Acknowledgments

Credits

Documentation

<table>
<thead>
<tr>
<th>Writing</th>
<th>Marc-david Cohen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing</td>
<td>Virginia Clark, Donna Sawyer</td>
</tr>
<tr>
<td>Documentation Support</td>
<td>Tim Arnold, Michelle Opp</td>
</tr>
<tr>
<td>Technical Review</td>
<td>Marc-david Cohen, Gehan Corea, Edward P. Hughes, John Jasperse, Charles B. Kelly, Phil Meanor</td>
</tr>
</tbody>
</table>

Software

The QSIM Application was implemented by the Operations Research and Development Department. Substantial support was given to the project by other members of the Analytical Solutions Division. Core Development Division, Display Products Division, Graphics Division, and the Host Systems Division also contributed to this product.

| QSIM Application | Hong Chen, Phil Meanor, Marc-david Cohen, Charles B. Kelly |

Support Groups

<table>
<thead>
<tr>
<th>Software Testing</th>
<th>Tao Huang, John Jasperse, Bengt Pederson, Rob Pratt, Nitin Agarwal, Marianne Bohinski, Edward P. Hughes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Support</td>
<td>Tonya Chapman</td>
</tr>
</tbody>
</table>
Acknowledgments

Many people have been instrumental in the development of SAS/OR software. The individuals acknowledged here have been especially helpful.

Lance Broad       New Zealand Ministry of Forestry
Richard Brockmeier  Union Electric Company
Ken Cruthers       Goodyear Tire & Rubber Company
Patricia Duffy     Auburn University
Richard A. Ehrhardt  University of North Carolina at Greensboro
Paul Hardy          Babcock & Wilcox
Don Henderson       ORI Consulting Group
Dave Jennings       Lockheed Martin
Vidyadhar G. Kulkarni University of North Carolina at Chapel Hill
Wayne Maruska       Basin Electric Power Cooperative
Bruce Reed          Auburn University
Charles Rissmiller  Lockheed Martin
David Rubin         University of North Carolina at Chapel Hill
John Stone          North Carolina State University
Keith R. Weiss      ICI Americas Inc.

The final responsibility for the SAS System lies with SAS Institute alone. We hope that you will always let us know your opinions about the SAS System and its documentation. It is through your participation that SAS software is continuously improved.
What’s New in SAS/OR 9 and 9.1

Overview

SAS/OR software contains several new and enhanced features since SAS 8.2. Brief descriptions of the new features appear in the following sections. For more information, refer to the SAS/OR documentation, which is now available in the following six volumes:

- SAS/OR User’s Guide: Bills of Material Processing
- SAS/OR User’s Guide: Constraint Programming
- SAS/OR User’s Guide: Local Search Optimization
- SAS/OR User’s Guide: Mathematical Programming
- SAS/OR User’s Guide: The QSIM Application

The online help can also be found under the corresponding classification.

The BOM Procedure

The BOM procedure in SAS/OR User’s Guide: Bills of Material Processing was introduced in Version 8.2 of the SAS System to perform bill of material processing. Several new features have been added to the procedure, enabling it to read all product structure records from a product structure data file and all part “master” records from a part master file, and compose the combined information into indented bills of material. This data structure mirrors the most common method for storing bill-of-material data in enterprise settings; the part master file contains data on each part while the product structure file holds data describing the various part-component relationships represented in bills of material.

The PMDATA= option on the PROC BOM statement enables you to specify the name of the Part Master data set. If you do not specify this option, PROC BOM uses the Product Structure data set (as specified in the DATA= option) as the Part Master data set. The BOM procedure now looks up the Part, LeadTime, Requirement, QtyOnHand, and ID variables in the Part Master data set. On the other hand, the Component and Quantity variables remain in the Product Structure data set.

You can use the NRELATIONSHIPS= (or NRELTSES=) option to specify the number of parent-component relationships in the Product Structure data set. You have greater control over the handling of redundant relationships in the Product Structure data set using the DUPLICATE= option.
Several options have been added to the STRUCTURE statement enabling you to specify information related to the parent-component relationships. In particular, the variable specified with the PARENT= option identifies the parent item, while the variables listed in the LTOFFSET= option specify lead-time offset information. You can also specify variables identifying scrap factor information for all parent-component relationships using the SFACTOR= option. The RID= option identifies all variables in the Product Structure data set that are to be included in the Indented BOM output data set.

**The CLP Procedure (Experimental)**

The new CLP procedure in *SAS/OR User’s Guide: Constraint Programming* is an experimental finite domain constraint programming solver for solving constraint satisfaction problems (CSPs) with linear, logical, global, and scheduling constraints. In addition to having an expressive syntax for representing CSPs, the solver features powerful built-in consistency routines and constraint propagation algorithms, a choice of nondeterministic search strategies, and controls for guiding the search mechanism that enable you to solve a diverse array of combinatorial problems.

**The CPM Procedure**

The CPM procedure in *SAS/OR User’s Guide: Project Management* adds more options for describing resource consumption by activities, enhancing its applicability to production scheduling models.

A new keyword, RESUSAGE, has been added to the list of values for the OBSTYPE variable in the Resource data set. This keyword enables you to specify whether a resource is consumed at the beginning or at the end of a given activity.

The MILESTONERESOURCE option enables you to specify a nonzero usage of consumable resources for milestone activities. For example, this option is useful if you wish to designate specific milestones to be the points of payment for a subcontractor. The MILESTONENORESOURCE option is the current default behavior of the CPM procedure, which indicates that all resource requirements are to be ignored for milestone activities.

**The GA Procedure (Experimental)**

The new GA procedure in *SAS/OR User’s Guide: Local Search Optimization* facilitates the application of genetic algorithms to general optimization. Genetic algorithms adapt the biological processes of natural selection and evolution to search for optimal solutions. The procedure can be applied to optimize problems involving integer, continuous, binary, or combinatorial variables. The GA procedure is especially useful for finding optima for problems where the objective function may have discontinuities or may not otherwise be suitable for optimization by traditional calculus-based methods.
The GANTT Procedure

The GANTT procedure in SAS/OR User's Guide: Project Management includes a new option for controlling the width of the Gantt chart. The CHARTWIDTH= option specifies the width of the axis area as a percentage of the total Gantt chart width. This option enables you to generate Gantt charts that are consistent in appearance, independent of the total time spanned by the project.

The LP Procedure

The performances of primal and dual simplex algorithms in the LP procedure (SAS/OR User's Guide: Mathematical Programming) have been significantly improved on large scale linear or mixed integer programming problems.

The PM Procedure

The new options added to the CPM procedure are also available with PROC PM.

The QP Procedure (Experimental)

The new QP procedure in SAS/OR User’s Guide: Mathematical Programming implements a primal-dual predictor-corrector interior-point algorithm for large sparse quadratic programs. Depending on the distribution of the eigenvalues of the Hessian matrix, $H$, two main classes of quadratic programs are distinguished (assuming minimization):

- convex: $H$ is positive semi-definite
- nonconvex: $H$ has at least one negative eigenvalue

Diagonal and nonseparable Hessian matrices are recognized and handled automatically.

Bill of Material Post Processing Macros

Several macros enable users to generate miscellaneous reports using the Indented BOM output data set from the BOM procedure in SAS/OR User's Guide: Bills of Material Processing. Other transactional macros perform specialized transactions for maintaining and updating the bills of material for a product, product line, plant, or company.
Introduction

Purpose

*SAS/OR User’s Guide: The QSIM Application* provides a complete reference for the QSIM Application, which is used to build and analyze models of queueing systems using discrete event simulation.

Accessing the SAS/OR Sample Library

The SAS/OR sample library includes many examples that illustrate the use of SAS/OR software. To access these sample programs, select Learning to Use SAS->Sample SAS Programs from the SAS Help and Documentation window, and then select SAS/OR from the list of available topics.

Online Help System and Updates

You can access online help information about SAS/OR software in two ways, depending on whether you are using the SAS windowing environment in the command line mode or the pull-down menu mode.

If you are using a command line, you can access the SAS/OR help menus by typing help or on the command line. If you are using the pull-down menus, you can select SAS Help and Documentation->SAS Products from the Help pull-down menu, and then select SAS/OR from the list of available topics.

Additional Documentation for SAS/OR Software

In addition to *SAS/OR User’s Guide: The QSIM Application*, you may find these other documents helpful when using SAS/OR software:

*SAS/OR User’s Guide: Bills of Material Processing*

provides documentation for the BOM procedure and all bill-of-material post-processing SAS macros. The BOM procedure and SAS macros provide the ability to generate different reports and to perform several transactions to maintain and update bills of material.

*SAS/OR User’s Guide: Constraint Programming*

provides documentation for the constraint programming procedure in SAS/OR software. This book serves as the primary documentation for the CLP procedure, an experimental procedure new to SAS/OR software.
Introduction

SAS/OR User’s Guide: Local Search Optimization
provides documentation for the local search optimization procedure in SAS/OR software. This book serves as the primary documentation for the GA procedure, an experimental procedure that uses genetic algorithms to solve optimization problems.

SAS/OR User’s Guide: Mathematical Programming
provides documentation for the mathematical programming procedures in SAS/OR software. This book serves as the primary documentation for optimization procedures, such as the ASSIGN, LP, INTPOINT, NETFLOW, NLP, and TRANS procedures, and the new procedure, QP, for solving quadratic programming problems.

provides documentation for the project management procedures in SAS/OR software. This book serves as the primary documentation for the CPM, DTREE, GANTT, NETDRAW, and PM procedures, as well as the PROJMAN Application, a graphical user interface for project management.

SAS/OR Software: Project Management Examples, Version 6
contains a series of examples that illustrate how to use SAS/OR software to manage projects. Each chapter contains a complete project management scenario and describes how to use PROC GANTT, PROC CPM, and PROC NETDRAW, in addition to other reporting and graphing procedures in the SAS System, to perform the necessary project management tasks.

SAS/IRP User’s Guide: Inventory Replenishment Planning
provides documentation for SAS/IRP software. This book serves as the primary documentation for the IRP procedure for determining replenishment policies, as well as the %IRPSIM SAS programming macro for simulating replenishment policies.
The QSIM Application is a SAS application for modeling and analyzing queueing systems using discrete event simulation. These models are used in a wide variety of scenarios that might be encountered in network and telecommunications systems, manufacturing systems, and transportation networks. For example, before instituting a re-engineered business process, you could use the QSIM Application to model the new process and study the model behavior to gain insight into how the new process might behave.

Figure 1.1. An M/M/1 Queueing Model

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

Figure 1.1 shows a simple M/M/1 queuing model built with the QSIM Application. An M/M/1 queuing model has a Poisson arrival process, exponential service times for a single server, and a FIFO queueing discipline. You can build a model of such a queueing system, control the simulation of the model, and produce summary statistics from the simulation sample path from within the application. You can save the model and the sample path in SAS data sets for reuse and further analysis. The application is designed to simplify model building by encouraging the construction of
hierarchical models based on user-built model components that can be stored, shared, and replicated easily.

In this context a model is a directed network. Transactions flow through the network changing the state of the model upon arrival at vertices or nodes. The type of state change that occurs depends on the current state of the simulation and the particulars of the model.

**Invoking the QSIM Application**

You invoke the application by typing QSIM on a command line or selecting Solutions ➤ from the menu bar then Analysis ➤ Queueing Simulation from the pull-down menu.

**User Interface**

When you invoke the QSIM Application, a Simulation window and a palette opens, as shown in Figure 1.2. The Simulation window has two panels; the command panel on the top and the model panel on the bottom. The palette contains the components that can be used to build a model. You build the model in the model panel, using the mouse to drag and drop model components from the palette into the model panel. These model components are connected by arcs to produce a directed network representation of the model. You can change parameters and other properties of model components using pop-up menus from the components.

![Figure 1.2. The Palette and the Simulation Window](image)
If you don’t see the component that you want in the palette, use the scrollbar on its right side to scroll the palette. You pick up the component from the palette and drag it to the model panel and then drop it. For example, to place a Sampler in the Simulation window, you pick up the \text{\texttt{\textbf{Sampler}}} component in the palette and then drop it on the model panel.

The command panel contains command buttons. These are push buttons that execute commands. For example, the animate button, \text{\texttt{\textbf{\{}}}}, is a command button that toggles animation on and off when pressed.

The QSIM Application also supports cursor tracking. As you move the cursor over a component, the shape of the cursor will change depending on where the cursor is in the component. When the cursor changes shape, you can click the left mouse button and elicit location-dependent behavior such as resizing the component, drawing an arc, or replicating (cloning) a component. For example, if you click when the cursor is a \text{\texttt{\textbf{+}}}, you can draw an arc from that component to the next component that you click. Also, picking and dragging a component with the cursor will move the component. Finally, clicking the right mouse button will always display a pop-up menu.

This paradigm for the user interface is used throughout the QSIM Application.

\textbf{Figure 1.3.} The \texttt{Sampler} Pop-Up Menu
Figure 1.3 on page 9 shows the pop-up menu on the Sampler obtained by clicking the right mouse button. Notice that the first item in the menu names the type of component. The submenu from the Sampler entry lists the actions that are specific to that component. In this case, it includes: Control Panel..., which opens a window that enables you to control the component, an action often used; Start, which starts the transaction sampling, an action seldom needed; and Stop, which stops the transaction sampling, also an action seldom needed. Other general actions include Expose/Hide Detail for changing the Sampler representation from an image to a line drawing, and Tools for other less frequently used tools.

Model components are connected by arcs. Transactions flow down arcs between components. You can connect two components with an arc by placing the cursor over the right side of a component. Notice that the cursor shape changes to +. If you click the left mouse button you will have a rubberband line to the cursor. Now when you select another component, an arc will connect the two components. If you move the cursor out of the Simulation window while the rubberband line is connected to it, the window may start power scrolling. Bringing the cursor back into the window will stop the scrolling. Clicking outside the window will take you out of the rubberband line mode.

Model components have control panels associated with them. With these panels, you specify parameters and control the behavior of the component. The simulation also has a control panel, as shown in Figure 1.5 on page 12. It is displayed by selecting the command button.
The Command Buttons

Figure 1.4. The Command Panel and Buttons

The command buttons shown in Figure 1.4 initiate global simulation commands. Notice that the stop button is pressed. This means that the simulation is temporarily halted. If the button were pressed again, the simulation would proceed from the point at which it was interrupted. From left to right the command buttons are

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>shows the simulation time. By default time is measured in dimensionless units, but it can be changed to any unit from milliseconds to years using the pull down menu Options ➤ Clock.... Pressing on this button toggles between the simulation time and the clock icon.</td>
</tr>
<tr>
<td></td>
<td>starts the simulation.</td>
</tr>
<tr>
<td></td>
<td>stops the simulation.</td>
</tr>
<tr>
<td></td>
<td>resets the simulation.</td>
</tr>
<tr>
<td></td>
<td>toggles transaction animation on and off. When animation is on, you can see transactions that originate at Samplers traversing the network.</td>
</tr>
<tr>
<td></td>
<td>aligns components vertically and horizontally.</td>
</tr>
<tr>
<td></td>
<td>raises the Debug Control Panel.</td>
</tr>
<tr>
<td></td>
<td>raises the Components Control Panel.</td>
</tr>
<tr>
<td></td>
<td>raises the Viewport Window.</td>
</tr>
<tr>
<td></td>
<td>saves the simulation model.</td>
</tr>
<tr>
<td></td>
<td>raises the Component Attributes Panel for changing colors, fonts, and images.</td>
</tr>
</tbody>
</table>

The Component Control Panel

The Component Control Panel gives you control over individual components in your models. You select components in the Components list box and then press buttons in the window to perform actions. The Components list box shows the highest level of components in the model. Since components can be nested into a hierarchy, there can be multiple levels.
Chapter 1. Getting Started

Figure 1.5. The Component Control Panel

If you select a component and press one of the five buttons to the right of the Components list box, the component executes that action. For example, selecting Sampler in the list box and pressing the Start button starts transaction sampling in that sampler.

