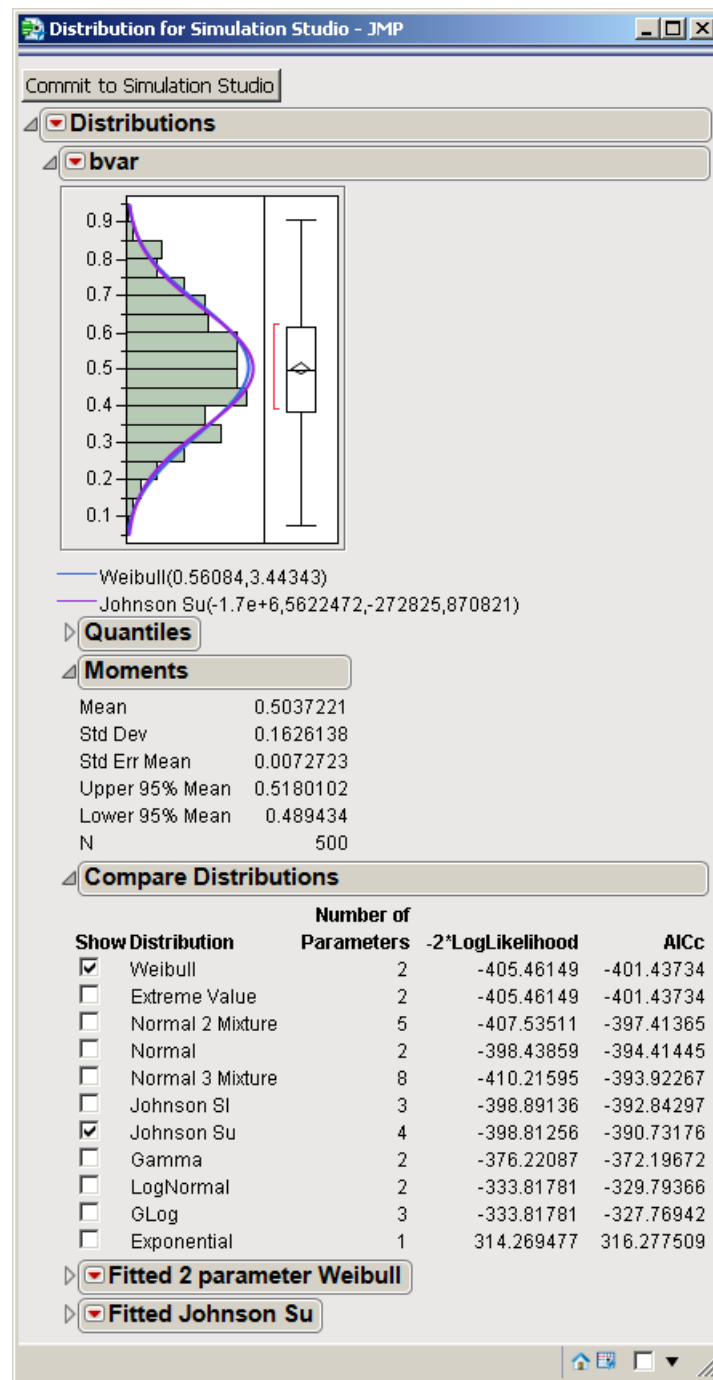
9. In the **Compare Distributions** section, select the check box beside each distribution that you are interested in viewing. Figure D.3 shows the Distribution for Simulation Studio - JMP window for a data column labeled *bvar*. The first distribution in the list (*Weibull*) is the top-ranked fit. The distribution *Johnson Su* is also checked. For each selected distribution, the corresponding density curve is shown overlaid on a histogram of the data. The fitted parameters are also provided for each selected distribution.

10. After viewing the fits, click **Commit to Simulation Studio** at the top of the window. A dialog box appears with a list of the selected distributions.

11. Click the distribution that you want to pass back to Simulation Studio to use in the model. Figure D.4 shows the Distribution Selection dialog box with the *2 parameter Weibull* selected.

12. Click **OK** in this dialog box. The **Type** and **Parameters** fields of the Block Properties for Numeric Source dialog box are populated with the appropriate information from the JMP distribution-fitting tool.

Figure D.4 Distribution Selection Dialog Box
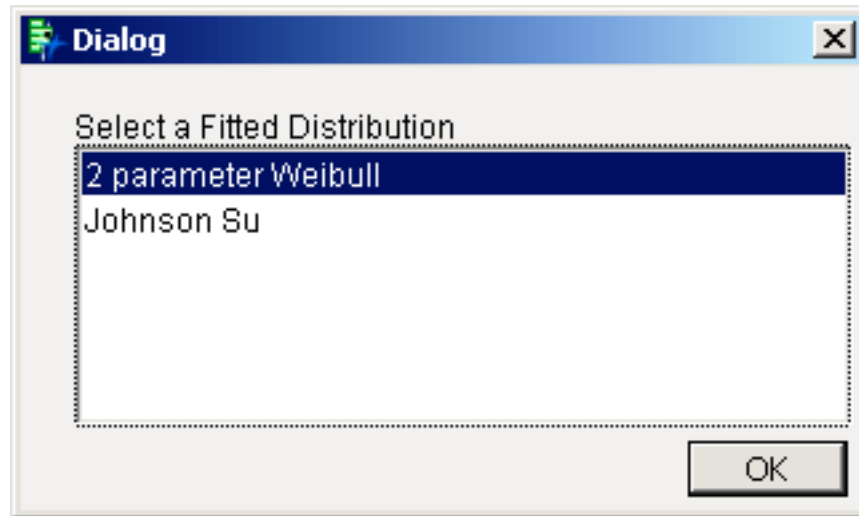
If you select a distribution in JMP that is not currently supported in Simulation Studio, then you receive an unsupported distribution error message.

The JMP **Fit All** option does not include bounded distributions (such as beta or Johnson bounded) in its automated fitting algortihm. However, these bounded distributions can be fit using the **Continuous Fit** option from the JMP menu that is associated with the current data set in the Distribution for Simulation Studio – JMP window, as shown in Figure D.5. The parameters for fitted distributions that are obtained by using the **Continuous Fit** option can still be automatically passed back to Simulation Studio by clicking the **Commit to Simulation Studio** button.

Figure D.5 Continuous Fit Option
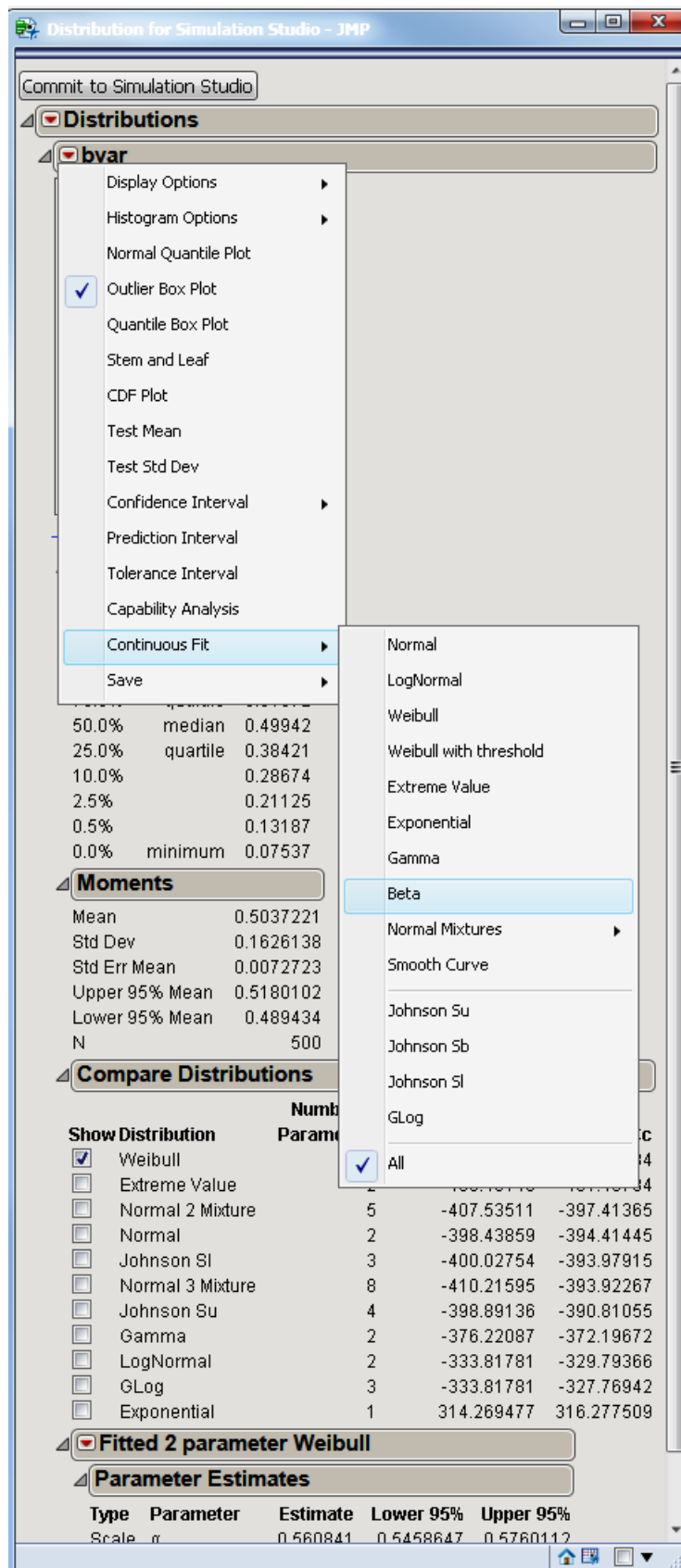
Use JMP Software for General Input Analysis

It is also possible to access the JMP distribution-fitting tool outside of a Numeric Source block. Unlike the method described in the previous section, this method does not automatically pass fitted distribution parameters back to Simulation Studio. To access the JMP distribution-fitting tool:

1. Make sure the JMP server has been launched. (See the section "Launching Local SAS and JMP Servers" on page 21 in Chapter 3, "Introduction to SAS Simulation Studio," for details.)

2. From the Simulation Studio menu, select **Analyze ▶ Fit Distribution**, as shown in Figure D.6.

Figure D.6 Input Analysis Menu Entry
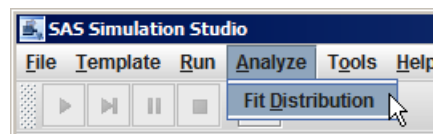
3. Simulation Studio displays a message box to alert you that the JMP server is waiting for input from you. See Figure D.2. Click **OK** in this message box.

4. Change focus to your JMP window, which shows the Open Data File dialog box.

5. In this dialog box, select the location of the data file that you want to analyze and click **Open**.

6. The JMP distribution-fitting tool opens. Select appropriate variables from your data set. Figure D.7 shows an example of this step for a data set that contains one variable labeled *bvar*.

Figure D.7 JMP Distribution Fitting Dialog
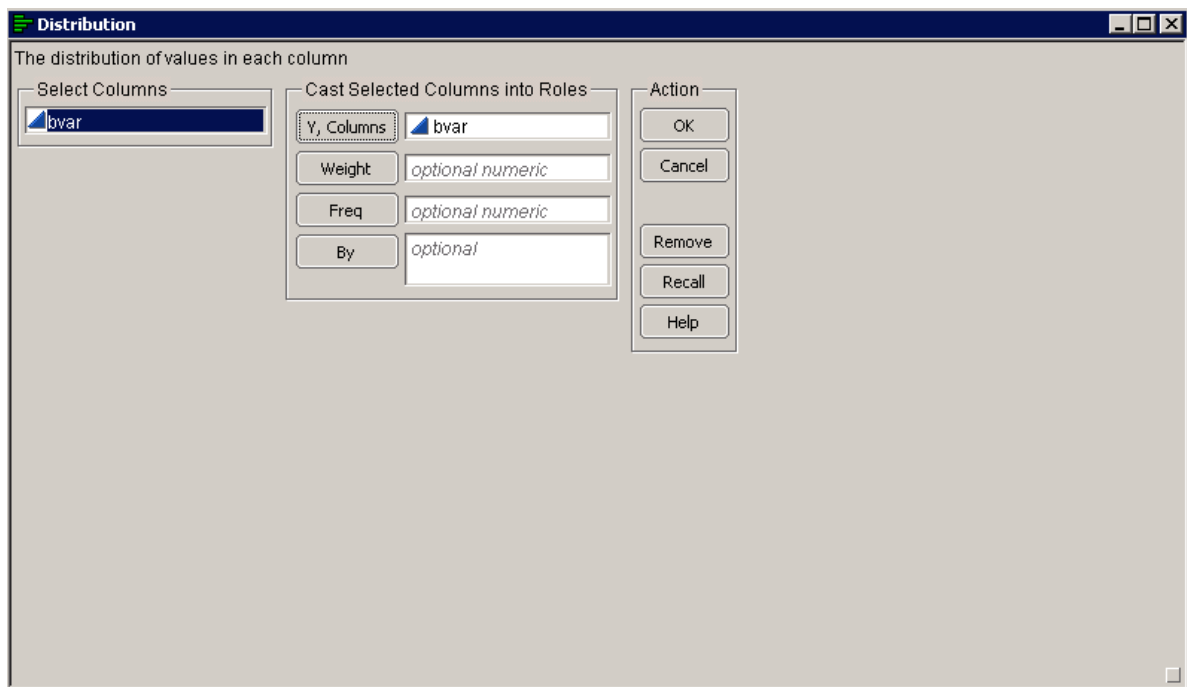
7. Use the JMP distribution-fitting tool to estimate an appropriate statistical distribution for the data. See the JMP documentation for specific information about fitting distributions to data.

8. From this point, the distribution fitting information from JMP can be used in a Simulation Studio model. However, you must manually enter the fit information in Simulation Studio: In the Block Properties for Numeric Source dialog box, select the entry for the distribution you want to use from the **Type** field. Enter the parameters for that distribution into the appropriate fields in the dialog box.

Note: The JMP definition for some distributions might be different from the Simulation Studio definition, so be careful when you map distribution parameters from a JMP distribution to a Simulation Studio distribution. It is also possible that JMP software provides support for distributions that Simulation Studio does not, and vice versa. The automated method for input analysis outlined in the previous section handles any required parameter mappings for you and also does not allow you to select a distribution in JMP that is not supported in Simulation Studio.

Appendix E
Examples of Simulation Studio Models

Contents

Overview of Simulation Studio Model Examples

This chapter provides examples of several modeling structures and illustrates uses and combinations of various blocks. The examples are meant only to show how you can use Simulation Studio to model various applications. They are not meant to show how you would analyze or evaluate these models or identify optimal parameterizations. The actual model construction process is not included in these example descriptions. The projects in this chapter can be found in the folder `\projects\examples` in the directory where Simulation Studio is installed. The default install location on the Windows operating system is `\Program Files\SASHome\SASSimulationStudio\<release_number>`, and on the Linux operating system it is `/usr/local/SASHome/SASSimulationStudio/<release_number>`.

A Simple M/M/1 Queueing Model

Chapter 2, "Overview of SAS Simulation Studio," first introduced this example, and it is discussed here because of its wide applicability. You can find the project files for this example in the folder `\projects\ examples\docMM1` in your Simulation Studio installation directory.

An M/M/1 queueing model can be used to represent many different real-life situations such as customers checking out at a supermarket, customers at a bank, and so on. This model illustrates the basic concepts involved in building models in Simulation Studio, and it is a good starting point for constructing more sophisticated models. In some ways this example is analogous to the "hello, world" introductory example used to illustrate many programming languages.

Figure E.1 An M/M/1 Queueing Model
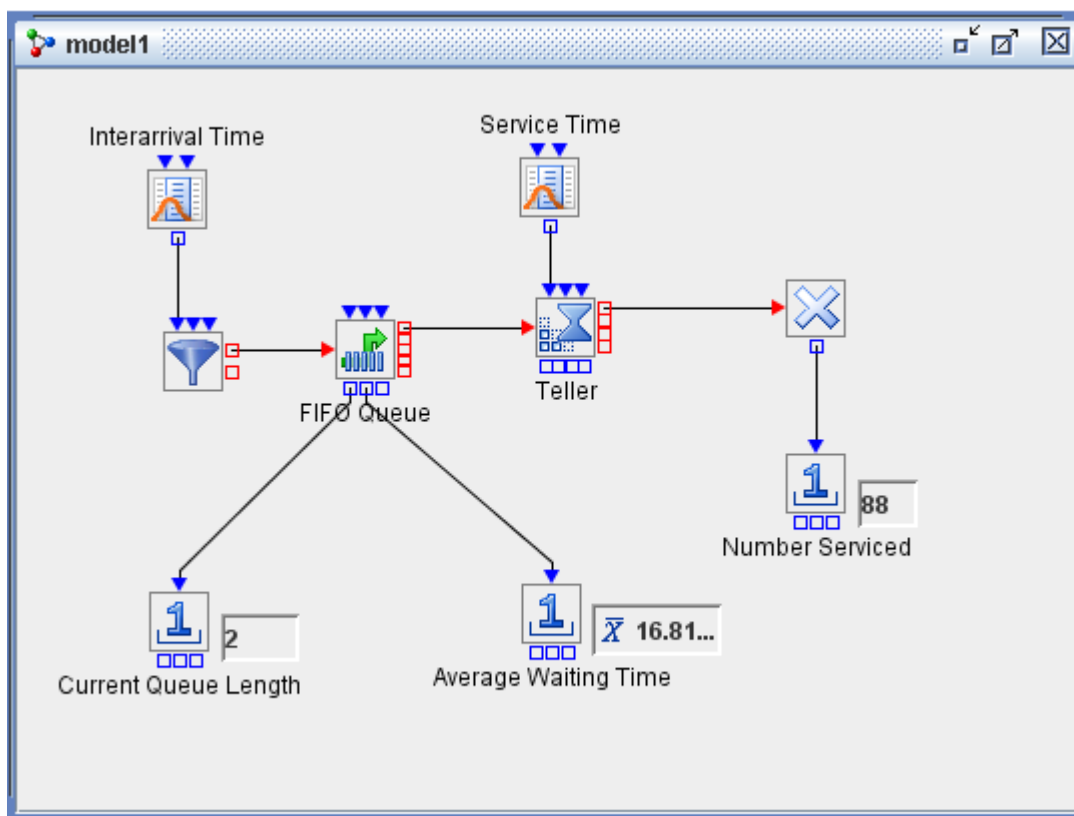
The details for building and running the M/M/1 queueing model depicted in Figure E.1 are provided in Chapter 2, "Overview of SAS Simulation Studio"; rather than repeating them here, this section provides suggestions for experimenting with this model to familiarize yourself with various features and functionality in Simulation Studio.

This model provides a good vehicle for acquainting yourself with the Log and Trace features in Simulation Studio. If you delete the link between the blocks labeled Interarrival Time and Arriving Customers and then attempt to run the model, a SEVERE level message is posted to the log with the description "Arriving Customers has no inter-arrival time connections." The model does not run because SEVERE log messages always halt the execution of the simulation model.

Reconnect the Interarrival Time and Arriving Customer blocks and change the distribution associated with the Interarrival Time block to one that is likely to produce negative numbers, such as the uniform distribution with parameters min=–2 and max=2. When you run the model now, you see WARNING messages posted to log with the message "Arriving Customers inter-arrival time value is negative; using 0.0 instead." The simulation continues to run after WARNING messages are posted.

If you enable the Tracer (Chapter 11, "Model Debugging and Verification") and then run the model, trace messages are generated by the various blocks and displayed in the Trace window. Each line in the Trace window contains the name of the block that creates the message and a short description of the event. An example trace message here might be "Numeric Source: Sampling, value = 0.136." The Tracer facility can generate many, many trace messages. See the section "Tracing Configuration" on page 115 for details about how to reduce the number of generated trace messages.

You can also use this model to practice defining factors, responses, and anchors and then use them to set up a simple experiment. Details are found in Chapter 5, "Experiments." For this example you can define a factor for changing the capacity of the Teller block and a response for recording the average wait time at the Queue block. After you create anchors between the new factor and response to the appropriate blocks in the model, you can include the factor and response in an Experiment window. After the factor and response are included in an Experiment window, you can create multiple design points with different values for the capacity factor, run the experiment, and compare the results.

It is easy to extend this M/M/1 model to incorporate many other Simulation Studio blocks and features such as data collection, plots, and so on to familiarize yourself with these capabilities so that you can apply them later in more sophisticated models.

Using the Stats Collector Block for Output Analysis

Chapter 13, "Data Input, Collection, and Analysis," first introduced this example. You can find the project files for this example in the folder `\projects\examples\docStatsCollector` in your Simulation Studio installation directory.

You can use the data collection blocks described in the section "Block Data Storage" on page 125 to collect both time-dependent and time-independent data and, in some cases, to compute statistics. For example, as shown in the simple bank system example in Figure E.2, you can use a Number Holder block (labeled NumberInSystem) connected to a Stats Collector block to record data about the total number of customers in a bank. Each time the value in the Number Holder block is updated, that value and the current time are stored by the Stats Collector block. At the end of the run, the Stats Collector block computes the user-defined time-average number in the system, as displayed in the table labeled NumberInSystem Statistics. The output that is generated by the Stats Collector block can also be saved as a SAS data set.

Although you can use the Number Holder block to collect data (in the form of time/value pairs) and compute observation-based statistics (such as mean, minimum, and maximum), you should not use this block to compute statistics for time-dependent data, such as queue length. In Figure E.2, a Number Holder block labeled Non-WeightedAvgQueueLength is connected to the OutLength port of the FIFO Queue block, and the **Display** option in the Number Holder block is set to **Mean**. This is not the correct way to compute the average queue length because the mean that the Number Holder block computes is not time-weighted.

The correct time-weighted average queue length is computed by connecting a Stats Collector block to the OutLength port of the FIFO Queue block. This mean value, in the "Time-Weighted Average Queue Length" table in Figure E.2, matches the value AvgQLength that is computed by the Queue Stats Collector block, as displayed in the "Queue Stats Collector Results" table.

**Figure E.2** Stats Collector Block Example
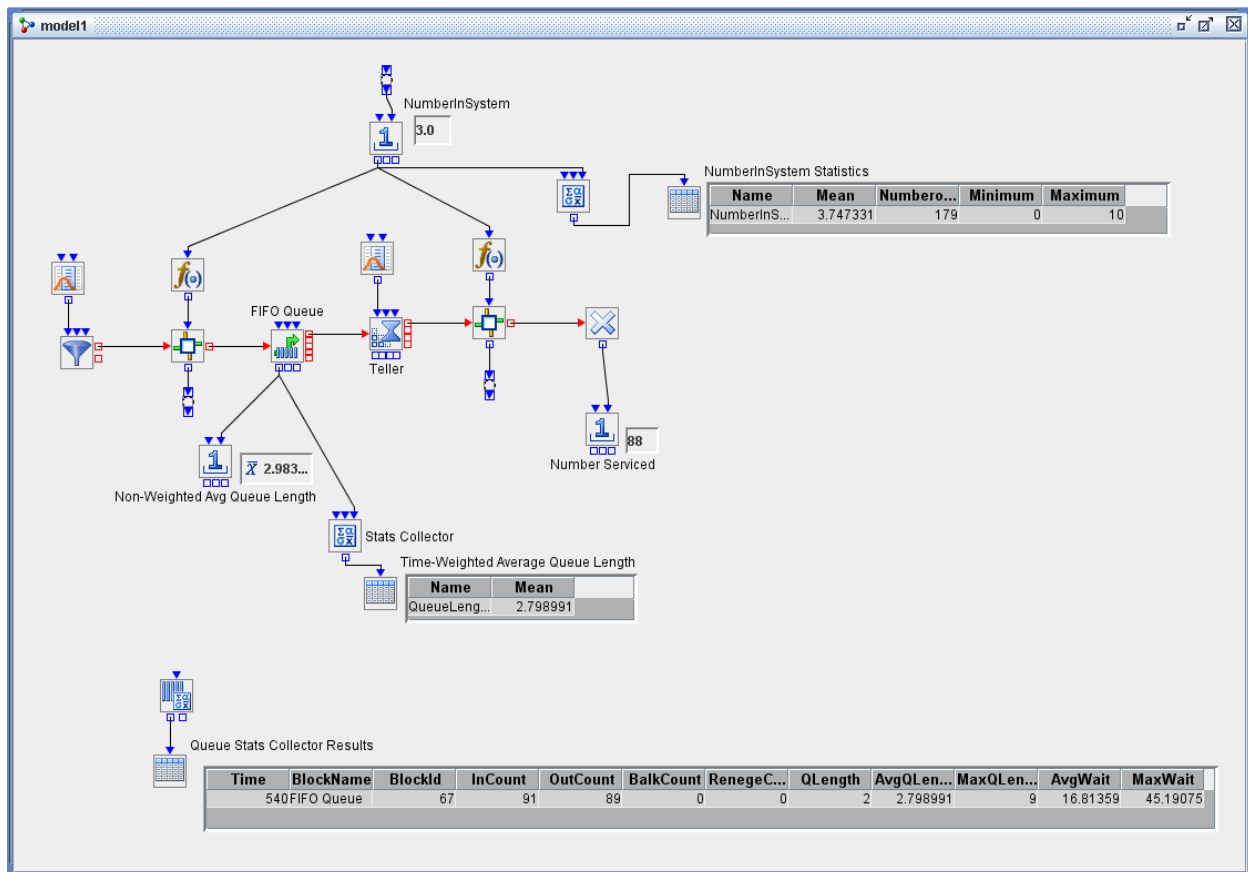
Routing to Shortest Queue

This example demonstrates how to use Switch and Formula blocks to route entities to the queue that has the shortest length when multiple queues are available. It also uses the Queue Stats Collector block, the Bucket block, and various Plot blocks to illustrate statistics collection and visualization. You can find the project files for this example in the folder `\projects\examples\docQueueRouting` in your Simulation Studio installation directory.

Entities are created according to an exponential distribution with a mean of 1. Figure E.3 shows three queues in which entities wait for a single server. Entities are routed to the queue with the shortest length. If all three queues have the same length, the entity routes to Queue1. The time it takes for each entity to be served is sampled from an exponential distribution with a mean of 1. The simulation is run for 5,000 time units, and the Entity Generator block shuts down after 4,970 time units to make sure that entities are being pulled from all three queues. (By default, the server checks Queue1 first to determine whether any entities are waiting, and then Queue2, and then Queue3. Thus entities move out of Queue3 only if Queue1 and Queue2 are empty.)

Figure E.3 Routing Example
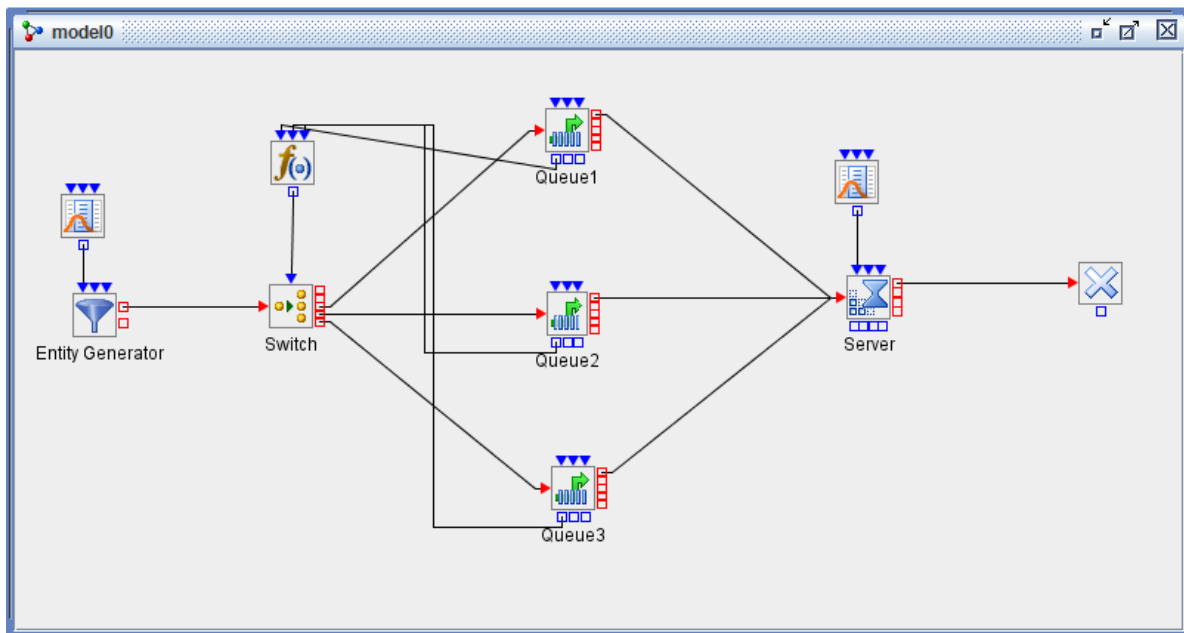
After the Entity Generator block creates an entity, the entity flows to the Switch block for routing to the desired queue. When an entity arrives at the Switch block, the Switch block pulls a value from the Formula block attached to the Switch block's InData port. Figure E.4 shows the Formula block's expression. The Formula block pulls the queue length from each of the queues in the model and then returns a value of 1, 2, or 3 (indicating the shortest queue) to the Switch block based on the comparison of the queue lengths.

Figure E.4 Routing Formula

The Switch block attempts to match the value returned by the Formula block with the cases defined on the Switch block. (See Figure E.5.) The entity is then pushed out the port associated with the matched case.

