THE POWER TO KNOW®

SAS/OR® 9.2 User's Guide QSIM Application

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2008. *SAS/OR® 9.2 User's Guide: QSIM Application*. Cary, NC: SAS Institute Inc.

SAS/OR® 9.2 User's Guide: QSIM Application

ISBN 978-1-59047-944-5

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, March 2008
2nd electronic book, February 2009

1st printing, March 2009

Contents

Acknowledgments

Credits

Documentation

Writing	Marc-david Cohen
Editing	Virginia Clark, Donna Sawyer, Anne Jones
Documentation Support	Tim Arnold, Michelle Opp, Remya Chandran
Technical Review	Marc-david Cohen, Gehan Corea, Edward P. Hughes, John Jasperse, Charles B. Kelly, Phil Meanor

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

Software Testing	Emily Lada, Tao Huang, John Jasperse, Bengt Pederson, Rob Pratt, Nitin Agarwal, Marianne Bohinski, Edward P. Hughes
Technical Support	Tonya Chapman

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 9.2

Overview

SAS/OR 9.2 continues the improvements delivered starting with SAS/OR 9.1.3 release 3.1 and release 3.2. Several new and enhanced features expand the scale and scope of problems that SAS/OR can address. These enhancements also make it easier for you to use the capabilities of SAS/OR. Brief descriptions of these new features are presented in the following sections. For more information, see the SAS/OR documentation, available in the following 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: Project Management*
- *SAS/OR User's Guide: The QSIM Application*

Online help can also be found under the corresponding classification.

The NETFLOW Procedure

The NETFLOW procedure for network flow optimization contains a new feature that enables you to specify and solve generalized network problems. In generalized networks, the amount of flow that enters an arc might not equal the amount of flow that leaves the arc, signifying a loss or a gain as flow traverses the arc. A new PROC NETFLOW option, GENNET, indicates that the network is generalized. Generalized networks have a broad range of practical applications, including the following:

- transportation of perishable goods (weight loss due to drying)
- financial investment account balances (interest rates)
- manufacturing (yield ratios)
- electrical power generation (loss during transmission along lines)

Another new option, EXCESS=, enables you to use PROC NETFLOW to solve an even wider variety of network flow optimization problems for both standard and generalized networks. As a result, PROC NETFLOW is equipped to deal with many frequently encountered challenges to successful network flow optimization, such as the following:

- networks with excess supply or demand
- networks that contain nodes with unknown supply and demand values
- networks with nodes that have range constraints on supply and demand

In SAS/OR 9.2, the MPSOUT= option directs the NETFLOW procedure to save input problem data in an MPS-format SAS data set. Invoking the MPSOUT= option causes the NETFLOW procedure to output the data and halt without attempting optimization. The MPS-format SAS data set corresponds closely to the MPS-format text file (commonly used in the optimization community). Problems that are specified in this format can be solved by using the OPTLP procedure.

The INTPOINT Procedure

In SAS/OR 9.2, the MPSOUT= option directs the INTPOINT procedure to save input problem data in an MPS-format SAS data set. Invoking the MPSOUT= option causes the INTPOINT procedure to output the data and halt without attempting optimization. The MPS-format SAS data set corresponds closely to the MPS-format text file (commonly used in the optimization community). Problems that are specified in this format can be solved by using the OPTLP procedure.

The LP Procedure

In SAS/OR 9.2, the MPSOUT= option directs the LP procedure to save input problem data in an MPS-format SAS data set. Invoking the MPSOUT= option causes the LP procedure to output the data and halt without attempting optimization. The MPS-format SAS data set corresponds closely to the MPS-format text file (commonly used in the optimization community). Problems that are specified in this format can be solved by using the OPTLP or OPTMILP procedure.

The OPTLP Procedure

The OPTLP procedure enables you to choose from three linear programming solvers: primal simplex, dual simplex, and interior point (experimental). The simplex solvers implement a two-phase simplex method, and the interior point solver implements a primal-dual predictor-corrector algorithm.

The TIMETYPE= option enables you to specify the type of time (real time or CPU time) that can be limited via the MAXTIME= option and reported via the _OROPTLP_ macro variable.

PROC OPTLP accepts linear programming problems that are submitted in an MPS-format SAS data set. The MPS-format SAS data set corresponds closely to the MPS-format text file (commonly used in the optimization community). Problem data in formats that are used by the LP, INTPOINT, and NETFLOW procedures can be converted into MPS-format SAS data sets by using the new MPSOUT= option in each of these procedures.

New in SAS/OR 9.2, the experimental IIS= option enables you to identify, for an infeasible problem, constraints and variable bounds that form an irreducible infeasible set (IIS). Identifying an IIS can be very helpful in diagnosing and remedying infeasibility in a linear program. Information about the IIS is contained in the PRIMALOUT= and DUALOUT= data sets.

Also new in SAS/OR 9.2, the value "2" for the PRINTLEVEL= option directs the OPTLP procedure to produce an ODS table called "ProblemStatistics" in addition to the "ProblemSummary" and "SolutionSummary" ODS tables that are produced for PRINTLEVEL=1.

The OPTMILP Procedure

The OPTMILP procedure solves mixed-integer linear programming problems with an LP-based branch-and-bound algorithm that has been completely rewritten for this release. The algorithm also implements advanced techniques including presolvers, cutting planes, and primal heuristics. The resulting improvements in efficiency enable you to use PROC OPTMILP to solve larger and more complex optimization problems than you could solve with previous releases of SAS/OR.

PROC OPTMILP accepts mixed-integer linear programming problems that are submitted in an MPS-format SAS data set.

New in SAS/OR 9.2, the value "2" for the PRINTLEVEL= option directs the OPTMILP procedure to produce an ODS table called "ProblemStatistics" in addition to the "ProblemSummary" and "SolutionSummary" ODS tables that are produced for PRINTLEVEL=1.

The OPTMODEL Procedure

The OPTMODEL procedure provides a modeling environment that is tailored to building, solving, and maintaining optimization models. This makes the process of translating the symbolic formulation of an optimization model into PROC OPTMODEL virtually transparent, since the modeling language mimics the symbolic algebra of the formulation as closely as possible. PROC OPTMODEL also streamlines and simplifies the critical process of populating optimization models with data from SAS data sets. All of this transparency produces models that are more easily inspected for completeness and correctness, more easily corrected, and more easily modified, whether through structural changes or through the substitution of new data for old.

The OPTMODEL procedure comprises the powerful OPTMODEL modeling language and state-of-the-art solvers for several classes of mathematical programming problems.

Seven solvers are available to OPTMODEL as listed in Table 1:

Table 1. List of OPTMODEL Solvers

Problem	**Solver**
linear programming	**LP**
mixed integer programming	**MILP**
quadratic programming (experimental)	**QP**
nonlinear programming, unconstrained	**NLPU**
general nonlinear programming	**NLPC**
general nonlinear programming	**SQP**
general nonlinear programming (experimental)	**IPNLP**

New in SAS/OR 9.2, the experimental IIS= option for the LP solver enables you to identify, for an infeasible linear program, constraints and variable bounds that form an irreducible infeasible set (IIS). Identifying an IIS can be very helpful in diagnosing and remedying infeasibility in a linear program.

The OPTQP Procedure

The OPTQP procedure solves quadratic programming problems with a new infeasible primal-dual predictor-corrector interior point algorithm. Performance is excellent for both sparse and dense quadratic programming problems, and PROC OPTQP excels at solving large problems efficiently.