- The Start button starts all the selected components.
- The Stop button stops all the selected components.
- The Reset button resets all selected components.
- The Controls button raises the control panels for all selected components.
- The Analyze button executes statistical analyses on all selected components.

The two buttons < and > are used for navigating into and out of compound components in the Components list box.

Before performing any statistical analysis, data must be collected. The Data Collection check box controls saving the simulation sample path into a SAS data set which is then used for the data analysis.
A Simple M/M/1 Queueing Model

A simple example illustrates some of the concepts involved in model building. The network shown in Figure 1.6 models an M/M/1 queue. Transactions originate at the source node Sampler. The user chooses the interval between transactions, called the *inter-arrival time*, to be a sample of a random variable (from one of several distributions), a fixed amount, or the value of a variable read from a SAS data set. By default, the inter-arrival time is an observation of an exponential random variable with parameter 1. This models a Poisson process for transaction arrivals.

![M/M/1 Queueing Model](image)

**Figure 1.6.** An M/M/1 Queueing Model

**Queueing**

When the transaction leaves the Sampler, it flows down the arc to the FIFO Queue. It is important to note that the movement of the transaction down the arc does not advance the simulation clock. On transaction arrival at the FIFO Queue, the queue broadcasts messages down arcs asking nodes downstream if they are busy. The responses depend on the types of components that are connected to the queue and the state of the simulation when the message is received. (Details are given in the next chapter.) It is important to note that broadcasting and evaluation of these messages does not advance the simulation clock. If the queue gets a response that there is a non-busy node, then it sends the transaction down the arc leading to that node. Otherwise, the transaction remains in the queue. When the simulation is first started, the Server is empty; when it gets the message “are you busy” from the queue, it responds “no.” As a result, the queue routes the transaction down the arc to the Server.

**Service**

When the transaction arrives at the Server, service is scheduled and the transaction ties up the server. By default, the service time is an observation of an exponential random variable with parameter 1. Both the service distribution and its parameters can be changed using the server’s control panel. While the server is serving this transaction, any “are you busy” messages sent to it result in a “yes” response. When service is complete, the server sends the transaction on any arcs directed away from it and also
sends a message up the arcs directed into it requesting an additional transaction. In this example, if the FIFO Queue is not empty, it will remove the transaction that has been there the longest and send it to the Server. By default, all queues in the system have a capacity of 50 transactions. Of course, this capacity can be changed through the user interface or programmatically, as discussed in the next chapter. Since by default the inter-arrival times and the service times are $\mathcal{E}(1)$, exponentially distributed with mean 1, the transaction time in the system would not have a stationary distribution if the queue had infinite capacity.

**Statistics**

Finally, the transaction flows to a Bucket, which collects transactions and can also save the value of transaction attributes in a SAS data set.

![Bucket Control Panel](image)

**Figure 1.7.** A Bucket Control Panel

From the bucket control panel (Figure 1.7) you can set the size of the transaction collection buffer and name the transaction attribute to accumulate. You can also name a SAS data set into which to collect the transaction attribute values.
Chapter 1. Getting Started

---

### The UNIVARIATE Procedure

**Variable:** value

**Moments**

<table>
<thead>
<tr>
<th>Moment</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1803</td>
</tr>
<tr>
<td>Mean</td>
<td>24.5093328</td>
</tr>
<tr>
<td>Std Deviation</td>
<td>14.8024337</td>
</tr>
<tr>
<td>Skewness</td>
<td>-0.1060516</td>
</tr>
<tr>
<td>Uncorrected SS</td>
<td>1477915.34</td>
</tr>
<tr>
<td>Coeff Variation</td>
<td>60.3950903</td>
</tr>
</tbody>
</table>

| Sum Weights           | 1803           |
| Sum Observations      | 44190.3271     |
| Variance              | 219.112044     |
| Kurtosis              | -1.2191446     |
| Corrected SS          | 394839.903     |
| Std Error Mean        | 0.34860532     |

**Basic Statistical Measures**

**Location**

- **Mean:** 24.50933
- **Median:** 26.21585
- **Mode:**

**Variability**

- **Std Deviation:** 14.80243
- **Variance:** 219.11204
- **Range:** 53.62360
- **Interquartile Range:** 26.10309

**Tests for Location: \( \mu_0 = 0 \)**

<table>
<thead>
<tr>
<th>Test</th>
<th>Statistic</th>
<th>p Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student’s t</td>
<td>t</td>
<td>(</td>
</tr>
<tr>
<td>Sign</td>
<td>M</td>
<td>( p &gt;</td>
</tr>
<tr>
<td>Signed Rank</td>
<td>S</td>
<td>( p &gt;</td>
</tr>
</tbody>
</table>

**Quantiles (Definition 5)**

<table>
<thead>
<tr>
<th>Quantile</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100% Max</td>
<td>53.62741148</td>
</tr>
<tr>
<td>99%</td>
<td>50.95981599</td>
</tr>
<tr>
<td>95%</td>
<td>46.44804080</td>
</tr>
<tr>
<td>90%</td>
<td>43.36915659</td>
</tr>
<tr>
<td>75% Q3</td>
<td>37.45295397</td>
</tr>
<tr>
<td>50% Median</td>
<td>26.21585009</td>
</tr>
<tr>
<td>25% Q1</td>
<td>11.34895903</td>
</tr>
<tr>
<td>10%</td>
<td>3.08194661</td>
</tr>
<tr>
<td>5%</td>
<td>1.38985610</td>
</tr>
<tr>
<td>1%</td>
<td>0.28776996</td>
</tr>
<tr>
<td>0% Min</td>
<td>0.00381383</td>
</tr>
</tbody>
</table>

**Extreme Observations**

<table>
<thead>
<tr>
<th>Value</th>
<th>Obs</th>
<th>Value</th>
<th>Obs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00381383</td>
<td>1442</td>
<td>51.9481</td>
<td>819</td>
</tr>
<tr>
<td>0.02449773</td>
<td>1486</td>
<td>52.1543</td>
<td>818</td>
</tr>
<tr>
<td>0.02846473</td>
<td>1458</td>
<td>52.1538</td>
<td>418</td>
</tr>
<tr>
<td>0.04001101</td>
<td>1249</td>
<td>52.4247</td>
<td>419</td>
</tr>
<tr>
<td>0.06573967</td>
<td>1700</td>
<td>53.6274</td>
<td>817</td>
</tr>
</tbody>
</table>

---

**Figure 1.8.** Statistics on a Transaction Attribute

If you collect data into a SAS data set and then press the **Analyze** button, univariate statistics will be calculated by the **UNIVARIATE** procedure (**Figure 1.8**) and a sample distribution function as shown in **Figure 1.9** on page 16 will be plotted.
Figure 1.9. Sample Distribution Function

See Chapter 8, “Analyzing the Sample Path,” for more information on collecting statistics and analyzing simulation data.
Chapter 2
Building a Model with Elementary Components

For ease in building and maintaining models, the QSIM Application promotes hierarchical model building, user component construction, and component reuse by providing a comprehensive set of primitive components and the ability to assemble components into compound components. For example, you can build a queue-server network, then encapsulate it, identify an image to represent it, use it to define a template for additional replication, and save the template in a SAS data set to be shared in other models and by other users. This chapter discusses the details of the elementary components used in model building.

There are several types of elementary components: sources, servers, queues, logic, holders, charts, and connectors. Although each of these has a special role, they have much in common, as is evident from their pop-up menus (displayed by pressing the right mouse button while pointing to the component).

![Figure 2.1. The Pop-Up Menu on the FIFO Queue Component](image)

Figure 2.1 shows a typical pop-up menu. The first entry in this pop-up menu is the name of the component. The submenu from that entry has **Control Panel...** as the first choice. Selecting it displays the component’s control panel, which enables you to set component parameters. Other choices include **Expose/Hide Details**, **Tools**, **Delete**, and **Help**. The **Expose/Hide Details** entry toggles the display of the component between an icon that represents the component, which you can change, and a drawn representation of the component. For some of the components, the drawn representation shows state information while the simulation is in progress and animating. For example, the family of queues slowly fills as transactions arrive and queue up for service.

Many of the components also have internal state information that changes as the result of transaction arrival and other kinds of message sending. There are five general types of actions that either change component state or return information about the component state: transaction arrival, request for transaction, are you busy message, query message, and trigger message. The **query message** is sent from the **Modifier** components and **formulas**. These messages make requests about the state of the component, for example, the number of transactions waiting in a queue. The **trigger message** is
sent when a transaction arrives at a **Trigger** component, and it is used to change the state of the component. For example, since the **Sampler** services the “start” message, a transaction arriving at a Trigger component can start a Sampler. The sections that follow document the elementary components and show in tables the types of state and information messages that the components service.

Each elementary component has a Control Panel associated with it. This panel provides you access to parameters that control the behavior and appearance of the component.

The system assigns a unique component ID to each of the elementary components. This is done so that you can unambiguously identify each component that appears in a list box. For example, when you instantiate a Server all that appears in the Simulation window is the icon 🔄. If you have several of these icons and you look at a Trigger control panel, such as the one shown in Figure 2.14 on page 37, you cannot distinguish them unless you give each a unique label. By default, each will have a unique ID, which is appended to the name of the object and displayed in the list box. As an alternative, you can give the component a label, which will be displayed in the list box. By default, the iconic representation of a component in the Simulation window includes the unique ID.

### Source Components

The source components are sources of transactions for the simulation network. There are two types of sources: Sampler and Transaction Pool.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>Sampler</td>
<td>generates transactions with prescribed inter-arrival times</td>
</tr>
<tr>
<td>🔄</td>
<td>Transaction Pool</td>
<td>a source of transactions</td>
</tr>
</tbody>
</table>

The Sampler generates a transaction, and then waits a specified time interval before generating another transaction. The time between transactions, called the inter-arrival time, can be a sample of a random variable (from one of several distributions), a fixed amount, or the value of a variable read from a SAS data set. By default, the inter-arrival time is an observation of an exponential random variable with parameter 1. This means that by default the Samplers follow a Poisson arrival process.
Figure 2.2 shows the Control Panel for the Sampler. The combo box labeled “Inter-Arrival-Time Distribution” enables you to specify the type of distribution. If you press the down arrow a list of distributions is displayed. You select the inter-arrival time distribution by selecting one of these. See Chapter 6, “Random and Exogenous Variation in the Model,” for information on the choices of distributions.

The control panel also has a slider for setting the capacity of the Sampler, that is, the number of transactions that will be generated before the Sampler shuts off. The “Transaction Image” button on the control panel enables you to choose a bitmap image that would flow through the network when the simulation is being animated.

The Transaction Pool (the other source component) differs from the Sampler only in that it does not create transactions unless it receives a request for transaction message. In other ways, it is identical to the Sampler.

The following documents the logic of the source components.

**Request for Transaction**

A Sampler passes requests to arcs leading into it. A Transaction Pool generates a transaction and initiates its flow.

**Are You Busy Message**

The source components pass this request to arcs leading out of them and return their answers.

**Query Message**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>capacity</td>
<td>returns the capacity</td>
</tr>
<tr>
<td>id</td>
<td>the source component’s unique identifier</td>
</tr>
<tr>
<td>on</td>
<td>TRUE if the source component is started, else FALSE</td>
</tr>
<tr>
<td>remaining</td>
<td>returns the number of transactions remaining to be sent</td>
</tr>
<tr>
<td>size</td>
<td>returns the number of transactions sent</td>
</tr>
</tbody>
</table>
Chapter 2. Building a Model with Elementary Components

### Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>stops the source component and resets the number of transactions remaining to be sent to 1</td>
</tr>
<tr>
<td>setCapacity</td>
<td>sets the capacity from the transaction attribute “capacity”</td>
</tr>
<tr>
<td>setDistribution</td>
<td>sets the distribution from the transaction attribute “distribution.”</td>
</tr>
<tr>
<td>setParameter1</td>
<td>sets the first parameter in the distribution from the transaction attribute “parameter1”</td>
</tr>
<tr>
<td>setParameter2</td>
<td>sets the second parameter in the distribution from the transaction attribute “parameter2”</td>
</tr>
<tr>
<td>start</td>
<td>starts the source component</td>
</tr>
<tr>
<td>stop</td>
<td>stops the source component</td>
</tr>
</tbody>
</table>

### Server Components

Server components model a resource used by a transaction for a specified amount of time. There are two types of servers: Server and MServer.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Server Icon]</td>
<td>Server</td>
<td>provides service for a transaction</td>
</tr>
<tr>
<td>![MServer Icon]</td>
<td>MServer</td>
<td>provides service simultaneously for multiple transactions</td>
</tr>
</tbody>
</table>

The Server holds the transaction while it is served. The service time can be a sample of a random variable (from one of several distributions), a fixed amount, or the value of a variable read from a SAS data set. By default, the service time is an observation of an exponential random variable with parameter 1.

An MServer, or multiple-server, can service multiple transactions simultaneously. The capacity of an MServer is set using the slider labeled “Capacity” on its control panel.
Notice that in the lower-right corner of the server components, there is a small rectangle. This is a Balk node. If a transaction arrives at a Server when it is busy or at an MServer when it is at capacity, the transaction will flow out the Balk node. Consider, for example, a situation where transactions are either serviced upon arrival by server 1 or, if server 1 is not free, wait for service from server 2. This is modeled by the network in Figure 2.4.

When the transaction leaves the Server or MServer, it has an attribute as named in the control panel that contains the time that the transaction spent in the server. This attribute can be used for controlling the simulation logic and for measuring the performance of the simulation by displaying it in one of the chart components or saving it in a SAS data set.

Figure 2.3. The Multiple-Server Control Panel

The following documents the logic of the server components.

Transaction Arrival

If the server is busy, at capacity, or stopped, the transaction flows out the Balk node; otherwise, service is scheduled. On service completion, a request for transaction message is sent to arcs directed into the server. If a transaction is found, then its flow is initiated. Regardless, the transaction that just finished service flows on each of the arcs directed out of the server.
Chapter 2. Building a Model with Elementary Components

Request for Transaction

If the server is not busy or stopped, then pass on the request to all arcs directed into the server. The order in which the requests for service are issued is determined by the order of the components in the “Pull from” list box on the Server Control Panel. Also, if a component is not included in the “Pull from” list box, then the request for transaction message is not propagated on the arc leading to that component.