Figure E.5 Routing Switch Cases
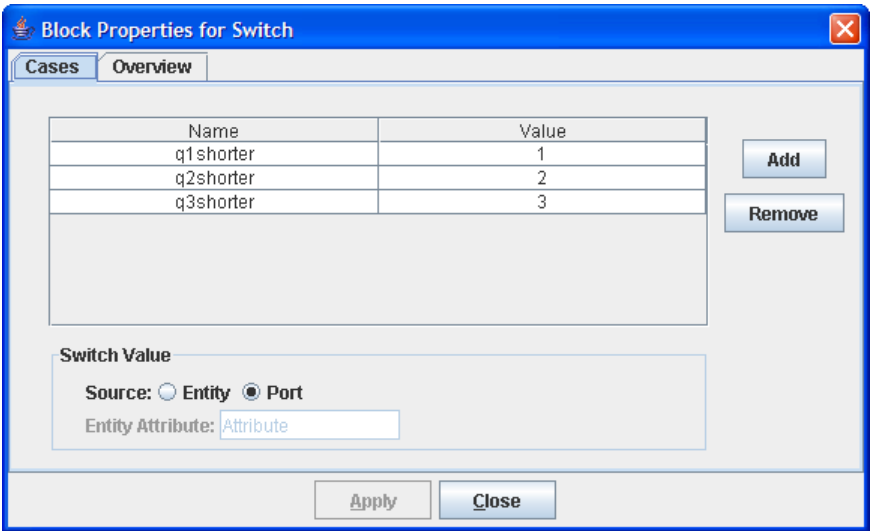
When the Server block becomes available, it attempts to pull an entity from a link connected to its InEntity port. In this example, three links are connected to the Server block's InEntity port. By convention in Simulation Studio, the pull is attempted from the first link connected to the Server block's InEntity port during the model construction process. If this is unsuccessful, the Switch block attempts to pull from the second link, and so on. In this example, the link from Queue1 to the Server block was created first, followed by the link from Queue2 to the Server block, and finally from Queue3 to the Server block.

Figure E.6 shows the model in Figure E.3, extended to use the Queue Stats Collector block and the Bucket block to collect statistics and data.

Figure E.6 Sample Routing Example Results
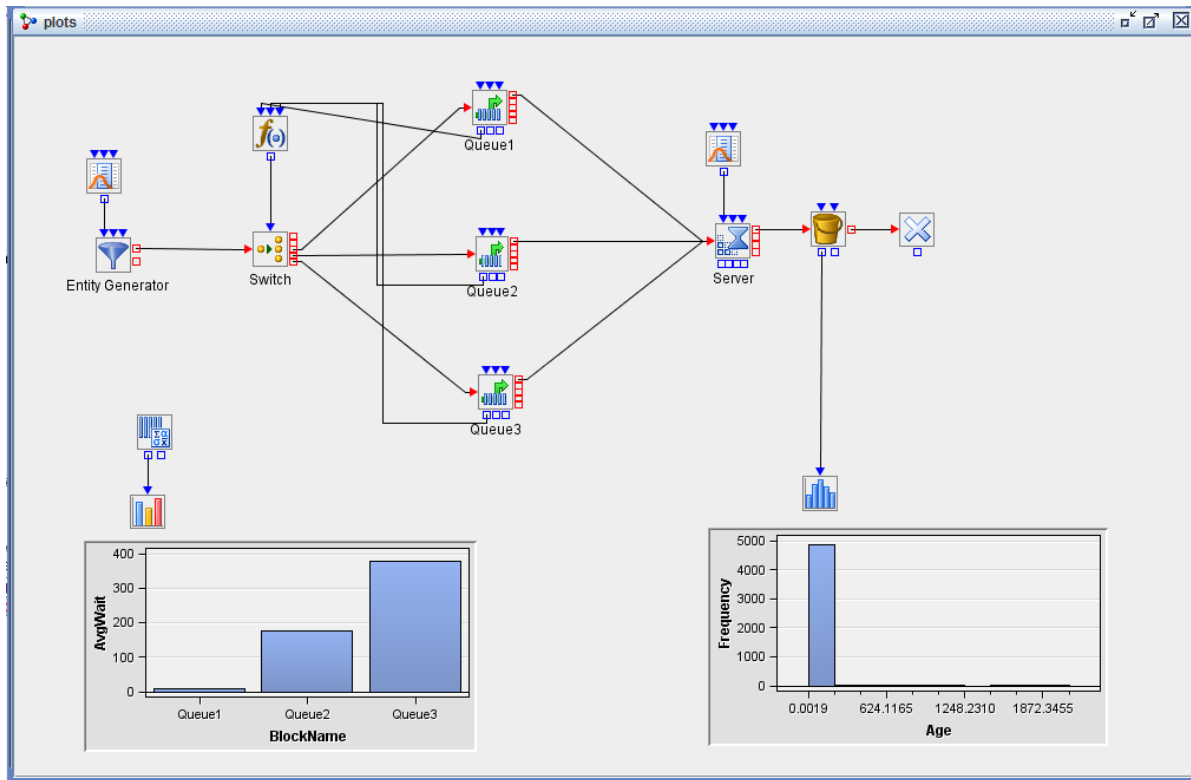
The Bucket block is configured to collect the **age** attribute of every entity that passes through it and store the value in a SimDataModel. The SimDataModel is passed to a Histogram block where the user has selected to display a histogram of the **age** variable from the incoming SimDataModel.

The Queue Stats Collector block in the model has been configured to collect data on all three queues. (See Figure E.7.)

Figure E.7 Queue Stats Collector Dialog Box
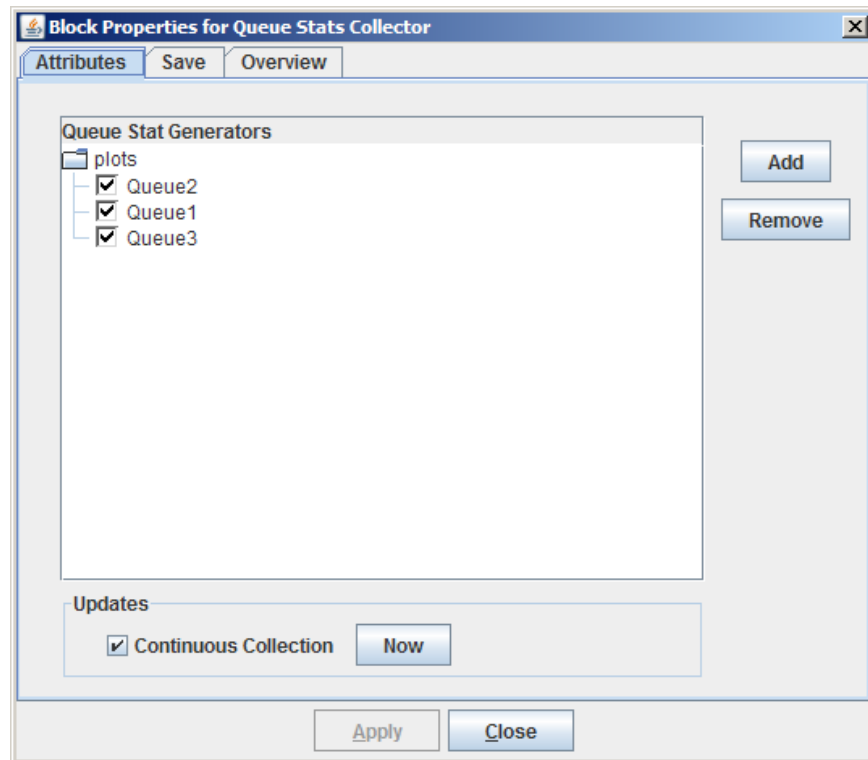
By default, the Queue Stats Collector block saves the following information for each queue it monitors:

Time	time the statistic was recorded
BlockName	name of the queue
BlockId	numeric ID of the queue
InCount	number of entities that enter the queue
OutCount	number of entities that exit the queue via the OutEntity port
BalkCount	number of entities that exit the queue via the OutBalk port
RenegeCount	number of entities that exit the queue via the OutRenege port
QLength	length of the queue at time Time
AvgQLength	average length of the queue
MaxQLength	maximum length of the queue
AvgWait	average wait time in the queue
MaxWait	maximum wait time in the queue

The data are saved in a SimDataModel that is accessible through the OutData port of the Queue Stats Collector block. For this example the Queue Stats Collector block sends its SimDataModel to a Bar Chart block, where the AvgQLength is displayed for each queue. (See Figure E.6.)

Reneging from a Queue

This model demonstrates the reneging feature of a Queue block along with the use of the Modifier, Extractor, and Gate blocks. You can find the project files for this example in the folder `\projects\examples\docRenege` in your Simulation Studio installation directory. Two very different applications of the Extractor block are depicted in this example. A special feature of the Number Holder block is also illustrated here.

This example models a queueing system in which customers arrive randomly over time with one server to process customers. Individual customers wait in the queue for service on a first-in-first-out basis.

After waiting 5 minutes in the queue, a customer reneges (that is, leaves the queue and the system) if the amount of time that customer requires for service is greater than 3.5. The goal is to estimate the average time between customers who renege.

Figure E.8 Reneging Example
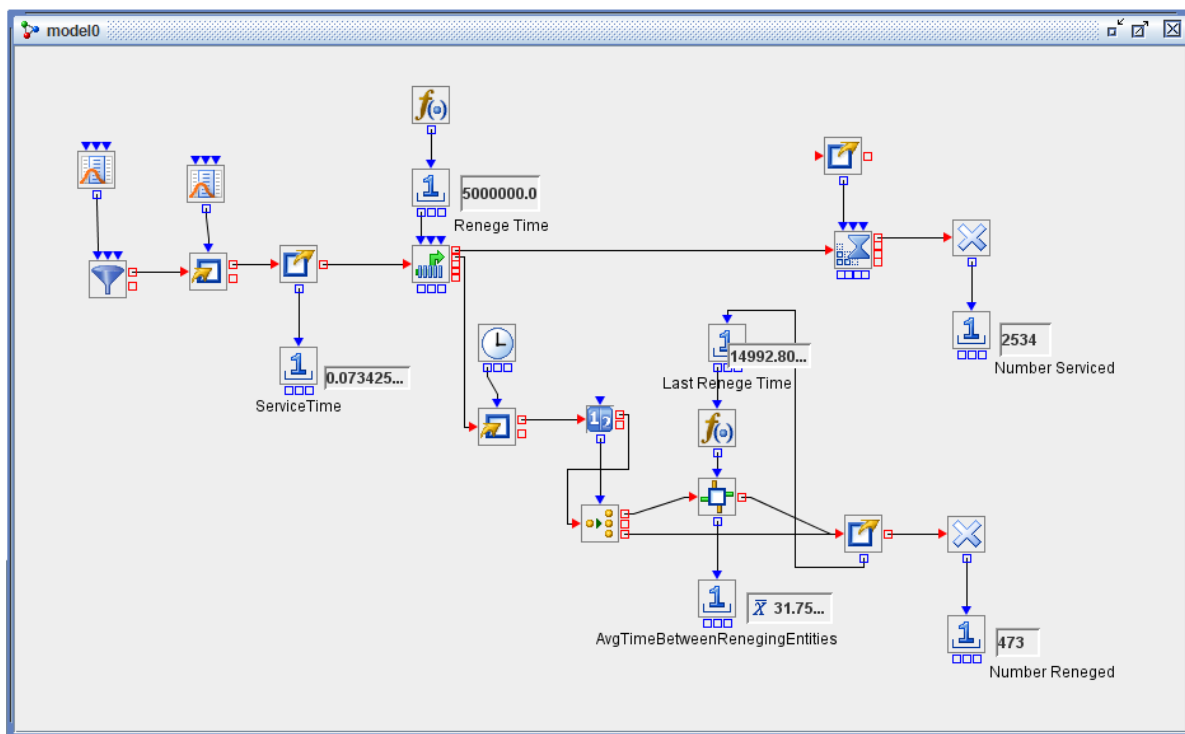
The arrival of customers is modeled by using an Entity Generator block with a Numeric Source block attached to its InterArrivalTime port. In the Numeric Source block, an exponential distribution with a mean of 5 is specified.

After entities are generated, they are sent to a Modifier block where an attribute called **servicetime** is assigned. The **servicetime** for each entity is sampled from an exponential distribution with a mean of 4.651. (The Extractor block immediately following the first Modifier block is used only to verify that the value is set in the Modifier block; it serves no other purpose in this model.)

Next the entities are sent to the Queue block (FIFO policy). As each entity enters the queue, a renege time is computed and assigned using a Formula block. Figure E.9 shows the properties dialog box for the

Formula block connected to the Queue block. In the Formula block, the **servicetime** attribute for the entity is compared to the value 3.5. If the **servicetime** is greater than 3.5, then a renege time of 5 is returned. Otherwise, a very large renege time (specifically, 5,000,000) is returned. For those entities with a **servicetime** of 3.5 or less, their renege time is set sufficiently large to ensure that they wait until they can be serviced and do not leave the queue. Note that the reneging option in the Queue block properties dialog box must be selected for reneging to be used by the queue.

Figure E.9 Renege Time Formula Dialog Box
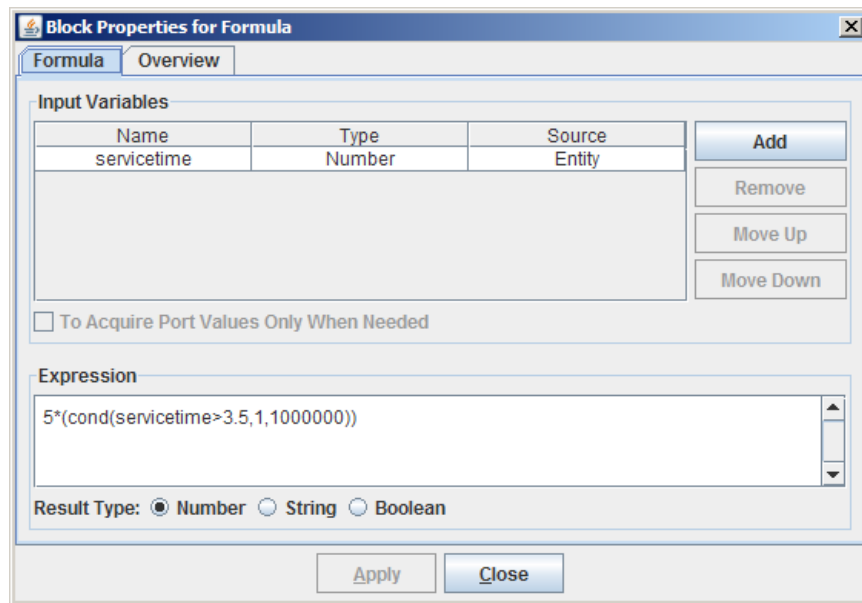
Entities that do not renege are processed in the Server block. The InServiceTime port of the Server block is connected to an Extractor block. When an entity arrives at the Server block, the Server block passes the entity to the Extractor block. The Extractor block extracts the **servicetime** attribute from the entity and passes the value to the Server block for use as the server processing time. Note that use of the Extractor block here does not require connections to its InEntity or OutEntity ports.

Entities that renege from the Queue block are sent out its OutRenege port, and the time at which they renege is stored in the entity by using a Modifier block. In this Modifier block the entity attribute **renegetime** is assigned the value Time Now. The entity is then sent to a Count block to determine whether it is the first entity to renege. The value of Count is passed to a Switch block. If the entity is the first entity to renege, the Switch block returns a value of 1 and the entity is sent to an Extractor block where the **renegetime** attribute value is extracted and passed into the Number Holder block labeled Last Renege Time.

If the entity is not the first entity to renege, it is sent to a Gate block, and a Formula block computes the time between reneging entities by subtracting the previous entity's **renegetime** from the current entity's **renegetime**. The **renegetime** for the previous entity is stored in the Number Holder block labeled LastRenegeTime. In order for this computation to work, the From Upstream option in the LastRenegeTime Number Holder block properties dialog box must be cleared, as shown in Figure E.10. If the From Upstream option is selected, then the Number Holder block pulls a value from upstream, which in this case means it pulls the value from the Extractor block. If this happens, the value of the previous entity's **renegetime** is replaced with the current entity's **renegetime**, resulting in a value of zero for the time between reneging customers.

Figure E.10 LastRenegeTime Number Holder Dialog Box
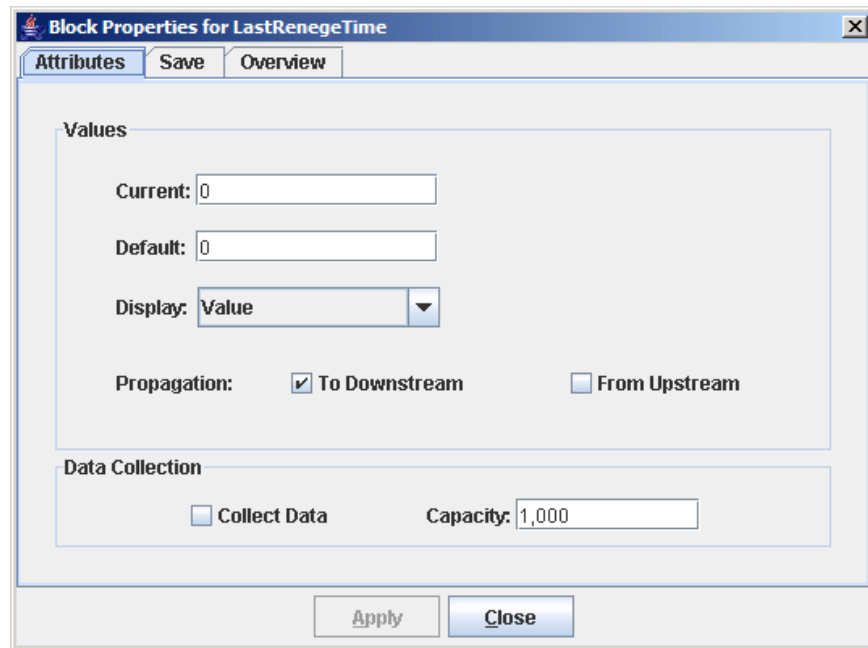
After the time between reneging entities is computed, the value is passed from the Gate block to the Number Holder block labeled AvgTimeBetweenRenegingEntities. The entity is then sent to the Extractor block. The **renegetime** attribute for the entity is extracted and sent to the Number Holder block LastRenegeTime, to be used when the next entity reneges from the system. The entity is then destroyed.

Repair Shop Model

Like the M/M/1 Queueing Model example discussed earlier, this model was also first introduced in Chapter 2, "Overview of SAS Simulation Studio," and the details and motivation for the model are found there. This section presents an enhancement to the original Repair Shop Model, which is shown in Figure E.11 and corresponds to **model1** in the project docRepairshop found in the folder \projects\examples in the directory where Simulation Studio is installed.

Figure E.11 The Repair Shop Model
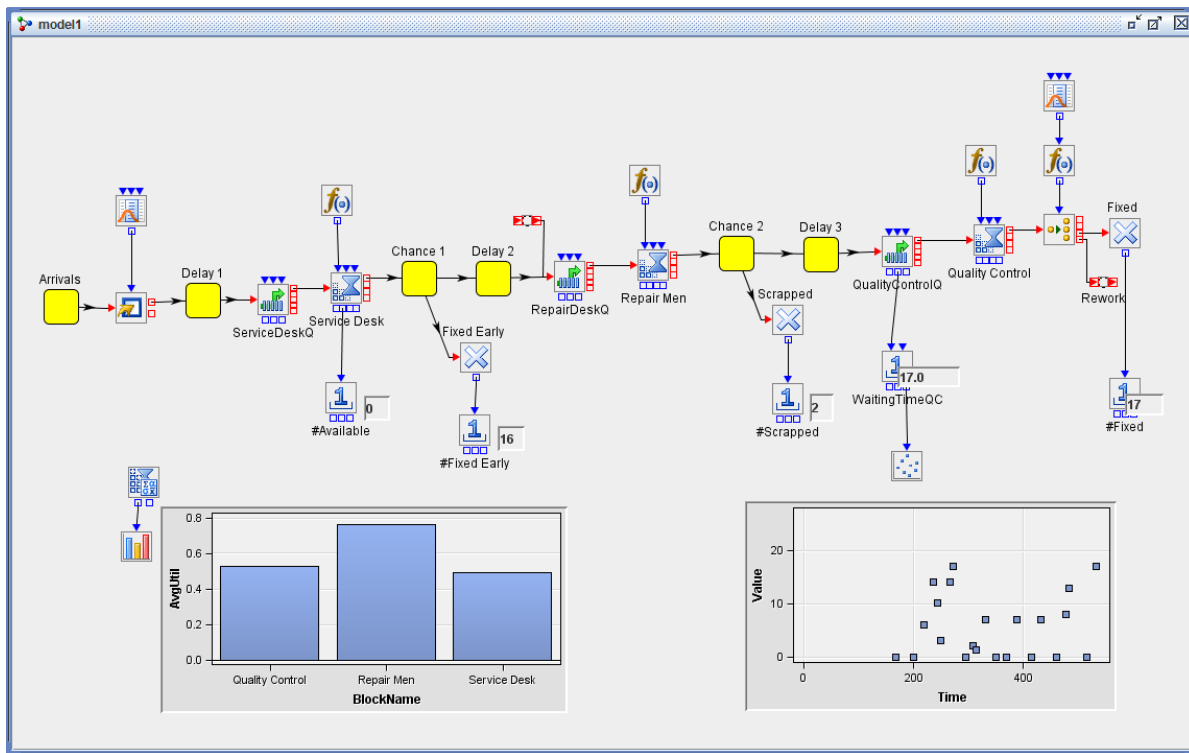
In this model, an attribute (named **PartType**) is added to the Part entities and is then used to dynamically generate a service time for a particular Part entity at each of the Server blocks in the model. In Figure E.11, a Modifier block has been added after the Arrivals compound block where the attribute **PartType** is assigned to be a random sample from the discrete uniform distribution on the interval [1,3]. At the Service Desk Server block, a Formula block is used to set the service time based on the value of **PartType** using the following expression: switch(PartType==1,5,PartType==2,10,15). Similar expressions are used at the Repair and Quality Control Server blocks so that the service time at each station is based on the value of **PartType**.

The Repair Shop model also provides an appropriately sized model for exploring the Simulation Studio linkage with the JMP routines for design of experiments. Details for defining factor and responses for the repair shop example and using JMP to generate a design are provided in Appendix C, "Design of Experiments."

PERT Network Model

This example is a program evaluation and review technique (PERT) network model of a repair and retrofit project. You can find the project files for this example in the folder `\projects\examples\docPERT` in your Simulation Studio installation directory. All activity times are assumed to be triangularly distributed. The activities relate to power units, instrumentation, and a new assembly, and they involve standard types of operations.

At the beginning of the project, three parallel activities can be performed: the disassembly of power units and instrumentation (Activity 1), the installation of a new assembly (Activity 2), and the preparation for a retrofit check (Activity 3). Cleaning, inspecting, and repairing the power units (Activity 4) and calibrating the instrumentation (Activity 5) can be done only after the power units and instrumentation have been disassembled. Thus, Activities 4 and 5 must follow Activity 1 in the network. Following the installation of the new assembly (Activity 2) and after the instruments have been calibrated (Activity 5), a check of interfaces (Activity 6) and a check of the new assembly (Activity 7) can be made. The retrofit check (Activity 9) can be made after the assembly is checked (Activity 7) and the preparation for the retrofit check (Activity 3) has been completed. The assembly and test of power units (Activity 8) can be performed following the cleaning and maintenance of power units (Activity 4). The project is considered completed when all nine activities are completed. Since Activities 6, 8, and 9 require the other activities to precede them, their completion signifies the end of the project. The goal is to estimate the project completion time.

Figure E.12 PERT Network Model
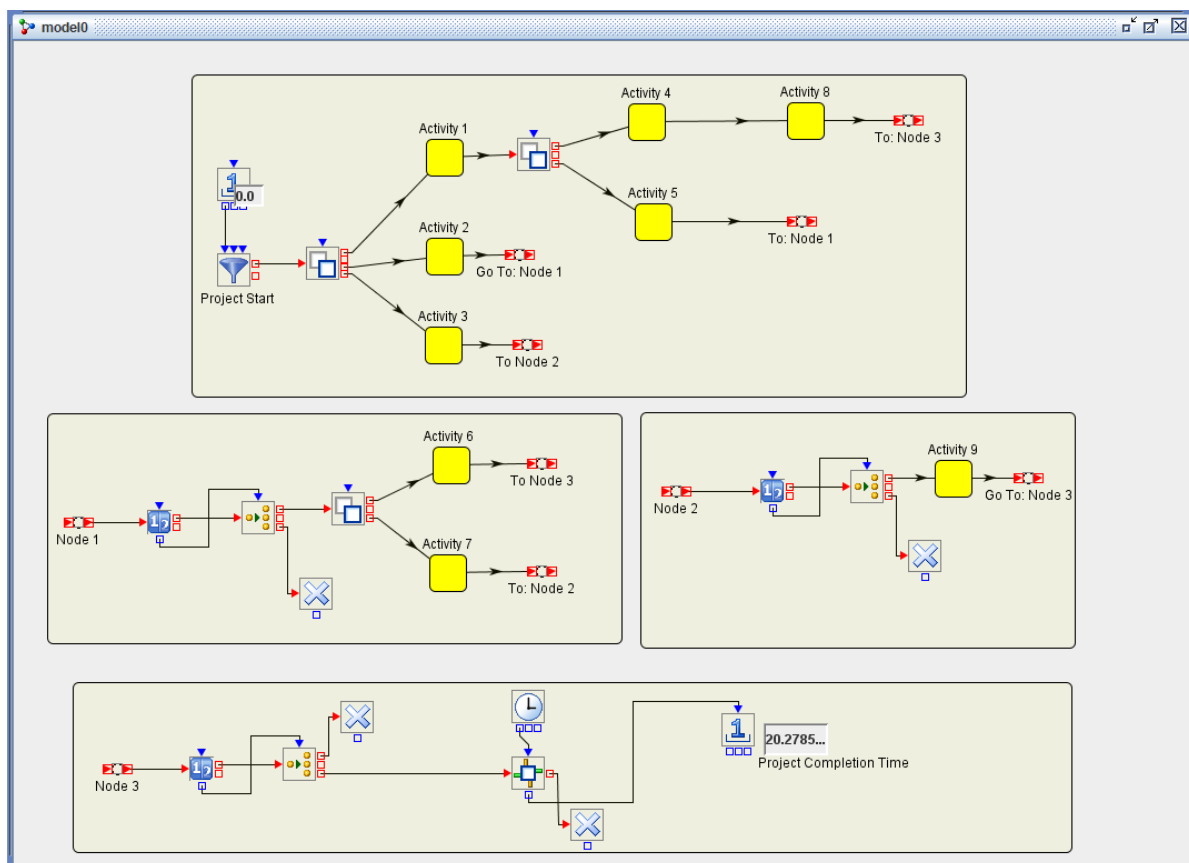
This model uses a common compound block (which consists of a Numeric Source block and a Delay block) to model the individual activities. Figure E.13 shows the structure of this compound block.

Figure E.13 PERT Model Activity Compound Block
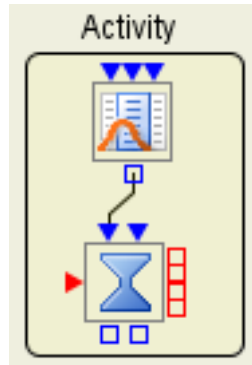
As was mentioned earlier, a triangular distribution is associated with each Numeric Source block. The distributional parameters for each of the activities are shown in Figure E.14.

Figure E.14 PERT Model Activity Table

Table of Activities			
Activity #	Min	Max	Mode
1	1	5	3
2	3	9	6
3	10	19	13
4	3	12	9
5	1	8	3
6	8	16	9
7	4	13	7
8	3	9	6
9	1	8	3

A Clone block is used to initiate parallel activities. When an entity enters a Clone block, the Clone block makes copies of the original entity and sends them out its various ports, depending on the cloning directives in Clone block. Figure E.15 shows the cloning directives for the first Clone block an entity encounters in this model. This Clone block has two additional output ports and sends one cloned entity out each port. This simulates the initiation of the disassembly of power units and instrumentation (Activity 1), the installation of a new assembly (Activity 2), and the preparation for a retrofit check (Activity 3) from the initial entity.

Figure E.15 PERT Network Cloning Directives
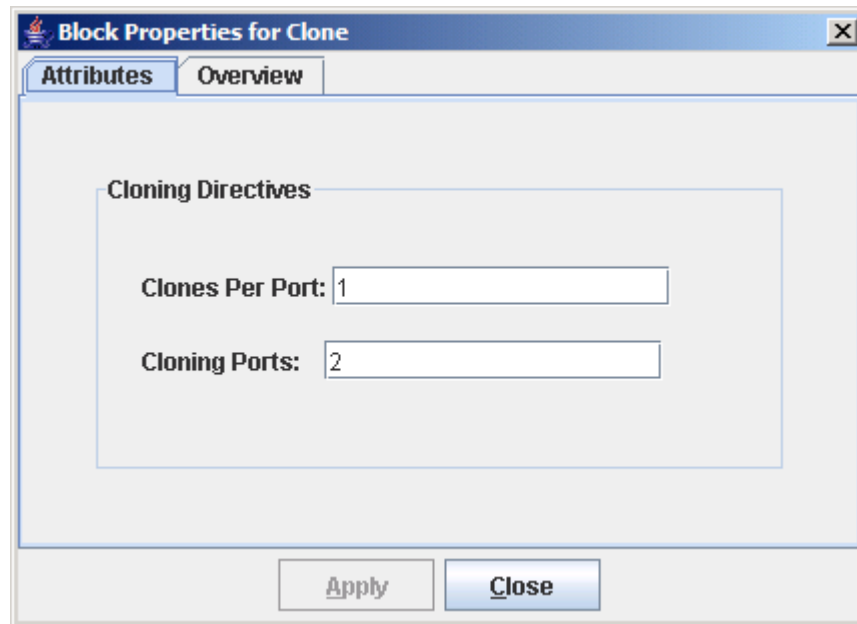
The combination of a Counter block with a Switch block is used in multiple places in the model. The Counter block simply counts how many entities have flowed through it and makes this count available via its OutCount port. Every time an entity enters a Switch block, the Switch block pulls the count value from its associated Counter block and then routes the entity accordingly. Each Switch block is essentially waiting until N entities have reached it (indicating completion of all preceding activities) before initiating the next activity in the model.