PROC OPTQP accepts quadratic programming problems that are submitted in a QPS-format SAS data set. The QPS-format SAS data set corresponds closely to the format of the QPS text file (a widely accepted extension of the MPS format).

Earned Value Management Macros

The set of earned value management macros complements the current SAS/OR procedures for project and resource scheduling (PROC CPM and PROC PM) by providing diagnostic information about the execution of scheduled projects. Earned value management (EVM) is growing in prominence and acceptance in the project management community due to its ability to turn information about partially completed projects into valid, early projections of overall project performance. EVM measures current project execution against the project execution plan on a cost and schedule basis.

SAS/OR provides two sets of EVM macros: a set of four analytical macros to compute EVM metrics, and a set of six macros to create graphical reports based on these metrics. A wide variety of EVM metrics and performance projections, for both task-by-task and project-wide evaluations, are supported.

Microsoft Project Conversion Macros

The SAS macros %MDBTOPM and %MP2KTOPM have been used in previous releases of SAS/OR to convert files saved by Microsoft Project 98 and Microsoft Project 2000 (and later), respectively, into SAS data sets that can be used as input for project scheduling with SAS/OR. Now these two macros are combined in the SAS macro %MSPTOSAS, which converts Microsoft Project 98 (and later) data. This macro generates the necessary SAS data sets, determines the values of the relevant options, and invokes PROC PM in SAS/OR with the converted project data. The %MSPTOSAS macro enables you to use Microsoft Project for the input of project data and still take advantage of the excellent project and resource scheduling capabilities of SAS/OR.

In SAS/OR 9.2, the experimental %SASTOMSP macro converts data sets used by the CPM and PM procedures into an MDB file that is readable by Microsoft Project. The macro converts information that is common to both PROC CPM / PROC PM and Microsoft Project, including hierarchical relationships, precedence relationships, time constraints, resource availabilities, resource requirements, project calendars, resource calendars, task calendars, holiday information, and work-shift information. In addition, the early and late schedules, the actual start and finish times, the resource-constrained schedule, and the baseline schedule are also extracted and stored as start-finish variables.

Execution of the %MSPTOSAS and %SASTOMSP macros requires SAS/ACCESS software.

The GA Procedure

The GA procedure solves optimization problems through the use of genetic algorithms. The procedure uses functions and call routines to set parameters such as crossover operators and mutation probabilities for genetic algorithm optimization. In SAS/OR 9.2, the routines that are used to specify procedure-supplied mutation and crossover operators (SetMut and SetCross), objective functions (SetObj), and selection options (SetSel) have been revised to a more flexible and readable form. The operator type is now specified as a parameter in these routines instead of being incorporated into the name of a separate call routine. Parameters for each operator type are now specified as property name-value pairs.

Note: Several call routines that were available in SAS/OR 9.1.3 have been replaced by new call routines and are not available in SAS/OR 9.2. Table 2 lists the routines and their replacements.

Table 2. PROC GA Routines Replaced in SAS/OR 9.2

New Routine	Routines Replaced
Cross call	CrossSimple call, Cross2Point call, CrossUniform call, CrossArithmetic call, CrossHeuristic call, CrossOrder call, CrossPMatch call, CrossCycle call
Mutate call	MutDelta call, MutUniform call, MutSwap call, MutInvert call
SetCross call	SetCrossSimple call, SetCross2Point call, SetCrossUniform call, SetCrossArithmetic call, SetCrossHeuristic call, SetCrossOrder call, SetCrossPMatch call, SetCrossCycle call
SetMut call	SetMutRoutine call, SetMutDelta call, SetMutUniform call, SetMutSwap call, SetMutInvert call
SetObj call	SetObjTSP call
SetSel call	SetSelTournament call, SetSelDuel call

In addition, the following new routines are provided:

- Objective function
- ReadCompare call
- SetCompareRoutine call
- SetObjFunc call
- SetProperty call
- ShellSort call
- Shuffle call

The Boolean encoding has been modified so that 0/1 values can be read from and written to solution segments directly, instead of requiring the PackBits and UnpackBits routines. In addition, each Boolean value is represented by one variable in a LASTGEN= or FIRSTGEN= data set, similar to the other encodings.

If the FIRSTGEN= data set has a field named "OBJECTIVE," then in the Initialize call, the value of that field (if nonmissing) is used as the initial objective value for the solution represented by that observation.

The default crossover and mutation probabilities have been changed to 0.

New options have been implemented for the Initialize call and the ReEvaluate call.

New in SAS/OR 9.2, the option LIBRARY= specifies an external library of routines. The NOVALIDATE= and NOVALIDATEWARNING= options control the level of feasibility checks performed by the GA procedure.

The CLP Procedure (Experimental)

The CLP procedure features improved algorithms for the “alldifferent” constraint as well as several extensions to the edgefinder algorithm for resource-constrained scheduling. The EDGEFINDER option can now determine whether an activity must be the first (last) to be scheduled from among a set of activities, while the NF= and NL= options specify the level of propagation for the “not first” and “not last” extensions. A new activity selection strategy RJRAND and a corresponding activity assignment strategy MAXTW have been added; these strategies tend to favor right-justified schedules. The MAXTIME= option enables you to specify a time limit on the CPU time for controlling execution times.

Chapter 1
Introduction

Chapter Contents

Chapter 1
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 in SAS/OR software.

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, which 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 the INTPOINT, LP, NETFLOW, and NLP procedures, in addition to the newer OPTMODEL, OPTLP, OPTQP, and OPTMILP procedures, the various solvers called by the OPTMODEL procedure, and the MPS-format SAS data set specification.

SAS/OR User's Guide: Project Management
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, the earned value management macros, the Microsoft Project conversion macros, and the PROJMAN application.

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.

Chapter 2
Getting Started

Chapter Contents

Chapter 2
Getting Started

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.

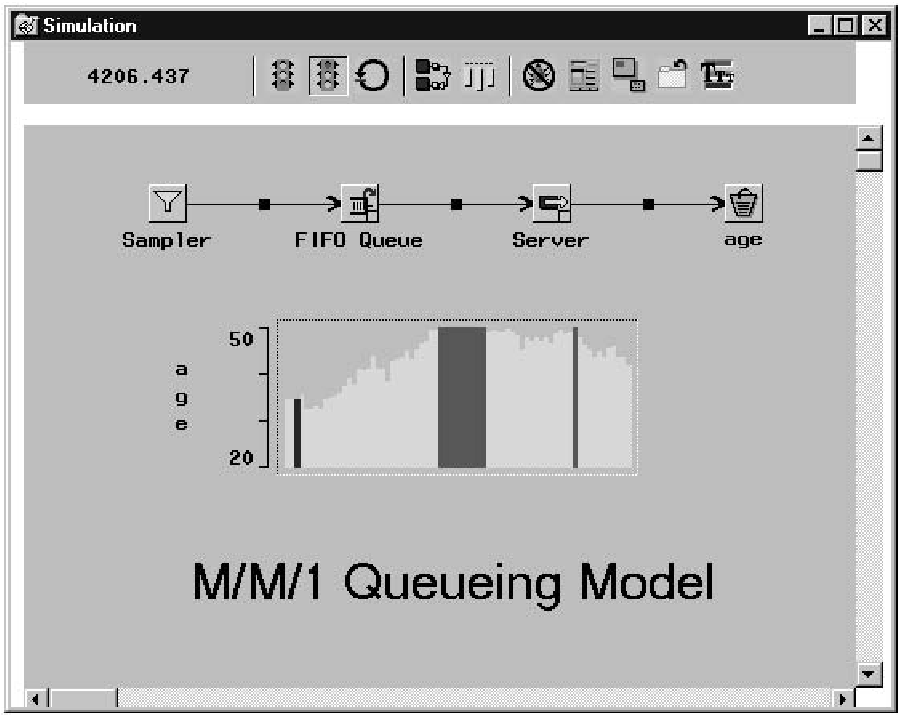


Figure 2.1. An M/M/1 Queueing Model

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 2.1 shows a simple M/M/1 queueing model built with the QSIM Application. An M/M/1 queueing 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 2.2. 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.