Are You Busy Message

If the server is not busy and not stopped, then return FALSE; otherwise, return TRUE.

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>returns TRUE if the Server is busy or the MServer is at capacity or either is stopped; else, returns FALSE</td>
</tr>
<tr>
<td>capacityIs</td>
<td>returns the capacity of the MServer</td>
</tr>
<tr>
<td>full</td>
<td>returns TRUE if the Server is busy or the MServer is at capacity; else, returns FALSE</td>
</tr>
<tr>
<td>id</td>
<td>returns the server’s unique identifier</td>
</tr>
<tr>
<td>off</td>
<td>returns TRUE if the Server or MServer is stopped</td>
</tr>
<tr>
<td>sizeIs</td>
<td>returns the number of multiple-server units that are busy</td>
</tr>
<tr>
<td>space</td>
<td>returns TRUE if the Server is free or the MServer is not at capacity; else, return FALSE</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>preempt</td>
<td>removes all the transactions that are being served. They flow out of the Balk node.</td>
</tr>
<tr>
<td>preemptContinue</td>
<td>removes all the transactions that are being served. They flow out of the Balk node. The server requests transactions from upstream components.</td>
</tr>
<tr>
<td>removeIt</td>
<td>removes the transaction at the Trigger, if it is also being served. It flows out of the Balk node.</td>
</tr>
<tr>
<td>reset</td>
<td>resets the Server and MServer, destroying all waiting transactions</td>
</tr>
<tr>
<td>seize</td>
<td>attempts to obtain service for the transaction that arrived at the Trigger</td>
</tr>
<tr>
<td>setCapacity</td>
<td>sets the MServer capacity from the transaction attribute “capacity”</td>
</tr>
<tr>
<td>setDistribution</td>
<td>sets the distribution from the transaction attribute “distribution”</td>
</tr>
<tr>
<td>setParameter1</td>
<td>sets the first parameter in the distribution from the transaction attribute “parameter1”</td>
</tr>
<tr>
<td>setParameter2</td>
<td>sets the second parameter in the distribution from the transaction attribute “parameter2”</td>
</tr>
<tr>
<td>start</td>
<td>starts the Server component</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Server component. Transactions in service have normal completion.</td>
</tr>
</tbody>
</table>
Queue Components

Queue components are transient storage for transactions. There are three types of queues: FIFO Queues, LIFO Queues, and Priority Queues.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![FIFO Icon]</td>
<td>FIFO Queues</td>
<td>first-in-first-out</td>
</tr>
<tr>
<td>![LIFO Icon]</td>
<td>LIFO Queues</td>
<td>last-in-first-out</td>
</tr>
<tr>
<td>![Priority Icon]</td>
<td>Priority Queues</td>
<td>priority</td>
</tr>
</tbody>
</table>

Each type of queue can behave as a buffer. This means that when the transaction first arrives, the queue will not try to route it to a nonbusy component but will wait for a request for transaction message from a downstream component before sending it on. In addition, you can have the queue behave as a buffer for some downstream components and as a standard queue for others. Those components in the “Don’t push to:” list box in the Queue Control Panel (see Figure 2.5) define components for which the queue acts as a buffer. Those components in the “Push to:” list box define components for which the queue acts as a standard queue.

The LIFO and FIFO queues order transactions according to their arrival time. The Priority Queue uses the value of the numeric transaction attribute named “priority” to determine placement location in the queue. This default name can be changed. The priority attribute can be assigned to a transaction by the Modifier component, discussed in the “Logic Components” section on page 26. By default, the smaller the value of the attribute, the higher placement in the queue and the sooner the element will leave the queue. Although this is the default priority order, it can be changed by unselecting the “Ascending Priority Order” check box on the control panel shown in Figure 2.5.
When each transaction leaves the queue, it has an attribute with the time it spent in the queue. The name of this attribute can be specified in the queue control panel. See Figure 2.5 for where to give the attribute name. By default the attribute name for all queues is “queue.”

The following documents the logic of the Queue components.

**Transaction Arrival**

If the queue is off or at capacity, the transaction flows out the Balk node; otherwise, it sends the message *are you busy* to the nodes on arcs directed away from the queue and listed in the Push to list box. If FALSE is returned, then route the transaction there; otherwise, queue the transaction.

**Request for Transaction**

If the queue is not empty (size > 0), then remove the next transaction according to the type of queue and send it out the arc directed to the component that made the request; otherwise, return FALSE.

**Are You Busy Message**

always returns FALSE.
Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>capacity</td>
<td>returns the queue’s capacity</td>
</tr>
<tr>
<td>id</td>
<td>returns the queue’s unique identifier</td>
</tr>
<tr>
<td>releaseType</td>
<td>returns a string naming the way that the last transaction</td>
</tr>
<tr>
<td></td>
<td>was released from the balk node. Possible values are: “balk,” “empty,”</td>
</tr>
<tr>
<td></td>
<td>“filter,” “filterOne,” and “releaseOne.”</td>
</tr>
<tr>
<td>size</td>
<td>returns the number of transactions that are in the queue</td>
</tr>
<tr>
<td>space</td>
<td>returns TRUE if there is unused capacity in the queue</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>balk</td>
<td>causes the transaction at the Trigger to leave the queue from the Balk node</td>
</tr>
<tr>
<td>empty</td>
<td>empties the queue of all transactions. Note that the transactions do not leave via the Balk node.</td>
</tr>
<tr>
<td>filter</td>
<td>evaluates a formula for each transaction in the queue. If the formula evaluates to TRUE, the transaction balks; otherwise, it maintains its place in the queue. The formula that is evaluated should be in an attribute named “formula” in the triggering transaction.</td>
</tr>
<tr>
<td>filterOne</td>
<td>evaluates a formula for each transaction in the queue. The first transaction for which the formula evaluates to TRUE balks. The formula that is evaluated should be in an attribute named “formula” in the triggering transaction.</td>
</tr>
<tr>
<td>insert</td>
<td>inserts the transaction at the Trigger into the queue</td>
</tr>
<tr>
<td>releaseOne</td>
<td>releases one transaction from the queue via the the Balk node</td>
</tr>
<tr>
<td>reset</td>
<td>destroys all transactions in the queue</td>
</tr>
<tr>
<td>start</td>
<td>starts the queue</td>
</tr>
<tr>
<td>stop</td>
<td>stops the queue</td>
</tr>
</tbody>
</table>
Logic Components

The logic components fall into two categories: those that control the flow of transactions (Adder, Splitter, Router, and Switch), and those that change the state of the simulation (Modifier and Trigger).

<table>
<thead>
<tr>
<th>Icon</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Adder" /></td>
<td>Adder</td>
<td>assemble multiple transactions</td>
</tr>
<tr>
<td><img src="image" alt="Splitter" /></td>
<td>Splitter</td>
<td>split single transactions</td>
</tr>
<tr>
<td><img src="image" alt="Modifier" /></td>
<td>Modifier</td>
<td>assign an attribute to transactions</td>
</tr>
<tr>
<td><img src="image" alt="MModerifier" /></td>
<td>MModifier</td>
<td>assign multiple attributes to transactions</td>
</tr>
<tr>
<td><img src="image" alt="Trigger" /></td>
<td>Trigger</td>
<td>change a component’s state</td>
</tr>
<tr>
<td><img src="image" alt="MTrigger" /></td>
<td>MTrigger</td>
<td>change multiple components’ state</td>
</tr>
<tr>
<td><img src="image" alt="Router" /></td>
<td>Router</td>
<td>direct flow as a function of system state</td>
</tr>
<tr>
<td><img src="image" alt="Switch" /></td>
<td>Switch</td>
<td>direct flow as a function of system state</td>
</tr>
</tbody>
</table>

Trigger Component

When a transaction arrives at a Trigger component, it initiates a message being sent to another component. For example, Figure 2.6 shows the control panel for a Trigger component. Notice that the FIFO Queue component is selected and that the insert trigger is also selected.

![Figure 2.6](image)  The Trigger Control Panel
When a transaction arrives at this Trigger, the “insert” message is sent to the specific queue selected in the control panel. As documented in the “Queue Components” section on page 23, the transaction that arrives at the Trigger is the one inserted into the queue named “FIFO Queue.”

Notice the check box labeled Schedule Trigger Event in the trigger control panel. You select this check box to delay execution of the trigger event. You can specify the length of the delay by pressing the Event Interval button. This opens a Distribution window (like the one shown in Figure 4.3 on page 52) from which you can choose a distribution, a fixed interval, or a numeric variable in a SAS data set.

The Trigger Interval field provides a mechanism for disabling the trigger for some transactions. If the trigger interval is one, then every other transaction will activate the trigger logic. If the trigger interval is two, then every third transaction will activate the trigger logic, and so on.

The Trigger Value push button provides a mechanism for associating a value with the trigger. This is used with the “setFromTrigger” trigger message in Holders.

The following documents the logic of the Trigger component.

**Transaction Arrival**

executes the trigger; then the transaction flows down each arc directed away from the component.

**Request for Transaction**

The request is sent up each arc directed into the component.

**Are You Busy Message**

If any of the components on arcs directed out of the Trigger is busy, then return TRUE; else, return FALSE.

**Query Message**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
<tr>
<td>value</td>
<td>returns the value associated with the Trigger</td>
</tr>
</tbody>
</table>

**Trigger Message**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Trigger</td>
</tr>
<tr>
<td>start</td>
<td>starts the Trigger</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Trigger</td>
</tr>
</tbody>
</table>
**MultiTrigger Component**

When a transaction arrives at a MTrigger component, it initiates the sending of a set of messages to a set of components, one message to each component. For example, Figure 2.7 shows the control panel for a MTrigger component.

Notice that the Triggers list box contains two entries, one labeled Server $\rightarrow$ preempt and the other labeled Trigger. The first one indicates that an arriving transaction will cause the “preempt” message to be sent to Server. The second one, labeled Trigger, has not been specified but is selected. Selecting this and pressing the **Edit** button raises the second window which looks like the Trigger Control Panel. Notice that the Server named Server is selected in that window. Also notice that **seize** has been selected. This means that the second message triggered by an arriving transaction will send the “seize” message to the component labeled Server.

The following documents the logic of the MTrigger component.

**Transaction Arrival**

sends each message to the appropriate component, then the transaction flows down each arc directed away from the component.

**Request for Transaction**

The request is sent up each arc directed into the component.

**Are You Busy Message**

If any of the components on arcs directed out of the MultiTrigger is busy, then return TRUE; else, return FALSE.

**Query Message**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
</tbody>
</table>
**Trigger Message**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Multi-Trigger</td>
</tr>
<tr>
<td>start</td>
<td>starts the Multi-Trigger</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Multi-Trigger</td>
</tr>
</tbody>
</table>

**Modifier Component**

The Modifier component assigns an attribute to a transaction. The control panel, shown in Figure 2.8, provides a field for entering the attribute name and a set of radio buttons for specifying how a value for that attribute is calculated. It can be the result of a simple character or numeric assignment, a formula evaluation, sampling of a random variable, or the value read from a variable in a data set. Regardless, the result of evaluation is the value given to the attribute. This attribute-value combination is unique to the transaction, and the transaction carries it on its route through the simulation network.

![Modifier Control Panel](image)

**Figure 2.8.** The Modifier Control Panel

By default, the value is calculated when the transaction arrives at the Modifier. However, if you select the Delay Formula Evaluation check box, the formula is not evaluated when the transaction arrives at the modifier but is itself the value of the attribute. This feature is used with the “filter” trigger message on queues.

The following documents the logic of the Modifier component.

**Transaction Arrival**

assigns the attribute value pair; then the transaction flows down each arc directed away from the component.

**Request for Transaction**

The request is sent up each arc directed into the component.

**Are You Busy Message**

If any of the components on arcs directed out of the Modifier is busy, then return TRUE; else, return FALSE.
Chapter 2. Building a Model with Elementary Components

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Modifier</td>
</tr>
<tr>
<td>start</td>
<td>starts the Modifier</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Modifier</td>
</tr>
</tbody>
</table>

MultiModifier Component

The MModifier component assigns multiple attributes to a transaction. The control panel, shown in Figure 2.9, provides a field for entering the attribute name and an Ok push button for adding the attribute to the attributes list.

![MultiModifier Control Panel](image)

Figure 2.9. The MModifier Control Panel

The list shows two attributes, named “class” and “priority.” By default, when a transaction arrives at the component, each of the attributes is assigned a value as is done in the Modifier component. You specify the details of how each attribute is evaluated by selecting the attribute and pushing the Edit button. This raises a control panel like the one for the Modifier as shown above. These attribute-value combinations are unique to the transaction, and the transaction carries them on its route through the simulation network.

The following documents the logic of the MModifier component.
Chapter 2. Building a Model with Elementary Components

Transaction Arrival

assigns the attribute; then the transaction flows down each arc directed away from the component.

Request for Transaction

The request is sent up each arc directed into the component.

Are You Busy Message

If any of the components on arcs directed out of the Modifier is busy, then return TRUE; else, return FALSE.

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Multi-modifier</td>
</tr>
<tr>
<td>start</td>
<td>starts the Multi-modifier</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Multi-modifier</td>
</tr>
</tbody>
</table>

Switch Component

The Router and Switch components are for controlling the flow of transactions as a function of the state of the simulation. The Router can have a formula associated with each arc directed away from it. When a transaction arrives at the Router, each formula is evaluated and the transaction flows down all arcs with formulas that evaluate to TRUE. The Switch is similar to the Router, but it has only one formula associated with it. The formula evaluation is interpreted as a case, which identifies an arc or set of arcs down which the transaction should flow. If the evaluation does not identify a valid case, the transaction flows out the Balk node.
Figure 2.10. Switch Control Panel

Figure 2.10 shows the control panel for a switch connected to two queues as in Figure 2.11. Selecting the **Formula** button displays a Formula Manager Window (see Chapter 4). There you build, verify, and save the formula associated with the switch. When a transaction arrives at the switch, the formula associated with the switch is evaluated. This value is compared to each of the cases listed in the Switch control panel. The transaction flows down the arcs associated with each of the cases that match. You can associate arcs with a case by selecting a case and pressing the **Edit** button. This displays the Switch Control Panel, as shown in Figure 2.10. In this window you select one or more of the listed components. For example, a Switch can be used to direct transactions to the smaller of two queues.

Figure 2.11. Switch Controlled Queue Selection

The following documents the logic of the Switch component.

**Transaction Arrival**

evaluates the formula for the switch. The transaction flows down the arcs leaving the switch that have case values matching the formula evaluation. If there is no match, the transaction flows out the balk node.
Request for Transaction

The request is sent up each arc directed into the component.

Are You Busy Message

If any of the components on arcs directed out of the Switch is busy, then return TRUE; else, return FALSE.

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Switch</td>
</tr>
<tr>
<td>start</td>
<td>starts the Switch</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Switch</td>
</tr>
</tbody>
</table>

Router Component

The Router and Switch components are for controlling the flow of transactions as a function of the state of the simulation. The Router can have a formula associated with each arc directed away from it. When a transaction arrives at the Router, each formula is evaluated and the transaction flows down all arcs with formulas that evaluate to TRUE. The Switch is similar to the Router, but it has only one formula associated with it. The formula evaluation is interpreted as a case, which identifies an arc or set of arcs down which the transaction should flow. If the evaluation does not identify a valid case, the transaction flows out the Balk node.