Each execution of the simulation model results in one estimate of how long it might take to complete the project. A large number of replications of the model execution are needed to produce enough data to construct a valid estimate for project completion time.

Modeling the Banking System by Using Mobile Resources

Chapter 10, "Resources," first introduced this example. You can find the project files for this example in the folder `\projects\examples\docMM1Resources` in your Simulation Studio installation directory.

In Chapter 2, "Overview of SAS Simulation Studio," and in the section "A Simple M/M/1 Queueing Model" on page 284, a model of a simple banking system with one teller is used to illustrate some of the basic concepts that are involved in building models in Simulation Studio. In that example, the bank teller is modeled by using a stationary resource (in particular, a Server block). In this section, the same system is modeled by using the bank teller as a mobile resource (a resource entity). These two examples demonstrate the conceptual differences between stationary resources and mobile resources. All blocks that are used in these models can be found in the Standard and Resource templates.

To summarize the modeling requirements for this banking system, these models assume that customers arrive at the bank at a rate of 10 per hour (so that the interarrival time between customers is a sample from the exponential distribution with a mean of 6 minutes). Customers wait in a single line on a first-come, first-served basis. The models also assume that the teller has a service rate of 12 customers per hour (so that the service time for each customer is a sample from the exponential distribution with a mean of 5 minutes).

Figure E.1 shows the original version of the model. The bank teller (represented by the Server block) is a stationary resource in the original model and is created during the model building phase. As a stationary resource, the bank teller never flows or moves through the model. A customer arrives at the bank teller, the bank teller services the customer, and the customer exits the system.

Figure E.16 shows an alternative model of the same system. In this model, the bank teller is modeled as a resource entity. The following description of the model is a high-level overview. For more information about the individual blocks used in the model, see Appendix A, "Templates."

Figure E.16 The Banking System Model Using Mobile Resources
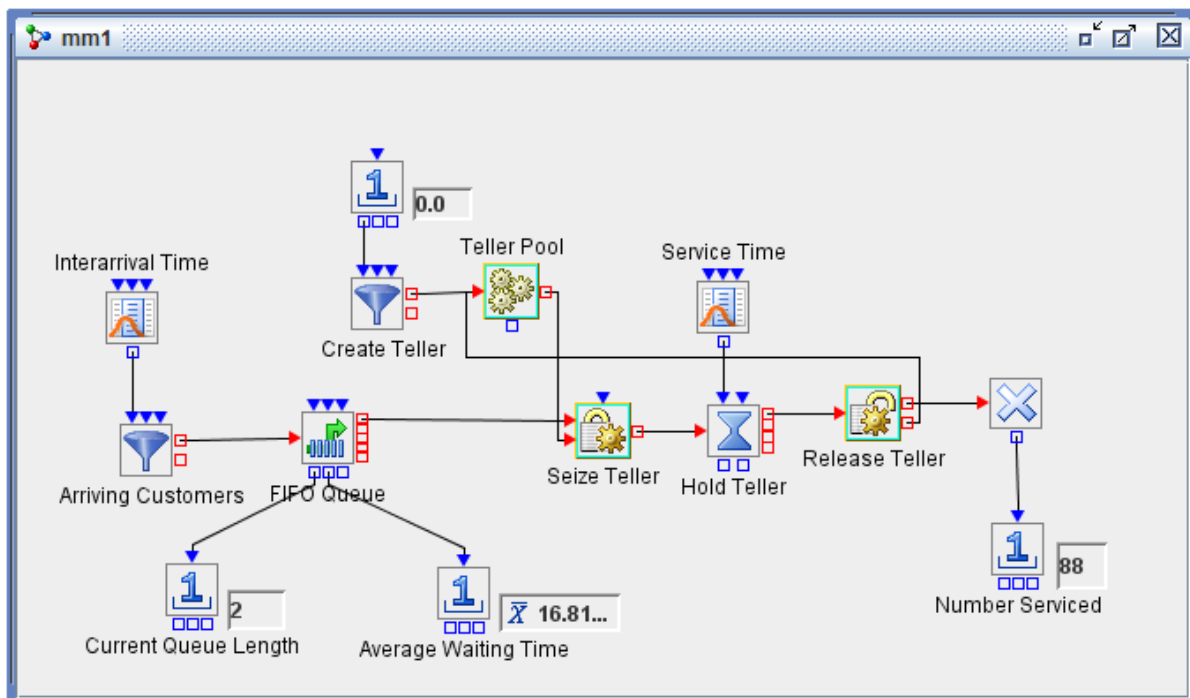
The customer arrival process is the same as in the original model—an Entity Generator block creates customers and sends them to a Queue block to wait for service. To model the bank teller as a resource entity, first a new resource entity type named BankTellers is created by using the Entity Types dialog box, available by right-clicking on the model name in the project tree and selecting **Entity Types**. For more information about the Entity Types dialog box, see Chapter 8, "Entities." Figure E.17 shows the attributes that are associated with the BankTellers resource entity type. The ResourceUnits attribute has value 1 because there is only one teller.

Figure E.17 BankTellers Entity Type

In the Block Properties dialog box for the Entity Generator block labeled "Create Teller," the BankTellers entity type is selected from the list in the Name field of the **EntityTypes** tab, as shown in Figure E.18.

Figure E.18 EntityTypes Tab in the Create Teller Block Properties Dialog Box

Because the model requires only one bank teller, you set the **Maximum Number of Entities** field value to 1 on the **Attributes** tab of the Create Teller Block Properties dialog box, as shown in Figure E.19. The bank teller resource entity must be created before the simulation clock begins to advance. Therefore, you need to set the **Start Time** property to 0 and also select the **At Start Time** option in the **First Entity Creation** area. As soon as the bank teller resource entity is created, it is sent to a Resource Pool block (labeled Teller Pool) to wait until it is needed by a customer entity.

Figure E.19 Attributes Tab in the Create Teller Block Properties Dialog Box

In this example, a Seize block (labeled Seize Teller), a Resource Pool block (labeled Teller Pool), a Delay block (labeled Hold Teller), and a Release block (labeled Release Teller) work together to mimic the functionality of the Server block (labeled Teller) in the original model. When a customer entity arrives at the FIFO Queue block, the FIFO Queue block notifies the Seize Teller block that a customer is waiting. The Seize Teller block then checks whether the bank teller resource entity is available in the Teller Pool block. If it is not available, then the customer entity stays in the queue. If the bank teller resource entity is available, then the Seize Teller block accepts the customer entity from the Queue block, pulls the bank teller resource entity from the Teller Pool block, and attaches it to the customer entity (forming a hierarchy of entities). The customer entity is then sent to the Hold Teller block where the customer entity (along with the bank teller resource entity) is held until its service is completed. It is then routed to the Release Teller block. The Release Teller block extracts the bank teller resource entity from the customer entity and sends the customer entity to the Disposer block to exit the model. The bank teller resource entity is routed back to the Teller Pool block.

A quick inspection of the values in the Number Holder blocks in both models at the end of the simulation run shows that both the original model and the new resource entity model produce the same results.

For this simple banking system, it is not necessary to use a resource entity to model the bank teller. However, in a more realistic model of a banking system, it might be necessary to seize multiple resources simultaneously, to vary the number of available resources according to a schedule, or to preempt a resource either because of a failure or because a higher priority entity arrives. It is not possible to model these scenarios using only stationary resources. In general, mobile resources offer more modeling flexibility and options.

The Banking System Model with Resource Scheduling

The resource facilities in Simulation Studio provide more advanced functionality than demonstrated in the previous simple banking system example, described in the section "Modeling the Banking System by Using Mobile Resources" on page 299. This example, first introduced in Chapter 10, "Resources," illustrates some of this additional functionality. The basic premise remains the same: customers arrive at the bank and wait to be served by a bank teller. However, in this new model there are three tellers, not all of whom are available during the entire simulation run. This model also collects utilization statistics for the bank teller resource entities.

Figure E.20 shows the new model. You can find the project files for this example (named `projects\docSchedule`) in the `examples` folder in your Simulation Studio installation directory. The most obvious difference between this resource model and the previous one is the addition of the Resource Agenda, Resource Scheduler, and Resource Stats Collector blocks. A less obvious difference is the creation, storage, and allocation approach that is used here for the three bank teller resources.

Figure E.20 Banking System Model That Uses Resource Scheduling

There are two options for creating the three bank teller resources. Either you can create three BankTeller resource entity objects, each of which has a ResourceUnits value of 1, or you can create one BankTeller resource entity object that has a ResourceUnits value of 3. To demonstrate the merging/splitting feature in Simulation Studio, this model uses the latter approach.

The Create Teller block generates one BankTeller resource entity object and passes it to the Teller Pool block, just as in the previous model. This time, the resource entity attribute of the resource entity object is equal to `3` instead of `1`. To make efficient use of the ResourceUnits in the BankTeller resource entity object, it is necessary to use the Resource Pool block's merging and splitting resource entities capabilities. Selecting the **Merge/Split resource units among resource entities of same type** check box in the Block Properties for Teller Pool dialog box (see Figure E.21) enables the block to look at the ResourceUnits attribute of its held resource entities and possibly subdivide a resource entity into two resource entities, one of which matches the needs of an incoming resource request. In this example, the Seize Teller block requests a BankTeller resource entity that has one ResourceUnit. With the merge/split option selected, the Teller Pool block can take a BankTeller resource entity that has a ResourceUnits value of 3, create a new BankTeller resource entity object that has a ResourceUnits value of 1, and decrease the ResourceUnits value of the existing BankTeller resource entity (already in the pool) to 2. The new BankTeller resource entity object (which has a ResourceUnits value of 1) is sent to the Seize Teller block to satisfy its request.

Similarly, when the BankTeller resource entity object returns to the Teller Pool block, its ResourceUnits can be merged with a BankTeller resource entity already in the pool, and the incoming BankTeller resource entity object is disposed.

Figure E.21 Teller Pool Block Properties Dialog Box

For this model, all three bank tellers might not be available during the entire simulation—maybe they take a staggered lunch break. The previous model used a total simulation time of 9 hours (540 minutes). Assume in this model that for the first 4 hours (240 minutes) of the work day, all three bank tellers are available. For the next hour, two of the tellers go on lunch break and when they return, the third teller takes an hour lunch

break. When the third teller returns from lunch break, all three tellers are available for the remainder of the work day.

A Resource Agenda block and a Resource Scheduler block are used together to implement the scheduling functionality in this model. A Resource Agenda block is used to create a list of resource adjustment actions (collectively known as a *resource agenda*) to be performed during the simulation run. The resource agenda information is passed to a Resource Scheduler block to arrange and perform the resource adjustment actions on specific resource entities. Figure E.22 shows the Resource Agenda Block Properties dialog box for this model. Each row or entry in the agenda represents a resource adjustment action and consists of three pieces of information: Duration, Value, and Value Type. For a complete description of each of these fields, see the Resource Agenda block description in Appendix A, "Templates." For this model, these entries represent the changes in the number of bank tellers throughout the simulation period.

Figure E.22 Resource Agenda Block Properties Dialog Box

The Resource Scheduler block receives the resource agenda at the beginning of the simulation run through its InAgenda input value port, and the scheduler performs the sequence of resource adjustments on a specified group of targeted resource entities. The block properties dialog box that is associated with the Resource Scheduler block in this model is shown in Figure E.23. Each row in the **Appointments** table is called an *appointment*. For more information about using the Resource Scheduler block, see Appendix A, "Templates." Only the **Start Time**, **Agenda**, and **Search Targets By** fields are discussed here. The **Start Time** field specifies the simulation time to activate the associated agenda. The **Agenda** field supplies the name of the incoming agenda to use for this appointment. (Multiple Resource Agenda blocks can be linked to the same Resource Scheduler block, each sending a named agenda to the scheduler.) The **Entity Type** field under **Search Targets By** indicates which resource entities the associated agenda in the appointment applies to. For this model, the agenda that is created in the Resource Agenda block is activated at a Start Time of 0 (when the simulation run begins) and is applied to the BankTellers resource entity objects in the model. The **Immediate**

Actions options selected here indicate that resource entities that are in a seized state are not preempted. For more information about preemption, see the section "Preempting Resource Entities" on page 111.
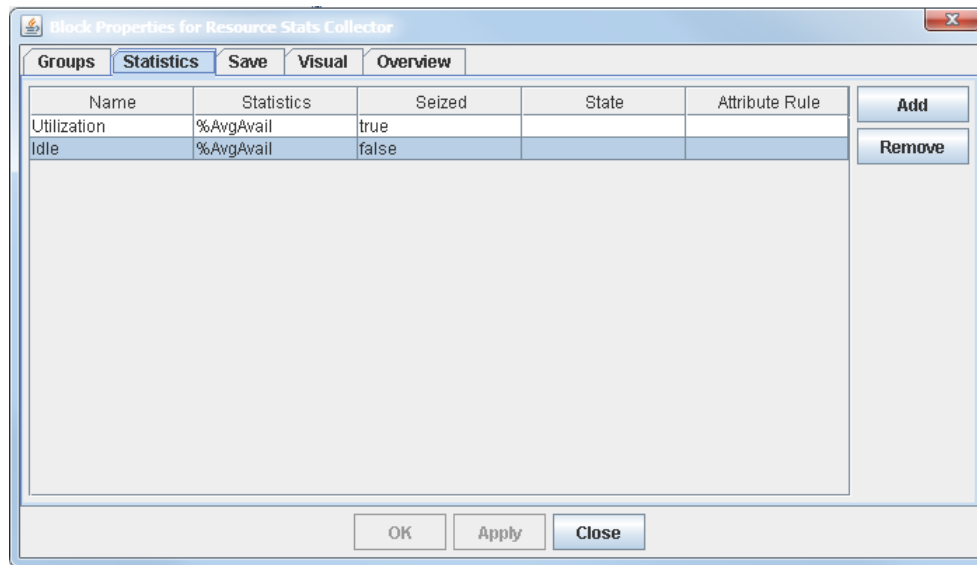
Figure E.23 Resource Scheduler Block Properties Dialog Box
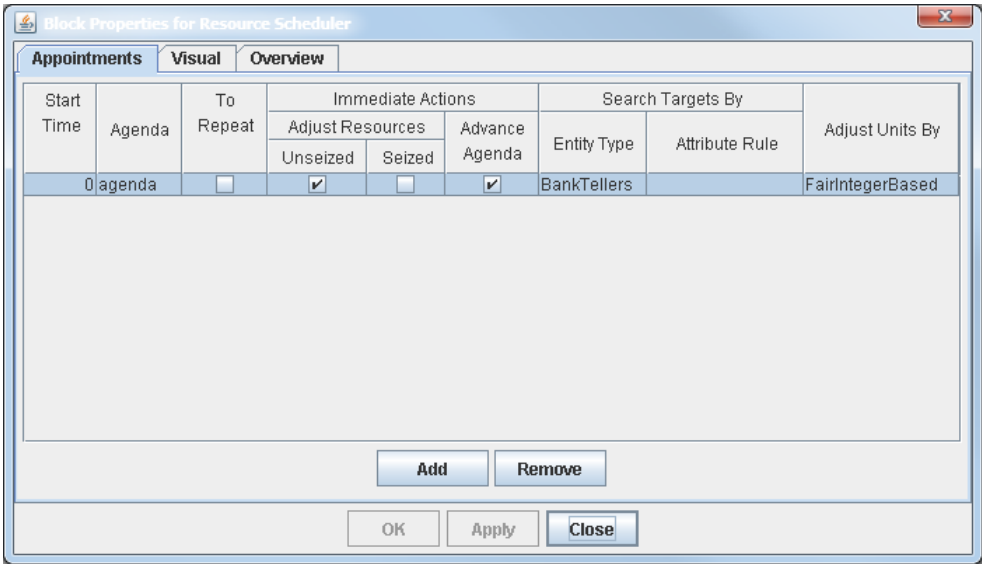
The final new block that is added to this resource model is the Resource Stats Collector block. This block collects statistics about resource entities during the simulation run. The Resource Stats Collector requires a minimum of two pieces of information: the resource entities that you want to collect statistics about and the statistics that you want to compute. The Resource Stats Collector Block Properties dialog box has separate tabs for you to enter this information. The **Groups** tab, as shown in Figure E.24, identifies the targeted resource entities for statistics collection. In this example, resource entities of the BankTellers Entity Type have been targeted. The **Statistics** tab, as shown in Figure E.25, specifies the statistics that you want to compute. For more information about the columns in this table, see the Resource Stats Collector block overview in Appendix A, "Templates."

Figure E.24 Groups Tab in Resource Stats Collector Block Properties Dialog Box
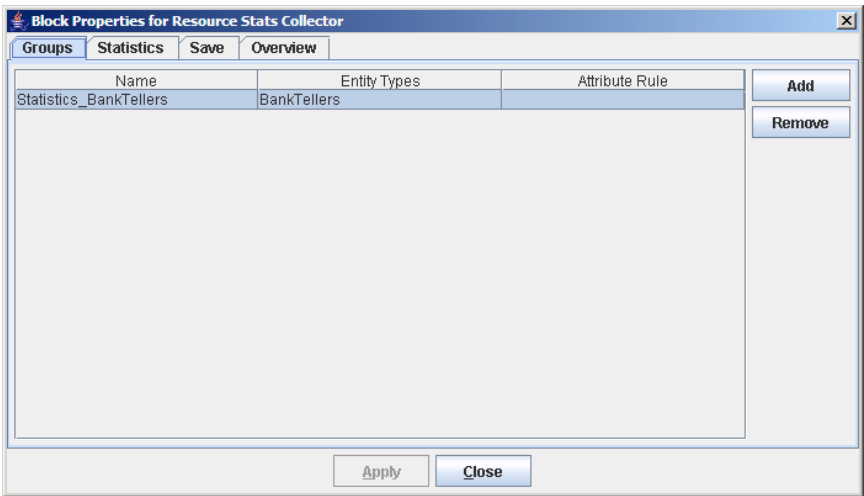
Figure E.25 Statistics Tab in Resource Stats Collector Block Properties Dialog Box

In this example, two statistics are defined: Utilization and Idle. Both are computed using the **%AvgAvail** statistic type. For the statistic named Utilization, the **Seized** option is set to true. In this case, the result is the percentage of time that resource entities of type BankTellers are seized, relative to the time-average number of available units of resource entities of type BankTellers during the simulation run. For the statistics named Idle, the **Seized** option is set to false. In this case, the result is the percentage of time that resource entities of type BankTellers are not seized, relative to the time-average number of available units of resource entities of type BankTellers during the simulation run.

The Resource Stats Collector block provides an option to save the defined statistics to a file at the end of each run (that option was selected for this example). You can also attach a plot block to the OutData outport of the Resource Stats Collector block to display the statistics during the simulation run. In this example, a Table block displays the defined statistics. At the end of the run, the Utilization statistic equals 0.361853, and the Idle statistic equals 0.638147 (as shown in Figure E.20).

This example provides a glimpse into the modeling capabilities and potential applications of the Resource Agenda, Resource Scheduler and Resource Stats Collector blocks. Although these blocks are more complex than many of the previously demonstrated blocks, they also provide powerful and flexible modeling functionality.

Priority-Based Preemption of Service

This example illustrates how to use several of the more advanced Simulation Studio blocks (Gate, Clone, Entity Group, Entity Filter) to model a system in which higher-priority customers can preempt lower-priority customers who are already receiving service. The preempted customers do not leave the system but instead wait for a server to become available again so that they can complete their service at a later time. You can find the project files for this example in the folder `\projects\examples\docPriorityPreempt` in your Simulation Studio installation directory. Figure E.26 below depicts this model.

Figure E.26 Priority-Based Preemption Example
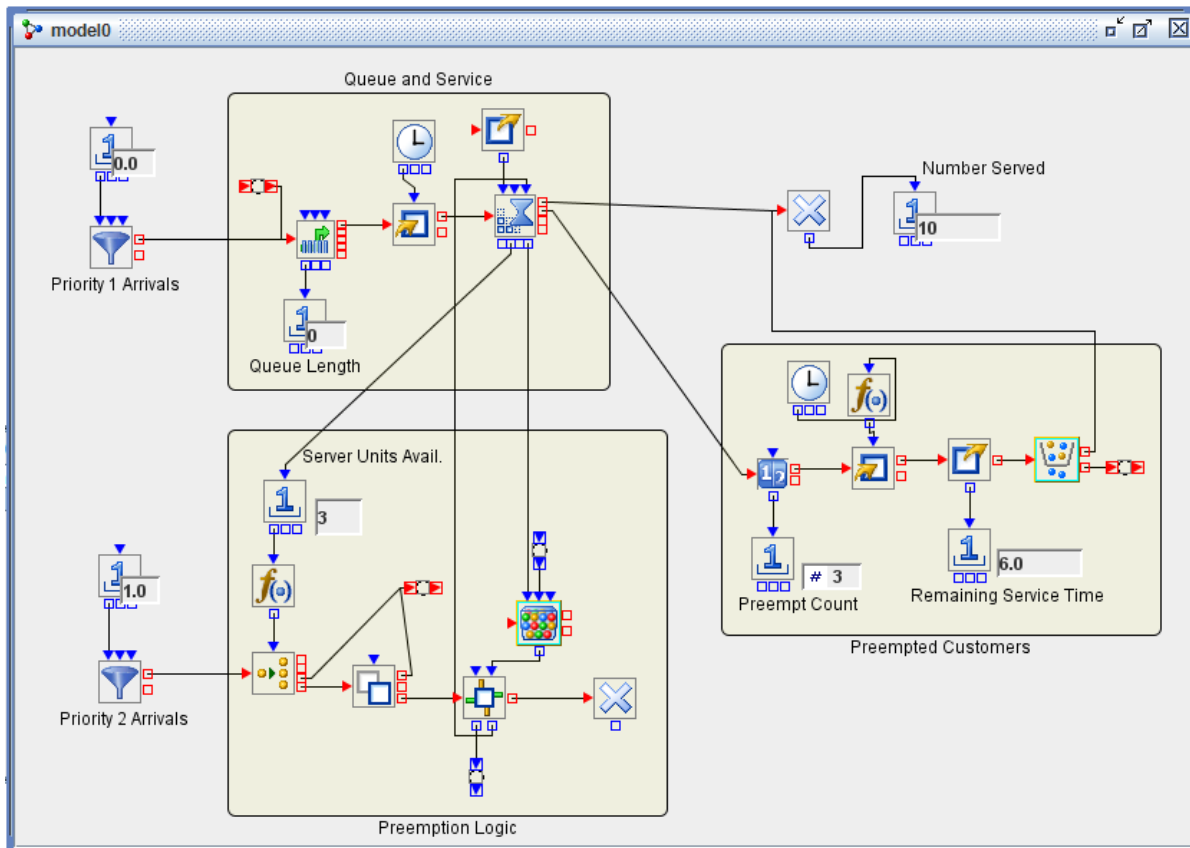
Entities are created by two distinct Entity Generator blocks—one for Priority 1 (lower priority) and one for Priority 2 (higher priority). Five Priority 1 entities are created (arrive) at time zero and five Priority 2 entities arrive one per time unit, starting at time 2. All entities are created from the entity type named Arrival, defined for this model with additional attributes **ServiceTime** and **Priority**, as shown in Figure E.27. **ServiceTime** for each entity is assigned a value of 10 units, and the **Priority** attribute is defined with a default value of 0 that is overwritten with 1 or 2 by the respective Entity Generator blocks.

Figure E.27 Entity Types Dialog Box for the Priority-Based Preemption Model

The entities enter a Queue block (Priority queueing policy) and await service from one of three servers. Priority 1 entities enter the Queue block immediately, but due to the fact that Priority 2 entities can preempt Priority 1 entities, their path (through the Preemption Logic section of the model) is more complex. A Priority 2 entity first enters a Switch block that receives input on available servers in the Server block (via output from the OutAvailable port on the Server block, fed through a Number Holder block for monitoring and then via a Formula block that evaluates whether the number of available servers is zero or positive). If a server is available, then the entity is routed to the Priority Queue block via a Connector; if no server is available, then the model must check to see whether a Priority 1 entity is currently in service that can be preempted.

In this case, the Priority 2 entity is first sent through a Clone block, which creates an additional copy of the entity. The original entity is routed to the Priority Queue block via the Connector, awaiting possible preemption of a Priority 1 entity, while its clone is sent to a Gate block. The Gate block is designed to pull and push values each time an entity passes through it. In this case, the Gate block pushes a true value to the InUpate port of an Entity Group block, causing it to pull values to create a new entity group. This link is made via Connectors for visual simplicity. This Entity Group block is intended to identify one Priority 1 entity in service that can be preempted. It pulls values from the OutHoldings port of the Server block, which supplies data on entities currently in service, and selects one with Priority value less than 2, as shown in the properties dialog box in Figure E.28.

Figure E.28 Properties Dialog Box for Entity Group Block

This dialog box specifies that the group be created from entities with a Priority value less than 2 (here, that is equivalent to Priority=1) and that the group has a maximum count of 1. Thus the Entity Group block either identifies a single Priority 1 entity currently in service that can be preempted or finds that none exists. In either case, it sends its entity group (with either one member or none) back to the Gate block via a connection between the OutSubgroup1 port of the Entity Group block and the InServiceIn port of the Gate block. The Gate block then sends the entity group out its InServiceOut port to the InPreempt port of the Server block, effectively telling the Server block which in-service entity (if any) it should preempt.

The InServiceIn and InServiceOut ports on the Gate block are created by defining an attribute **InService** for the Gate block with type Entity Group; the SignalIn and SignalOut ports are created similarly by defining a Boolean attribute **Signal**. The SignalIn port is not needed in this model. The properties dialog box for the Gate block is shown in Figure E.29. Note that the checked box in the **Default** column for **Signal** indicates that its value is true by default—needed in order to signal the Entity Group block to attempt to find a Priority 1 entity to preempt.

Figure E.29 Defining InService and Signal Attributes for the Gate Block
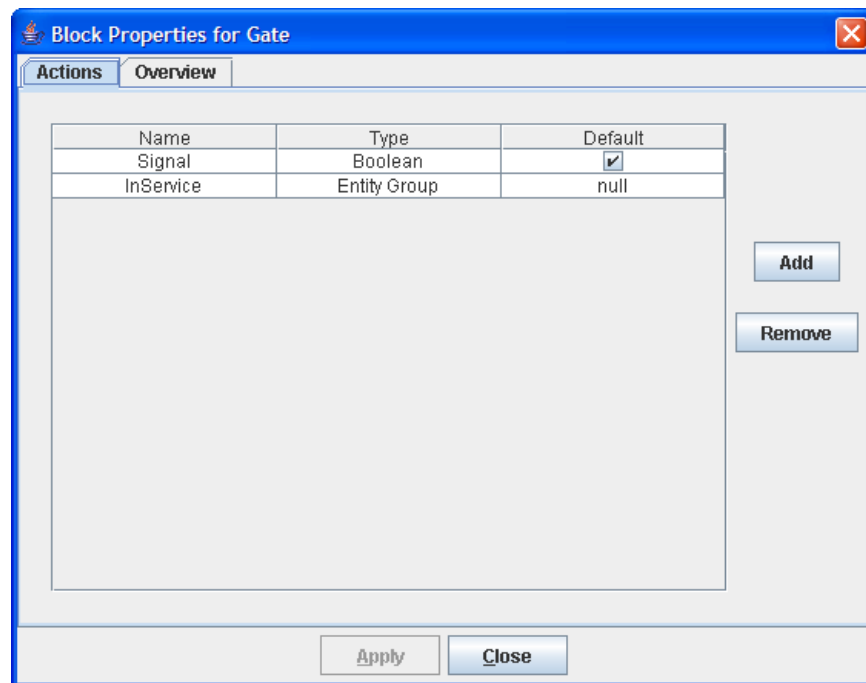
When a Priority 1 entity is preempted from the server, it is sent via the OutPreempt port on the server to a Count block (in the Preempted Customers Compound block) and then to a Modifier block where the remaining service time is computed. Before an entity enters the Server block, the current time is saved in the attribute EnterServTime. If an entity is preempted from service, the EnterServTime attribute is used by a Formula block to compute the remaining service time. The preempted entity is then routed to an Entity Filter block where the value of the attribute ServiceTime is checked. If the remaining ServiceTime is greater than 0, the preempted entity is routed back to the queue. If the remaining ServiceTime equals 0, the entity is routed to the Disposer block and leaves the system.

A Model of an Incoming Call Center

This example demonstrates the use of both regular entities and resource entities to model the operations and performance of an incoming call center, in which a finite number of telephone lines are allocated among callers who want to conduct one of two types of business. Several of the standard Simulation Studio blocks are used along with some advanced blocks and some blocks specialized for resource entities. The model is shown in Figure E.30 and you can find the project files for this example in the folder `\projects\` `examples\docCallCenter` in your Simulation Studio installation directory.

Callers choose whether to use the call center's automated call routing system or to speak with an operator. They also choose one of two activities: placing an order or speaking with customer service. Calls might be lost initially due to a lack of open phone lines (the caller gets a busy signal) or when a caller is forced to wait an excessive amount of time to speak with an operator or service representative.