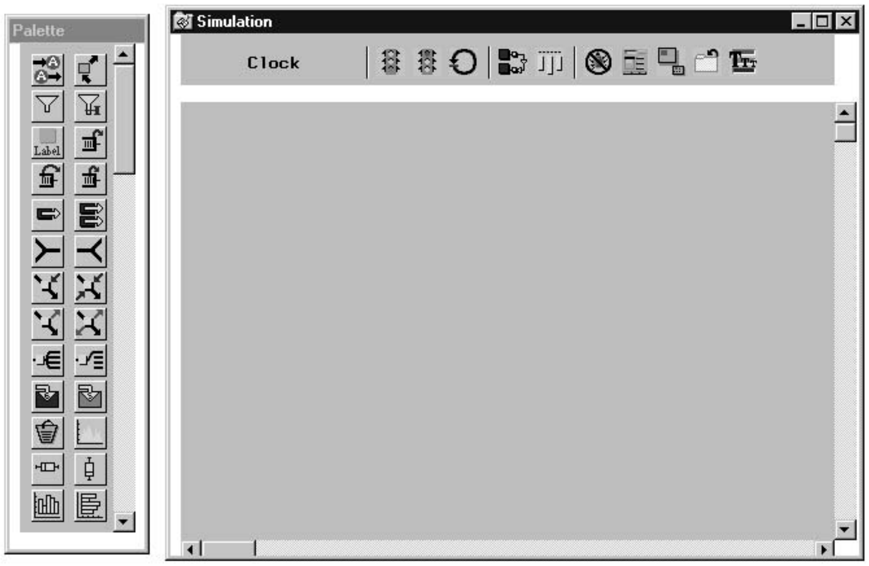


Figure 2.2. The Palette and the Simulation Window

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 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, , 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 +, 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.

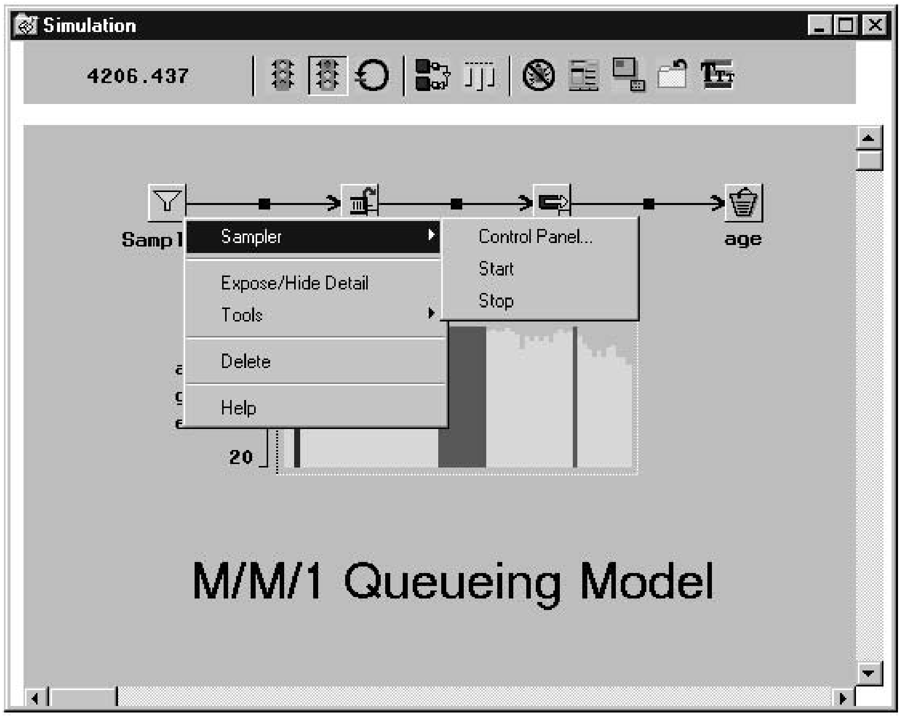


Figure 2.3. The **Sampler** Pop-Up Menu

Figure 2.3 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 2.5 on page 12. It is displayed by selecting the command button.

The Command Buttons

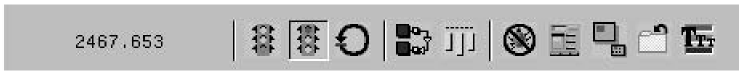

2467.653

Figure 2.4. The Command Panel and Buttons

The command buttons shown in Figure 2.4 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

Button	Description
Clock	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.
	starts the simulation.
	stops the simulation.
	resets the simulation.
	toggles transaction animation on and off. When animation is on, you can see transactions that originate at **Samplers** traversing the network.
	aligns components vertically and horizontally.
	raises the Debug Control Panel.
	raises the Components Control Panel.
	raises the Viewport Window.
	saves the simulation model.
	raises the Component Attributes Panel for changing colors, fonts, and images.

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.

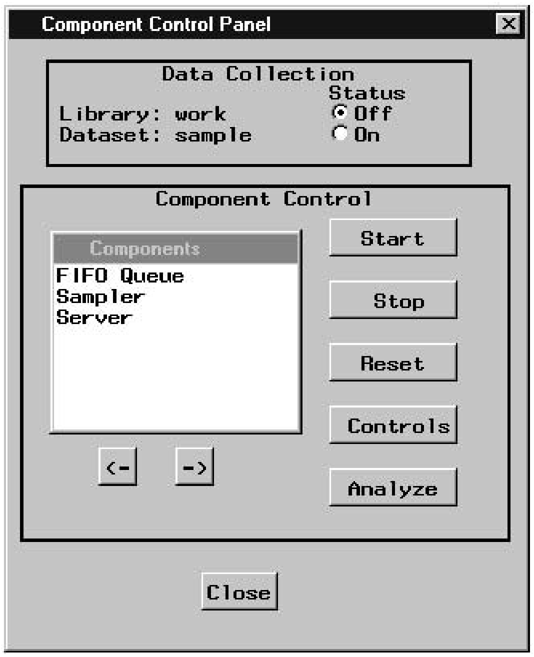


Figure 2.5. The Component Control Panel

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 2.6 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.