The following documents the logic of the Router component.

Transaction Arrival

evaluates the formula for each arc leaving the router. If an evaluation returns TRUE, then the transaction flows down the associated arc.

Request for Transaction

The request is sent up each arc directed into the component.

Are You Busy Message

If any of the components on arcs directed out of the Router is busy, then return TRUE; else, return FALSE.
Chapter 2. Building a Model with Elementary Components

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Router</td>
</tr>
<tr>
<td>start</td>
<td>starts the Router</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Router</td>
</tr>
</tbody>
</table>

Adder Component

The Adder component is useful when you want to model the assembly process such as putting two parts together. The Adder guarantees that this will occur only when both of the parts are available.

The following documents the logic of the Adder component.

Transaction Arrival

If all components on arcs leading into the Adder can initiate flow, then initiate a transaction from each and generate a new transaction to travel down each arc directed away from the Adder; otherwise, the transaction flows out the balk node.

Request for Transaction

If all components on arcs leading into the Adder can initiate flow, then initiate a transaction from each and generate a new transaction to travel down each arc directed away from the Adder.

Are You Busy Message

If all components on arcs leading into the Adder can initiate flow, then return TRUE; else, return FALSE.

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Adder</td>
</tr>
<tr>
<td>start</td>
<td>starts the Adder</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Adder</td>
</tr>
</tbody>
</table>
Splitter Component

The transaction entering the Splitter leaves down all arcs away from the splitter. A single instance of the transaction leaves the Splitter multiple times so it is in multiple places at once. It appears that these are multiple copies but they all refer to the same transaction instance. This means that any change to the transaction can be detected in multiple places in the model. This is the same behavior as with a Port or Connector. The additional behavior that is provided by the Splitter comes when an “Are You Busy” message is sent to it. This behavior differs from both the Port and Connector.

The following list documents the logic of the Splitter component.

Transaction Arrival

The transaction flows down each arc directed away from the component.

Request for Transaction

If none of the components on arcs directed out of the Splitter is busy, then pass on the request to all components on arcs leading into the Splitter; else, deny the request.

Are You Busy Message

If any of the components on arcs directed out of the Splitter is busy, then return TRUE; else, return FALSE.

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>resets the Splitter</td>
</tr>
<tr>
<td>start</td>
<td>starts the Splitter</td>
</tr>
<tr>
<td>stop</td>
<td>stops the Splitter</td>
</tr>
</tbody>
</table>

Holder Components

There are two types of holders: StringHolder and NumberHolder. These are used to hold strings and numbers for maintaining user-defined state information.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>NumberHolder</td>
<td>storage for a number</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>StringHolder</td>
<td>storage for a string</td>
</tr>
</tbody>
</table>
NumberHolder Components

There are various ways to both save information and retrieve information from holders. The values of attributes carried by transactions can be saved in a holder when the transaction enters the holder. Alternatively, a value can be saved in a holder when a transaction enters a trigger in some other part of the simulation model. For example, suppose that you want to save the value of a transaction attribute called “weight” in a NumberHolder.

![Figure 2.12. Number Holder Saving “weight” Attribute](image)

In the model fragment in Figure 2.12, when the transaction arrives at the Trigger, the value of the weight attribute in that transaction is saved in the NumberHolder. Now, another part of the simulation can query the NumberHolder to find the current value of weight. You could also route the transaction directly to the NumberHolder and update its state that way.

You specify the name of the attribute that is stored in the NumberHolder in the NumberHolder Control Panel, which is displayed by selecting Control Panel... from the pop-up menu on the NumberHolder.

![Figure 2.13. Number Holder Control Panel](image)

Notice that the NumberHolder control panel shown in Figure 2.13 has the attribute name “weight” in the Attribute Name field.
Chapter 2. Building a Model with Elementary Components

The transaction sets the NumberHolder when it arrives at the Trigger because the Trigger Control Panel, as shown in Figure 2.14, has the NumberHolder component selected and the setFromAttribute selected.

![Figure 2.14. A Trigger Control Panel](image)

The **Initial Value** field in the holder control panel provides a way of initializing the holder. This is useful when using the holder as a counter of resources. When the NumberHolder decrements, there is one less available resource. Other parts of the model may query the holder to see if there are resources available for certain activities. In this case it may be desirable to have an initial pool available.

The **Disable Reset** check box will disable the resetting of the last value in the holder when the simulation reset button is pressed. If not checked, when the reset button is pressed the holder is reset to its initial value. If checked, the reset button has no effect on the holder.

The following documents the logic of the NumberHolder component.

**Transaction Arrival**

sets the value as specified in the NumberHolder Control Panel; then flows the transaction to each arc directed away from the component.

**Request for Transaction**

passes on the request to all arcs directed into the component.

**Are You Busy Message**

If any component on an arc leading out of the Trigger is busy, then return TRUE; otherwise, return FALSE.
Chapter 2. Building a Model with Elementary Components

Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentValue</td>
<td>returns the value in the holder at the current time</td>
</tr>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
<tr>
<td>value</td>
<td>returns the value in the holder when the transaction passed through it</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>adds the transaction attribute to the value</td>
</tr>
<tr>
<td>-</td>
<td>subtracts the transaction attribute from the value</td>
</tr>
<tr>
<td>clearSetFromAttribute</td>
<td>clears the value then sets it</td>
</tr>
<tr>
<td>controls</td>
<td>displays the Holder Control Panel</td>
</tr>
<tr>
<td>decrement</td>
<td>decrements the value</td>
</tr>
<tr>
<td>increment</td>
<td>increments the value</td>
</tr>
<tr>
<td>print</td>
<td>prints the value on the SAS Log window</td>
</tr>
<tr>
<td>reset</td>
<td>resets the value</td>
</tr>
<tr>
<td>setFromAttribute</td>
<td>sets the value from the transaction attribute</td>
</tr>
<tr>
<td>setFromTrigger</td>
<td>sets the value that is assigned with the Trigger Value button in the Trigger Control Panel</td>
</tr>
<tr>
<td>setTimenow</td>
<td>sets the simulation time into the value</td>
</tr>
<tr>
<td>start</td>
<td>starts the holder</td>
</tr>
<tr>
<td>stop</td>
<td>stops the holder</td>
</tr>
</tbody>
</table>

StringHolder Components

The following list documents the logic of the StringHolder component.

Transaction Arrival

sets the value as specified in the StringHolder Control Panel; then flows the transaction to each arc directed away from the component.

Request for Transaction

passes on the request to all arcs directed into the component.

Are You Busy Message

If any component on an arc leading out of the Trigger is busy, then return TRUE; otherwise, return FALSE.
Query Message

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentValue</td>
<td>returns the value in the holder at the current time</td>
</tr>
<tr>
<td>id</td>
<td>returns the component’s unique identifier</td>
</tr>
<tr>
<td>value</td>
<td>returns the value in the holder when the transaction passed through it</td>
</tr>
</tbody>
</table>

Trigger Message

<table>
<thead>
<tr>
<th>Trigger Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearSetFromAttribute</td>
<td>clears the value then sets it</td>
</tr>
<tr>
<td>controls</td>
<td>displays the Holder Control Panel</td>
</tr>
<tr>
<td>print</td>
<td>prints the value on the SAS Log window</td>
</tr>
<tr>
<td>reset</td>
<td>resets the value</td>
</tr>
<tr>
<td>setFromAttribute</td>
<td>sets the value from the transaction attribute</td>
</tr>
<tr>
<td>setFromTrigger</td>
<td>sets the value that is assigned with the Trigger Value button in the Trigger Control Panel</td>
</tr>
<tr>
<td>start</td>
<td>starts the holder</td>
</tr>
<tr>
<td>stop</td>
<td>stops the holder</td>
</tr>
</tbody>
</table>

Chart Components

There are five types of charts and a Bucket component. The charts are used to display information about the performance of the system. The bucket collects data. Charts can be used in two ways: you can drag and drop a component (queue, server, or NumberHolder, for example) on a chart and then choose the attribute of the component you want to display in the chart and the frequency with which to sample the component. In those cases, the chart will instantiate a bucket within it to collect the data. On the other hand, if you drop a bucket into a chart, the chart uses that instant as its data source. This is used to display attribute data in transactions.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucket</td>
<td>collect value of an attribute for analysis</td>
<td></td>
</tr>
<tr>
<td>V Histogram</td>
<td>vertical histogram of numeric data</td>
<td></td>
</tr>
<tr>
<td>HH Histogram</td>
<td>horizontal histogram of numeric data</td>
<td></td>
</tr>
<tr>
<td>V Box Plot</td>
<td>vertical box plot of numeric data</td>
<td></td>
</tr>
<tr>
<td>H Box Plot</td>
<td>horizontal box plot of numeric data</td>
<td></td>
</tr>
<tr>
<td>Line Plot</td>
<td>plot of numeric data over simulation time</td>
<td></td>
</tr>
</tbody>
</table>
Bucket

The bucket is a component for collecting statistics on an attribute and saving its values in a SAS data set. You name an attribute for which you want to collect statistics in the Bucket control panel. The attribute “age” is the default collected. You can also specify the buffer size, which is the number of values of the attribute that is used in calculating statistics and displaying in the chart component. There is also a way to name a data set into which the attribute’s values are saved and a way to start and stop data collection. In addition, the ‘Reset’ button empties the buffer and the ‘Ok’ button sets the buffer size and attribute names.

![Bucket control panel](image)

**Figure 2.15.** A Bucket control panel

When you select the Collect Data check box, for each transaction the values of the monitored attribute are saved in a SAS data set. Chapter 7, “Saving and Restoring,” discusses the details of this data set. You can analyze the data you collect by pressing the Analyze button which executes PROC UNIVARIATE and PROC G PLOT. Chapter 8, “Analyzing the Sample Path,” discusses the details of the type of analysis.

Box Plot

The Box Plot shows the minimum, maximum, and quartiles of the attribute that it is monitoring. This attribute can be named in a bucket that is dropped on the box plot or can be one of the states of a component that is dropped on the box plot. If a bucket is dropped on the box plot then the bin controls for the box plot are those of the bucket. If a component is dropped on the box plot then there is a hidden bucket associated with the box plot and the bin controls are associated with the box plot and are accessible from the box plot control panel.
Figure 2.16. A Box Plot Control Panel

The box plot control panel has buttons to enable or disable automatic scaling, refresh the plot, raise the local bin controls, and to specify auxiliary controls. The latter two buttons apply if you drop a component on a box plot.

You can also drop box plots onto other box plots. This enables you to collect statistics on the minimum, maximum, and quartiles of the attribute. It is a type of batch means.

Histogram

The histogram shows a distribution histogram of the selected attribute. The Histogram control panel shows the minimum, maximum, number, mean, and standard deviation of the attribute that it is monitoring. This attribute can be named in a bucket that is dropped on the histogram or can be one of the states of a component that is dropped on the histogram. If a bucket is dropped on the histogram, then the bin controls for the histogram are those of the bucket. If a component is dropped on the histogram, then the bin controls are associated with the histogram and are accessible from the histogram control panel.

Figure 2.17. A Histogram and Control Panel
Chapter 2. Building a Model with Elementary Components

The histogram control panel has buttons to enable or disable automatic scaling, refresh the plot, raise the local bin controls, and to specify auxiliary controls. The latter two buttons apply if you drop a component on a histogram.

You can also drop histograms onto other histograms. This enables you to collect statistics on the minimum, maximum, mean, and standard deviation of the attribute. It is a type of *batch means*.

**Line Plot**

The line plot shows the attribute’s sample path. The line plot control panel shows the minimum, maximum, number, mean, and standard deviation of the attribute that it is monitoring. This attribute can be named in a bucket that is dropped on the line plot or can be one of the states of a component that is dropped on the line plot. If a bucket is dropped on the line plot, then the bin controls for the line plot are those of the bucket. If a component is dropped on the line plot, then the bin controls are associated with the line plot and are accessible from the line plot control panel.

**Figure 2.18.** A Line Plot of Transaction Age

The line plot control panel has buttons to enable or disable automatic scaling, refresh the plot, raise the local bin controls, and to specify auxiliary controls. The latter two buttons apply if you drop a component on a line plot.

**Figure 2.19.** A Line Plot Control Panel

You can also drop line plots onto other line plots. This enables you to collect statistics on the minimum, maximum, mean, and standard deviations of the attribute. It is a type of *batch means*. 
Port, Connector, and Label Components

Ports and Connectors aid in connecting components to each other and are useful when building hierarchical models and assembling components into larger aggregate components. You can annotate the simulation with text using Labels. In addition, you can attach labels to many of the elementary components. You do this by selecting Tools ▶ Add label from the pop-up menu on the component you want to annotate. Then, type the text you want to appear in the label.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Port Icon" /></td>
<td>Port</td>
<td>for connecting multiple components</td>
</tr>
<tr>
<td><img src="image" alt="Connector Icon" /></td>
<td>Connector</td>
<td>for connecting multiple components without using arcs</td>
</tr>
<tr>
<td><img src="image" alt="Label Icon" /></td>
<td>Label</td>
<td>for annotating the model</td>
</tr>
</tbody>
</table>

Figure 2.20 shows an example with a connector labeled “a.” When a transaction flows into connector “a,” it will flow out of all other connectors “a.”

![Figure 2.20. Simple Example Using a Connector](image)

The three “a” connectors are treated as identical. An equivalent model using an instance of a Port is shown in Figure 2.21.

![Figure 2.21. Simple Example Using a Port](image)

Here, the port explicitly connects the three components, which were implicitly connected using the Connector. In addition, ports have a special role in compound components. In this setting, they can be used to create special connections from the outside of compound components to the inside of compound components. See Chapter 5, “Building a Model with Compound Components,” for more details on this special function.
You duplicate a Connector by selecting **Duplicate** on the pop-up menu. If you type in the interior of the Connector, then all the duplicates of that connector will display the same text.

## Connecting Components

The examples presented thus far use arcs to connect components. This section describes arcs and some of the features they provide in more detail. If you click on the right side of a component, when the cursor is in the “+” shape, a rubberband line displays from the component to the cursor. If you don’t see this line, it means that the component doesn’t support arcs directed away from it. You will not get an error message. If there is a rubberband line attached to the cursor, when you click in another component that supports arcs directed towards it, the rubberband line is replaced by a solid arc. If, while the rubberband line is connected to the cursor, you move the cursor to the right or below the window border, the window will scroll automatically. This is power scrolling, and it allows you to connect components that may not be visible in the window simultaneously. If you click outside the window border while power scrolling, then the rubberband line is dropped.

There are two types of arcs: regular arcs and segmented arcs. As the name implies, segmented arcs are composed of multiple line segments. **Figure 2.22** shows the two types of arcs.

![Figure 2.22. Two Types of Arcs](image)

If you click on the simulation window background while a rubberband line is connected to the cursor, the selected point ends one line segment and begins another. In this way you can create circuitous routes between components.