**Figure E.30** Incoming Call Center Model
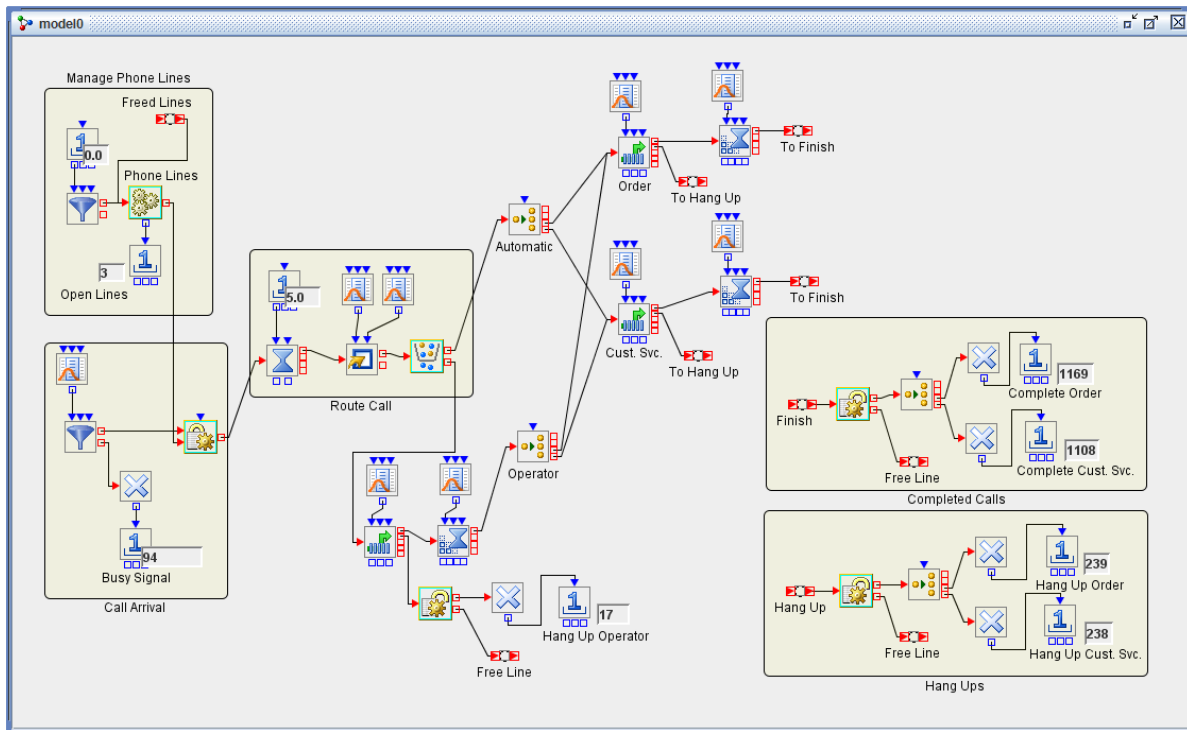
NOTE: Although the time units of the simulation clock in Simulation Studio do not denote any specific time units, for the purposes of this model each time unit represents one second.

The Manage Phone Lines section of the model creates and maintains the resource entities in this model, representing the available telephone lines in the call center. An Entity Generator block creates 15 Telephone Line resource entities at time zero and routes them to the Resource Pool block labeled Phone Lines; the Number Holder block attached to its OutLength port reports the current number of available lines.

In the Call Arrival section, incoming calls are created as regular entities. Calls arrive via an Entity Generator block according to an exponential distribution with mean 30. Each entity is created as a member of an entity type named Caller, with an attribute named **Choice** that is used to designate the type of business that the caller wishes to conduct. A second attribute, **Operator**, specifies whether the caller wishes to speak with an operator or use the automated routing system solely.

These Caller entities proceed immediately to a Seize block, which attempts to allocate one Telephone Line resource entity to the Caller entity. If no telephone lines are available, the caller receives a busy signal and hangs up; this is modeled by the Caller entity exiting its Entity Generator block via the OutBalk port to a Disposer block. A Number Holder blocks tallies these calls.

If a Telephone Line is allocated, the Caller entity moves next to the Route Call section of the model. A Delay block simulates the time (5 seconds) taken by the initial dialogue of the automated answering system, and the Caller entity moves next to a Modifier block that randomly assigns values to the **Choice** attribute (1=place order, 2=customer service) and the **Operator** attribute (0=use automated system, 1=speak with operator). An Entity Filter block checks the value of the **Operator** attribute, and routes the Caller entity accordingly. The properties dialog box for this Entity Filter block, shown in Figure E.31, show that it simply checks whether the value of **Operator** is equal to zero.

Figure E.31 Properties Dialog Box for Entity Filter
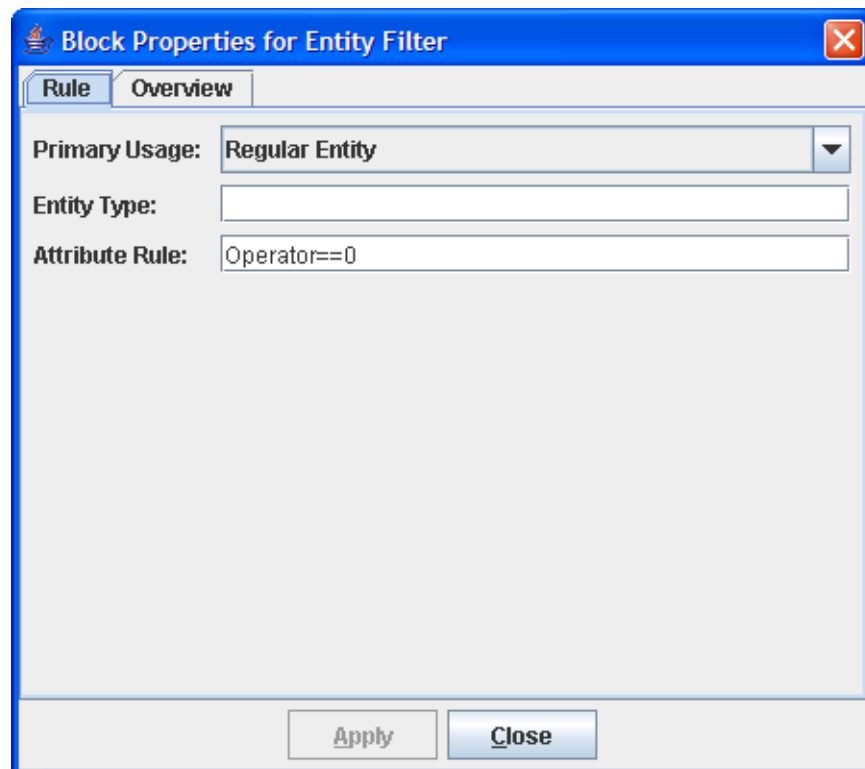
If the caller prefers to speak with an operator (the value of the **Operator** attribute is not zero), the Caller entity is routed to the lower section of the model, which consists chiefly of a Queue block (FIFO queueing policy) and a Server block (capacity 2, indicating two operators on staff). The Caller entity might renege from this queue, indicating a hang-up by a caller who has been waiting too long to speak with an operator. The distribution of the renege time is uniformly distributed from 75 to 120; this indicates that each caller waits at this point between 75 and 120 seconds before hanging up. Service time with an operator is exponentially distributed with mean 45 seconds.

If the caller hangs up while awaiting an operator, the Caller entity passes out the OutRenege port of the Queue block to a Release block that frees up the Telephone Line resource entity, sending it back to the Phone Lines Resource Pool block via a Connector. The Caller entity proceeds to a disposer, is counted, and exits the system.

If the caller completes service with the operator, the Caller entity moves next to the Operator Switch block, which routes the Caller entity according to the value of the **Choice** attribute. An identical Switch block, labeled Automatic, is encountered by Caller entities that exit the Entity Filter block with an **Operator** attribute value of zero. These two Switch blocks could easily be combined but are modeled separately for the sake of clarity.

The Switch blocks route Caller entities to either the Order queue or the Cust. Svc. queue, both located in the upper right corner of the model. The model is identical in each case, except for differences in renege time and service time distribution parameters. For each, the Caller entity might renege (the caller might hang up) and if so is routed (via a Connector) to the Hang Ups section of the model. Otherwise the Caller entity eventually proceeds to the corresponding Server block (capacity 4 for Order and 3 for Cust. Svc.) and then is routed (via a Connector) to the Completed Calls section of the model.

In both the Hang Ups section and the Completed Calls section the treatment of the Caller entity is identical. First, a Release block releases the Telephone Line resource entity back to the Phone Lines Resource Pool block. Next, a Switch block routes the Caller entity to a specific Disposer block and Number Holder block based on the value of the **Choice** attribute; this enables hang ups and completed calls according to the type of service desired or provided.

The model is run for 86,400 seconds, equal to 24 hours of continuous operation of the call center. Tracking of the number of busy signals, hang ups, and completed calls in each category can provide invaluable information about the performance of the call center under varying conditions. This model can be made even more useful by specifying key controls (number of lines, staffing levels, service times, and so on) as factors and key performance indicators (the aforementioned counts, staff utilization, queue lengths, and so on) as responses so that experimental design can be used to create a number of different scenarios for which the simulation can be run and the results tracked.

Modeling Assembly Operation and Parts Inventory System

This example shows how regular entities and resource entities, along with both standard and resource-oriented Simulation Studio blocks, can be used to model an assembly system and an associated parts inventory system. Each order (subassembly) arrives with a need for a given number of each of two parts. The parts needed are withdrawn from inventory and the assembly operation is executed; the completed order leaves the system. The part inventories are checked and replenished on a periodic basis. If the needed parts for an order are not immediately available, then the order must await inventory replenishment before it can proceed to the assembly operation. The model of this system is shown in Figure E.32 and you can find the project files for this example in the folder `\projects\examples\docInventory` in your Simulation Studio installation directory.

Figure E.32 Assembly and Parts Inventory Model
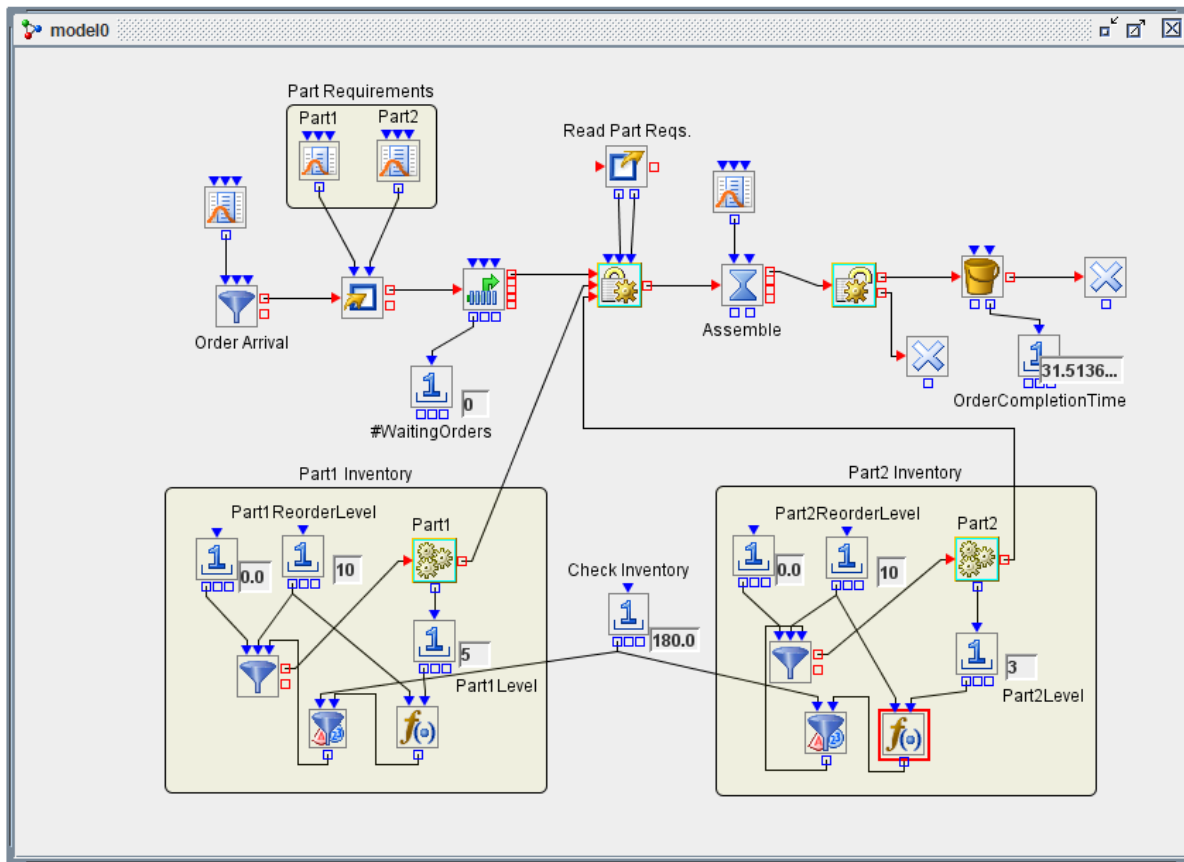
Entities that represent orders are created by the Entity Generator block labeled Order Arrival. Next, a Modifier block creates attributes **NumPart1** and **NumPart2**, which correspond to the quantities of Part1 and Part2 needed, for each order, drawing values from the Numeric Source blocks in the Part Requirements compound block. The order then proceeds to a Queue block (FIFO policy) where it waits until its needed parts are available. The length of the queue is monitored.

Following the Queue block is a Seize block that executes the procurement of needed parts. For this Seize block two resource ports, Part1 and Part2, were created as shown in the properties dialog box in Figure E.33.

Each defined resource port for a Seize block creates a resource entity input port through which units of the corresponding resource entity enter the Seize block and are given to the requesting entity (the order). Because the **Units** column for each port is left blank, additional input ports called Part1_units and Part2_units are created so that the Seize block can pull the needed units of each resource for the current requesting entity. In this model, an Extractor block supplies the values of the **NumPart1** and **NumPart2** attributes to be used for the number of units for Part1 and Part2, respectively.

Figure E.33 Properties Dialog Box for Seize Block
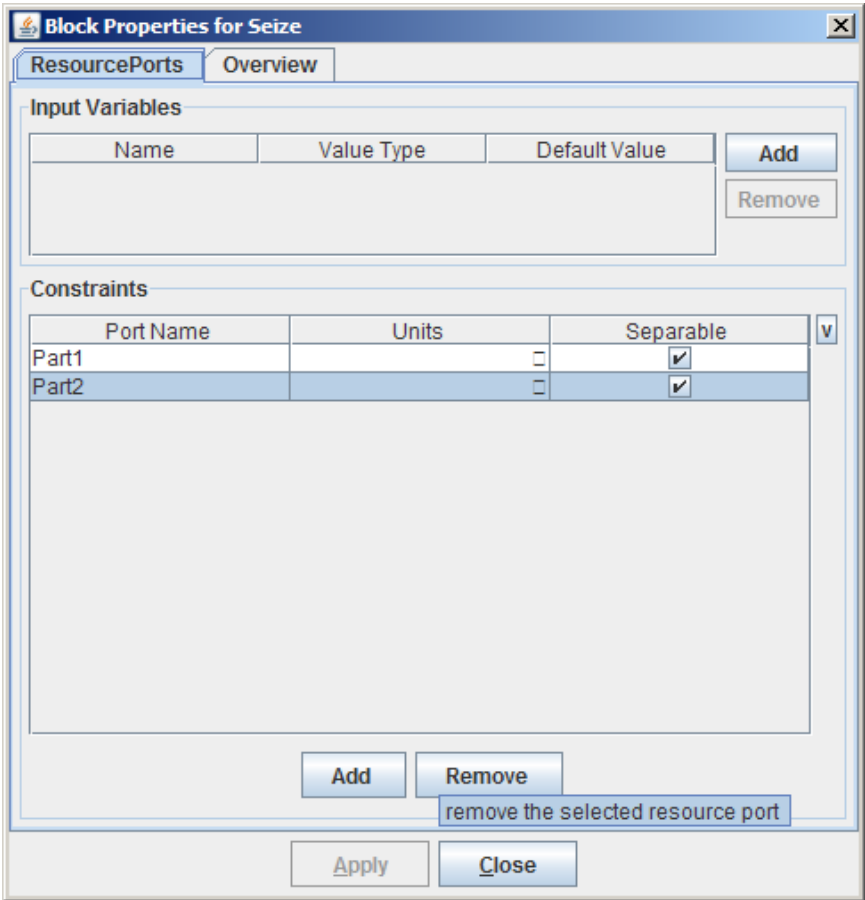
The order proceeds next to a Delay block that models the time needed for assembly (distributed uniformly between 30 and 60). Once assembly is complete, a Release block releases the previously seized units of the resource entities Part1 and Part2. In this case, since the parts are consumed during the assembly operation, they are routed to a Disposer block upon release. If the resource entities represented nondisposable resources, then they would be routed back to their respective resource pools or elsewhere in the model upon release.

Finally, the order proceeds to a Bucket block, which records the current age (time in the model) of each entity and (via a connection to the Bucket block's OutLatestAge port) passes the information to a Number Holder block for reporting and possible data collection. The order entity then exits the system via a Disposer block.

In the lower half of the model are two areas (expanded compound blocks) labeled Part1 Inventory and Part 2 Inventory. These sections of the model simulate inventory management and replenishment for the two parts. The functionality for both parts is identical; so this example focuses on Part1. Inventory is checked every 180

time units and, if the inventory of Part1 is below the Part1 reorder level, an order equal to the Part1 reorder level is placed.

In order to simulate this inventory policy, a Number Holder block supplies a constant value of 180 as the InterValue Time for a Value Generator block. Thus, every 180 time units the Value Generator block pulls a Boolean value through its InValue port from a Formula Block that compares the current Part1 inventory level to its reorder level. The formula produces a true value (indicating the need for inventory replenishment) if inventory is under the reorder level and a false value otherwise. The details are shown in the properties dialog box for the Formula Block in Figure E.34.

Figure E.34 Comparing Current Inventory to Reorder Level in the Formula Block
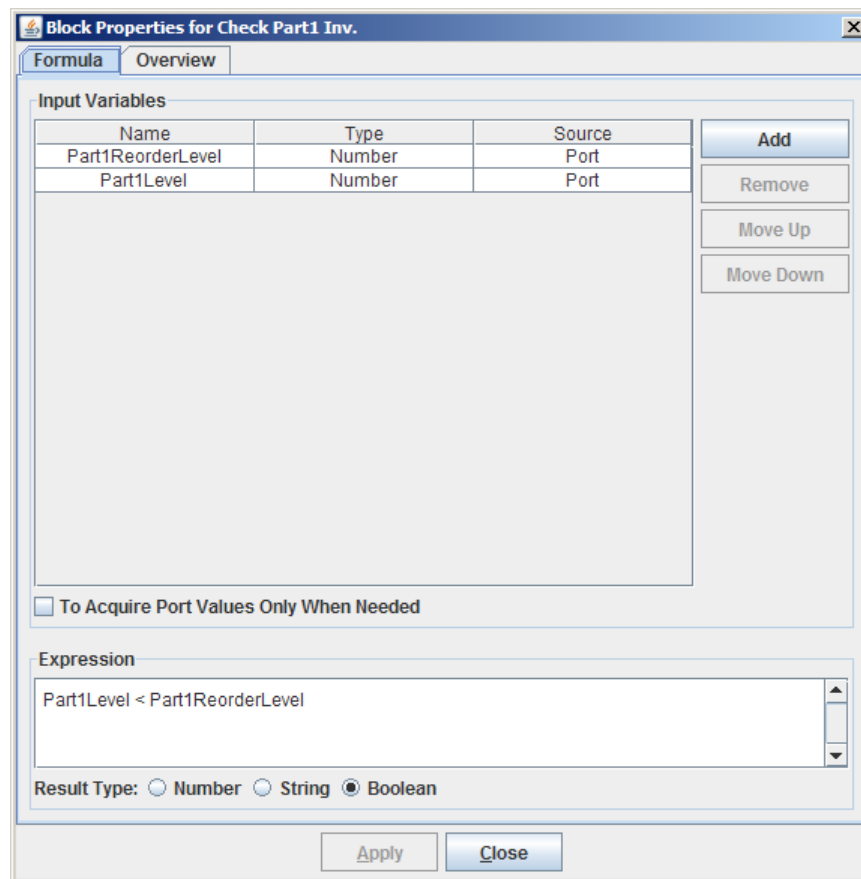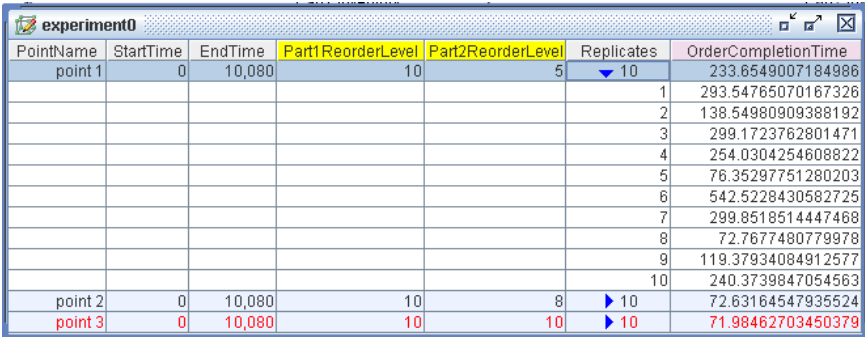
This Boolean value flows out the OutValue port of the Value Generator block to the Signal input port of an Entity Generator block that represents inventory replenishment; if the value is true, it signals the Entity Generator block to produce entities according to its configuration—to replenish the Part1 inventory. The BatchSize input port of the Entity Generator is connected to the Number Holder block that holds the reorder level (10) for Part1, and the InterArrivalTime input port is attached to a Number Holder block that holds the value zero. Collectively, this means that upon receiving a true value from the Value Generator block, the Entity Generator block creates 10 Part1 resource entities immediately, sending them to the Part1 Resource Pool block. This completes the inventory replenishment operation, and all units of Part1 are available to be given to requesting order entities via the Seize block.

The reorder levels for Part1 and Part2 are declared as factors, and the order completion time is declared as a response; these factors and responses are included in the Experiment window. Three design points, with varying reorder levels for Part1 and Part2, are run for 10,080 time units, equal to one week if each

time unit corresponds to one minute. Details about creating factors and responses and including them in the Experiment window are provided in Appendix C, "Design of Experiments."

Each design point is run for 10 replications, and the results are recorded. The Experiment window for this model is shown in Figure E.35. The first design point is expanded to show all 10 replications.

**Figure E.35** Experiment Window for Assembly and Inventory Model

Using the SAS Program Block to Analyze Simulation Results

This example shows you how to use the SAS Program block to analyze data from a run of the simulation model, analyze the data by using SAS procedures, and produce SAS graphical output from the analyses. Although this example uses a SAS program, you can also use the SAS Program block with a program written in JMP Script. This model simulates an M/M/1 system; arrivals and service times are exponentially distributed, and there is a single server. The model is shown in Figure E.36, and you can find the project files for this example in the folder `\projects\examples\docMM1SAS` in your Simulation Studio installation directory.

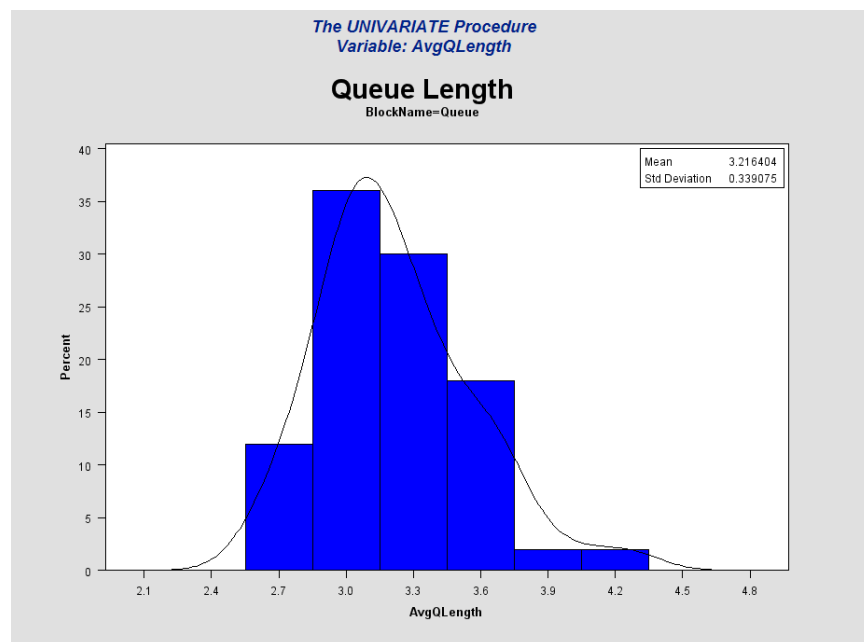**Figure E.36** M/M/1 Model with a SAS Program Block
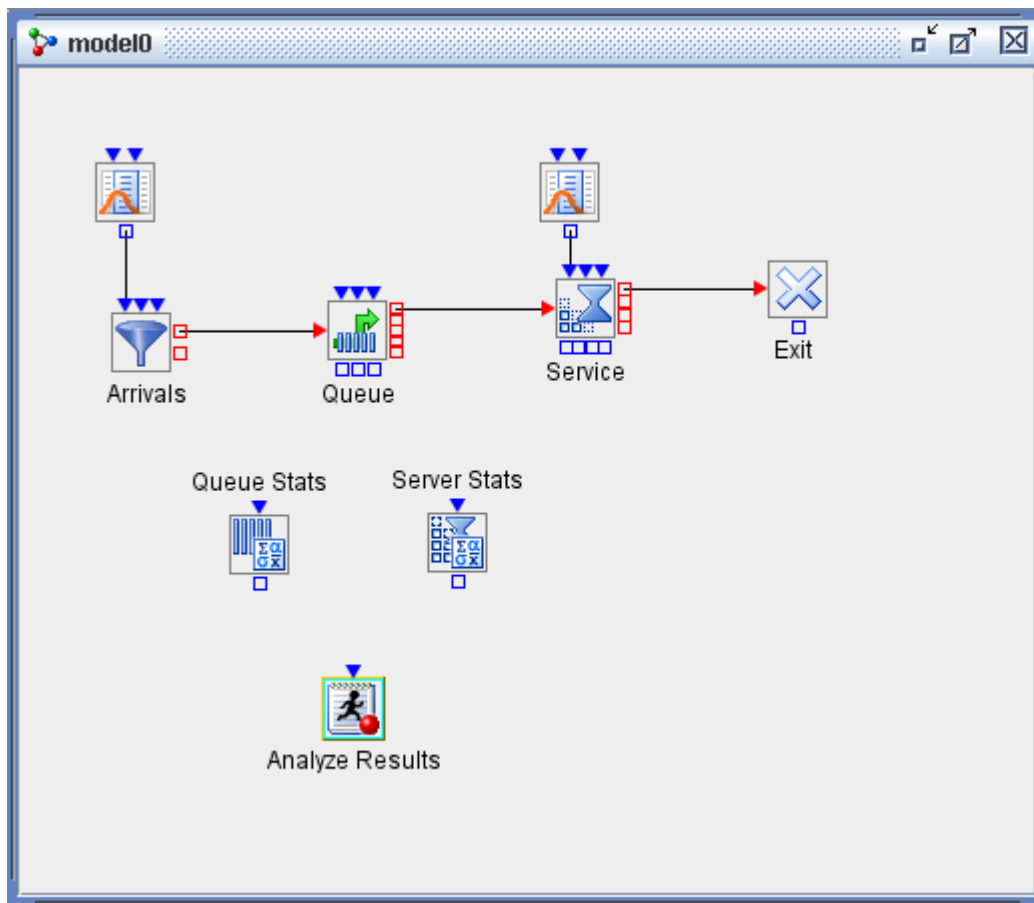
Two additional blocks gather data from the model: a Queue Stats Collector block and a Server Stats Collector block. The Queue Stats Collector block can collect data from every Queue block in the model, and the Server Stats Collector block can collect data from every Server block. This model contains only one Queue block and one Server block, so the properties dialog box for the Queue Stats Collector block lists only one possible source of data, as shown in Figure E.37.

Figure E.37 Properties Dialog Box for Queue Stats Collector Block
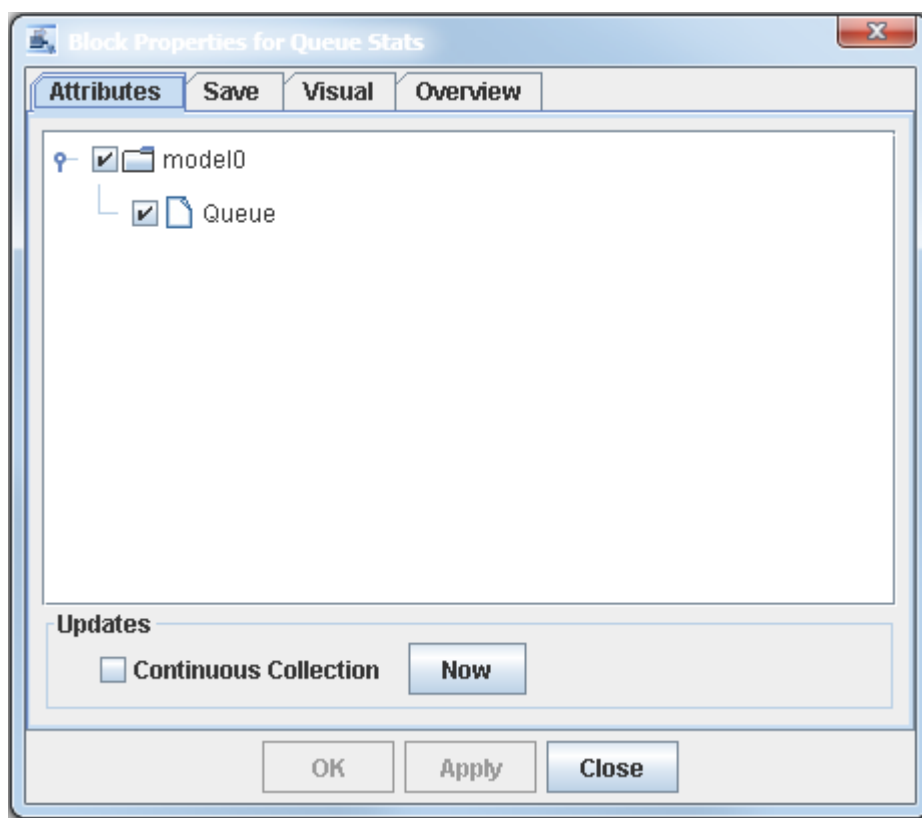
The remaining block in the model is the SAS Program block labeled Analyze Results. This block runs the SAS program that is specified in the **SAS Code Path** field of its properties dialog box after all replications have finished executing. (See Figure E.38.)