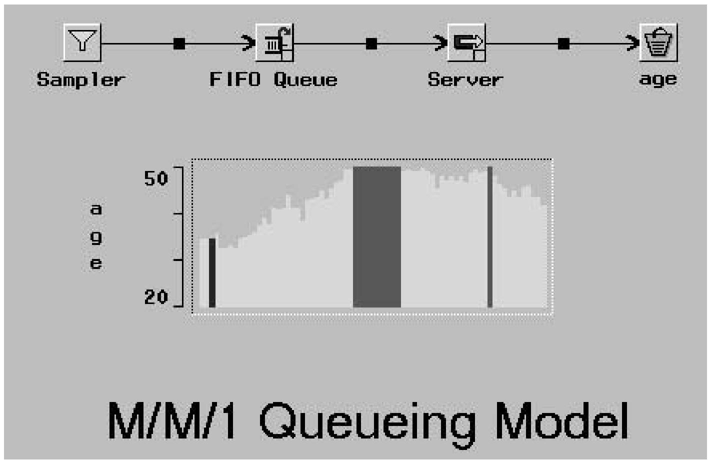


Figure 2.6. An M/M/1 Queueing Model

Queueing

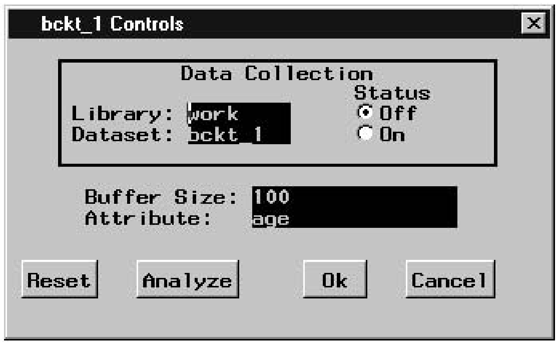
When the transaction leaves the Sampler, it flows down the arc to the FIFO Queue. It is important to note that the movement of the transaction down the arc does not advance the simulation clock. On transaction arrival at the FIFO Queue, the queue broadcasts messages down arcs asking nodes downstream 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.



Library: work
Dataset: bckt_1
Status: [✓ Off] On
Buffer Size: 100
Attribute: age

Figure 2.7. A Bucket Control Panel

From the bucket control panel (Figure 2.7) 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.

```
---------------------------------- name=age ----------------------------------

                           The UNIVARIATE Procedure
                               Variable:  value

                                   Moments

        N                        1803    Sum Weights               1803
        Mean               24.5093328    Sum Observations    44190.3271
        Std Deviation      14.8024337    Variance            219.112044
        Skewness           -0.1060516    Kurtosis            -1.2191446
        Uncorrected SS     1477915.34    Corrected SS        394839.903
        Coeff Variation    60.3950903    Std Error Mean      0.34860632


                          Basic Statistical Measures

                 Location                    Variability

             Mean     24.50933     Std Deviation           14.80243
             Median   26.21585     Variance               219.11204
             Mode      .           Range                   53.62360
                                   Interquartile Range     26.10309


                          Tests for Location: Mu0=0

                Test           -Statistic-    -----p Value------

                Student's t    t  70.30662    Pr > |t|    <.0001
                Sign           M     901.5    Pr >= |M|   <.0001
                Signed Rank    S    813153    Pr >= |S|   <.0001


                           Quantiles (Definition 5)

                          Quantile          Estimate

                          100% Max       53.62741148
                          99%            50.95981599
                          95%            46.44804080
                          90%            43.36915659
                          75% Q3         37.45295397
                          50% Median     26.21585009
                          25% Q1         11.34985903
                          10%             3.08194661
                          5%              1.38985610
                          1%              0.28776996
                          0% Min          0.00381383


                             Extreme Observations

                 -------Lowest-------        -----Highest-----

                       Value      Obs            Value      Obs

                  0.00381383     1442          51.9481      819
                  0.02449773     1486          52.1543      818
                  0.02846473     1458          52.1938      418
                  0.04001101     1249          52.4247      419
                  0.06573967     1700          53.6274      817
```

Figure 2.8. 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 2.8) and a sample distribution function as shown in Figure 2.9 on page 16 will be plotted.

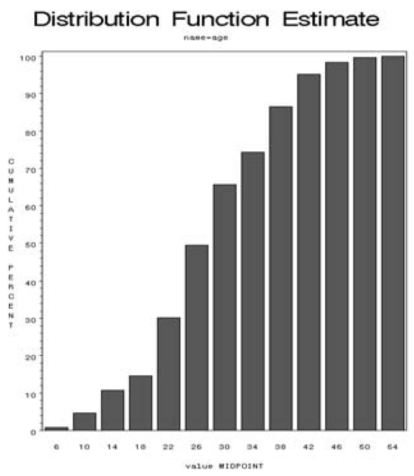


Figure 2.9. Sample Distribution Function

See Chapter 9, “Analyzing the Sample Path,” for more information about collecting statistics and analyzing simulation data.

Chapter 3
Building a Model with Elementary Components

Chapter Contents

Chapter 3
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).

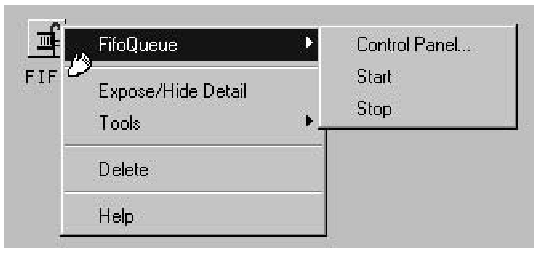


Figure 3.1. The Pop-Up Menu on the FIFO Queue Component

Figure 3.1 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 3.14 on page 39, 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.

Icon	Component	Description
	Sampler	generates transactions with prescribed inter-arrival times
	Transaction Pool	a source of transactions

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.

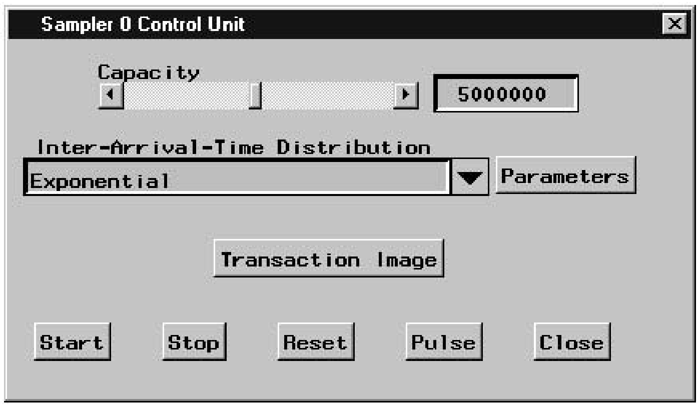


Figure 3.2. The Sampler Control Panel

Figure 3.2 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 7, “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

Keyword	**Meaning**
capacity	returns the capacity
id	the source component’s unique identifier
on	TRUE if the source component is started, else FALSE
remaining	returns the number of transactions remaining to be sent
size	returns the number of transactions sent

Trigger Message

Keyword	Meaning
reset	stops the source component and resets the number of transactions remaining to be sent to 1
setCapacity	sets the capacity from the transaction attribute “capacity”
setDistribution	sets the distribution from the transaction attribute “distribution.” This attribute should be a character string whose value is one of the distributions: Exponential, Gamma, Erlang, Uniform, IUniform, and Deterministic.
setParameter1	sets the first parameter in the distribution from the transaction attribute “parameter1”
setParameter2	sets the second parameter in the distribution from the transaction attribute “parameter2”
start	starts the source component
stop	stops the source component

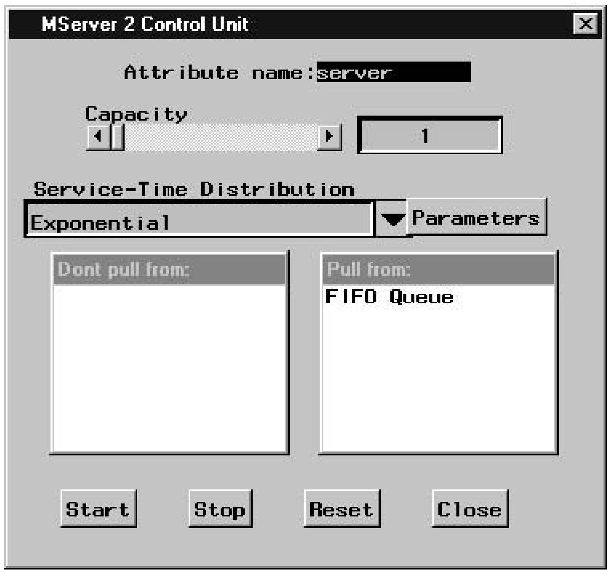
Server Components

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

Icon	Component	Description
▣	Server	provides service for a transaction
▣	MServer	provides service simultaneously for multiple transactions

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.