Notice the rectangular handle in the center of the arc. This is the arc’s “hot spot.” If you click the right mouse button while the cursor is over the hot spot, a menu associated with the arc pops up.
Figure 2.23. Pop-Up Menu for a Segmented Arc

In the pop-up menu for segmented arcs in Figure 2.23, there is a selection called **Perpendicular**. This selection causes the arc to be drawn so that the line segments are perpendicular to each other. As the numerous selections in Figure 2.23 show, a full range of capabilities are available.
Transactions are generated in three components: Sampler, Transaction Pool, and Adder. Transactions are discrete entities that traverse the simulation network, and they can be used to represent physical and conceptual things such as a partially assembled refrigerator on an assembly line, a telephone call in a phone system, or an event such as a check on the size of a queue.

You can view the movement of transactions through the network by pressing the Animate button on the command panel. Bitmap images can be assigned to transactions in each of the three components in which they are generated. For example, the Transaction Image push button on the Sampler control panel opens a window from which you can choose an image to display on the transactions that originate at that Sampler. Figure 3.1 shows an example. Although Figure 3.1 shows the image SASHELP.ORIMG24.PHONE, any image in a SAS catalog can be used.

When there are no references to transactions in the simulation, the transaction is disposed of. This means that if a transaction does not reside in a queue, is busy in a server, is traversing an arc, or is being processed by another elementary component, it will be disposed of. You do not have to explicitly destroy transactions.

Attributes

You assign an attribute to a transaction in the Modifier component. There are three types of attributes: numeric, character, and unevaluated formulas. An attribute is attached to the transaction for the duration of the transaction, but it can be given another value at any time.

History

The transaction maintains a history of the time it has spent traversing the network. This history is saved in transaction attributes. For example, when a transaction enters a queue, it records the time it entered the queue in an attribute that is named in the
queue control panel (see Figure 2.5 on page 24). When the transaction leaves the queue, it updates the attribute with the total time it spent in the queue. When a transaction enters a Bucket, if the Collect Data check box is set (see Figure 8.8 on page 81), then any attribute in its history can be written to a SAS data set. You have to name the attribute for which you want to save data in the bucket control panel.

By default, the total age of the transaction is also a part of its history. This is the quantity that is, by default, collected in Buckets. You can access it with the attribute named “age.”

It is important to note that the transaction only spends simulation time in queues and servers. All the real time that is spent in other components and traversing arcs do not result in passage of simulation time.

**Trigger Messages**

The Trigger component includes an entry labeled “Transaction” in the Components list. Figure 3.2 shows a Trigger control panel with this entry selected.

![Trigger Control Panel]

Notice the entries in the Trigger list box. These entries are possible behaviors to trigger when a transaction arrives at the Trigger component. In this example, because Transaction is selected, the target of the trigger is the transaction itself. The removeFromServers action causes the transaction to send that message to all servers in the simulation which in turn causes removal of that transaction from any servers in which it is receiving service. The other actions are described in the following list.
**removeFromServers** removes the transaction from any server by sending the *removeIt* message to all servers.

**removeFromQueues** removes the transaction from any queue by sending the *balk* message to all queues.

**routeToId** routes the transaction to the component that has id equal to the value of the *routeToId* attribute attached to the transaction.

Note that when using the *routeToId* trigger, you need to assign an attribute to the transaction with the name “routeToId” and the value the id of a component. You could assign that id as a numeric attribute when the transaction is in a modifier. This approach may not work if you save the model and then subsequently reload it because the id number of the component may change between the save and the load. You can avoid this problem by assigning the routeToId attribute as the “id” of a model component.

These options provide you additional modeling flexibility. Two examples, “Servers as Resources II” and “Special Routing” in Chapter 9, “Selected Examples,” illustrate two uses of the routeToId trigger.

---

**Timing Transaction Arrivals**

There are some issues concerning timing of transaction arrivals at a component. Consider model fragment A in Figure 3.3.

![Figure 3.3](image)

**Figure 3.3.** Model Fragment A

Although transactions leaving the multiple-server will arrive at the Trigger and the Bucket at the same simulation time, the logic and behavior of each of these components will be executed in sequence. There is no guarantee which will occur first.

![Figure 3.4](image)

**Figure 3.4.** Model Fragment B

To guarantee that the Trigger “Server Off” executes before the transaction traverses to the Bucket, connect the components as in Figure 3.4.
Chapter 4
Formulas

The Router, Switch, and Modifier components use formulas to control routing and set attributes. Formulas are specified in a Formula Manager window. Figure 6.6 shows one such window with the formula $5.0 + X$, where $X$ is a random variable with distribution $E(1)$, exponentially distributed with mean 1.

Figure 4.1. Formula Manager Window

Formulas are built using an interface that is similar to that used when building models. The palette has a set of icons that can be dragged and dropped into the Formula Manager window. You use these to build expressions. The following list shows some of the elements that are used to build expressions.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="another formula" /></td>
<td>another formula</td>
</tr>
<tr>
<td><img src="image" alt="a transaction attribute" /></td>
<td>a transaction attribute</td>
</tr>
<tr>
<td><img src="image" alt="the simulation clock" /></td>
<td>the simulation clock</td>
</tr>
<tr>
<td><img src="image" alt="a component in the model" /></td>
<td>a component in the model</td>
</tr>
<tr>
<td><img src="image" alt="an observation of a variable in a SAS data set" /></td>
<td>an observation of a variable in a SAS data set</td>
</tr>
<tr>
<td><img src="image" alt="a number constant" /></td>
<td>a number constant</td>
</tr>
<tr>
<td><img src="image" alt="a string constant" /></td>
<td>a string constant</td>
</tr>
</tbody>
</table>
As with components, each of these elements has a pop-up menu associated with it. For example, the random variable icon shown in Figure 6.6 has a pop-up menu, shown in Figure 4.2.

**Figure 4.2.** Pop-up Menu on the Random Variable Formula Element

If you select **Edit**, the window shown in Figure 4.3 is displayed. From this window, you select the distribution for the random variable. When the formula is evaluated (to set an attribute or determine transaction routing, for example), an observation of the random variable is made.

**Figure 4.3.** Control Panel for the Random Variable Formula Element

Another important formula element is the model element. Recall the example discussed in the “Switch Component” section on page 31. There, the model in Figure 2.11 on page 32 routes transactions to the shortest queue. The control panel on the Switch shown in Figure 2.10 on page 32 has TRUE as a case for routing to component Queue 1. This means that you want the formula associated with this switch to return TRUE if Queue 1 is shorter than Queue 2. To accomplish this, you compare the sizes of the two queues and return TRUE if the size of Queue 1 is less than the size of Queue 2.

**Figure 4.4.** Formula for Comparing the Size of Two Queues

Figure 4.4 shows the formula for accomplishing this. The edit window for the left-hand model element is shown in Figure 4.5.
Chapter 4. Formulas

Figure 4.5. The Model Element Edit Window

The Model list box contains the simulation components. When you select one of these components, the Query list box displays query messages for that component. When the formula is evaluated, the result of the query of the identified component will be used. For example, when the formula shown in Figure 4.4 is evaluated, the model element will query Queue 1 with the message size, which will return the number of elements in its queue. Similarly, if you edit the model element on the right-hand side of the > and set it to query size of Queue 2, the formula will evaluate to TRUE if Queue 1 has fewer elements than Queue 2. In this case, the switch will send the transaction to Queue 1.

You should be careful when validating your models that they are behaving as you want them. There is a great amount of error-checking done when formulas are evaluated, particularly with regard to type, but not every error will be detected. For example, a Router may expect a TRUE or FALSE returned from a formula, but you could enter a valid formula that evaluates to a number or a string. The QSIM Application would not detect this type of error but would function as if a nonzero number were TRUE.

The formula syntax is standard with two exceptions. First, functions of one argument, such as the trigonometric functions, are specified in reverse polish style. Thus, an expression like \( \log(X) \) would be displayed as in Figure 4.6.

Figure 4.6. A Formula for the Log of a Random Variable

Functions of two arguments, such as max and min, are specified between the arguments. So, an expression like \( \max(X, \pi) \) would be displayed as in Figure 4.7.

Figure 4.7. A Formula for the Maximum of a Random Variable and \( \pi \)
Chapter 5
Building a Model with Compound Components

The ability to assemble elementary components into larger aggregates is an important feature of the QSIM Application. It encourages hierarchical model building, information hiding, and component reuse. This chapter discusses the details of model building using compound components.

Assembling Components into Compound Components

The pop-up menu on the Simulation window (see Figure 5.2) has as its first entry Assemble Components. When you choose this selection, you get a rubberband rectangle with which you can sweep out an area on the Simulation Window. Any components that are completely within this region will be encapsulated in a compound component.
Figure 5.2. The Simulation Window Pop-up Menu

Figure 5.1 shows an example of a compound component. This encapsulated queue and server can now be treated as a single unit, called a compound component. It has a pop-up menu, shown in Figure 5.3, which includes Assemble Components and Disassemble Components for assembling additional compound components within it and for removing the encapsulation.

Figure 5.3. The Compound Component Pop-up Menu

This compound component also includes Edit, a selection that opens a separate window for editing the contents of the compound component, and Expose/Hide Details, a selection that toggles the compound component with an icon. The default icon, can be replaced with a user icon by using the Component Attributes Panel selection on the Command Panel (see Figure 8.2 on page 76).
You can also pick up components and move them into and out of compound components. Compound components themselves can be moved into and out of compound components as well. And, it is not necessary to use drag and drop to accomplish this. You simply move the component by pressing the mouse button down as you scroll over the component when you see the hand icon. When you release the mouse button, the component will be in the new position within the compound component.

**Nested Compound Components**

It is often useful to nest compound components within one another when making reusable components. One issue that arises when doing this is how to expose externally connections to internal component logic. The Port component is particularly helpful to accomplish this when nested compound components are built for subsequent reuse. Consider an example of a queue-server combination. By adding two Ports, one for transactions to enter the queue and the one for transactions to leave after service, and then assembling the ports and the compound component into a larger unit, you can further encapsulate the queue-server combination as shown in Figure 5.4.

![Figure 5.4. A Multi-Level Compound Component with Queue, Server, and Ports](image)

**Controlling Subcomponent Exposure**

Another feature that helps in building hierarchical models is the ability to limit the exposure of compound components. Even though the model in Figure 5.4 is nested and allows you to connect to ports, it still exposes submodel detail. However, the pop-up menu on the compound component also has an entry labeled *Expose/Hide Detail*. Selection of this entry hides the detail in the compound component behind the icon. With the inner compound component hidden, the model shown in Figure 5.4 looks like Figure 5.5.
Chapter 5. Building a Model with Compound Components

Figure 5.5. A Multi-Level Compound Component with Ports

You can expose the detail in the inner Queue/Server component by selecting Expose/Hide Detail again on the component labeled Logic. With these features, you assemble components with complex behavior, attach ports to the substructures, then hide the detail, exposing only the structure necessary for using the compound component.

Not only can you hide the visual details of compound components, but you can also hide the details of the logic in a compound component. Just hide the details of the Logic component in Figure 5.5 and you obtain the example shown in Figure 5.6.

Figure 5.6. A Multi-Level Compound Component with Details Hidden

Notice that the Ports remain exposed. That is because the Toggle Exposure was selected from the pop-up menu on the Port as shown in Figure 5.7.

Figure 5.7. A Multi-Level Compound Component with Queue, Server, and Ports

The pop-up menu on compound components in Figure 5.3 shows the selection Select Components... When this is selected for the component labeled Logic, the window in Figure 5.8 is displayed. In this window, you can select those components that you want exposed when queries are made on the Logic compound component.
Chapter 5. Building a Model with Compound Components

Initially, the Exposed Components list box contains all the components in the Logic compound component. These are the only components that will be exposed. You move components between the Exposed Components and Non-Exposed Components list boxes by simply selecting them. Notice that only FIFO Queue will be exposed. For example, consider the model shown in Figure 5.9.

If you select the Controls... from the pop-up menu on the Queue-Server Component then only the FIFO Queue Control Panel will be raised, as shown in Figure 5.10.
This is because you had previously (in Figure 5.8) selected only that element to be exposed.

**Editing Components**

Suppose that you want to edit the Logic component in the model in Figure 5.9. One way is to visually expose the detail in an edit window by selecting Edit... from the pop-up menu on compound components (see Figure 5.3). Figure 5.11 shows such an edit window for the Logic component.

In this new window, you can modify the detail structure of the Logic compound component.
Palettes for Reusing Components

Another powerful feature of compound components is the ability to use them as templates for replication. You do this by simply dragging and dropping them into a palette. Notice the compound component at the bottom of the palette in Figure 5.12. Since this is in the palette and is a template, it can be replicated.

Figure 5.12. The Toolbar with the New “Logic” Button

Now, this icon can be replicated by simply dragging it onto the Simulation window as you would any other component in the palette. You can change the image on the icon by using the pop-up menu on the icon and selecting Graphic Attributes. Furthermore, this icon or template can be saved in a SAS dataset by saving the palette. See Chapter 7, “Saving and Restoring,” for details.

Note that it is possible to create compound components and, as a result, templates, that reference components not in the compound component and template itself. For example, you can have a Trigger as a template that references a queue not in that template. When the template is replicated, the Trigger will no longer reference the queue, and its reference to a component in the model will have to be re-established. However, any references to components within the template will be preserved in replicates.
Chapter 6
Random and Exogenous Variation in the Model

Random and exogenous sources of variation play a central role in discrete event simulation. The Sampler, Server, MServer, and formulas are the principle sources of this variation in the QSIM models. However, there are other situations where variation can be meaningfully incorporated into models. For example, you may want to delay the effect of a trigger for some random or fixed amount of time. In each of these cases you have access to a combo box, such as the one in the Sampler control panel shown in Figure 6.1, labeled to show the use of the source of variation. In this case, it is the time between transaction arrivals to the system. If you click on the down arrow, a list that includes the possible distributions is displayed.

![Figure 6.1. Sources of Variation](image)

When you select one of these distributions, the selection is displayed in the text area to the left of the down arrow. For each distribution you select, you can set one or more parameters which further define the choice.