Figure E.38 Properties Dialog Box for SAS Program Block
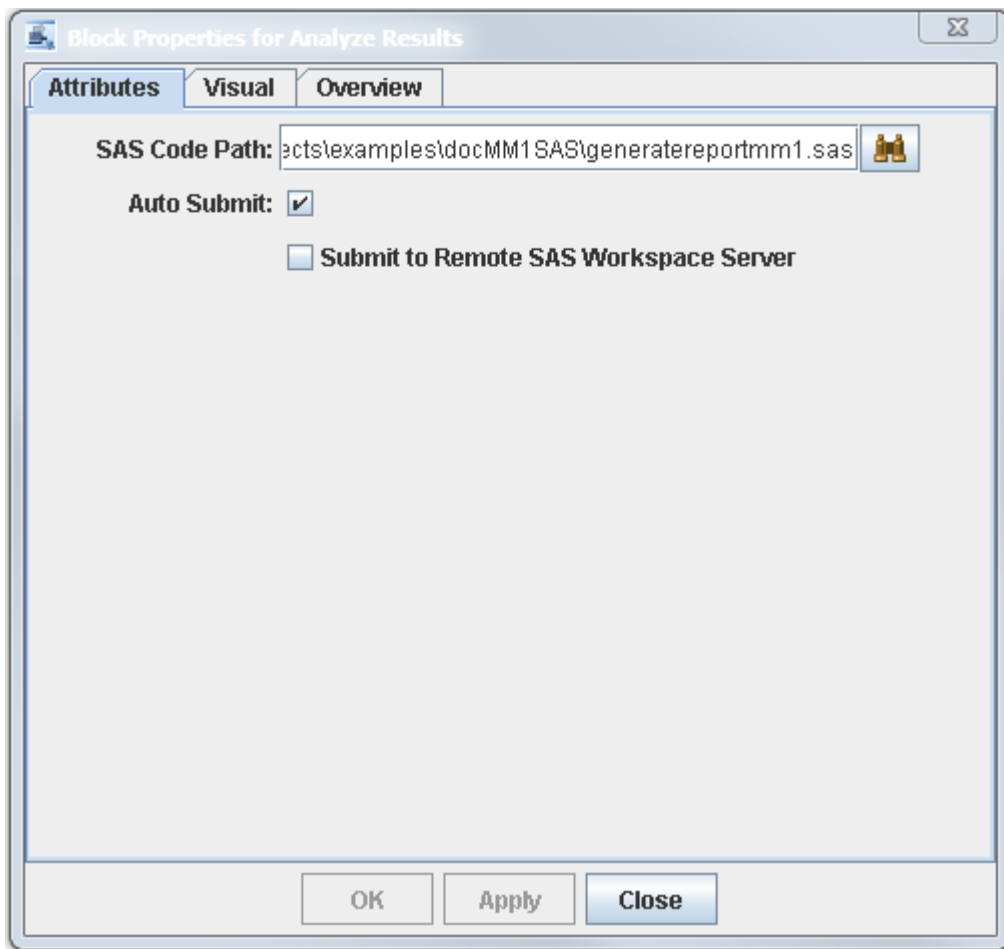
The SAS program `generatereportmm1.sas` uses the MEANS and UNIVARIATE procedures in Base SAS to analyze the queue waiting times, the length of the queue, and the utilization of the single server. It produces output in HTML format, excerpts of which are shown in Figure E.39 and Figure E.40. To generate data for these analyses, the model is run for 50 replications of 10,000 time units each. **NOTE:** You might need to modify the two LIBNAME statements in `generatereportmm1.sas` to point to the appropriate location of the simulation output data on your machine (especially if you installed Simulation Studio in a nonstandard location or if you are running the model on the Linux operating system). You can also change the ODS output paths in `generatereportmm1.sas` to save the resulting HTML output to a different location.

Figure E.39 PROC MEANS Output for M/M/1 Model

Queue Statistics Over All Replications

The MEANS Procedure

BlockName	N Obs	Variable	Mean	Std Dev	Minimum	Maximum
Queue	50	AvgQLength	3.2164040	0.3390747	2.6030158	4.2613799
		MaxQLength	29.2800000	7.0190412	20.0000000	56.0000000
		AvgWait	3.2110029	0.3290056	2.6171484	4.1807029
		MaxWait	25.3736926	5.5577984	18.3473203	44.3458727

Server Statistics Over All Replications

The MEANS Procedure

BlockName	N Obs	Variable	Mean	Std Dev	Minimum	Maximum
Server	50	AvgUtil	0.8023242	0.0098741	0.7852575	0.8357475
		MaxUtil	1.0000000	0	1.0000000	1.0000000

Figure E.40 PROC UNIVARIATE Analysis of Queue Waiting Times for M/M/1 Model

Machining Center Model

This example demonstrates how you can use the Observation Source and Dataset Holder blocks to read in and store a SAS data set or JMP table that is used repeatedly by all entities. You can find the project files for this example in the folder `\projects\examples\docDatasetHolder` in your Simulation Studio installation directory. In Figure E.41, five different types of parts arrive at a machining center for processing at four different stations: milling (station 1), turning (station 2), drilling (station 3), and chamfering (station 4).

Figure E.41 Machining Center Model
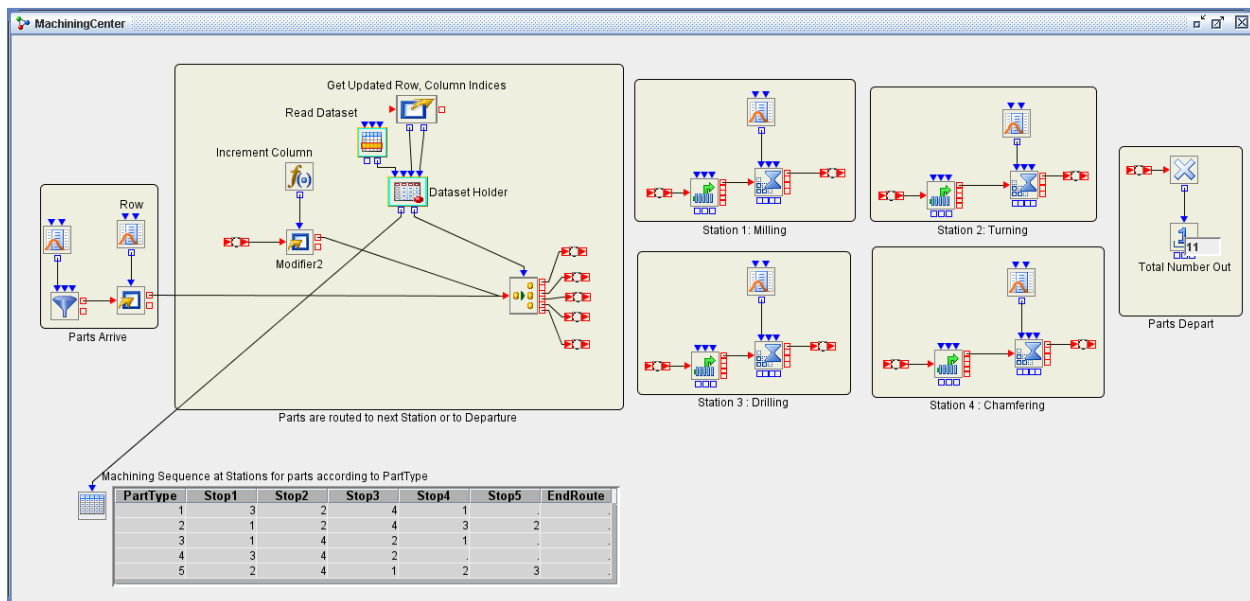
The interarrival time of parts to the center has an exponential distribution with a mean of 2 minutes. Each part that arrives has an equal probability of being one of the five types; each part type follows a different machining sequence. For example, parts of Type 1 visit the stations in the following order: chamfering, turning, drilling, and milling. Parts of Type 2 visit the station in the following order: milling, turning, chamfering, drilling, and turning (they visit the turning station twice).

Each of the four stations can work on one part at a time, and a queue in front of each station holds waiting parts. At each station, part-processing times are exponentially distributed with a mean of 1 minute (for milling), 1.5 minutes (for turning), 1.6 minutes (for drilling), and 1.8 minutes (for chamfering).

The entities in this model are the parts. You define an entity type called Parts, which includes Row and Column attributes. After each Parts entity is generated, it is sent to a Modifier block where the value of the Row attribute is randomly assigned from the discrete uniform distribution on 0 to 4. This determines the part type (because the row attribute is used to access a specific row in a data set, the values must be between 0 and 4, rather than between 1 and 5). The initial value of the Column attribute is set to 1 in the Entity Types dialog box.

An Observation Source block reads in the entire machining sequence data set, and the data model that represents that data set is passed from the OutData port of the Observation Source block to the InData port of the Dataset Holder block where it is held until needed. You can display the machining sequence data set in the top left of Figure E.41 by connecting the OutData port of the Dataset Holder block to the InData port of a Table block. Each row in the data set represents the machining sequence for a particular part type. The end of the machining sequence is indicated with a missing value "." in the sequence data set. In the Dataset Holder block, one query output called NextStation is created. The value to be returned by this query (Target Type) is of type Number. Because the default values for the Row Index and Column Index fields are blank, the row and column values are entity-dependent and are pulled from the InRow and InColumn ports of the Dataset Holder block.

When a Parts entity flows into the Switch block, it is routed either to one of the four stations for processing or to the Parts Depart compound block (signaling that it has completed its machining sequence). The InSwitchValue port of the Switch block is connected to an output port of the DataSetHolder block called NextStation. When machining at any station is complete, the Parts entity attribute Column is incremented by 1 (see the Formula block labeled IncrementColumn). The Extractor block uses this updated value to determine the next station in the sequence by passing the Row and updated Column attribute values as inputs to the InRow and InColumn ports of the Dataset Holder block.

The value NextStation is then passed to the Switch block to route the entity to the correct station. When machining is complete, the value extracted from the DataSet Holder block is ".". When "." is passed to the Switch block, the entity is forced to be routed out the OutDefault port of the Switch block because the value "." does not match any of the defined case values. The completed parts are routed to the Parts Depart compound block, which computes the total number of parts processed.

In this example, the Dataset Holder block is used to hold a data set that is used repeatedly by all entities. An alternative is to store the information in the machining sequence data set as attributes on the entities, but that would result in the same data being stored multiple times.

Using the Observation Source Block to Set Entity Attributes

This simple queueing example contains two models to demonstrate features of the Observation Source block and how it can be used to quickly set multiple entity attributes in a Modifier block. You can find the project files for this example in the folder `\projects\examples\docObsSource` in your Simulation Studio installation directory.

In `model0` in Figure E.42, you create entities and send them to a Modifier block where you set five attributes.

Figure E.42 Observation Source Block Example
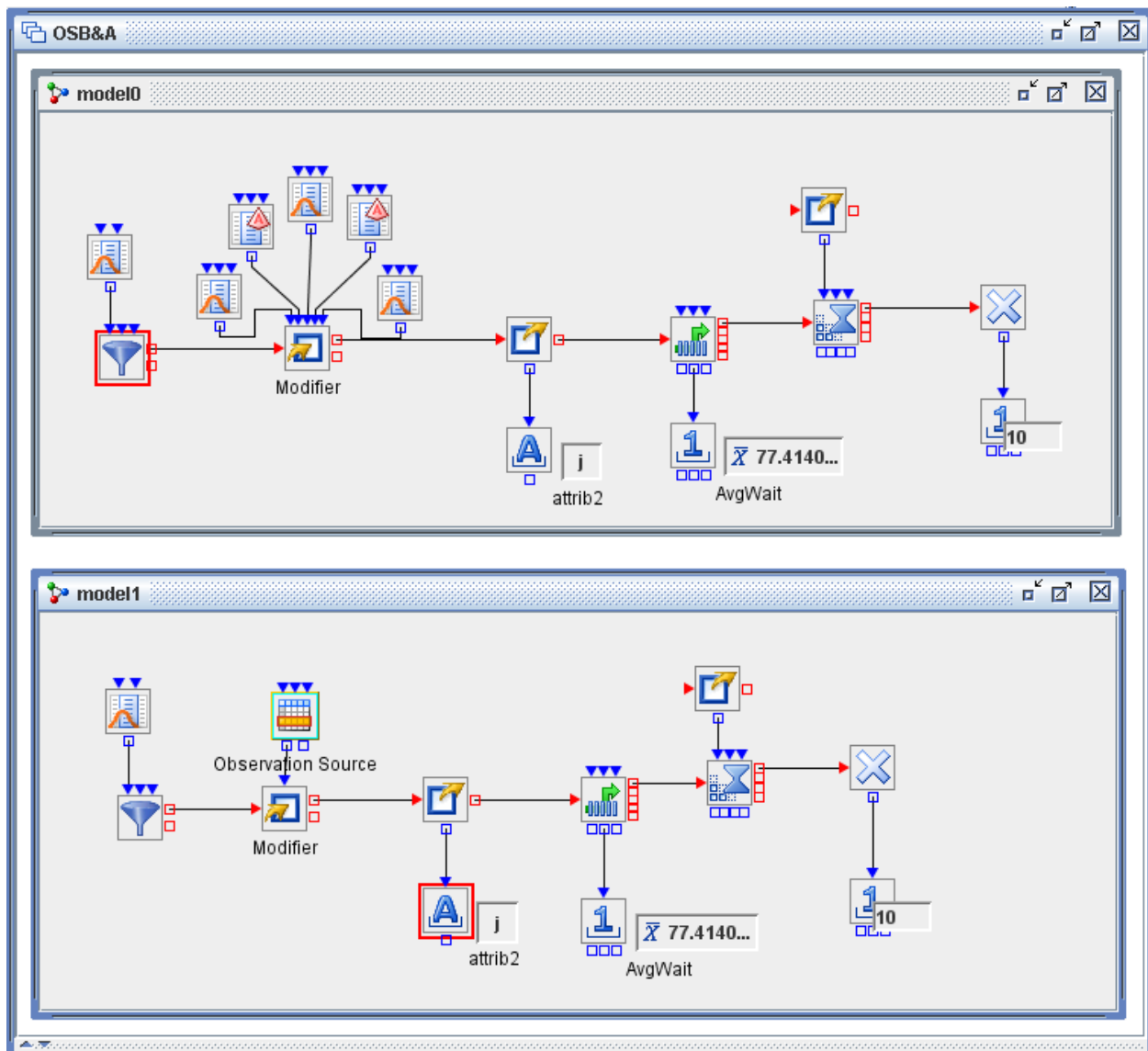
Each attribute is read from a separate column in a SAS data set using Numeric Source and Text Source blocks. After the attributes are set, the value of `attrib2` is extracted and sent to a String Holder block. The entity then waits in a queue for the Server to become available. The service time for each entity is the value of the attribute `attrib5`. The value of `attrib5` is extracted using an Extractor block, which is connected to the InServiceTime port of the Server block. When an Extractor block is used in this way, it does not require connections to its InEntity and OutEntity ports. After being serviced, the entity leaves the system.

`model1` in Figure E.42 is the same model as `model0` except that an Observation Source block is used to read in the attribute values from the same data set as is used in `model0`. In the Modifier block in `model1`, you select **An Observation Value Input** for the **Assigned from** option and you select **Full Assignment**, as shown in Figure E.43. This means that as each entity enters the Modifier block, an entire row from the SAS data set is read in and each column is assigned as an attribute on the entity (using the column names in the data set for the attribute names). This eliminates the need to use five separate Numeric Source and Text Source blocks to read in the attribute values.

Figure E.43 Modifier Block Dialog Box
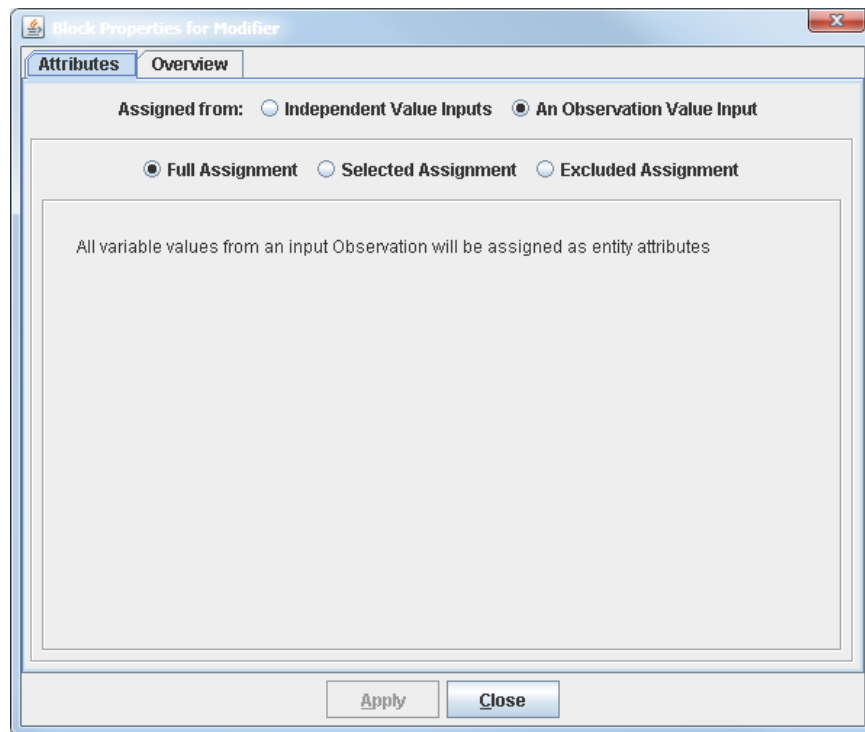
Using the Dataset Writer Block to Save Data during a Run

This simple queueing example demonstrates the Dataset Writer block functionality. You can find the project files for this example in the folder `\projects\examples\docDatasetWriter` in your Simulation Studio installation directory.

In Figure E.44, the Entity Generator block in the upper left corner creates entities every two minutes and sends them to a Queue block where they wait for a server to become available. After being serviced, each

entity passes through a Bucket block where the attributes Time and Age are collected. Then the entity leaves the system.

Every five minutes, the Entity Generator block in the lower left creates an entity. The entity is sent to a Gate block where first a Boolean signal (with value true) is sent to the InSaveNow port of the Dataset Writer block. Once the true Boolean signal arrives at the InSaveNow port, it is used as a signal to save the contents of the data model currently provided to the Dataset Writer block through the InData port. In this example, the InData port is connected to the OutData port of the Bucket, so the information collected up to that point by the Bucket is saved. Since the InPolicy port of the Dataset Writer block has a connection, the location of the saved data is set dynamically. The following string expression is passed from the Formula block to the InPolicy port of the Dataset Writer block to set the location: `concat("result",toString(timeNow()),".sas7bdat")`. For example, the first data set saved is named `result5.sas7bdat`.

After the entity generated by the second Entity Generator block signals the Dataset Writer block to save the Bucket data, another Boolean true signal is generated by the Gate block and sent to the InClearData port of the Bucket. This signal clears all data that have been collected by the Bucket block up to that time during the simulation execution. So the first data set saved by the Dataset Writer block contains the data collected by the Bucket between time 0 and time 5, and similarly the second data set saved contains the data collected by the Bucket between time 5 and time 10.

Figure E.44 Dataset Writer Block Example
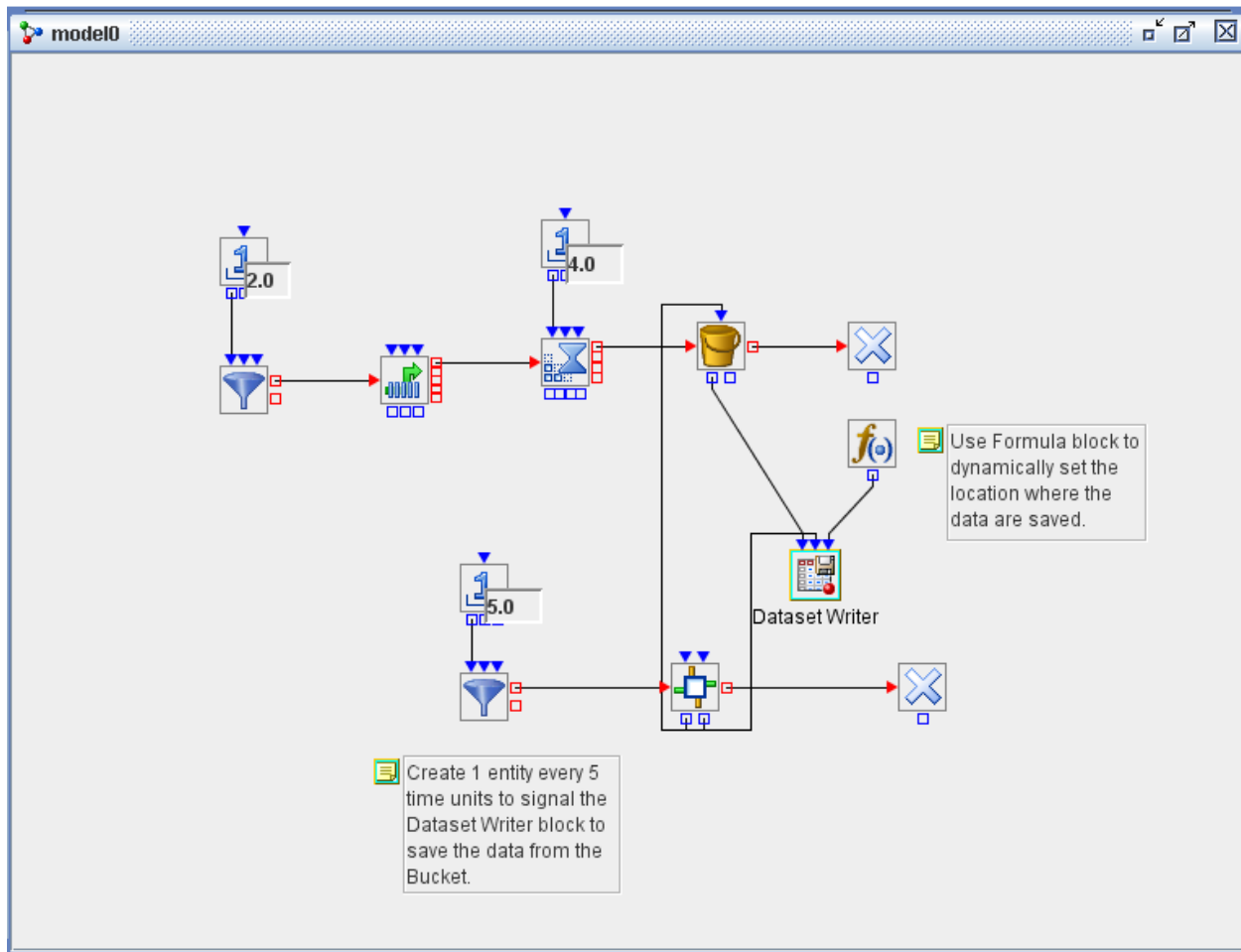
Using the Steady State Block for Output Analysis

This example demonstrates the functionality of the Steady State block, which is located in the Output Analysis template. You can find the project files for this example in the folder `\projects\examples\docSteadyState` in your Simulation Studio installation directory.

Parts arrive at a repair facility at a rate of one part per hour and wait in a queue to be serviced by a machine that processes them at a rate of 1.25 parts per hour, or 48 minutes per part. Suppose this processing facility never shuts down and you are interested in estimating the long-run average time that parts wait to be repaired. In Figure E.45, the OutWait port of the Queue block is connected to the InValue port of the Steady State block. The InValue port of the Steady State block is used for individual data values (such as waiting time values). The Steady State block can also take a data model object as input through its InData port (such as a data model collected by a Number Holder block).

Figure E.45 Steady State Block Example for Observation-Based Statistics
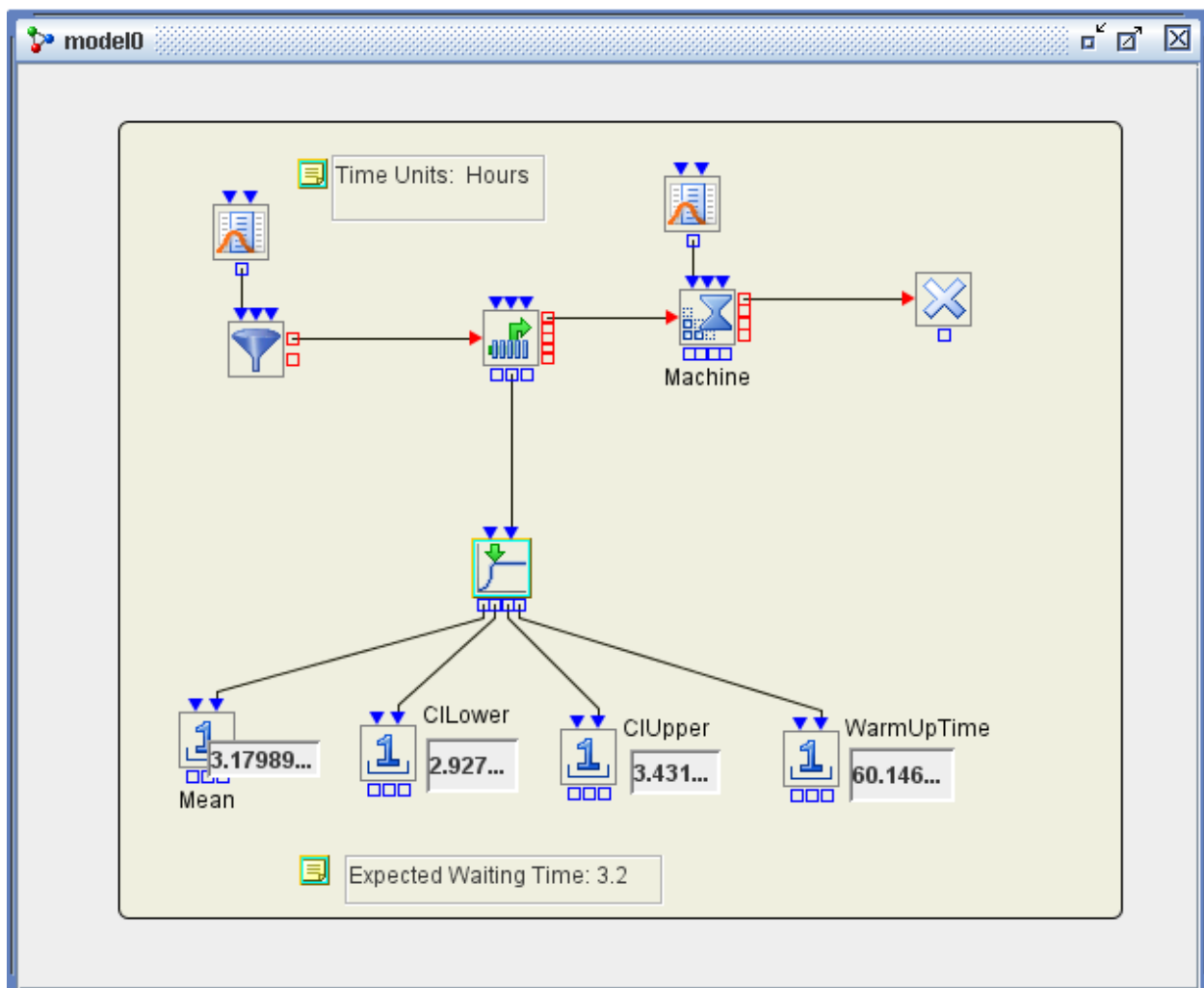
Figure E.46 shows the Block Properties dialog box for the Steady State block. Because the InValue port is used to pass individual data values, you do not need to specify a data model variable name. A precision value of 0.15 is specified, and the **Relative Precision** option is selected. This means that the Steady State block will return a confidence interval that is within ±15% of the estimated mean. The beta value is 0.10, indicating that you want a 90% confidence interval for the average waiting time. The option **Time Persistent** is not selected because waiting time is an observation-based statistic. For more information about observation-based and time-persistent statistics, see the section "Output Analysis" on page 128.