MServer 2 Control Unit

Attribute name: server

Capacity: 1

Service-Time Distribution: Exponential

Pull from:
FIFO Queue

Figure 3.3. The Multiple-Server 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 3.4.

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.

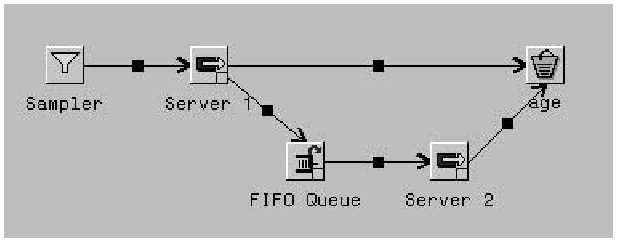


Figure 3.4. Server Balk Model

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.

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

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

Trigger Message

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

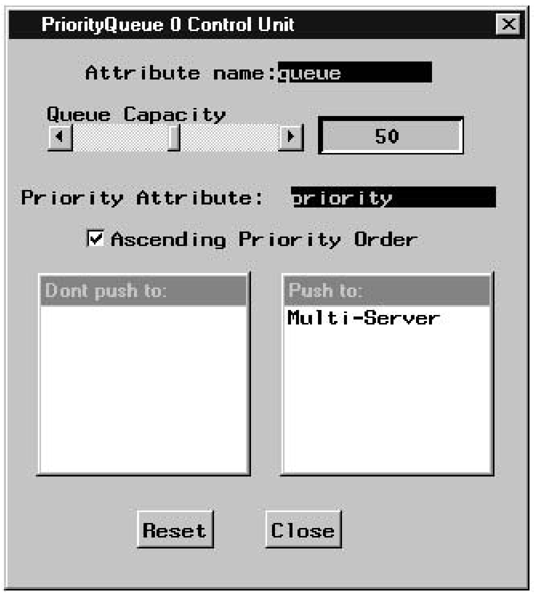
Queue Components

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

Icon	Component	Description
▨	FIFO Queues	first-in-first-out
▨	LIFO Queues	last-in-first-out
▨	Priority Queues	priority

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 3.5) 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 section “Logic Components” on page 28. 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 3.5.



Attribute name: queue

Queue Capacity 50

Priority Attribute: priority

☑ Ascending Priority Order

Dont push to:

Push to:
Multi-Server

Figure 3.5. The Priority Queue Control Panel

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 3.5 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

Keyword	Meaning
capacity	returns the queue's capacity
id	returns the queue's unique identifier
releaseType	returns a string naming the way that the last transaction was released from the balk node. Possible values are: "balk," "empty," "filter," "filterOne," and "releaseOne."
size	returns the number of transactions that are in the queue
space	returns TRUE if there is unused capacity in the queue

Trigger Message

Keyword	Meaning
balk	causes the transaction at the Trigger to leave the queue from the Balk node
empty	empties the queue of all transactions. Note that the transactions do not leave via the Balk node.
filter	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.
filterOne	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.
insert	inserts the transaction at the Trigger into the queue
releaseOne	releases one transaction from the queue via the Balk node
reset	destroys all transactions in the queue
start	starts the queue
stop	stops the queue

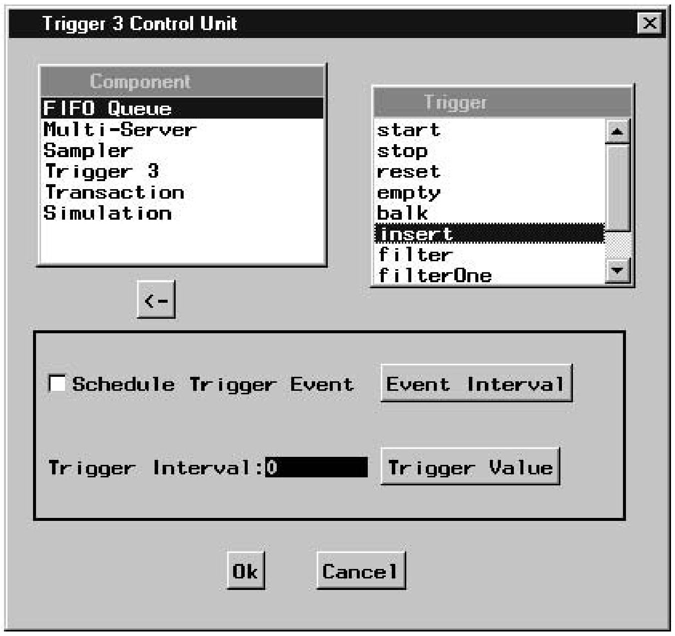
Logic Components

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).

Icon	Component	Description
	Adder	assemble multiple transactions
	Splitter	split single transactions
	Modifier	assign an attribute to transactions
	MModifier	assign multiple attributes to transactions
	Trigger	change a component's state
	MTrigger	change multiple components' state
	Router	direct flow as a function of system state
	Switch	direct flow as a function of system state

Trigger Component

When a transaction arrives at a Trigger component, it initiates a message being sent to another component. For example, Figure 3.6 shows the control panel for a Trigger component. Notice that the `FIFO Queue` component is selected and that the `insert` trigger is also selected.


Trigger 3 Control Unit

Component: FIFO Queue (selected), Multi-Server, Sampler, Trigger 3, Transaction, Simulation

Trigger: start, stop, reset, empty, balk, insert (selected), filter, filterOne

☐ Schedule Trigger Event

Trigger Interval: 0

Figure 3.6. 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 section "Queue Components" on page 25, 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 5.3 on page 56) 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

Keyword	Meaning
id	returns the component's unique identifier
value	returns the value associated with the Trigger

Trigger Message

Keyword	Meaning
reset	resets the Trigger
start	starts the Trigger
stop	stops the Trigger

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 3.7 shows the control panel for a MTrigger component.

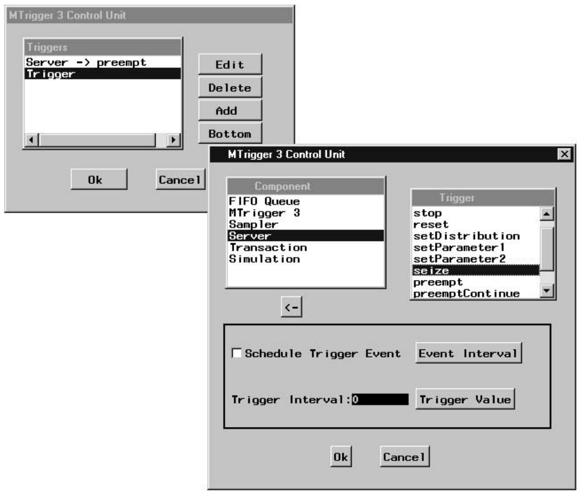


Figure 3.7. The MTrigger Control Panel

Notice that the Triggers list box contains two entries, one labeled `Server -> 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

Keyword	Meaning
id	returns the component's unique identifier

Trigger Message

Keyword	Meaning
reset	resets the Multi-Trigger
start	starts the Multi-Trigger
stop	stops the Multi-Trigger

Modifier Component

The Modifier component assigns an attribute to a transaction. The control panel, shown in Figure 3.8, 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.