There are two types of sources of variation: random and exogenous. These are not mutually exclusive. That is, an exogenous source of variation can be randomly generated.
Chapter 6. Random and Exogenous Variation in the Model

Random Sources of Variation

The random sources of variation are generated using pseudo-random number generators. The QSIM application provides a set of standard generators. These include:

- **Exponential**, with parameter \( \lambda \) and density function

\[
f(x) = \lambda \exp^{-\lambda x}, \quad \text{for } x \geq 0, \lambda \geq 0
\]

- **Nonhomogeneous Poisson**, with rate parameter \( \lambda(t) \) and density function

\[
f(x) = \frac{1}{\lambda(t)} \exp^{-x/\lambda(t)}, \quad \text{for } x \geq 0, \lambda(t) \geq 0
\]

where \( \lambda(t) \) is cyclic and continuous for all \( t \geq 0 \) with

\[
\lambda^* = \sup_{t \geq 0} \lambda(t)
\]

- **Gamma**, with parameters \( \lambda \) and \( n \) and density function

\[
f(x) = \frac{x^{n-1} \exp^{-x/\lambda}}{\lambda^n \Gamma(n)}, \quad \text{for } x \geq 0, \lambda > 0, n > 0
\]

- **Erlang**, with parameters \( \lambda \) and \( n \) and density function

\[
f(x) = \frac{x^{n-1} \exp^{-x/\lambda}}{\lambda^n \Gamma(n)}, \quad \text{for } x \geq 0, \lambda > 0, n \in \mathbb{Z}^+
\]

- **Uniform**, with parameter \( U \) and density function

\[
f(x) = \frac{1}{U}, \quad \text{for } U \geq 0, 0 \leq x < U
\]

- **IUniform**, with parameter \( U \) and density function

\[
f(x) = \frac{1}{\lfloor U \rfloor}, \quad \text{for } U \geq 1, 0 \leq x \leq \lfloor U \rfloor, x \in \mathbb{Z}^+
\]

- **Deterministic**, with parameter \( U \) and density function

\[
f(x) = 1, \quad \text{for } x = U
\]

Each of these generators has a control panel for setting parameters. For example, if you pick Exponential from the list box in Figure 6.1 and then click the **Parameters** button, the exponential control panel in Figure 6.2 will be displayed. In this window, you set the initial seed value and the mean \( 1/\lambda \).
Chapter 6. Random and Exogenous Variation in the Model

Figure 6.2. The Exponential Control Panel

The seed value for each distribution is initialized automatically. You have the capability of changing this value. The next section tells you how you can control seed values.

Note that, by default, the Exponential mean in Figure 6.2, is set to 1. You can change the mean with the slider. The slider has a range of 0 to 10 for this parameter. If 10 is not large enough for the mean, you simply click in the display and type the mean that you want.

Controlling the Seed Values

The QSIM application gives you considerable control over the streams of random numbers used in simulations. Each instance of a random number has its own unique generator. The control panel for each provides a mechanism for setting the seed and time units associated with the random variable. The exponential control panel in Figure 6.2 shows this. By default, a seed is selected when the component is instantiated. This seed is selected from a seeds dataset, named QSEEDS, in the SASHELP library and selected to guarantee no overlap with any other random variable seed values for 100,000 observations. Moreover, each time you run the QSIM Application, the selection of seeds continues from where it last left off. You can control these by raising the Seed Control panel by selecting the Options ➤ Seeds... from the pull-down menus (see Figure 6.3) on the Simulation window.

Figure 6.3. The Options Pull-down Menu

In the Seed Preferences window, shown in Figure 6.4, you can increase the default distance between seeds, and when you click the Reset Seeds button, you reset the seeds to the original values that were automatically assigned when the components were instantiated. This enables you to rerun the simulation with the same seed values used initially.
Chapter 6. Random and Exogenous Variation in the Model

Figure 6.4. The Seed Control Panel

The **Reset Seed Stream** button resets the seed stream to the beginning of the QSEEDS dataset. This will result in use of the seeds that were used when the application was first installed on your computer. You can also edit the QSEEDS dataset in the SASHELP directory to change the seed values, although this is not recommended.

Controlling Time Units

Each of the random variable control panels also shows a Units combo box. See Figure 6.2 for an example. With this you can identify a value for the time units. Then, with the Clock Options window, (see Figure 6.5) accessed by selecting **Options ➤ Clock Options...** from the Simulation window (Figure 6.3), you can assign time units to be used for the simulation clock.

Figure 6.5. The Clock Control Panel

If you select time units for the simulation of seconds and an inter-arrival time distribution for some random variable on a Sampler has time units of minutes, then the simulation will automatically perform the conversion in the sampling.

FormulaDistribution

QSIM provides a general function writing capability with the **FormulaDistribution** selection, shown in the list box in Figure 6.1. After you select it and then click the **Parameters** button, a Formula Manager window (as shown in Figure 4.1 on page 51) is displayed. From this window, you can write a function that returns a number that is used as the sample. For example, suppose you wanted a mixture distribution with density function

\[ f(x) = 0.5f_1(x) + 0.5f_2(x) \]
where \( f_1(x) \) is exponential with parameter \( \lambda \), and \( f_2(x) \) is uniform. Figure 6.6 shows a portion of the four windows needed to express this.

![Figure 6.6. An Example of a Mixture Distribution](image)

The bottom window is the FormulaDistribution Formula window, and it contains the conditional element \(?<\). Editing this element provides you with the capability to specify the condition \( X < 0.5 \). If the condition evaluates to TRUE, it returns the formula in the True Expression window \( X \), which is the appropriate exponential random variable; if it evaluates to FALSE, it returns the formula in the False Expression window \( X \), which is the uniform random variable.

**DataSource**

You have another opportunity to customize the source of variation with the **DataSource** selection shown in Figure 6.1. With this choice, you can select a SAS data set and a numeric variable. Whenever a sample is needed, an observation is read from the data set and the value of the selected variable is used as the sample value. You choose the data set by clicking the **Parameters** button, which displays the DataSource Control window in Figure 6.7.
Chapter 6. Random and Exogenous Variation in the Model

Suppose that you had executed the following DATA step, which sampled 10,000 observations from the mixture distribution with density function

\[ f(x) = .5f_1(x) + .5f_2(x) \]

where \( f_1(x) \) is exponential with parameter 1 and \( f_2(x) \) is uniform.

```
data mixture;
  keep sample;
  do i = 1 to 10000;
    if ranuni(123456789) < .5 then
      sample = ranexp(98765432);
    else sample = ranuni(54321678);
  output;
end;
```

Then, the selections highlighted in Figure 6.7 would result in the sample being used. Note that the simulation may require more than 10,000 observations of the random variable with this mixture density. In this case, the DataSource will rewind the data set to the beginning and reuse the sample. If this feature is not accounted for, it could result in some unexpected and incorrect estimates of performance measures.
Chapter 7
Saving and Restoring

There are several ways to save information about the simulation. You can save a picture of the model as a SAS Graph object, save the sample path of a simulation, and save a representation of the simulation model or a piece of the model. Each of these saved representations can be reused, each in its own special way.

![Figure 7.1. The Save As Selection on the Simulation Window](image)

**Figure 7.1** shows the pull-down menu on the Simulation window from which you choose whether you want to save a graph or a model.

**Models**

A model of the simulation includes the values of all parameters and settings, and the logical and physical structure of the model. It does not include information on transactions in service or those waiting for service, nor does it include information on which windows and panels are open. When you select **Model...** from the pull-down menu in Figure 7.1, the Save Model window in Figure 7.2 opens. The model is saved in a SAS data set whose name you enter in the Save Model window. You can restore a model by selecting **Open...** from the pull-down menu in Figure 7.1.

![Figure 7.2. Save the Simulation Model](image)
Since a model is saved in a SAS data set, it is possible to edit the model. However, this should not be done since it will result in errors when reading the model back into the simulation.

**Palettes**

The palette contains the default components that are discussed in Chapter 2, “Building a Model with Elementary Components,” and any compound components that you have assembled and added. As discussed above, these are dragged and dropped into the Simulation window when building simulation models. There are several features for manipulating and maintaining palettes.

**Figure 7.3.** The Simulation File Pull-down Menu

Figure 7.3 shows the **File ➤ New Palette...** pull-down menu on the Simulation window. This will open a new palette. You can have multiple palettes open at any time. The **File ➤ Clear Palette** pull-down menu on the palette, shown in Figure 7.4, empties the palette of all components. This enables you to bring up a new palette, clear it, and add whatever components you have built to it. You have created a customized palette.

**Figure 7.4.** The Palette File Pull-down Menu

The palette **File** pull-down menu also has an **Open...** and **Save...**. With these selections, you can save all the templates in a palette into a SAS data set which can then, in another invocation of the QSIM Application, be opened and loaded into a palette.
Figure 7.5.  Save Palette Window

In Figure 7.5, the palette is saved into a data set named SASUSER.BASE.

Figure 7.6.  Load Palettes Window

In Figure 7.6, SASUSER and BASE are selected so that when the Ok button is pushed, the palette will be populated with the templates that were saved in SASUSER.BASE.
Chapter 7. Saving and Restoring

Graphs

A graph is a visual representation of the model. There is no structural information about the model saved in the graph of the model. When you select **Graph** from the pull-down menu in Figure 7.1, the Save Graph window opens.

![Save Graph Window](image)

**Figure 7.7.** Save the Graph of the Simulation

In this window you enter the name of the entry in a catalog into which the graph representation is stored as a SAS/GRAPH grseg.

Sample Paths

There are two ways of accumulating sample path data. One is from the perspective of the components, and the other is from the perspective of the transactions. Each of these stores the sampled data in a SAS data set that can be analyzed by your own programs. One relies on another SAS data set named DICT, which contains a dictionary for tying together components and unique component IDs.

Unlike the other pieces that are saved (models, templates, and graphs), the sample paths cannot be reconstituted by the QSIM application or by other applications or procedures within the SAS System. However, this information can be useful in user-written SAS programs to further analyze these data.

The Component Dictionary Data Set

The component dictionary data set is always in WORK.DICT. It has two variables, CMPONENT and ID. **Figure 7.8** shows a printout of the data set for a simple M/M/1 model.

<table>
<thead>
<tr>
<th>Obs</th>
<th>component</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sampler</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>FIFOQueue</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Server</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 7.8.** An M/M/1 Example Dictionary Data Set

The value of the CMPONENT variable will be whatever label is associated with the component. Note that only elementary components will appear in this data set.
Component State Sample Paths

The component state information is saved in the data set WORK.SAMPLE. For each change of state in the simulation, a new observation is added. This data set contains three variables: TIMENOW, the simulation time when the state change occurred; ID, the id of the component that is changing state; and STATE, the value of the new state. Figure 7.9 shows an example. Notice that the ID corresponds to either the server or the queue as given in the dictionary in Figure 7.8.

<table>
<thead>
<tr>
<th>Obs</th>
<th>TIMENOW</th>
<th>ID</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.18227</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.65860</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1.24356</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1.24356</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1.24356</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1.52718</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1.92666</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2.40294</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4.15565</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4.53893</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 7.9. An M/M/1 Component State Sample Path Data Set

Sample path data are not saved to WORK.SAMPLE until the Collect Data check box in the Simulation control panel (see Figure 1.5 on page 12) is selected.

Transaction Sample Paths

The state information on the components in a transaction’s sample path can be saved to a data set in a Bucket. Figure 8.8 shows the Collect Data check box, which must be set to start saving data on the transactions arriving to that Bucket. These data are placed in a data set named for the number of the Bucket. You can see and change the default name in the Bucket control panel.

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>age</td>
<td>1.06129</td>
</tr>
<tr>
<td>2</td>
<td>age</td>
<td>0.65856</td>
</tr>
<tr>
<td>3</td>
<td>age</td>
<td>0.59527</td>
</tr>
<tr>
<td>4</td>
<td>age</td>
<td>0.38027</td>
</tr>
<tr>
<td>5</td>
<td>age</td>
<td>0.23192</td>
</tr>
<tr>
<td>6</td>
<td>age</td>
<td>0.14250</td>
</tr>
<tr>
<td>7</td>
<td>age</td>
<td>0.73510</td>
</tr>
<tr>
<td>8</td>
<td>age</td>
<td>0.25593</td>
</tr>
<tr>
<td>9</td>
<td>age</td>
<td>0.36469</td>
</tr>
<tr>
<td>10</td>
<td>age</td>
<td>0.53515</td>
</tr>
</tbody>
</table>

Figure 7.10. An M/M/1 Transaction State Sample Path Data Set
Chapter 8
Analyzing the Sample Path

The QSIM application provides some built-in analysis capabilities. You can obtain sample means, variance estimates, and other simple statistics on several measures of performance, including time spent in system components and resource utilization. You can also display simple line plots, histograms, and box plots to dynamically observe system behavior.

An additional feature is the ability to accumulate the sample paths in SAS data sets. See Chapter 7, “Saving and Restoring,” for more information. With these data in data sets, you can write your own programs to do more sophisticated analysis.

When calculating statistics, QSIM uses the following measures of system performance:

- number in a queue
- time in a queue
- number of multiple-servers busy
- server utilization
- time in service
- time in the system

In addition, statistics can be calculated on samples of random variables, which are user-defined attributes.

To illustrate some of the analysis features, this chapter uses the multiple-server, single-input queueing model shown in Figure 8.1.

Figure 8.1. A Single Queue 3-Server Model

The inter-arrival time distribution is $\mathcal{E}(.33)$ and the service time distribution for each server is $\mathcal{E}(1)$. Transactions queue in a FIFO discipline and go to the first available server. This corresponds to the type of queueing that you encounter at check-in in many airports.
Running a Simulation

Suppose that you have built the model in Figure 8.1. You start and stop this simulation using the button on the command panel in the simulation window. The button starts the simulation by sending a start message to each of the components in the model. Similarly, the button stops the simulation.

Figure 8.2. The Command Panel and Buttons

Also note that the button stops and resets the simulation by sending the reset message to each component. But first it stops the simulation, resets the simulation clock, and removes all pending actions. For example, all queues are emptied and transactions in service are discarded.

The button toggles animation on and off. Animation is one effective way to validate a model. You simply turn animation on and observe the simulation behavior. In addition, you can add Triggers and StringHolders to display and print state information to help you with model validation.

Another way to control the simulation is with the Component Control Panel as shown in Figure 8.3.

Figure 8.3. The Component Control Panel
Notice the buttons on the right side of the control panel. When any of these buttons are clicked, a message is sent to each of the selected components. For example, you can select a component from the Components list box and start it only by clicking the Start button. It sends the start message to the selected component. This is typically not the way to start the simulation since not all components will be started, but it is the typical way to analyze component performance with the Analyze button.

Statistics on Component State

The queues and servers are two component types for which performance measures are typically of interest. For queues you want to know the distribution on queue length, and for servers you want to know about server utilization. Other performance measures that may be of interest are time in the queue and time in service. These are discussed in the “Statistics on Transaction State” section on page 85.

There are two ways you can obtain statistics on queue length and server utilization. One way uses the chart components and the other uses the Simulation control panel shown in Figure 8.3. When the data collection status is “On” in this control panel, all changes of state are recorded to a SAS data set. These data can be summarized through the Simulation Control Panel and through the Analyze button.

Queue Length

The QSIM Application will generate and execute SAS code that calculates summary statistics on the length of a queue. When you want to start collecting the sample data, you select the On radio button in the Simulation control panel. This starts data collection. When you are ready to generate sample statistics, select the component for which you want the statistics and then click the Analyze button. For example, you display data on the queue length on the single-queue multiple-server model in Figure 8.1 by selecting FIFOQueue 1 in the Simulation Control panel and then by pressing Analyze.
Chapter 8. Analyzing the Sample Path

Figure 8.4. Statistics on Queue Length

This generates the report shown in Figure 8.4 in the SAS Output Window. Figure 8.4 shows that the sample mean queue length is approximately 45.90 and that the sample variance is approximately 2.04.

The Analyze button also opens a graphics window with the histogram on the queue length as shown in Figure 8.5. This shows the sample distribution of queue length for the complete sample path that was collected from the time the Collect Data check box is selected until the time the Analyze button is selected.
It is important to note that the observations in the sample path are not independent. There can be a significant amount of autocorrelation in the observations of queue length.