Figure E.46 Steady State Block Dialog Box for Observation-Based Statistics
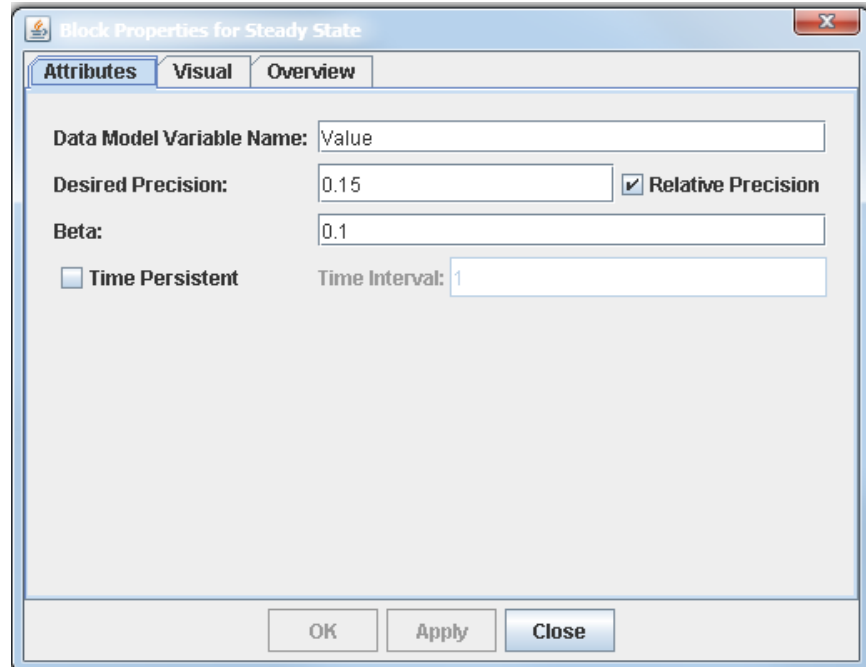
When the simulation is executed, the Steady State block controls the run length, and the simulation terminates as soon as the Steady State block delivers a confidence interval that satisfies the specified precision requirement. At that point, the estimated mean and the lower and upper limits of the confidence interval are returned. The estimated warm-up time is also returned at the point during the run at which the algorithm detects the value. For this example in steady state operation, the expected waiting time is 3.2 time units. In Figure E.45, the confidence interval (2.93, 3.43) that is returned by the Steady State block covers this value. The estimated mean waiting time is 3.18, so the delivered confidence interval has an actual precision of about 7.9%. This is less than the requested precision level of 15%.

Suppose you are now interested in computing the long-run average number of parts in the system. Figure E.47 shows the same model as in Figure E.45, except two Formula blocks and two Gate blocks have been added to calculate and update the number of parts in the system. When a part arrives at the first Gate block, the Gate block triggers the execution of the expression in the first Formula block, which adds 1 to the current value in the Number Holder block labeled NumberInSystem. When the entity is ready to leave the system, it flows through a second Gate block and triggers the execution of the expression in the second Formula block. The second Formula block subtracts 1 from the current value in the NumberInSystem Number Holder block. The Gate blocks are connected to the Number Holder block by using value Connectors.

Figure E.47 Steady State Block Example for Time-Persistent Statistics
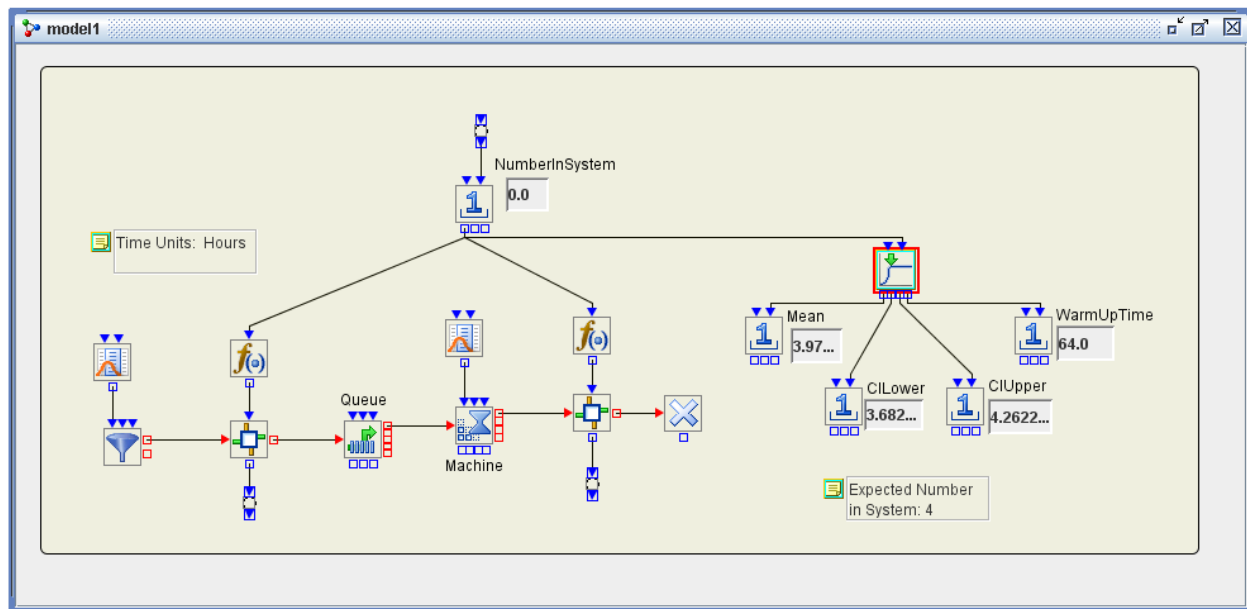
In the Steady State block properties dialog box in Figure E.48, the option **Time Persistent** is selected, and the **Time Interval** value is 1. This means that the computation of the time-averaged number in system statistic occurs every 1 time unit during the simulation run. The time-averaged values are then used in the spaced batch means algorithm for computing the confidence interval. In steady state operation, the expected number of parts in the system for this example is four. The confidence interval (3.682, 4.262) that is returned by the Steady State block covers this value.

Figure E.48 Steady State Block Dialog Box for Time-Persistent Statistics

Updating an Input Distribution

In certain scenarios, you might want to change a particular input distribution at the start of a simulation run. Other times, you might want to change a distribution dynamically during the simulation run. This example demonstrates both of these possibilities. You can find the project files for this example in the folder `\projects\examples\docUpdateDistribution` in your Simulation Studio installation directory.

The easiest way to change the parameters for a distribution at the beginning of the run is to use the DataStreamDescription factor for the Numeric Source block, as demonstrated in the model in Figure E.49. In this example, two processing times are randomly sampled for each entity and are saved as its attributes. These processing times are used downstream in the model as service times in two different Server blocks.

To explore different processing time distributions, you first define two factors. One factor, called ExponentialMean, is anchored to the DataStreamDescription factor candidate for the block labeled Numeric Source1. Assume that the distribution type for the first processing time is fixed as Exponential and only the mean of the distribution will vary. The block Numeric Source1 has its distribution type defined as Exponential in its Block Properties dialog box. After including the ExponentialMean factor in the Experiment window, you can vary the mean of the Exponential distribution for the Numeric Source1 block for different runs of the model in the Experiment window.

Figure E.49 Using Factors to Change an Input Distribution
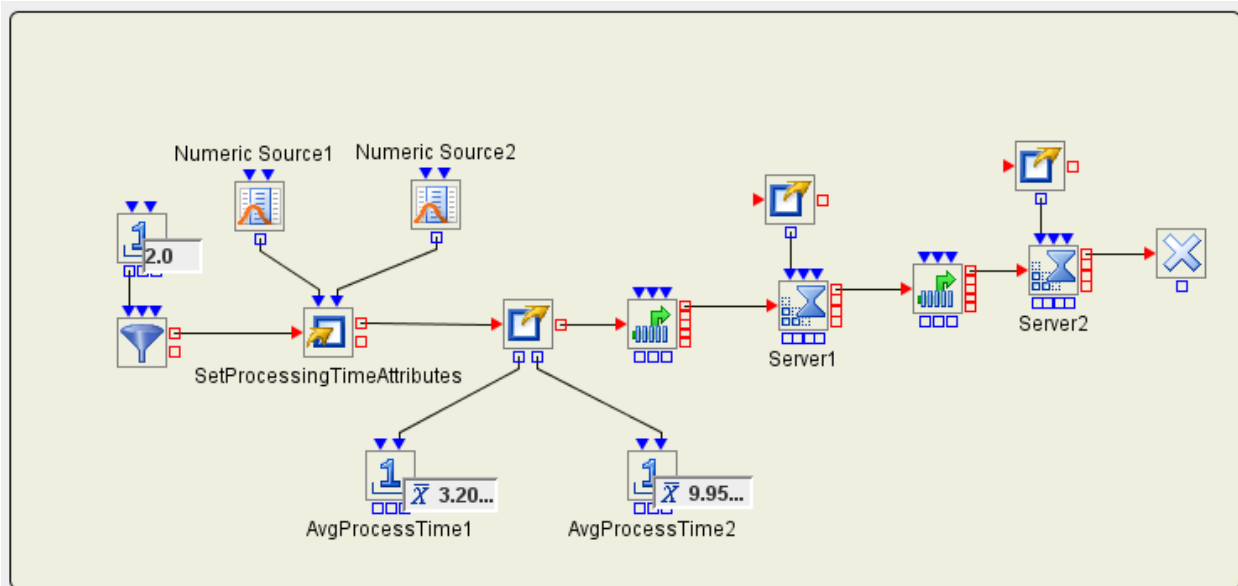
The second factor, called ProcessTime2, is anchored to the DataStreamDescription factor candidate for the block labeled Numeric Source2. For the second processing time, suppose you are interested in varying the type of its distribution. In this case, it does not matter what distribution type is currently specified in the Numeric Source2 Block Properties dialog box. The factor ProcessTime2 is used to set the distribution type and parameters in the Experiment window at the start of the run.

Figure E.50 shows the Anchors dialog box for the model, and Figure E.51 shows the Experiment window with the processing time factors included. Two responses have also been created for the average processing times. Each response is anchored to the MeanValue candidate of the corresponding Number Holder block. For the first design point in the Experiment window, the mean of the Exponential distribution for the block Numeric Source1 is set to 3, and the distribution for the block Numeric Source2 is set to Normal with a mean of 10 and a standard deviation of 1. The second design point specifies different values for each of these settings. The format for a DataStreamDescription factor is the same as for a value that is supplied to the InStreamPolicy port of the Numeric Source block. For more information about the format of the DataStreamDescription factor candidate, see the Numeric Source block overview in Appendix A, "Templates."

Figure E.50 Anchors Dialog Box for Updating Distributions Using Factors
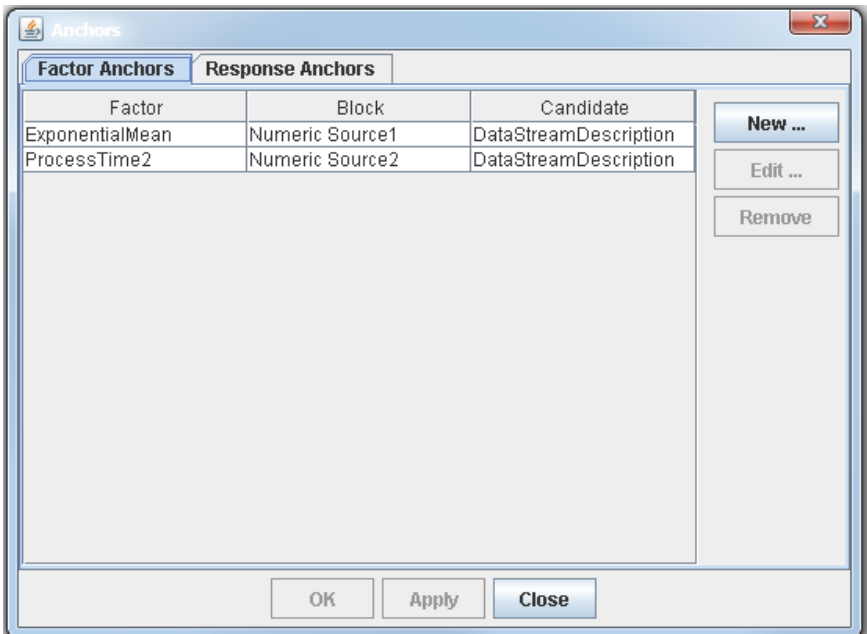
Figure E.51 Experiment Window for Updating Distributions Using Factors
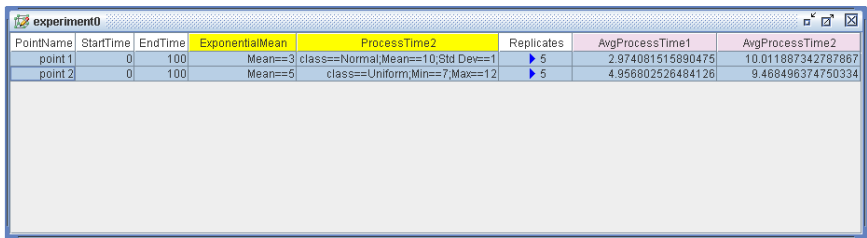
The following example demonstrates how the InStreamPolicy port of a Numeric Source block can be used to dynamically change a distribution during a run rather than at the beginning. In the model in Figure E.52, each entity samples a processing time from a different normal distribution. To accomplish this, an entity first randomly samples mean and standard deviation values and assigns those values as its attributes. Because the InStreamPolicy port of a Numeric Source block requires a string input, a Formula block is used to concatenate the mean and standard deviation values into a single string that the InStreamPolicy port can process.

Figure E.52 Dynamically Updating a Distribution during the Simulation Run
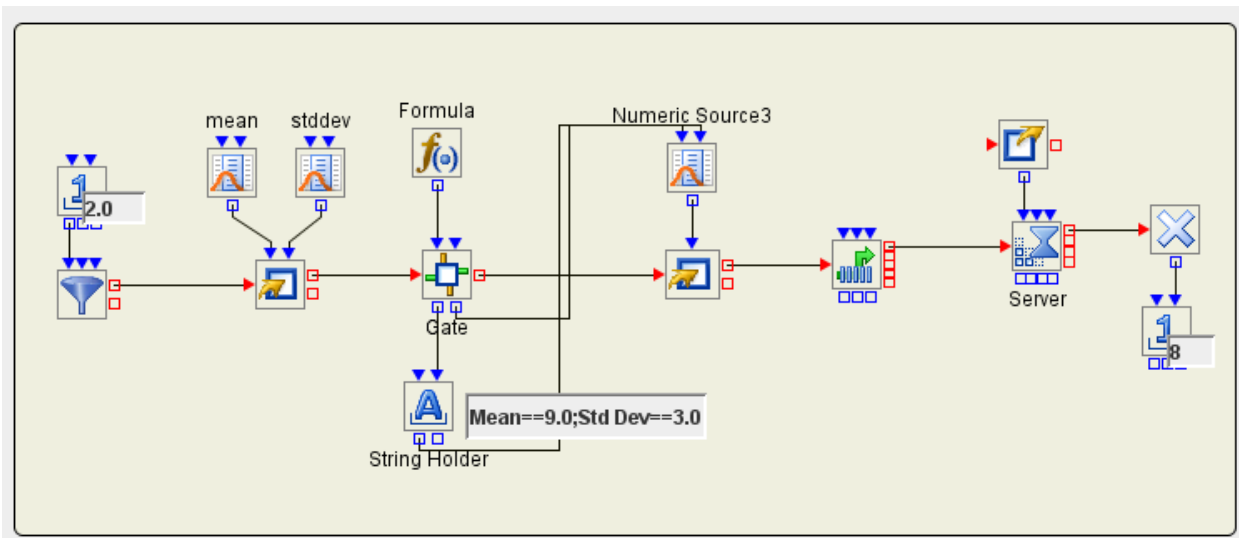
Figure E.53 shows the format of the expression used in the Formula block to generate the string for the InStreamPolicy port. As shown in Figure E.52, a Gate block triggers the evaluation of the expression in the Formula block, and the resulting string is then passed to a String Holder block, where it is held until needed.

**Figure E.53** Formula Block Properties Dialog Box for Generating the InStreamPolicy Port String
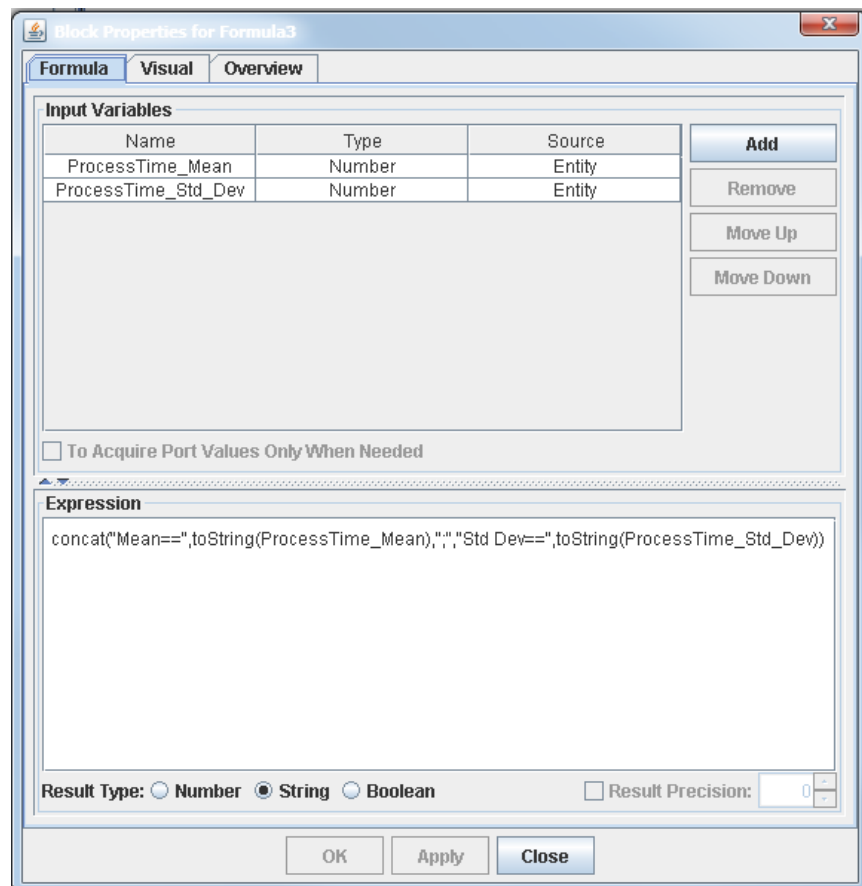
Next, the Gate block generates a `'true'` Boolean signal. This signal is passed to the InUpdate port of the Numeric Source block labeled Numeric Source3. When the Numeric Source block receives a `'true'` Boolean value in its InUpdate port, it pulls the value from its InStreamPolicy port, which in this example is connected to the OutData port of the String Holder block. The string that is held by the String Holder block is passed to the block labeled Numeric Source3, updating the parameters of the Normal distribution. Note that the order of actions in the Gate block is important because the string value that is pulled by the InStreamPolicy port must be available before the `'true'` Boolean value is passed to the InUpdate port of the Numeric Source block.

Extended Queueing Options

In this example, there are five separate models that illustrate the extended queueing controls in the Queue block. You can find the project files for this example in the folder `\projects\examples\docQueueOptions` in your Simulation Studio installation directory.

In the docQueueOptions project, model1 demonstrates the use of the **Nonblocking Pull Depth** option, as shown in Figure E.54. At time 0, two entities are created: one with attribute type 1 and one with attribute type 2. Both entities are routed to a Queue block, where the type 1 entity enters the queue first. Also at time 0, two resource entities are created, and an attribute called Specialty is assigned with value 2 for both entities. The resource entities are routed to a Resource Pool block. All entities are created at time 0, but the Entity Generators were added to the model so that type 1 entities are created first, then type 2 entities, and finally the resource entities. By default, when multiple events are scheduled at the same simulation time, precedence on the event calendar is given to the block that was created first. Block creation order is one way to control the precedence of block events. For more information about setting the rank order of blocks in a simulation model, see the section "Block Ranking" on page 67.

Figure E.54 Using the Nonblocking Pull Depth Queueing Option
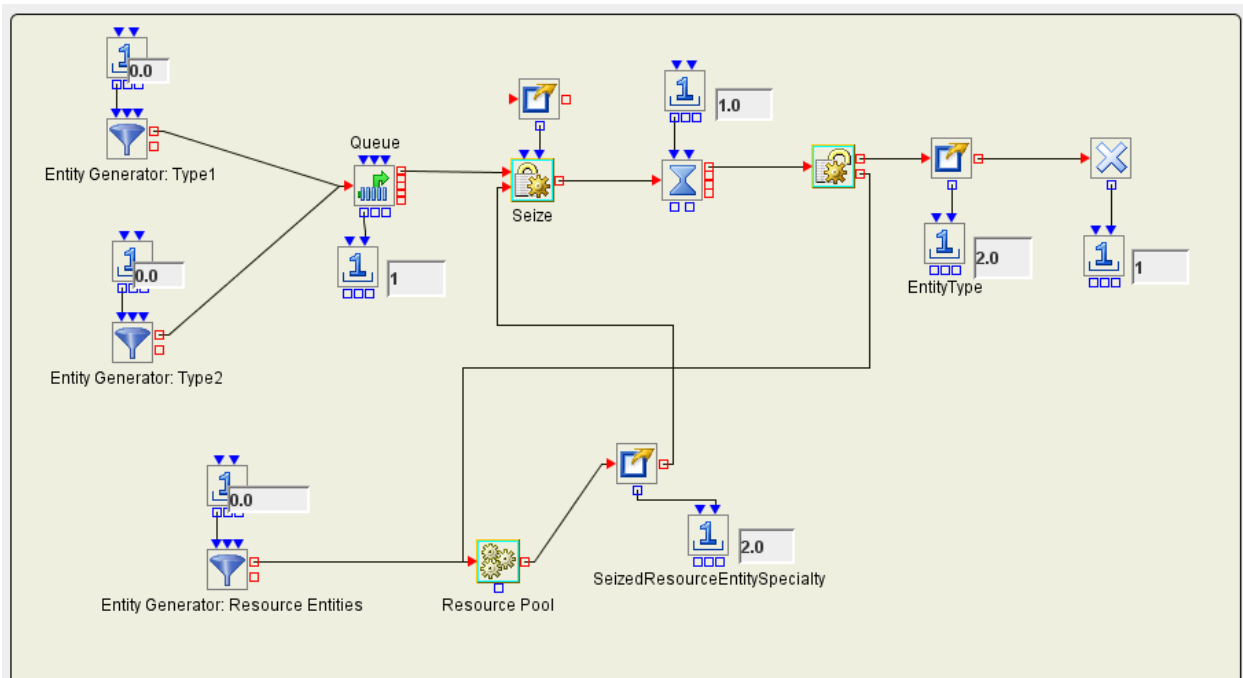
When the resource entities enter the Resource Pool block, the Seize block attempts to pull an entity from the Queue block that satisfies the attribute rule Specialty==Type. Because the first entity in the Queue block is type 1, it will wait because there is no resource entity with Specialty=1. In the Queue block, the **Nonblocking Pull Depth** option is set to a relative value of 1, as shown in Figure E.55. This means that any block attempting to pull an entity from the Queue block triggers a checking of 100% of the capacity of the queue for a match. In this case, the second entity in the queue is then checked to see whether it satisfies the attribute rule specified in the Seize block. Because the second entity is type 2, it seizes one of the resource entities with Specialty=2, proceeds to the Delay block, and exits the system. At the end of the simulation run, the type 1 entity remains in the Queue block. Note that setting the **Nonblocking Pull Depth** option to an absolute value of 2 would produce the same result (because there can never be more than two entities in the queue).

Figure E.55 Nonblocking Pull Depth Option in the Queue Block Properties Dialog Box
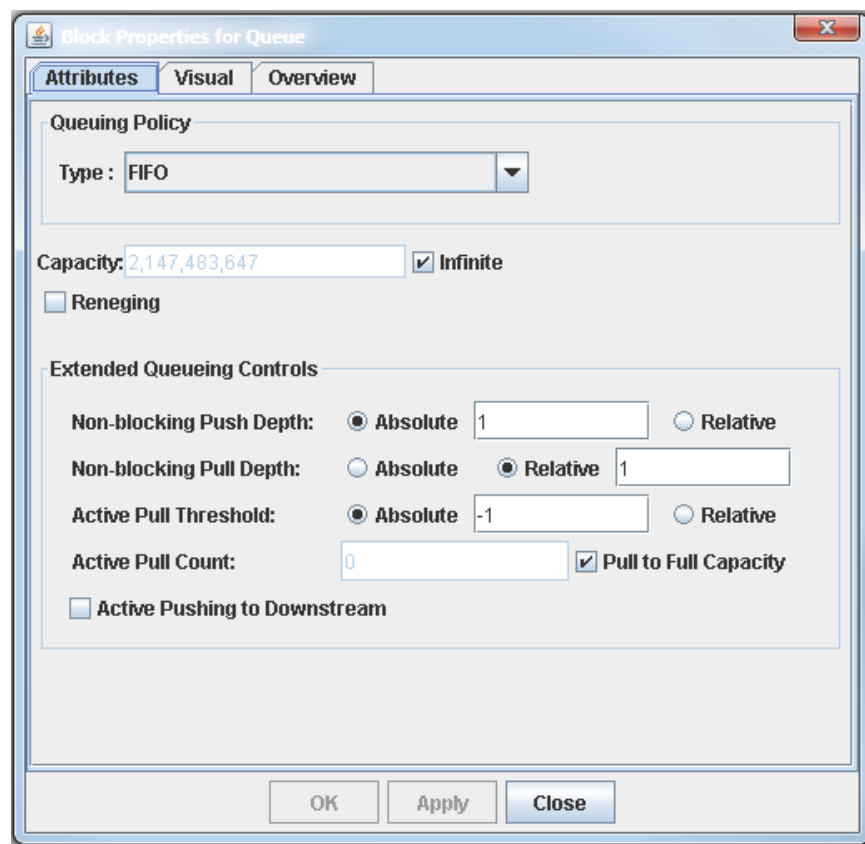
In this example, the **Nonblocking Pull Depth** option is used to allow other blocks to pull entities from the queue other than the one at the front. If the nonblocking pull depth was left at the default absolute value of 1, then no entities would leave the Queue block. The type 2 entity would be blocked by the type 1 entity because an absolute nonblocking pull depth of 1 means that only the first entity in the Queue block is checked when a downstream block attempts to pull an entity from the Queue block.

The second model in the docQueueOptions project, labeled model2 and shown in Figure E.56, demonstrates the use of the **Nonblocking Push Depth** option. One type 1 entity is created at time 0 and routed to the Queue block. Also at time 0, two resource entities with attribute Specialty=2 are created and routed to the Resource Pool block. The type 1 entity will wait in the Queue block because it does not satisfy the attribute rule Specialty==Type that is specified in the Seize block. At time 1, one type 2 entity is generated and routed to the Queue block. When the type 2 entity arrives at the Queue block, the block attempts to push an entity downstream. The **Nonblocking Push Depth** option in the Queue block has a relative value of 1 as shown in Figure E.57, which means that up to 100% of the queue capacity is checked when the Queue block attempts to push an entity out. (Note: Specifying an absolute nonblocking push depth of 2 would have the same effect.) Because the first entity in the Queue block does not satisfy the attribute rule for the downstream Seize block, the second entity is checked. The second entity does satisfy the Seize attribute rule, so the Queue block pushes that entity downstream to the Seize block. If the default absolute value of 1 was used for the nonblocking push depth in the Queue block, then the second incoming entity would not be pushed out of the Queue block at time 1, and it would wait in the Queue block behind the first entity. The second entity would be blocked by the first entity, and the simulation would end with two entities in the Queue block.