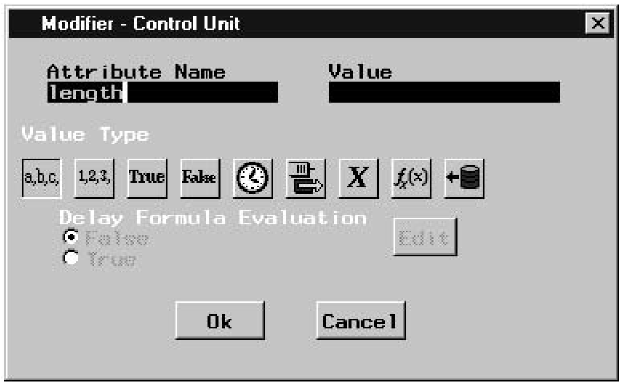


Figure 3.8. The Modifier Control Panel

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.

Query Message

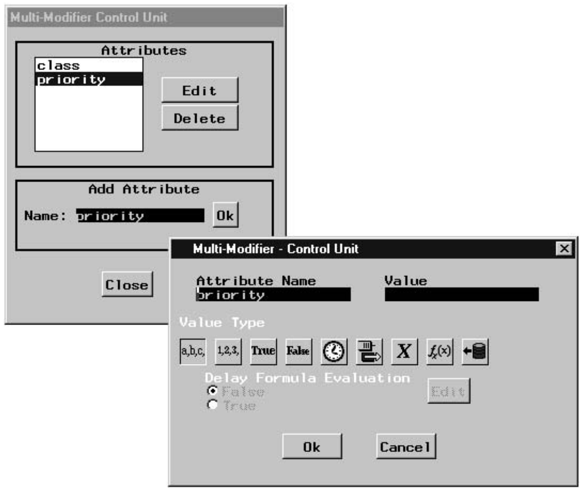
Keyword	Meaning
id	returns the component's unique identifier

Trigger Message

Keyword	Meaning
reset	resets the Modifier
start	starts the Modifier
stop	stops the Modifier

MultiModifier Component

The MModifier component assigns multiple attributes to a transaction. The control panel, shown in Figure 3.9, provides a field for entering the attribute name and an **Ok** push button for adding the attribute to the attributes list.



class
priority

Name: priority

Attribute Name: priority
Value:

Delay Formula Evaluation
[✓ False]
True

Figure 3.9. 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.

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

Keyword	Meaning
id	returns the component's unique identifier

Trigger Message

Keyword	Meaning
reset	resets the Multi-modifier
start	starts the Multi-modifier
stop	stops the Multi-modifier

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.

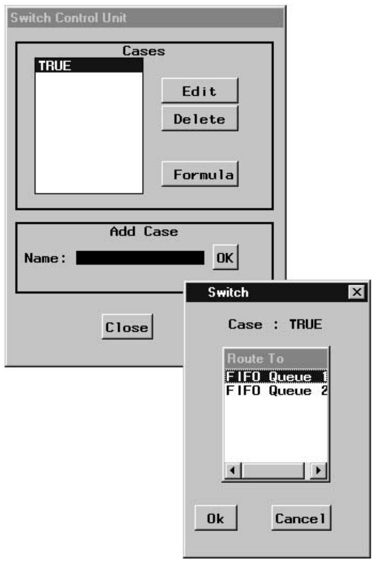


Figure 3.10. Switch Control Panel

Figure 3.10 shows the control panel for a switch connected to two queues as in Figure 3.11. Selecting the **Formula** button displays a Formula Manager Window (see Chapter 5). 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 3.10. 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.

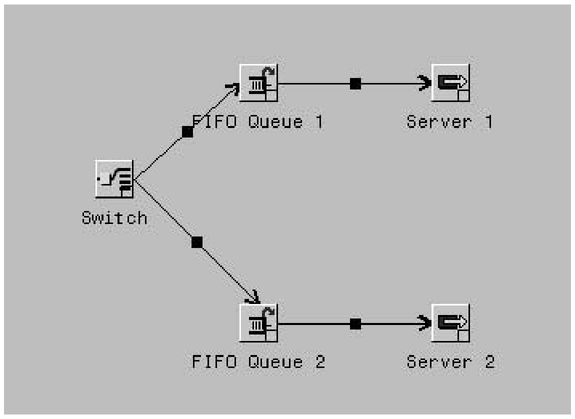


Figure 3.11. Switch Controlled Queue Selection

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

Keyword	Meaning
id	returns the component's unique identifier

Trigger Message

Keyword	Meaning
reset	resets the Switch
start	starts the Switch
stop	stops the Switch

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.

Query Message

Keyword	Meaning
id	returns the component's unique identifier

Trigger Message

Keyword	Meaning
reset	resets the Router
start	starts the Router
stop	stops the Router

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

Keyword	Meaning
id	returns the component's unique identifier

Trigger Message

Keyword	Meaning
reset	resets the Adder
start	starts the Adder
stop	stops the Adder

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

Keyword	**Meaning**
id	returns the component’s unique identifier

Trigger Message

Keyword	**Meaning**
reset	resets the Splitter
start	starts the Splitter
stop	stops the Splitter

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.

Icon	Component	Description
▨	NumberHolder	storage for a number
▨	StringHolder	storage for a string

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.

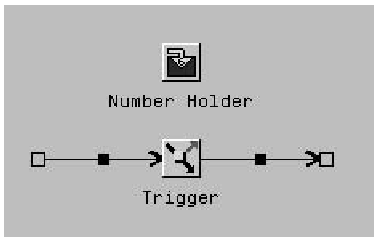
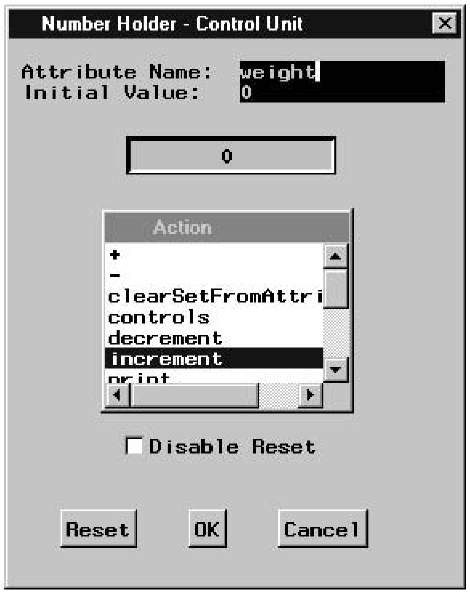


Figure 3.12. Number Holder Saving “weight” Attribute

In the model fragment in Figure 3.12, 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.


Number Holder - Control Unit

Attribute Name: weight
Initial Value: 0

0

Figure 3.13. Number Holder Control Panel

Notice that the NumberHolder control panel shown in Figure 3.13 has the attribute name “weight” in the **Attribute Name** field.

The transaction sets the NumberHolder when it arrives at the Trigger because the Trigger Control Panel, as shown in Figure 3.14, has the `NumberHolder` component selected and the `setFromAttribute` selected.