Another way to get Queue Length Statistics

Another way you can obtain statistics on queue length and server utilization is by using the chart components. You take the queue that you want to monitor and drag it into one of the chart components, the VHistogram for example. When you drop the queue, a control panel will open from which you can select the attribute of the component that you want sampled and the distribution to control sampling. Figure 8.6
shows an example. Here the size is selected to be monitored, and the component is to be sampled with a deterministic inter-sample time of one time unit. This is the default sampling distribution. This is different than using the Simulation control panel because in that case all state changes are captured, when using a chart component the queue state is sampled.

![Sample Component State Control Panel](image)

**Figure 8.6.** Sample Component State Control Panel

This control panel is opened when the queue labeled “ID: 1” in Figure 2.17 on page 41 is dropped on the histogram.

![Vertical Histogram of the FIFO Queue length](image)

**Figure 8.7.** Vertical Histogram of the FIFO Queue length

*Figure 2.17* also shows the histogram control panel. Notice that the control panel shows the simple statistics: minimum, maximum, mean, and standard deviation. Also note that when a component is dropped on a chart, a bucket is automatically associated with the chart component. The bucket is where the sample is collected. The controls for this implicit bucket can be viewed by clicking the Bin Controls button in the Histogram Controls window. This opens the control panel for the bucket as shown in *Figure 2.15* on page 40.
Figure 8.8. Bucket Control Panel

You can display more detailed statistics on queue length using SAS data sets. The “Data Collection” section in the Bucket control panel shows a data collection Status check box. If this is selected, then data on the component state changes are routed to the SAS data set named WORK.BCKT_1. If sample data have been collected for a time, then clicking the Analyze button will generate a univariate analysis performed by the UNIVARIATE procedure and a histogram.

Server Utilization

The utilization of a server is another measure of system performance that is often of interest. Estimates on the probability that a server is busy is one statistic for measuring utilization. If after collecting data using the Collect Data check box on the Simulation Control panel, you select a server from the components list box and click the Analyze button, you get a printout similar to that shown in Figure 8.9.
Chapter 8. Analyzing the Sample Path

The mean shown is approximately 0.77. This is an estimate of the probability that the server is busy.
Figure 8.10. Server Utilization

The pie chart in Figure 8.10 shows the percent of time that the server is busy.

**Another way to get Server Utilization Statistics**

Another way to view server utilization statistics is with a chart component. One component that can be used is the VHistogram. Figure 8.11 shows a VHistogram and a Histogram control panel. This histogram had an MServer dropped on it and is monitoring the number of busy servers.

Figure 8.11. Histogram of Server Utilization
When you click the **Analyze** button in the Simulation control panel (see **Figure 8.3**), the QSIM Application executes SAS code to extract and summarize the sample path saved in the **WORK.SAMPLE** data set. You can write a SAS program to extract and analyze these data. For example, suppose you want to subset the data on the time the queue is in each state. The following **DATA** step subsets **WORK.SAMPLE** with those observations that have state information on queue with ID 5:

```sas
data subset;
   set sample;
   if id=5;
      keep timenow number time;
      label number="Queue Length";
      time = ( timenow - lag(timenow) );
      number = lag(state);
run;
```

The resulting data set has three variables: **TIMENOW** for the time that the state changes, **NUMBER** for the number in the queue, and **TIME** for the length of time in that state. The following SAS code executes the **UNIVARIATE** procedure to produce summary statistics and the **GCHART** procedure to produce a histogram as shown in **Figure 8.5**.

```sas
proc univariate data=subset;
   weight time;
   var number;
run;

proc gchart;
   label = "Time in Queue";
   vbar number / subvar=time discrete;
run;
```

A similar SAS program subsets the data on server utilization and produces the output in **Figure 8.10** and **Figure 8.9**.

```sas
data subset;
   set sample;
   if id=4;
      keep timenow number time;
      label number="Utilization";
      time = ( timenow - lag(timenow) );
      number = lag(state);
run;

proc univariate data=subset;
   weight time;
   var busy;
run;
```
Statistics on Transaction State

As a transaction goes through the simulation, it maintains a history of the time spent at various components. When a transaction arrives at a Bucket, it can save any of its attributes. The name of the data set is derived from the number of buckets in the model and is displayed in the Bucket control panel as shown in Figure 2.15 on page 40.

Time in the System

By default, the Bucket component accumulates the age or system time of the transactions arriving to it. However, it can accumulate any attribute of the transaction, including user-defined attributes. The Bucket control panel has a check box which, when set, records the attribute values in the SAS data set. Figure 2.15 on page 40 shows such a control panel. If a simulation has been executing and the Collect Data check box has been set, then you can click the Analyze button to submit SAS code that calculates statistics on the sample.

Univariate statistics are calculated and histograms are printed for the attribute named in the Bucket control panel.

Figure 8.12. A Histogram Showing Time in the Queue

The control panel for the VHistogram maintains statistics on the sample in the Bucket and displays them as shown in Figure 8.12.
Chapter 9
Selected Examples

This chapter shows examples of several common modeling structures. These models address such subjects as queues with reneging, priority queues, batch arrivals, and servers that break down. The examples are meant only to show how you would model these typical situations using the QSIM Application. They are not meant to show how you would analyze these models to evaluate them or identify optimal parameterizations.

Queues with Reneging

When a customer arrives at a facility that includes queues and service, they may choose to enter a queue, if there is room, or leave the facility. Once in a queue, they may choose to leave it if they have waited too long. Not entering a queue and leaving a queue are two types of reneging. This example shows how to model some typical queues with reneging.

The model in Figure 9.1 shows a single queue for three servers. It models the M/M/c/K system where \( c = 3 \) and \( K = 50 \). This system has Poisson arrivals to a single queue with a capacity of \( K \) transactions for service by \( c \) parallel servers. Another way to model this system is with the MServer, as shown in Figure 9.1.

![Figure 9.1. An M/M/c/K Model](image)

This model is often compared to one with \( c \) parallel queues and servers, as shown in Figure 9.2.
Chapter 9. Selected Examples

Figure 9.2. A 3-Queue 3-Server Model

In this model, the Switch component directs the transaction to one of the three queues. In this case, the transaction is routed to the shortest length queue. This is accomplished with two formulas tied to the switch. A model for this is shown in Figure 2.11 on page 32.

Another variant on the parallel server models in Figure 9.1 and Figure 9.2 has customers entering a queue and, if they have waited for too long, deciding to switch to another queue. This decision-making and queue-switching policy is more complex, but it can be modeled as shown in Figure 9.3.

Figure 9.3. A 2-Queue 2-Server Model with Reneging

In this model, upon arrival, the transaction is assigned an attribute named “priority,” whose value is the current simulation time. This is done in the Modifier component labeled “priority.” Next, the transaction goes to a Switch, which compares the two queues and sends the transaction down the path leading to the shorter of the two queues. Next, the transaction encounters a Trigger, which schedules the transaction to balk when it has spent a given amount of time in the queue. The default is a random variable with exponential distribution with mean 1. When a transaction balks, it goes...
into another Switch, which checks whether the other queue is shorter. If it is, the transaction is routed to the Connector a and goes to the end of the other queue. Otherwise, the transaction goes back into the queue after scheduling, in the Trigger, another future check.

### Scanning a Queue

In the preceding example, transactions balked from a queue at a time scheduled, by a trigger, before the transaction entered the queue. Similar behavior can be modeled by periodically scanning all the transactions in a queue and balking those transactions that meet some criteria that may be based on the state of the system.

![Figure 9.4. Scanning a Queue](image)

Figure 9.4 shows a simple model where the scanning process controls the periodic searching of the queue in the main process. The sampler in the section labeled “Scanning Process” has deterministic, inter-arrival time distribution so that at fixed times a transaction gets a formula and goes to the Trigger that starts the scan of the queue. The formula has \( \text{random} \leq 0.8 \) where the transaction attribute is random. Since the trigger is set to the queue in the main process and the filter trigger message, when the transaction arrives at the Trigger, the queue is scanned and each transaction whose value of the \( \text{random} \) attribute is less than .8 is balked.

### Priority Queues

Many types of models require that multiple classes of transactions be served by a single server. For example, two different types of customers arrive at an auto repair shop. One type needs only minor repairs, and the other type needs more major work. If one class has priority on getting service, then a priority queue is the appropriate modeling choice. Figure 9.5 shows one such model having two classes.
Figure 9.5. A Priority Queue Example with Two Transaction Classes

The two classes of transactions arrive according to independent Poisson processes as represented by the two Samplers. Transactions travel to Modifiers that set the priority to be either a 1 or a 2. They then enter a Priority Queue with the priority level determining their position in the queue; the higher priority transactions are serviced before the lower priority transactions. When the transactions finish service, they enter a Switch that directs them to one of two Buckets as a function of priority class.

The model in Figure 9.5 assumes that if a Class 1 transaction is in service when a Class 2 transaction arrives, the Class 1 transaction completes service before the Class 2 transaction starts service ahead of any other Class 1 transactions in the queue. Figure 9.6 shows how you would modify the model if you wanted to preempt a Class 1 transaction that was in service when a Class 2 transaction arrived.

Figure 9.6. A Priority Queue Example with Two Transaction Classes and Preemption

For this preemption, you would store the class of the transaction currently in service in a number holder. This storage is done by the trigger just below the number holder. Then, when a Class 2 transaction arrives, it causes a check of the class of the transaction in service. If it is Class 1, then it is preempted. Notice that any preempted Class 1 transactions are routed back into the queue.
**Batch Arrivals I**

The model in Figure 9.7 shows one way to represent batch arrivals. The compound component labeled “Arrival Process” has a sampler with the batch inter-arrival time distribution set. When a transaction arrives in this process, it traverses to the Trigger labeled “Reset,” which resets the Sampler labeled “Batch Source.”

The transaction then goes to the Trigger labeled “Start,” which starts the Batch Source Sampler. The Batch Source Sampler has a deterministic inter-arrival time distribution with parameter 0 and capacity \( c \), the batch size. Because of this inter-arrival time distribution, when the Batch Source Sampler is started by the Start Trigger, the Batch Source generates \( c \) transactions at the current simulation time and sends them to the queue. This is the batch arrival of transactions.

Notice the LinePlot labeled “Number in the Queue.” The periodic discrete jumps in queue length show a batch arrival.
Chapter 9. Selected Examples

Batch Arrivals II

Another variant on batch arrivals has batch size as a random variable. A simple extension to the previous model provides this alternative.

Figure 9.8. Batch Arrivals

Figure 9.8 shows two components added to the model of Figure 9.7: a Modifier labeled “capacity” and a Trigger labeled “Set Capacity.” The capacity Modifier samples a uniform random variable on the interval \([0, 30]\) and sets it in the capacity attribute on the transaction in the Arrival Process. The Set Capacity Trigger then sets the capacity of the Batch Source Sampler to that value. Then, the Start Trigger starts the Batch Source arrivals as before.

Compare the LinePlot labeled “Number in the Queue” in this model to the previous example. Here, in addition to the timing of the discrete jumps in queue length, the size of the discrete jumps in queue length is random.

Nonhomogeneous Poisson Processes

In many situations, the arrival rate or service rate is determined by a Poisson process whose parameter varies as a function of time. For example, if the arrival rate to a fast food restaurant varies with the time of day and increases to a local maximum during meal times, you can sample from a nonhomogeneous Poisson process. In the QSIM Application, there are some limitations to the shape of the rate function that are allowed. This function must be cyclical and bounded. The software takes the absolute value of the rate function to guarantee that it is nonnegative.

In this example, shown in Figure 9.9, the model has deterministic arrival rate and nonhomogeneous Poisson service times.
Figure 9.9. Nonhomogeneous Poisson Service

The service rate is $9 + \cos(0.001t)$, where $t$ is the value of the simulation time when a sample is taken. This rate function is specified via the control panel for the random variable, as shown in Figure 9.10.

Figure 9.10. Nonhomogeneous Poisson Control Panel

In this window you can set two parameters of the process: the rate function and the maximum value that the rate function can take. The rate function is specified as a QSIM formula. When you click the Rate Function button, a Formula Manager window opens and allows you to specify the function.

Figure 9.11. Rate Function

Figure 9.11 shows the function used in this example. The maximum is needed by the algorithm that does the sampling. If this is not the correct maximum or the function specified is not cyclical, then the sample is not from the desired distribution.

When the transactions from this simple model are displayed in the LinePlot as shown in Figure 9.9, you can see the impact of the cyclical rate function on the transaction time in the system.
Markov-modulated Poisson Arrivals

A Markov-modulated Poisson Process (MMPP) is a Poisson process that has its parameter controlled by a Markov process. These arrival processes are typical in communications modeling where time-varying arrival rates capture some of the important correlations between inter-arrival times. This example has a Markov-modulated Poisson process that serves to control the arrival process to a single-queue, single-server queueing model.

Figure 9.12. Markov-Modulated Poisson Arrivals

Figure 9.12 shows one way to model an MMPP. The process labeled “Markov-modulated Poisson Process” samples from an MMPP distribution and sets the value of the parameter lambda, the mean inter-arrival time for an exponential random variable in the Sampler labeled “MMPP Arrivals.” In the upper process, lambda is given the values 10, .1, and 1, based on the state of a Markov chain. The state is changed in the Modifier components labeled “state.” Each has a conditional component driven by an observation of a uniform random variable. So, for a given state, the state is changed to the next state and the value of lambda is chosen for the MMPP Arrivals Sampler. The selected lambda is set in the MMPP Arrivals Sampler, and the process is delayed for an exponential amount of time whose parameter is state dependent. The transaction then goes to a switch that routes based on the state for the next state change.
State-Dependent Service

In many situations, the rate of service depends on the type of service being performed. For example, the time it takes for a teller to service a customer in a bank depends on the type of service requested. State-dependent service distributions are modeled similarly to the Markov-modulated Poisson arrivals. Consider the example shown in Figure 9.13, in which there are multiple classes of transactions to a single queue.

Figure 9.13. State Dependent Service

Figure 9.13 shows this model with the addition of a Modifier to set the exponential mean service time. When the transaction leaves the queue to begin service, it passes through a Trigger that sets the parameter for the service time as a function of the class of transaction that is to receive the service. In addition to the parameter, other models can change the shape of the service distribution.
You may have a need to model a server that periodically breaks down and is repaired. For example, a machine on an assembly line may periodically fail. Figure 9.14 shows such a model. The Server component labeled is the server that experiences down periods when it cannot service transactions.

The process in the compound component labeled “Breakdown Loop” models the breakdown behavior. The transaction pool has a capacity of 1 so that, when it is started, one transaction is generated that cycles through the breakdown loop for the rest of the simulation. This loop has two delays: Delay 1 models the time when the server is in operation; Delay 2 models the time when the server is broken. The two triggers, labeled “Down” and “Up,” stop and start the Server.
Another variant of the server breakdown model concerns what happens to the transaction that is in service when the breakdown occurs. In the model in Figure 9.14, even though the server is stopped when it breaks, the transaction in service completes service. The model in Figure 9.15 adds the preemption of the transaction in service, which is routed back into the queue.