Figure E.56 Using the Nonblocking Push Depth Queueing Option
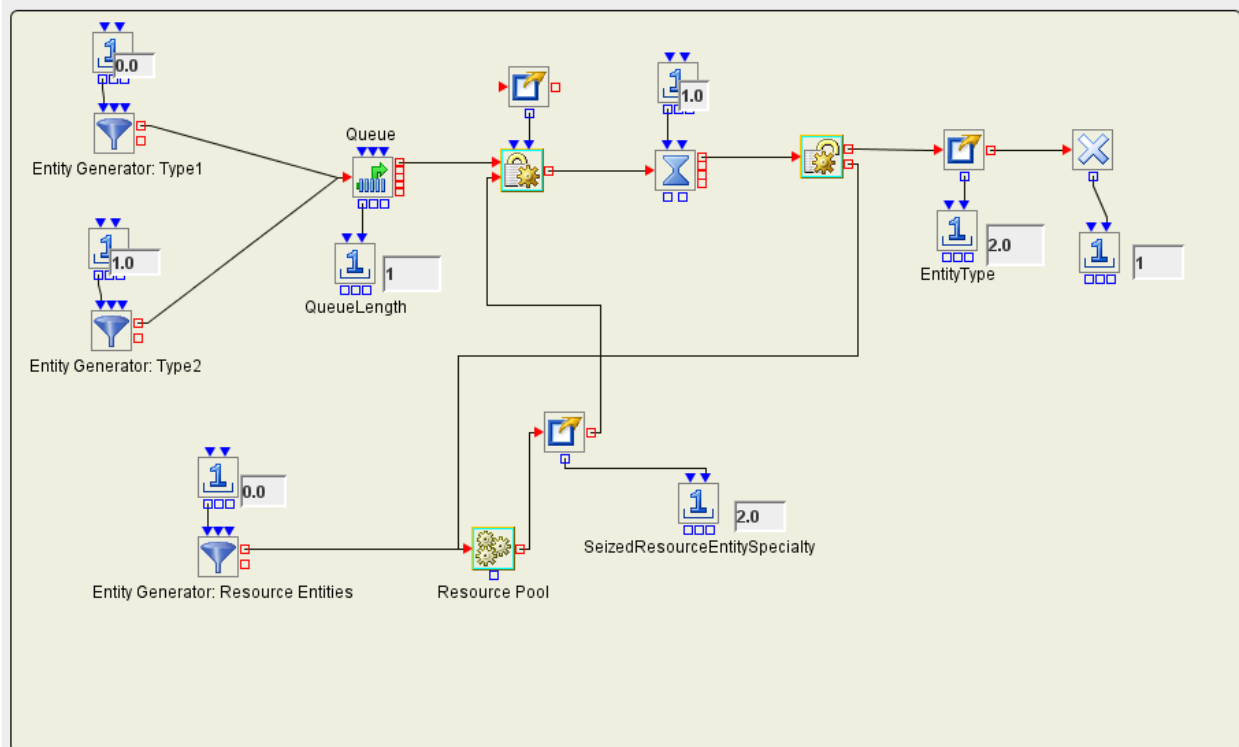
Figure E.57 Nonblocking Push Depth Option in the Queue Block Properties Dialog Box
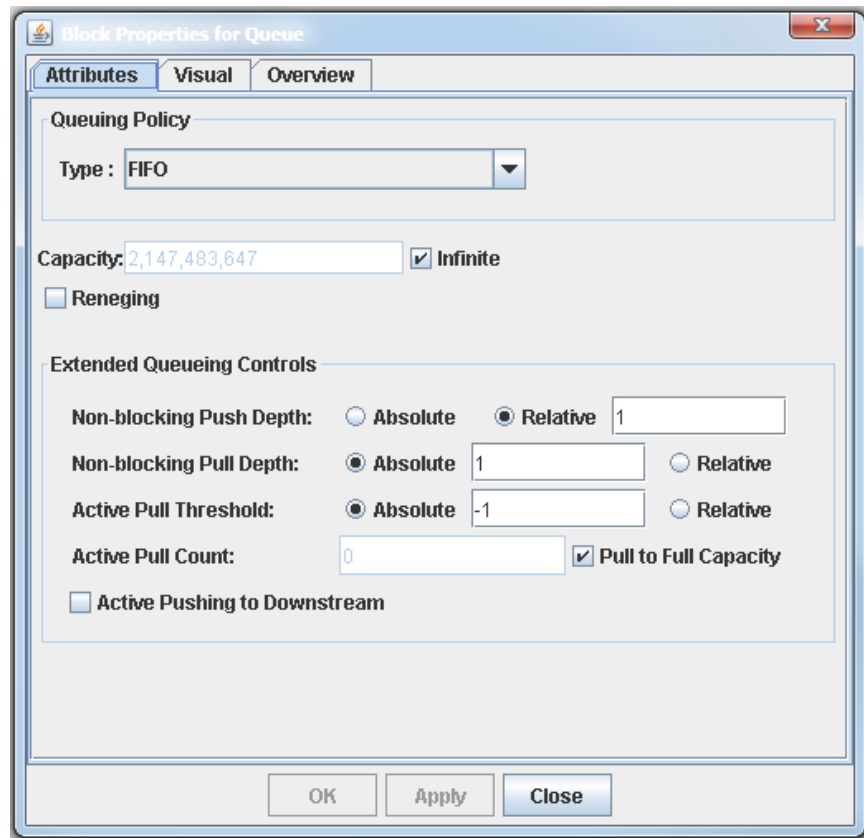
The model labeled model3 in the docQueueOptions project demonstrates the **Nonblocking Push Depth** and **Nonblocking Pull Depth** options together. In this model, shown in Figure E.58, 25 type 1 entities and 25 type 2 entities are generated with exponential interarrival times. At time 0, two resource entities are generated: one resource entity with attribute Specialty=1 and one resource entity with attribute Specialty=2. In the Queue block, both the nonblocking push depth and the nonblocking pull depth have a relative value of 1, as shown in Figure E.59. As type 1 and type 2 entities enter the Queue block, the relative nonblocking push depth of 1 (100% of the queue capacity) ensures that an incoming entity to the Queue block will not be blocked by an entity of a different type that is waiting for its resource entity. Furthermore, when a resource entity returns to the Resource Pool after being released by another entity, a relative nonblocking pull depth of 1 in the Queue block connected to the downstream Seize block ensures that all entities in the queue will be checked for a match to the returning resource entity (not just the entity at the front of the queue). Using both the **Nonblocking Pull Depth** and the **Nonblocking Push Depth** options in the Queue block allows entities to leave the Queue block as soon as their associated resource entity is available for seizing.

Figure E.58 Modeling a Nonblocking Queue
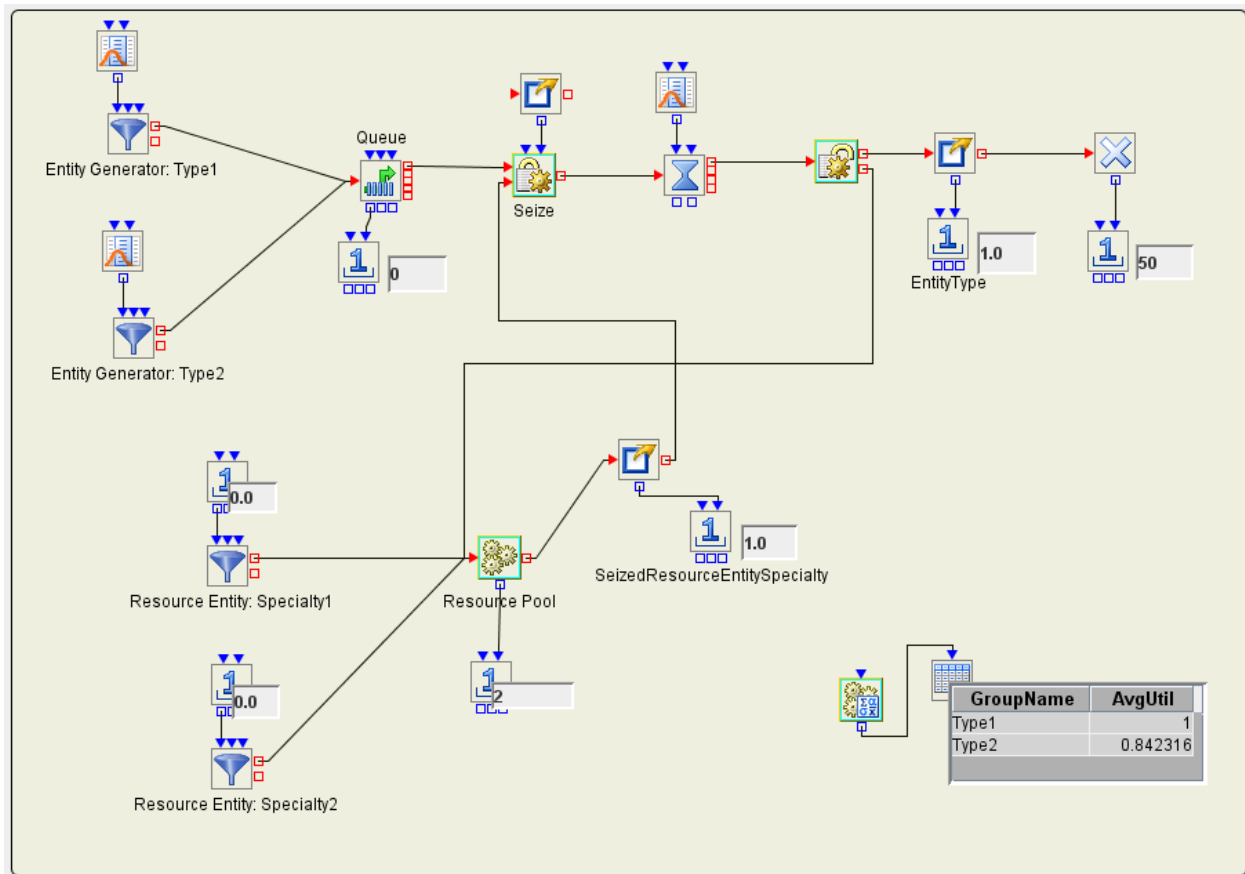
Figure E.59 Queue Block Properties Dialog Box for a Nonblocking Queue


If you run model3 with the default extended queueing options (the **Nonblocking Push Depth** and the **Nonblocking Pull Depth** options both have an absolute value of 1), then you will notice that the average utilization of the resource entity with the attribute Specialty=1 drops from 1.0 to 0.69 and the average utilization of the resource entity with Specialty=2 drops from 0.84 to 0.49. Also, the simulation run length increases from 30.38 time units to 47.58 time units. The lower resource utilizations and the longer simulation time are a result of entities being blocked in the Queue block, even when their matching resource is available. For example, a type 2 entity arrives at the Queue block at time 1.42 and waits behind two type 1 entities. At time 2.69, the resource entity with Specialty=2 is released and returns to the Resource Pool block. However, the type 2 entity in the Queue block does not seize the waiting Specialty=2 resource entity, because the nonblocking pull depth has a value of 1, which means that only the first entity in the queue is checked for a match. Because the first entity in the Queue block is type 1, the resource entity with Specialty=2 waits in the Resource Pool until the type 2 entity in the Queue block reaches the front of the queue and another entity arrives at the Queue block to trigger the block to attempt to push an entity downstream, which does not happen until time 6.3.

Model4 demonstrates the functionality of the **Active Pushing to Downstream** option in the Queue block, as shown in Figure E.60. At time 0, three entities are created and sent into a Queue block. There is a Valve block that is initially set to **Closed** downstream from the Queue block. At time five, another entity is generated and routed to a Gate block. In the Gate block, one action is defined that sends a `'true'` Boolean signal to the InSignal and the InFlowTrigger ports of the Valve block. This action changes the state of the Valve block from closed to open and sends out a message to blocks connected to the Valve block that its state has changed. With the **Active Pushing to Downstream** option selected in the Queue block as shown in Figure E.61, the Queue block pushes out the first entity and then attempts to push out as many entities as it can. Because downstream from the Queue block is a Disposer block (which can always take an entity from upstream), all three entities are pushed out of the Queue block. If the **Active Pushing to Downstream** option is not selected in the Queue block, then only one entity is pushed out of the Queue block when the InFlowTrigger port of the Valve block receives the `'true'` Boolean value.

Figure E.60 Using the Active Pushing to Downstream Queueing Option
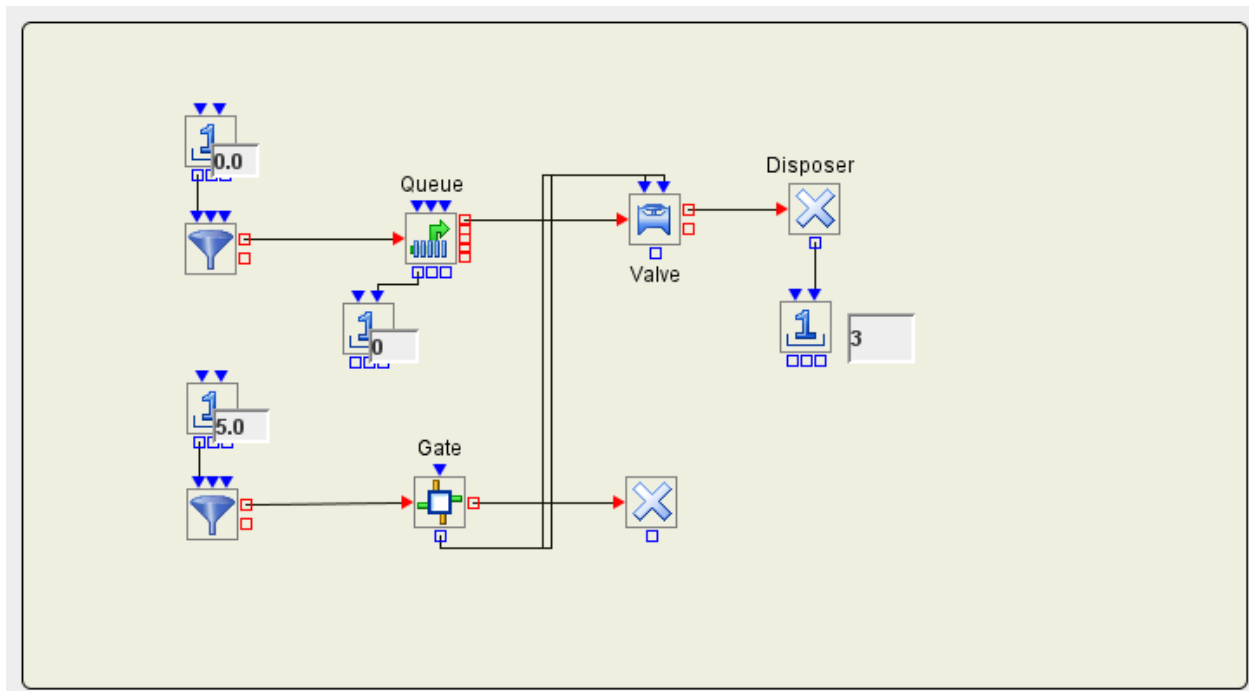
Figure E.61 Active Pushing to Downstream Option in the Queue Block Properties Dialog Box
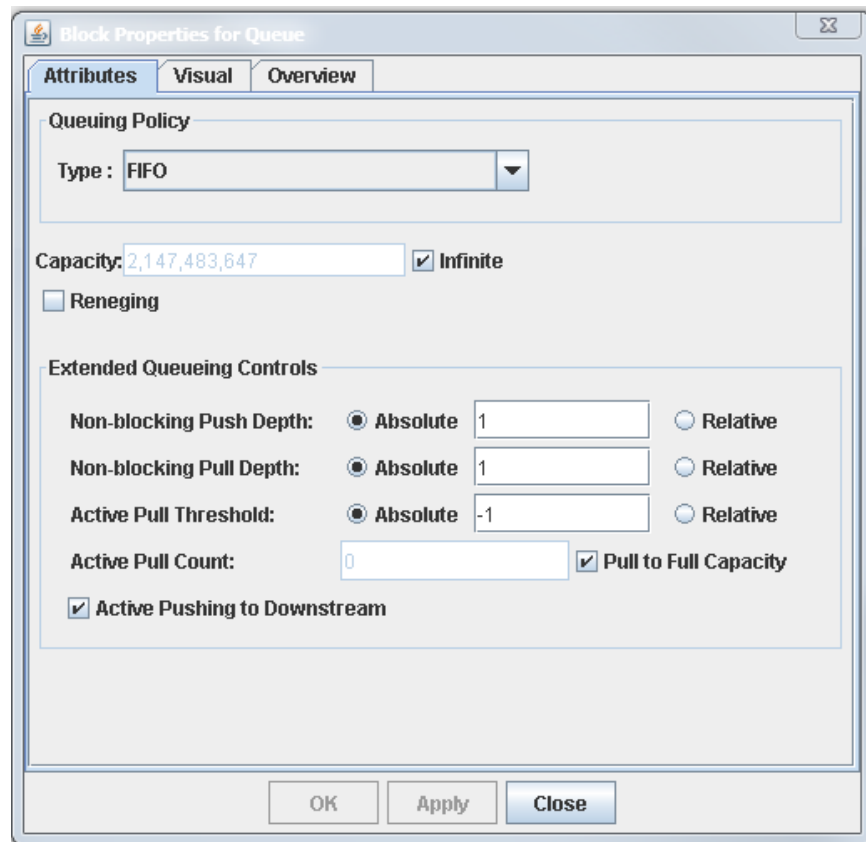
The model labeled model5 demonstrates the **Active Pull Threshold** option, as shown in Figure E.62. This model has two Queue blocks in series, where the Queue block labeled Queue2 has a fixed capacity of 5. The Server block has a capacity of 1. At time 0, 20 entities are created, and the first one flows through the two Queue blocks and starts service at the Server block. The next five entities wait in Queue2, and the remaining 14 entities wait in Queue1. The **Active Pull Threshold** option in the Queue2 block has an absolute value of –4, as shown in Figure E.63. This means that Queue2 will not actively pull entities from upstream until its length falls to 5–4=1. The **Pull to Full Capacity** option is also selected in Queue2, which means that when Queue2 reaches a length of 1 and pulls from upstream, it will pull as many entities as it can until the length of Queue2 reaches its capacity of 5. At time 12, the length of Queue2 falls to 1, and subsequently four entities are pulled from Queue1.

Figure E.62 Using the Active Pull Threshold Queueing Option
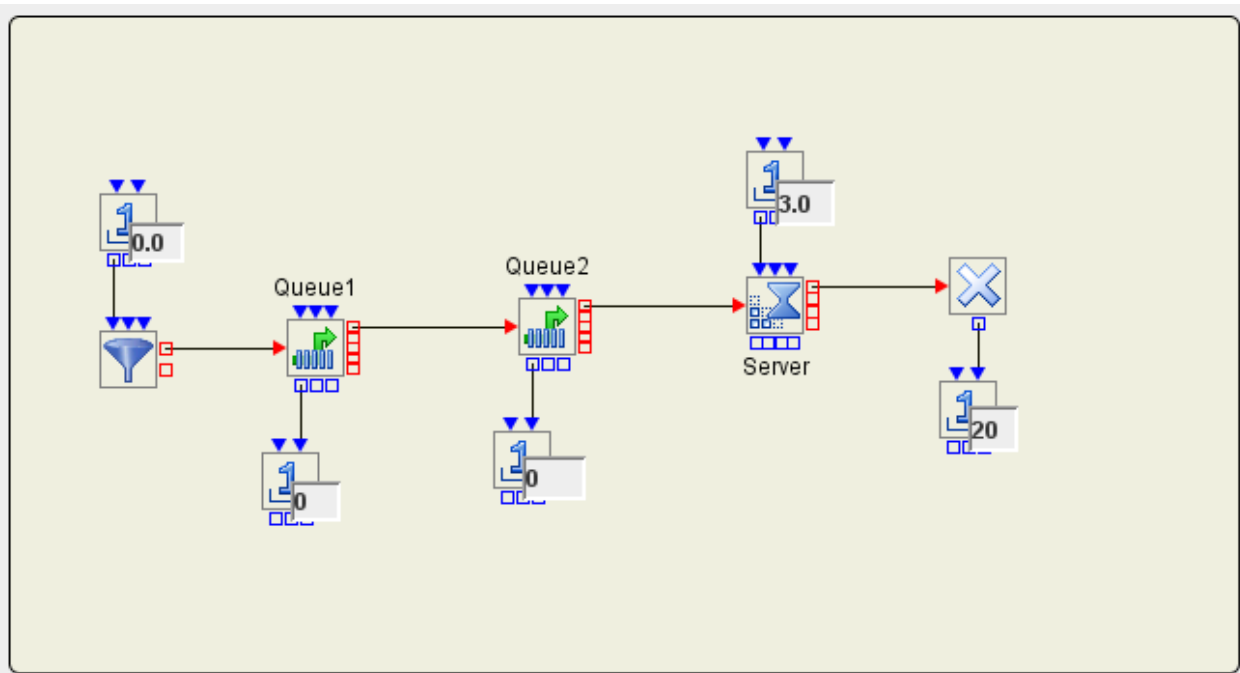
Figure E.63 Active Pull Threshold Option in the Queue Block Properties Dialog Box
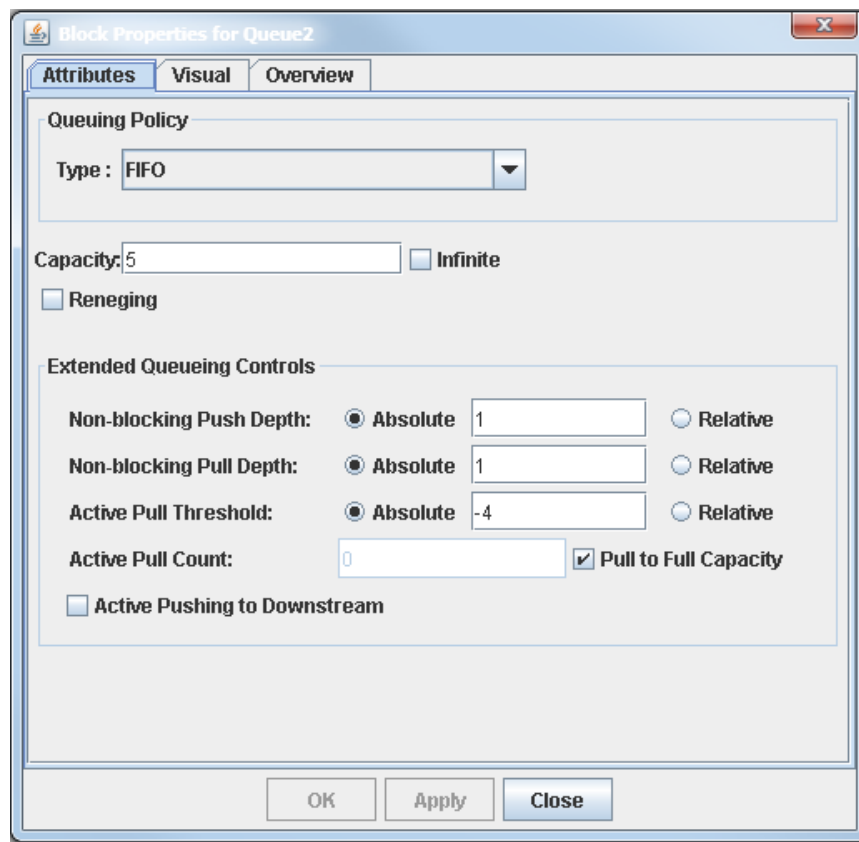
Using the Resource Schedule Entity

This example illustrates resource scheduling capabilities, including the use of the resource schedule entity type to dynamically create entries in the Resource Scheduler block. You can find the project files for this example in the folder `\projects\examples\docSchedEntity` in your Simulation Studio installation directory.

At a particular candy manufacturing plant, batches of candy arrive at a final coating process according to a uniform distribution with a minimum of 8 minutes and a maximum of 10 minutes. The batches are held in a single queue that feeds two identical coating pans. The coating time for a batch of candy is uniformly distributed with a minimum of 12 minutes and a maximum of 15 minutes. Each coating pan is subject to random failures. The time until failure for a coating pan is exponential with a mean of 5 days. The time to repair the coating machine after a failure occurs is uniformly distributed with a minimum of 12 hours and a maximum of 24 hours. Suppose the manufacturing facility operates 24 hours a day, 7 days a week, and you are interested in studying the output from this system over a six-month period (302,500 minutes). Figure E.64 shows the model for this system (corresponding to `model0` and `experiment0` in the `docSchedEntity` project).

Figure E.64 Using the Resource Scheduler Entity in a Candy Manufacturing Example
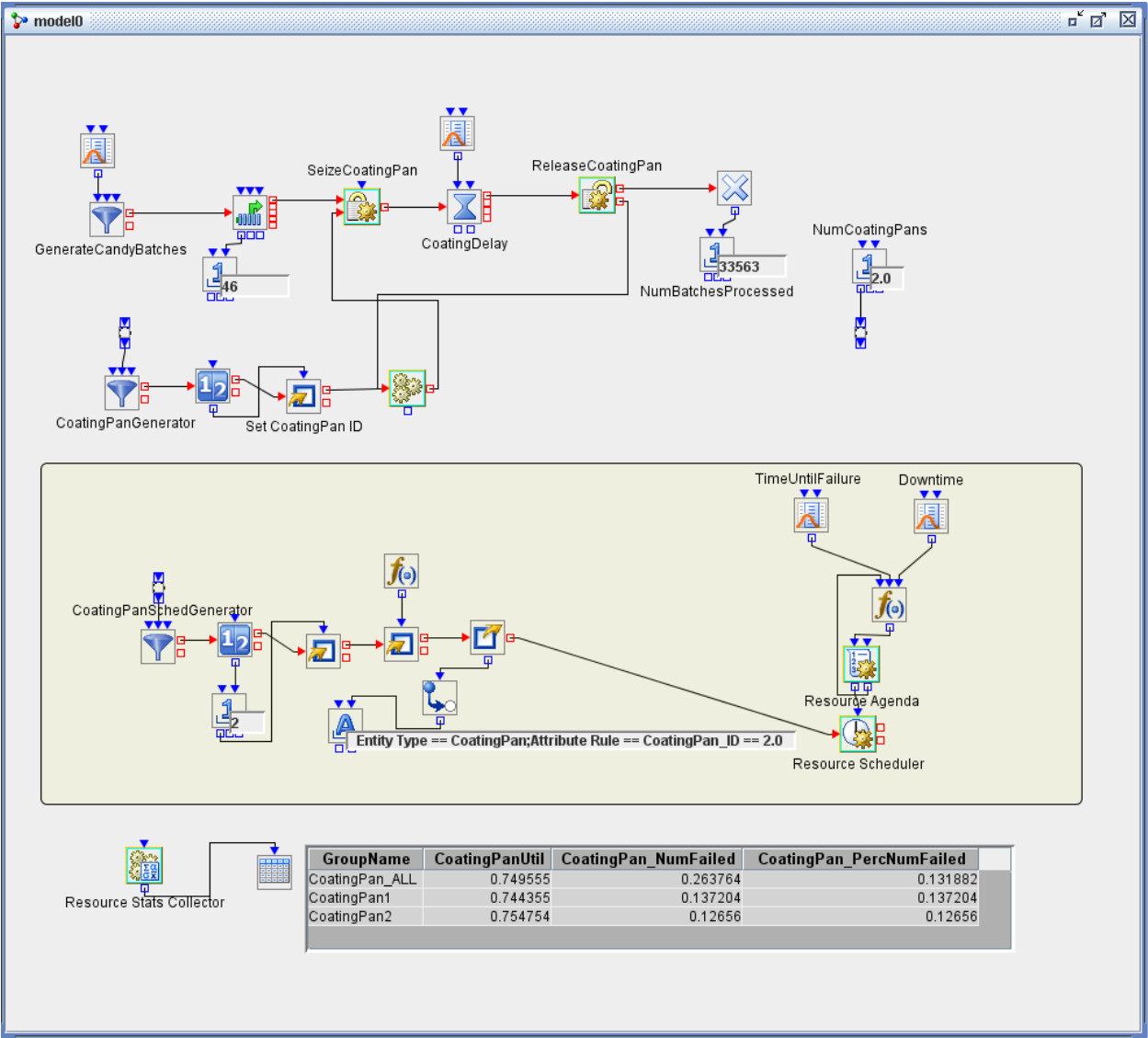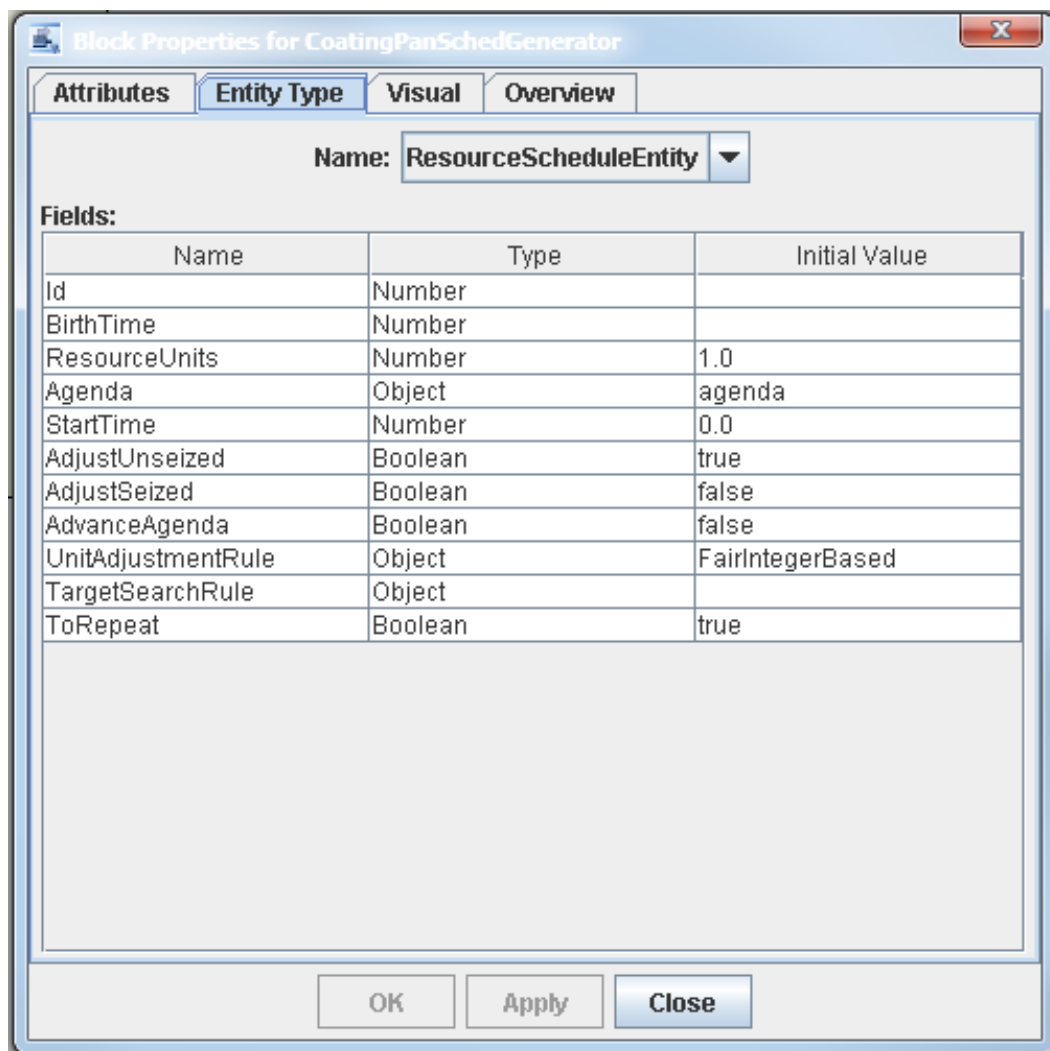

At the start of the simulation, the Number Holder block labeled NumCoatingPans is used to set the batch size for the Entity Generator block labeled CoatingPanGenerator that generates resource entities of the type CoatingPan. The CoatingPan resource entity type has an attribute called CoatingPan_ID, as defined in the Entity Types dialog box shown in Figure E.65, that is set using a Counter block. As batches of candy arrive, they wait in a queue for an available coating pan. After seizing a coating pan resource entity, the candy batch is delayed for the length of the coating time. After releasing the coating pan resource entity, the candy batch exits the system.