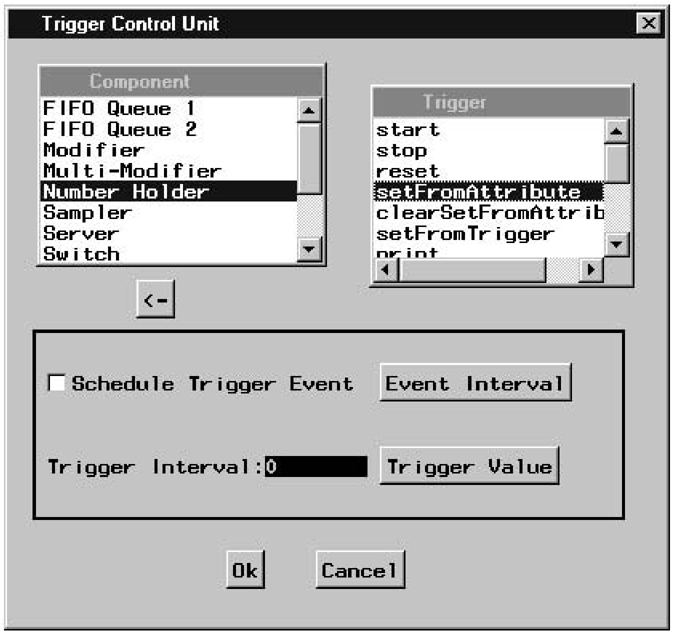


Figure 3.14. A Trigger Control Panel

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.

Query Message

Keyword	Meaning
currentValue	returns the value in the holder at the current time
id	returns the component's unique identifier
value	returns the value in the holder when the transaction passed through it

Trigger Message

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

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

Keyword	Meaning
currentValue	returns the value in the holder at the current time
id	returns the component's unique identifier
value	returns the value in the holder when the transaction passed through it

Trigger Message

clearSetFromAttribute	clears the value then sets it
controls	displays the Holder Control Panel
print	prints the value on the SAS Log window
reset	resets the value
setFromAttribute	sets the value from the transaction attribute
setFromTrigger	sets the value that is assigned with the **Trigger Value** button in the Trigger Control Panel
start	starts the holder
stop	stops the holder

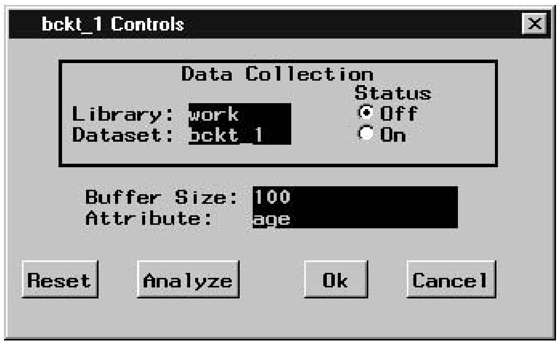
Chart Components

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.

Icon	Component	Description
🪣	Bucket	collect value of an attribute for analysis
📊	VHistogram	vertical histogram of numeric data
☰	HHistogram	horizontal histogram of numeric data
⌶	VBoxPlot	vertical box plot of numeric data
⊢⊣	HBoxPlot	horizontal box plot of numeric data
📈	LinePlot	plot of numeric data over simulation time

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.


bckt_1 Controls

Data Collection

Library: work
Dataset: bckt_1

Status
[✓ Off]
On

Buffer Size: 100
Attribute: age

Figure 3.15. 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 8, “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 GPLOT. Chapter 9, “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.

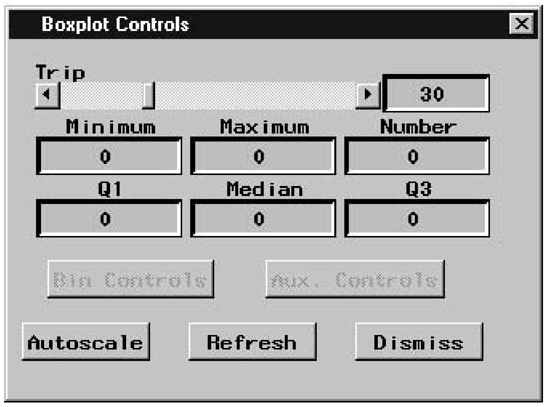


Figure 3.16. A Box Plot Control Panel

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.

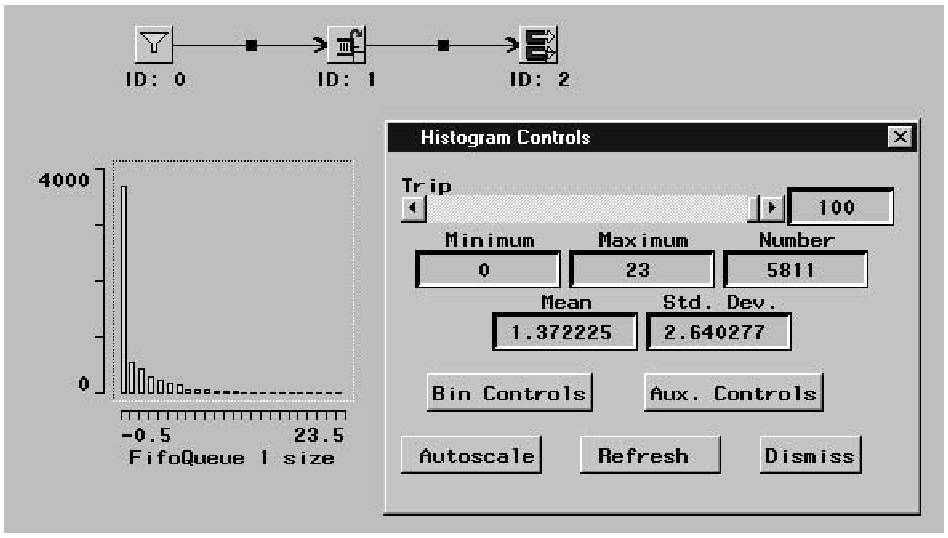


Figure 3.17. A Histogram and Control Panel

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.

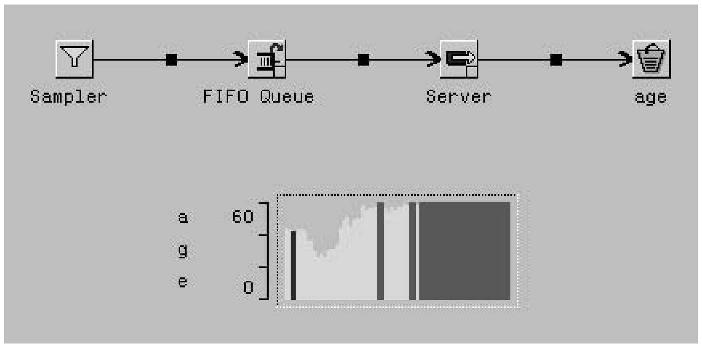


Figure 3.18. A Line Plot of Transaction Age

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.

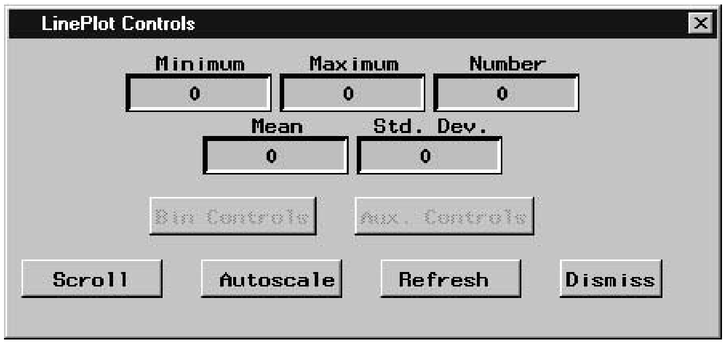


Figure 3.19. A Line Plot Control Panel

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.