By default, the transaction is placed at the end of the FIFO queue. So, if there were other transactions waiting for service, the preempted transaction would be behind them. Another variant on this model would place the preempted transaction into the front of the queue, even though the queue was a FIFO for nonpreempted transactions. This variant could be accomplished using a priority queue where the transaction priority is the simulation time at the time the transaction arrived to the queue and the queue has decreasing priority (see Figure 2.5). See the preceding example on priority queues.
Suppose that you want to service transactions in a batch where you start service simultaneously on all the items in the batch but the individual service times are independent and identically distributed. This might occur in a drying process, where you have arrivals to a drying machine determined by some arrival process. When there are enough items to fill the batch, the baking of all the items in the batch begins. However, as each item dries it is removed individually from the drying machine.

The model in Figure 9.16 accomplishes this batch service. In this model the multiple-server is set to the batch size and the Server On compound component turns the server on if it is empty and there are 10 or more transactions in the FIFOQueue. The Trigger labeled “Server Off” turns the server off when each transaction leaves service. When a server is off, all transactions currently in service complete normally, but the server will not send out messages for additional transactions. As a result, service on all transactions in process will complete, but additional arrivals to the system will queue until there are at least 10 and the server is empty.

Notice the LinePlot labeled “Server Utilization.” It shows the number of transactions in service over time. It demonstrates graphically the batch service and independent nature of the service completions.
Batch Service II

Some situations demand a somewhat different approach to batch service; for example, consider a washing machine. The machine is started when enough items have arrived for service to complete the batch. However, unlike the preceding example, all the items in the batch finish at the same time. The model in Figure 9.17 accomplishes this.

![Batch Service Diagram]

**Figure 9.17.** Batch Service

In this model the service distribution in the MServer labeled “Batch Server” is deterministic with a large parameter value, for example, $D$. The Server labeled “Delay” provides the actual sample of the service time for the entire batch. The Batch Server is turned on before the Delay and off after service for the batch is complete. Since turning the server off does not preempt transactions currently in service, there is another Trigger labeled “Preempt” that preempts all the transactions in the Batch Server. Since the transactions are preempted, they leave the server through the balk node.

Notice the LinePlot labeled “Server Utilization,” which shows the number of transactions in service over time. It demonstrates graphically the batch service and dependent nature of the service completions.

Because of the modeling technique used here, the service time distribution is the minimum of $D$ and $X$, an exponential random variable. If you want the service time distribution to be $X$, then use caution in choosing $D$ so that the probability that $X > D$ is very small and highly unlikely to occur within the number of samples planned.
Chapter 9. Selected Examples

Batch Service III

Another variant on batch service has the transactions accumulating into a batch according to the arrival process and has service scheduled as soon as the first transaction arrives. Service on the entire batch completes at once.

Figure 9.18. Batch Service

Figure 9.18 shows a model of this batch service. When the first transaction arrives to the Server labeled “Delay,” it initiates the definition of a batch. Any other transactions that arrive to that server are discarded through the balk node. When the delay is complete, the Trigger labeled “Preempt Service” terminates service on all the transactions in the MServer labeled “Batch Server.” As the transactions arrive, they accumulate in the MServer for batch service. This server has a deterministic service distribution with a large parameter value, for example, $D$. Note that because of the modeling technique used here, the service time distribution is the minimum of $D$ and $X$, an exponential random variable. If you want the service time distribution to be $X$, then use caution in choosing $D$ so that the probability that $X > D$ is very small and highly unlikely to occur within the number of samples planned.

Notice the LinePlot labeled “Server Utilization,” which shows the number of transactions in service over time. It demonstrates graphically the accumulation of the batch and the dependent nature of the service completions.
Assembly

In a model of manufacturing systems, there is often assembly of subunits into larger units. The assembly cannot occur unless all of the subunit pieces are available. An important component for modeling this behavior is the Adder.

Figure 9.19. Assembly Unit

Figure 9.19 shows the assembly of two subunits into a larger unit. Each subunit line produces components as modeled by servers 1 and 2. These subunits queue in the buffers at the end of the subunit assembly lines. When the multiple server is free, it requests a transaction from the Adder. The Adder requests one transaction from each of the lines going into it. If there is a transaction available from each of these lines, then it requests one. When all the transactions have arrived at the Adder, it generates a new transaction, which is sent down the arc to the multiple server.
**Servers as Resources I**

There are instances in which where the system needs to schedule concurrent service from multiple servers on a single transaction. In these situations, you can think of servers as resources that are being utilized by the transactions. For example, in an auto repair facility, several mechanics (modeled as servers) can work on a single car (the transaction) at a time. The **Splitter** is useful for treating servers as resources and capturing concurrent use of the resources.

**Figure 9.20.** Servers as Resources

Figure 9.20 shows a simple model with arrivals from two sources, each sending the transactions into a queue. If the two servers are free and there is a transaction in FIFO 1, then the first transaction inserted into the queue will flow to both the servers and service will start in each. The service times in each of these is independent (unless you construct and use a service time distribution that destroys this independence). When Server 2 becomes free, it requests a transaction. If Server 1 is busy, then the request can only be honored by a transaction in the FIFO 2 queue. When Server 1 becomes free, it requests a transaction that can only be honored if Server 2 is free and there is a transaction in FIFO 1.
Servers as Resources II

In the preceding example, the resources (servers) performed service independently on a transaction. However, there are situations where the resource may be used in a more controlled way. Suppose there are two parallel lines that each require the use of a shared resource (a crane, for example).

Figure 9.21. Servers as Resources

Figure 9.21 shows such a model. As in the last example, the Splitter is used to capture the shared use of the resources by a transaction. In addition, there is a Trigger after each of the servers in the parallel lines. These triggers release the transaction from service in any other servers. Therefore, the time the transaction uses the Crane is the minimum of the time scheduled for Crane use and the line service time. In particular, if the service time in Server 1 is \( X \) and the service time specified for the Crane is \( Y \), then the service time that the transaction actually receives in the Crane is \( \min(X, Y) \). This occurs because either the transaction finishes with the Crane before it is done with service in Server 1 (or Server 2) or it finishes with service in Server 1 (or Server 2) before the Crane service is completed. In this case, Trigger 1 (or Trigger 2) sends the “RemoveFromServers” messages (see Figure 3.2 on page 48), which removes that transaction from any servers in which it may be receiving service. In this case, the transaction can be explicitly removed from service by the Crane.
There are other ways that independent streams can share resources. One is illustrated in Figure 9.22.

**Figure 9.22.** Special Routing

Here the **Modifiers** labeled “routeToId” set the attribute “routeToId” to the id of “FIFO 1” or “FIFO 2.” When the transaction finishes with the service of the “Crane” and traverses to the **Trigger** labeled “routeToId,” it is routed to the component whose id is in its attribute “routeToId.” This is another way that transactions can be routed through the network.
Appendix A

References


Index

Default
+, 38
-, 38

A
Adder, 34, 47, 101
Analyze, 81, 85
analyze data, 40
animation, 47
arcs, 10, 44
   segmented, 45
are you busy message, 17
arrivals
   batch, 91, 92
attributes, 29, 30, 47, 51

B
balk, 25
Balk node, 21, 31, 33
Box Plot, 39, 40
Bucket, 40, 73, 85
Bucket Control Panel, 81
buffer, 23
busy, 22

capacity, 19, 25
cart chart components, 79
check box
   Collect Data, 81, 85
Clear Palette, 70
clearSetFromAttribute, 38, 39
Collect Data, 81, 85
collection buttons, 11
command panel, 11
Component Control Panel, 76
component dictionary data set, 72
component id, 72
component state sample path, 73, 77
compound components, 55
   controlling exposure, 57
   editing, 60
   pop-up menu, 56
connecting components, 44
Connector, 43, 89, 104
control panel
   Box Plot, 41
   Bucket, 40, 81
   Component, 76
   Compound Component, 59
   DataSource, 67
   Exponential Control Panel, 65
   Histogram, 41
   Line Plot, 42
   MModifier, 30
   Modifier, 29
   MServer, 21
   MTrigger, 28
   NumberHolder, 36
   Priority Queue, 24
   Sampler, 19, 63
   Simulation, 81
   Switch, 32
   Trigger, 26, 37
controls, 38, 39
currentValue, 38, 39

D
data collection, 40, 48
dataSource Control window, 67
dataSource Distribution, 67
decrement, 38
Deterministic Distribution, 64
distribution, 27, 63
   DataSource, 67
   Deterministic, 64
   Erlang, 64
   Exponential, 64
   Formula, 66
   Gamma, 64
   IUUniform, 64
   Markov-modulated Poisson, 94
   Mixture, 67
   Nonhomogeneous Poisson, 64, 92
   Uniform, 64

E
Edit, 60
elementary components, 17
   Box Plot, 40
   Bucket, 40
   Connector, 43
   FIFO Queue, 23
   Histogram, 41
   Holders, 35
   Label, 43
   LIFO Queue, 23
   Line Plot, 42
Logic Components, 26
MModifier, 30
Modifier, 29
MServer, 20
MTrigger, 28
NumberHolder, 36
Port, 43
Priority Queue, 23
Router, 33
Sampler, 18
Server, 20
Splitter, 35
StringHolder, 38
Switch, 31
Transaction Pool, 18
Trigger, 26
empty, 25
Erlang Distribution, 64
Exponential Distribution, 64
Expose/Hide Details, 17

F
FIFO Queue, 23
filter, 25
filterOne, 25
Formula Distribution, 66
Formula Manager, 51
Formula Manager Window, 32, 51, 93
formulas, 51

G
Gamma Distribution, 64
graphs, 72

H
Histogram, 39
histogram
queue length, 79
time in queue, 85
Holders, 35

I
id, 19, 22, 25, 27, 28, 30, 31, 33–35, 38, 39
images, 47
insert, 25
IUniform Distribution, 64

L
Label, 43
LIFO Queue, 23
Line Plot, 39, 42
logic
Adder, 34
MModifier, 30
Modifier, 29
MTrigger, 28
NumberHolder, 37
Queue components, 24
Router, 33
Server components, 21
Source components, 19
StringHolder, 38
Switch, 32
Trigger, 27
Logic Components, 26

M
M/M/1 Queue, 13
M/M/c/K, 87
Markov-modulated Poisson Process, 94
menu
Options, 65
MModifier, 30
model, 69
assembly, 101
batch arrivals, 91, 92
batch service, 98–100
queue with reneging, 87, 89
routing, 104
scanning a queue, 89
servers as resources, 102–104
service break down, 96, 97
state-dependent service, 95
Modifier, 29, 47, 51, 88
MServer, 20, 63
MTrigger, 28

N
New Palette, 70
Nonhomogeneous Poisson Distribution, 64
Nonhomogeneous Poisson Process, 92
Nonhomogeneous Possion Parameter Window, 93
number in a queue, 75
NumberHolder, 36
NumberHolder Control Panel, 36

O
off, 22
on, 19

P
Palette, 61, 70
palette, 9
component, 9
Formula Manager, 51
pie chart
server utilization, 83
pop-up menu, 10
pop-up menu selections
Assemble Components, 55, 56
Control Panel..., 10, 17
Disassemble Components, 56
Duplicate, 44
Edit, 56, 60
Expose/Hide Details, 10, 17, 56
on arcs, 44
Select Components..., 58
Start, 10
Stop, 10
  Toggle Exposure, 58
Tools, 10
Port, 43, 57
preempt, 23
preemptContinue, 23
print, 38, 39
Priority Queue, 23, 88, 89
pull down menus
  Clear Palette, 70
  New Palette, 70
pull-down menu selections
  Graph..., 72
  Model..., 69
  Open..., 69
  Save As, 69
pull-down menus
  Clock Options, 66
  Seeds..., 65
push button
  Analyze, 81, 85

Q
QSIM Application, 1
query message, 17
  busy, 22
  capacity, 19, 25
  currentValue, 38, 39
  id, 19, 22, 25, 27, 28, 30, 31, 33–35, 38, 39
  off, 22
  on, 19
  releaseType, 25
  remaining, 19
  size, 19, 25
  sizes, 22
  space, 22, 25
  value, 27, 38, 39
queue
  priority, 88
  renege, 87, 89
  scanning a queue, 89
Queue Components, 23
queue length statistics, 77

R
random variation, 63
releaseOne, 25
releaseType, 25
remaining, 19
removeFromQueues, 49
removeFromServers, 49
removeIt, 23
reset the simulation, 76
resetting information, 71
Router, 33, 51
routeTold, 49

S
sample path, 72, 75
Sampler, 10, 18, 47, 63
  batch arrivals, 91, 92
SAS data set, 40, 48, 81, 84
SAS Graph, 72
Save Graph Window, 72
Save Model Window, 69
saving information, 69
  component dictionary data set, 72
  component state sample path, 73
  graphs, 72
  model, 69
  Palette, 70, 71
  sample path, 72
  transaction state sample path, 73
seed values, 65
seize, 23
Select Components..., 58
Server, 63
  as resource, 102–104
  batch, 98–100
  break down, 96, 97
  state-dependent service, 95
Server Components, 20
server utilization, 75
server utilization statistics, 81
setCapacity, 20, 23
setDistribution, 23
setFromAttribute, 38, 39
setFromTrigger, 27, 38, 39
setParameter1, 20, 23
setParameter2, 20, 23
Simulation Control Panel, 81
simulation window
  pop-up menu, 56
  size, 19, 25
  sizes, 22
Source Components, 18
space, 22, 25
Splitter, 35, 102, 103
start the simulation, 76
state, 77
state change, 17, 26, 28, 31, 33
statistics, 40, 77
  queue length, 77
  server utilization, 81
  time in the system, 85
stop the simulation, 76
StringHolder, 38
Switch, 31, 51, 52, 88
system performance
  queue length, 77
  server utilization, 81
  time in the system, 85
time in a queue, 75
time in service, 75
time in system, 75
time in the system statistics, 85
time units, 66
Toggle Exposure, 58
transaction
    attributes, 29, 30, 47
generation, 18
    history, 47
timing of arrivals, 49
transaction arrival, 17
Transaction Pool, 18, 47
transaction state sample path, 73
transactions, 47
    saving state information, 73
Trigger, 26, 63
delayed trigger event, 27
    Trigger Interval, 27
    Trigger Value, 27
Trigger Control Pane, 37
trigger message, 17, 48
    +, 38
    -, 38
balk, 25
clearFromSetAttribute, 39
clearSetFromAttribute, 38
controls, 38, 39
decrement, 38
empty, 25
filter, 25
filterOne, 25
insert, 25
preempt, 23
preemptContinue, 23
print, 38, 39
releaseOne, 25
removeIt, 23
seize, 23
setCapacity, 20, 23
setDistribution, 23
setFromAttribute, 38, 39
setFromTrigger, 38, 39
setParameter1, 20, 23
setParameter2, 20, 23

U
Uniform Distribution, 64
units, 66
User Interface, 8

V
value, 27, 38, 39

W
window
Your Turn

If you have comments or suggestions about *SAS/OR® 9.1 User’s Guide: QSIM Application*, please send them to us on a photocopy of this page or send us electronic mail.

For comments about this book, please return the photocopy to
SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: yourturn@sas.com

For suggestions about the software, please return the photocopy to
SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: suggest@sas.com