To model the failure of the coating pans, the same number of resource schedule entities as the number of coating pan resource entities are generated at time 0. The Entity Generator block labeled CoatingPanSched-Generator has the entity type specified as the default resource schedule entity, as shown in Figure E.66. Each resource schedule entity is assigned an attribute called SchedCoatingPan_ID that matches the ID of one of the coating pan resource entities (with the value 1 or 2). The default attributes on the resource schedule entities correspond to each field that is required in order to define an appointment in the Resource Scheduler

block. For example, the TargetSearchRule attribute corresponds to the **Entity Type** and **Attribute Rule** fields in the Resource Scheduler Block Properties dialog box. Figure E.66 shows the default attribute values for a resource schedule entity. You update the attribute TargetSearchRule for each resource schedule entity in this example by using a Formula block to generate a string of the form `"Entity Type== <value>;` `Attribute Rule==<value>"`. Here, Entity Type is set to CoatingPan and the Attribute Rule is set to CoatingPan_ID == SchedCoatingPan_ID, as shown in Figure E.67. Note that any resource entity attribute that is used in the Attribute Rule (in this case, CoatingPan_ID) must be predefined in the Entity Types dialog box. Because the attribute TargetSearchRule is of the type object, a Caster block is used to cast the TargetSearchRule value to the type string so it can be displayed using a String Holder block (for debugging purposes). The resource schedule entities are then routed to a Resource Scheduler block.

Figure E.65 Entity Type Dialog Box for the Candy Coating Example

Figure E.66 Default Resource Schedule Entity Attributes

Figure E.67 Updating the Resource Schedule Entity TargetSearchRule Attribute Using a Formula Block

Because you create one resource schedule entity per coating pan resource entity, the model becomes scalable. That is, if the number of coating pans is changed (using the Number Holder block labeled NumCoatingPans), the part of the model that simulates the coating pan failures will not have to be changed. If you do not use the resource schedule entity to dynamically create appointments in the Resource Scheduler block, then you have to manually create one appointment row for each coating pan. For a model with a large number of resource entities that require a schedule, it is more efficient to use resource schedule entities to populate the appointments in the Resource Scheduler block.

When the resource schedule entities enter the Resource Scheduler block, an agenda is activated and processed sequentially for each entity. In this example, both resource schedule entities have the same agenda that alternates the targeted resource entity's state between functional and failed, where the targeted resource entities are the two resource entities of the type CoatingPan. The duration that each coating pan resource entity spends in the functional and failed states is sampled randomly. The InDuration port of the Resource Agenda block is connected to a Formula block that pulls from the appropriate time until failure or downtime distribution, depending on which row the Resource Agenda block is currently processing (as provided by the OutCurrentEntry port on the Resource Agenda block). For more information about resource entities and the Resource Scheduler and Resource Agenda blocks, see Chapter 10, "Resources."

If a coating pan entity happens to fail while it is in the Delay block for the coating time, then it will remain in the Delay block until the coating time ends. When it is released and enters the Resource Pool block, its state will be updated to failed and the Agenda block will schedule the time that it returns to the functional state. So the time that the coating pan resource entity remains in the failed state starts when it is unseized and enters the Resource Pool block. This behavior is dictated by the AdjustSeized and AdvanceAgenda attributes

on the resource schedule entities, as shown in Figure E.66. By default, the attribute AdjustSeized has a value of 'false', which means that the state of any seized resource entity will not be adjusted until it becomes unseized and enters a Resource Pool block. Also by default, the attribute AdvanceAgenda has a value of 'false', which means that the next agenda entry will not be processed until the resource entity state is set to failed. These resource schedule entity attributes can be changed using a Modifier block if necessary. For example, if you want to change the state of the coating pan resource entity immediately if it fails during the coating process, then you would set the attribute AdjustSeized to 'true'. If you do not connect the OutPreempt or the OutResource ports on the CoatingDelay block, then the coating pan resource entity, along with its associated candy batch entity, will remain in the CoatingDelay block until the coating time ends. If you want to immediately release the coating pan entity from the candy batch entity if there is a failure, then you use the OutResource port on the CoatingDelay block to route the failed coating pan resource entity elsewhere. You can also use the OutPreempt port to route the candy batch entity elsewhere in the model if you do not want it to complete its coating delay time if there is a resource failure.

In this example, the Entity Generator blocks labeled CoatingPanGenerator and CoatingPanSchedGenerator both generate entities at time 0. To ensure that the resource entities of the type CoatingPan are generated before the resource schedule entities, you use the Block Ranking dialog box to give the block CoatingPanGenerator the highest ranking in the model, as shown in Figure E.68. If the resource schedule entities are generated before the CoatingPan entities, it is possible that the resource schedule entities will enter the Resource Scheduler block and attempt to apply the agenda to a resource entity that does not exist yet. The blocks CoatingPanSchedGenerator and Resource Scheduler are given the second-highest priority in the model when there are ties on the event calendar.

**Figure E.68** Block Ranking Dialog Box for the Candy Coating Example
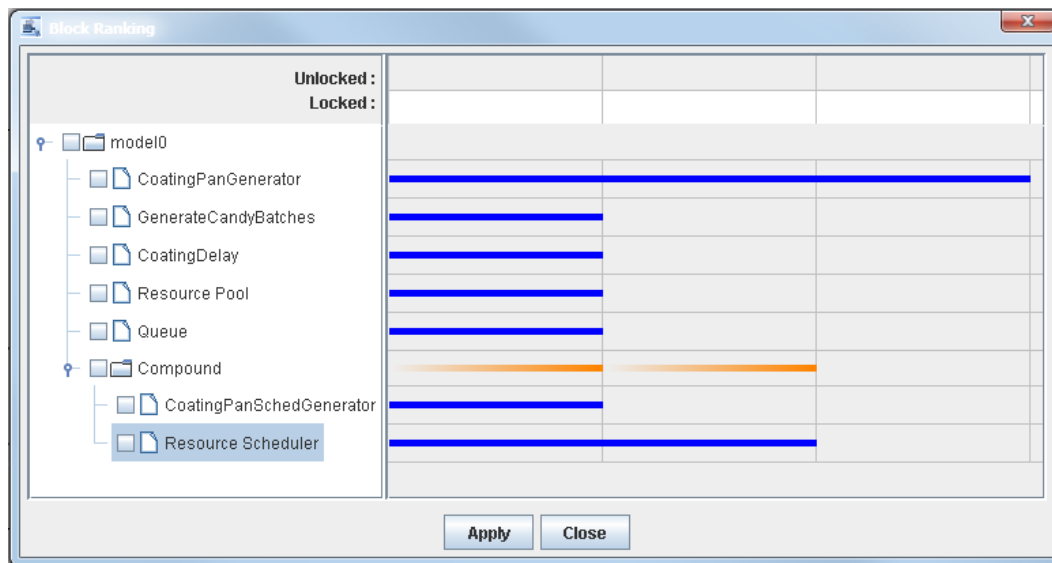
To collect resource statistics on the coating pan resource entities individually and as a group, you use a Resource Stats Collector block with three groups defined: CoatingPan_ALL, CoatingPan1, and CoatingPan2, as shown in Figure E.69. It is important to note that if an attribute rule is used in the Resource Stats Collector block to define a group, any attributes that are used in the rule must be predefined for the resource entity in the Entity Types dialog box (in this case, the attribute CoatingPan_ID is predefined in the Entity Types dialog box for the CoatingPan entity type, as shown in Figure E.65). For each of the three resource groups, the following statistics are calculated, as shown in Figure E.70:

- CoatingPanUtil: the utilization, or the percentage of resource units in the group that are seized relative to the time-averaged number of available resource units in the group during the simulation run

- CoatingPan_NumFailed: the time-weighted average number of resource units in the group that are in the failed state

- CoatingPan_PerNumFailed: the percentage of the resource units in the group that are in the failed state, relative to the time-averaged number of available resource units in the group during the simulation run. For the CoatingPan1 and CoatingPan2 groups, the CoatingPan_PerNumFailed statistic is equal to the CoatingPan_NumFailed statistic, because each group has only one resource entity in it.

Figure E.69 Groups Defined in the Resource Stats Collector Block
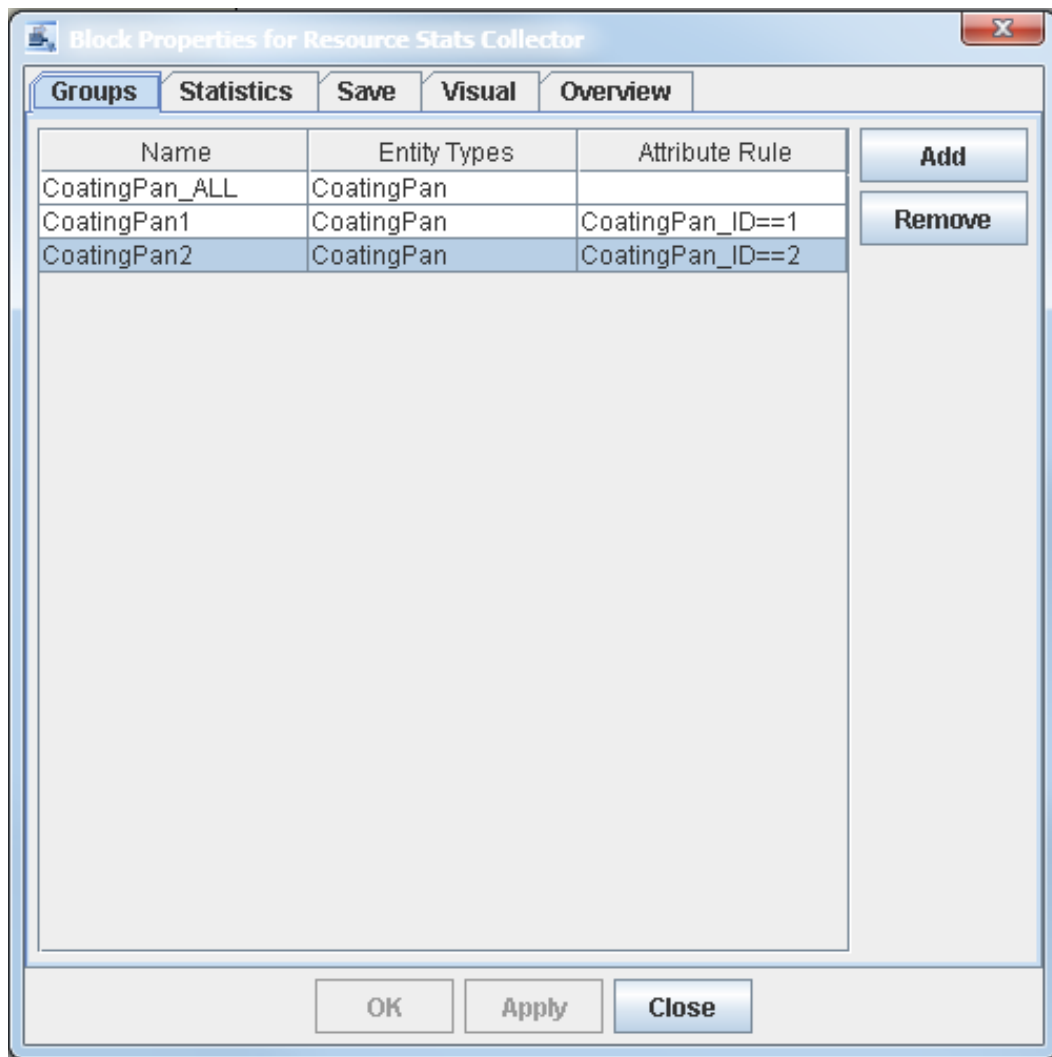
Figure E.70 Statistics Defined in the Resource Stats Collector Block
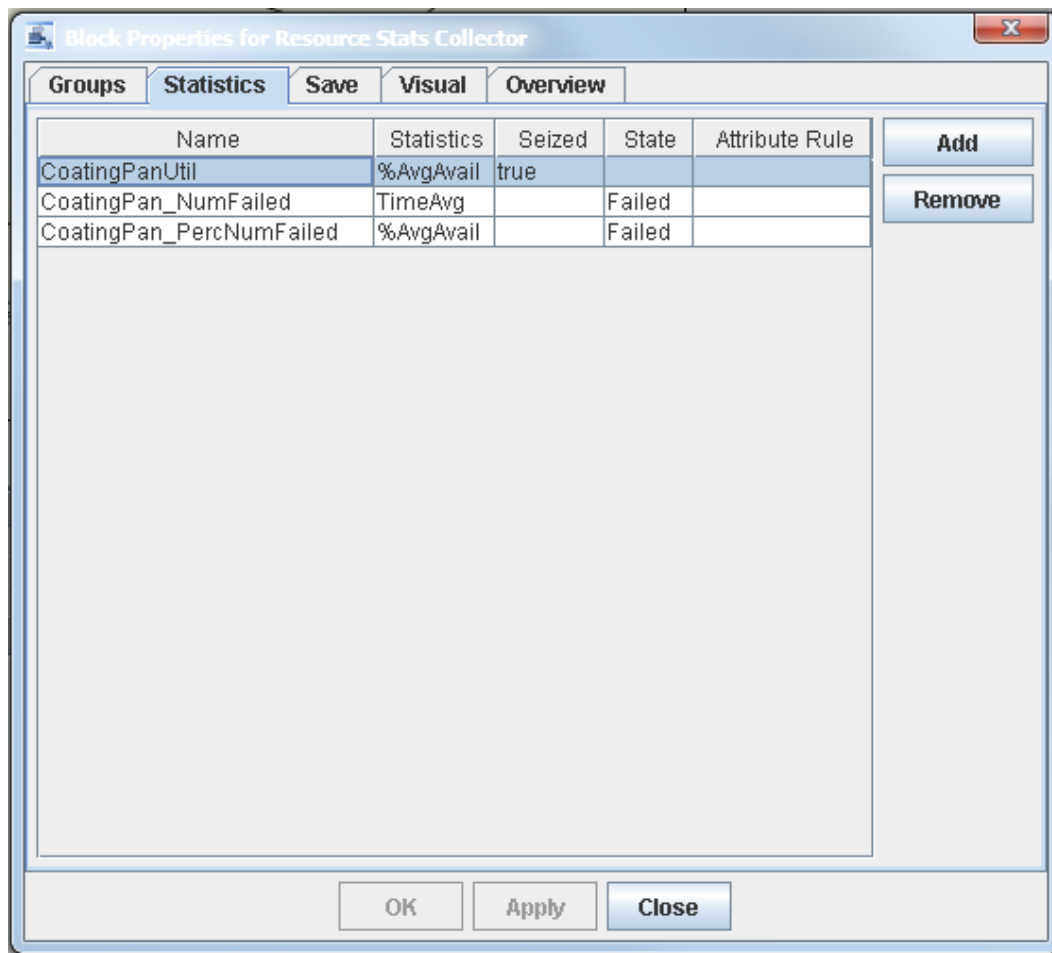
In this example, the simulation stops when the end time specified in the Experiment window is reached. However, suppose you want to stop the simulation when a certain number of candy batches have been coated. In `model1` of the `docSchedEntity` project, the **Maximum Number of Entities** field in the GenerateCandy-Batches Entity Generator block has been set to 1000. If you made only that change and ran the model with an end time of infinity in the Experiment window, then the model would never stop running. This is because the Resource Scheduler block will continue to schedule state changes for the coating pan resource entities indefinitely. To stop the model in this scenario, you use a Stopper block, as shown in Figure E.71. After the candy batches release the coating pan resource entity, they are counted, and that count is passed to a Switch block to determine whether the last batch has been processed. If the current entity is the last batch, then a Gate block is used to send a Boolean signal of `'true'` to a Stopper block to stop the simulation.

Figure E.71 Candy Coating Example with a Stopper Block
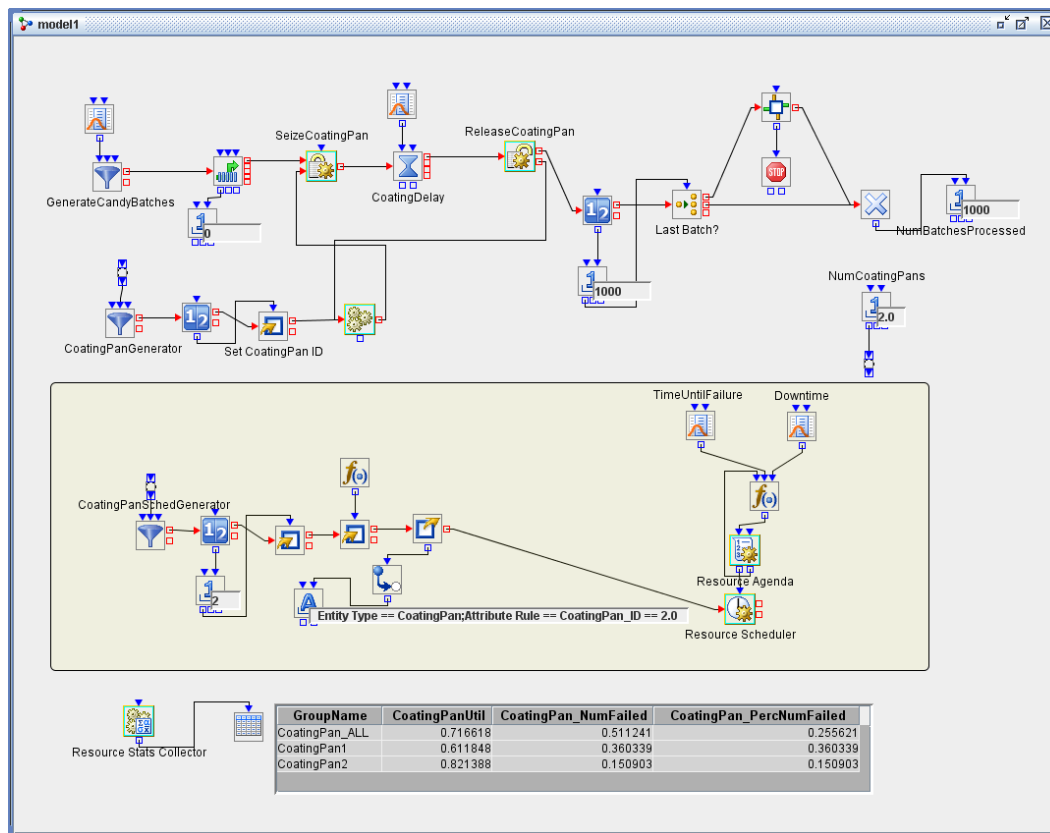
Appendix F
Expressions

Contents

Overview of Expressions

Expressions are used in various places in Simulation Studio:

1. in writing expressions in the **Expression** field of a Formula block;

2. in writing Boolean criteria in the **Attribute Rule** field of the Unbatch, Entity Filter, Entity Group Holder, Resource Stats Collector, and Resource Scheduler blocks; and

3. in writing Boolean criteria in the **Attributes** field of the Seize and Release blocks.

All blocks that have an **Attribute Rule** or **Attributes** field also have a pop-up editing window for writing expressions. You open the editing window for expressions by right-clicking in the **Attribute Rule** or **Attributes** field and selecting the option **Edit**. This appendix documents the operators and functions for Simulation Studio expressions.

Operators

The following Boolean operators can be used in an expression.
The syntax is *operand1 operator operand2* or *operator operand*.

&&   logical and (valid for two Boolean operands)

‖   logical or (valid for two Boolean operands)

!   logical not (valid for one Boolean operand)

The following arithmetic operators can be used in an expression.
The syntax is *operand1 operator operand2* or *operator operand*.

+	add (valid for two numeric operands)
–	subtract (valid for one or two numeric operands)
*	multiply (valid for two numeric operands)
/	divide (valid for two numeric operands)
%	remainder (valid for two numeric operands)

The following equality or inequality operators can be used in an expression.
The syntax is *operand1 operator operand2*.

==	equal to (valid for two numeric, Boolean, or text operands)
!=	not equal to (valid for two numeric, Boolean, or text operands)
<	less than (valid for two numeric operands)
>	greater than (valid for two numeric operands)
<=	less than or equal to (valid for two numeric operands)
>=	greater than or equal to (valid for two numeric operands)

Functions

The following function requires no arguments and can be used in an expression.

timeNow() returns the current simulation clock value.

The following arithmetic functions can be used in an expression.

abs(*arg1*) returns the absolute value of a single numeric argument.

floor(*arg1*) returns the largest integer that is less than or equal to a single numeric argument.

ceil(*arg1*) returns the smallest integer that is greater than or equal to a single numeric argument.

round(*arg1,arg2*) returns the numeric argument *arg1* rounded off to the specified precision *arg2*, where *arg2* > 0 rounds off *arg1* up to *arg2* decimal places to the left of the decimal point, *arg2* < 0 rounds off *arg1* up to *arg2* decimal places to the right of the decimal point, and *arg2*=0 rounds off *arg1* to the closest integer. The default value for *arg2* is 0. The rounding function returns 0 for any invalid value of *arg2* (where valid values of *arg2* depend on the value of *arg1*).

min(*arg1,arg2,...*) returns the minimum value among two or more numeric arguments.

max(*arg1,arg2,...*) returns the maximum value among two or more numeric arguments.

power(*arg1*,*arg2*)   returns the first numeric argument *arg1* raised to the power of the second numeric argument *arg2*.

sin(*arg1*)   returns the trigonometric sine of a single numeric radians argument.

cos(*arg1*)   returns the trigonometric cosine of a single numeric radians argument.

tan(*arg1*)   returns the trigonometric tangent of a single numeric radians argument.

asin(*arg1*)   returns the trigonometric arc sine of a single numeric radians argument.

acos(*arg1*)   returns the trigonometric arc cosine of a single numeric radians argument.

atan(*arg1*)   returns the trigonometric arc tangent of a single numeric radians argument.

sinh(*arg1*)   returns the trigonometric hyperbolic sine of a single numeric radians argument.

cosh(*arg1*)   returns the trigonometric hyperbolic cosine of a single numeric radians argument.

tanh(*arg1*)   returns the trigonometric hyperbolic tangent of a single numeric radians argument.

log(*arg1*)   returns the base 10 logarithm of a single numeric argument.

ln(*arg1*)   returns the natural logarithm of a single numeric argument.

exp(*arg1*)   returns *e* (Euler's number) raised to the power of a single numeric argument.

The following string functions can be used in an expression.

concat(*arg1*,*arg2*,...)   returns the concatenation of two or more string arguments.

substring(*string*,*index*,*length*)   returns the substring of *string* starting at the specified zero-based *index*. For *length* > 0, the substring starting at *index* is returned; for *length* < 0, the substring ending at *index* is returned; if *length* is not specified, the substring from *index* to the end of *string* is returned.

The following conversion functions can be used in an expression.

degrees(*arg1*)   returns the degrees equivalent of a single numeric radians argument.

radians(*arg1*)   returns the radians equivalent of a single numeric degrees argument.

toString(*arg1*)   returns a string representation of a single argument.

toNumber(*arg1*)   returns a numeric representation of a single argument (if possible).

toBoolean(*arg1*)   returns the Boolean value *false* for a numeric argument with value 0 or a string argument with value other than "true". Returns the Boolean value *true* for a nonzero numeric argument or a string argument with value "true".

The following argument-index functions can be used in an expression.

minindex(*arg1*,*arg2*,...)   returns the zero-based index of the argument with the smallest value among two or more numeric arguments.

maxindex(*arg1*,*arg2*,...)   returns the zero-based index of the argument with the largest value among two or more numeric arguments.

The following logical functions can be used in an expression.

cond has the following syntax: *cond(Boolean expression, true return value, false return value)*. The value returned by this function is determined by evaluating the Boolean expression that is the first argument. If the Boolean expression evaluates to true, the second argument is returned. Otherwise, the third argument is returned.

switch has the following syntax: *switch(Boolean expression 1, value 1, Boolean expression 2, value 2, ..., default value)*. The value returned by this function is the value argument immediately following the first Boolean expression argument that evaluates to true. The default value is returned if none of the Boolean expression arguments evaluate to true.

Examples

The following are some sample expressions and how they are evaluated in Simulation Studio.

1. The expression **(AttributeA || AttributeB) && (! AttributeC)** evaluates to true if the following two conditions are both satisfied: either AttributeA or AttributeB is true, and AttributeC is false.

2. Assume the following values:

   Attribute = 1, Attribute2 = 2, Attribute3 = 1, Attribute4 = 5, Attribute5 = 2, and Attribute6 = 4.

   ((Attribute + Attribute2 – Attribute3) * Attribute4 / Attribute5) % Attribute 6   evaluates to 1 because 1 + 2 – 1 is 2, the result 2 multiplied by 5 is 10, the result 10 divided by 2 is 5, and the remainder of 5 divided by 4 is 1.

   Attribute2 == Attribute5   evaluates to true because both Attribute2 and Attribute 5 have the same value.

   max(Attribute6,Attribute5,Attribute4)   evaluates to 5 because Attribute4 has a value of 5, which is larger than the values of the other two attributes.

   maxindex(Attribute6,Attribute5,Attribute4)   evaluates to 2 because Attribute4 has a value that is larger than the values of the other two attributes. Furthermore, the zero-based index of Attribute4 in the list of arguments is 2 (Attribute6 has index 0 and Attribute5 has index 1).

3. Assuming MyNum is –13.5, the expression **abs(floor(MyNum))** evaluates to 14 because the floor of –13.5 is –14 and the absolute value of –14 is 14.

4. If MyNum=2000/3=666.666666, then

round(MyNum)	=	667.0
round(MyNum,0)	=	667.0
round(MyNum,1)	=	670.0
round(MyNum,2)	=	700.0
round(MyNum,3)	=	1000.0
round(MyNum,-3)	=	666.667

5. Assume the following values: A="One", B="Two", and C="Three". Then the expression **concat(A,B,C)** evaluates to "OneTwoThree".

6. The expression **substring("Attribute1",2,4)** evaluates to "trib".

7. The expression **substring("Attribute1",2,-3)** evaluates to "Att".

8. Assume Input=3. Then the expression **cond(Input%2==0,1,2)** evaluates to a value of 2 because the Boolean expression in the first argument is false. Therefore, the third argument is returned.

9. Assume the following values: X=3, Y=2, and Z=4. Then the expression **cond(X < Y,0, cond(X < Z, 1, 2))** evaluates to 1 because X < Y is false and X < Z is true.

10. Assume Input=45. Then the expression **switch(Input <= 10, 1, Input > 10 && Input <= 20, 2, Input > 20 && Input <= 30 ,3, 0)** evaluates to 0 because the three Boolean expressions all evaluate to false. If Input=25, then the same expression evaluates to 3.
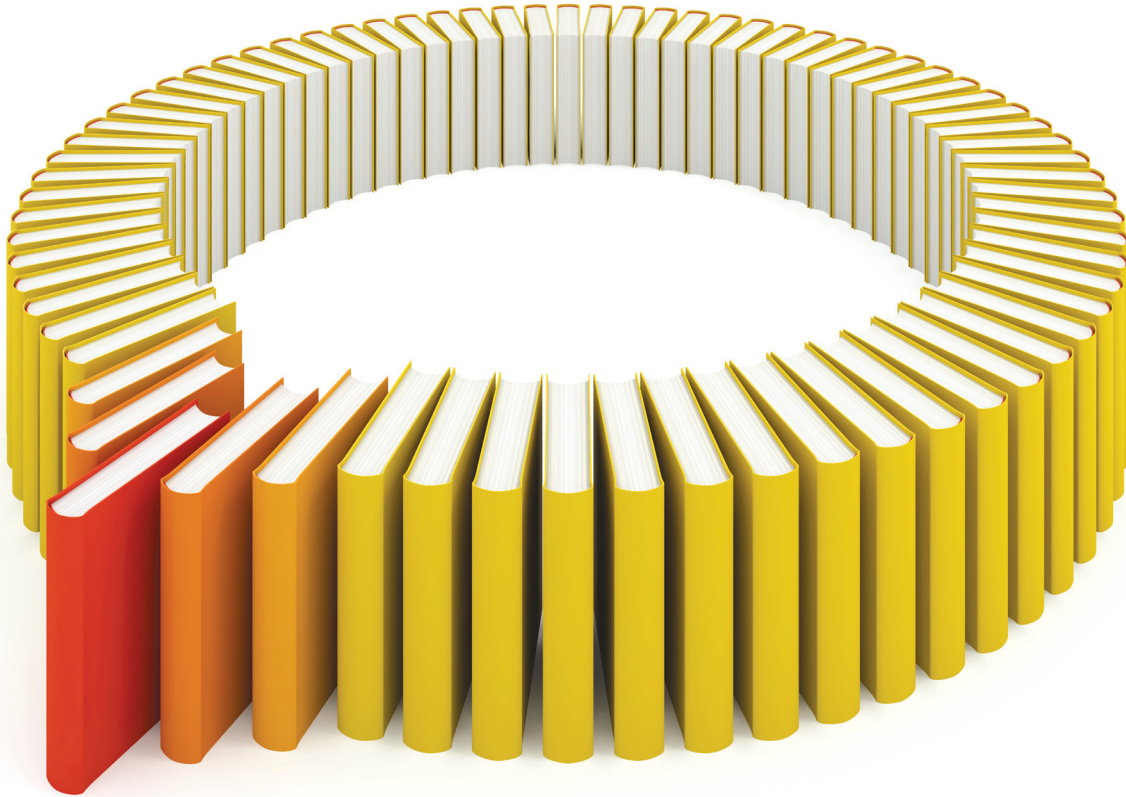
Index

Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

§sas
THE POWER TO KNOW®