Icon	Component	Description
	Port	for connecting multiple components
	Connector	for connecting multiple components without using arcs
	Label	for annotating the model

Figure 3.20 shows an example with a connector labeled "a." When a transaction flows into connector "a," it will flow out of all other connectors "a."

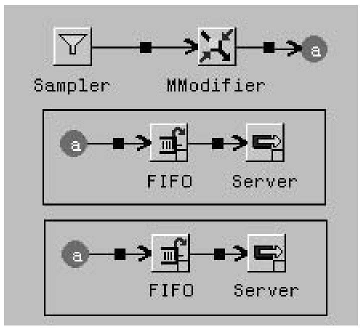


Figure 3.20. Simple Example Using a Connector

The three "a" connectors are treated as identical. An equivalent model using an instance of a Port is shown in Figure 3.21.

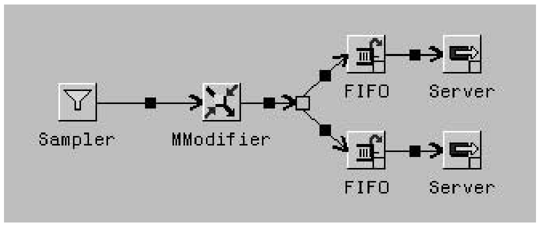


Figure 3.21. Simple Example Using a Port

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 6, "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 3.22 shows the two types of arcs.

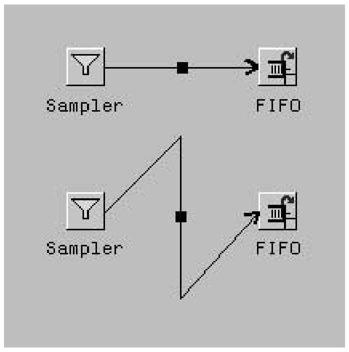


Figure 3.22. Two Types of Arcs

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.

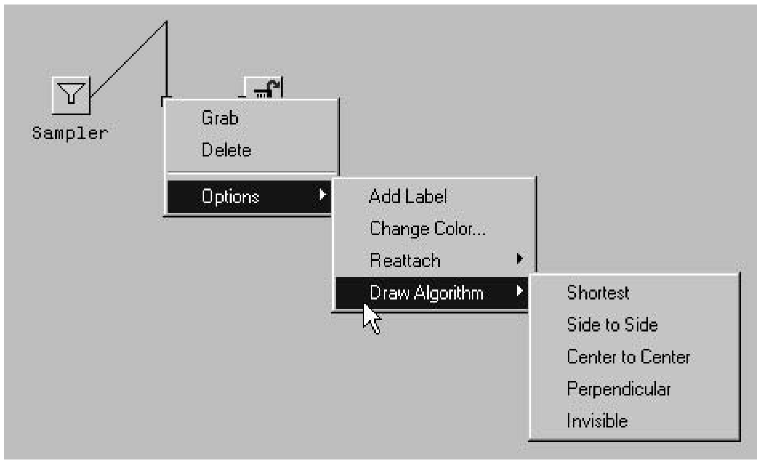


Figure 3.23. Pop-Up Menu for a Segmented Arc

In the pop-up menu for segmented arcs in Figure 3.23, 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 3.23 show, a full range of capabilities are available.

Chapter 4
Transactions

Chapter Contents

Chapter 4 Transactions

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 4.1 shows an example. Although Figure 4.1 shows the image SASHELP.ORIMG24.PHONE, any image in a SAS catalog can be used.

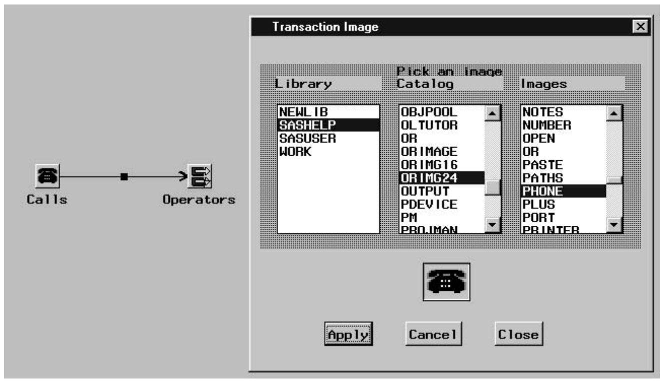


Figure 4.1. Setting an Image for a Sampler Transaction

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 3.5 on page 26). 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 9.8 on page 93), 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 4.2 shows a Trigger control panel with this entry selected.

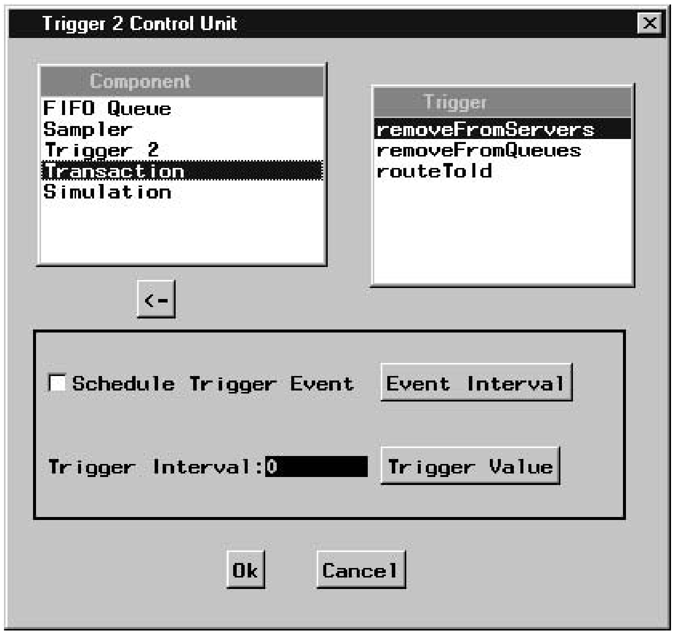


Figure 4.2. Transaction Trigger Messages

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 10, "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 4.3.

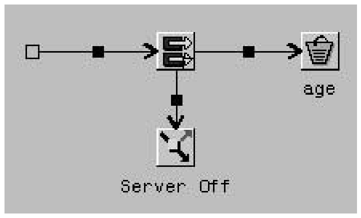


Figure 4.3. Model Fragment A

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.

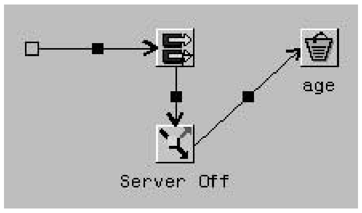


Figure 4.4. Model Fragment B

To guarantee that the Trigger "Server Off" executes before the transaction traverses to the Bucket, connect the components as in Figure 4.4.

Chapter 5
Formulas

The Router, Switch, and Modifier components use formulas to control routing and set attributes. Formulas are specified in a Formula Manager window. Figure 7.6 shows one such window with the formula $5.0 + X$, where X is a random variable with distribution $\mathcal{E}(1)$, exponentially distributed with mean 1.

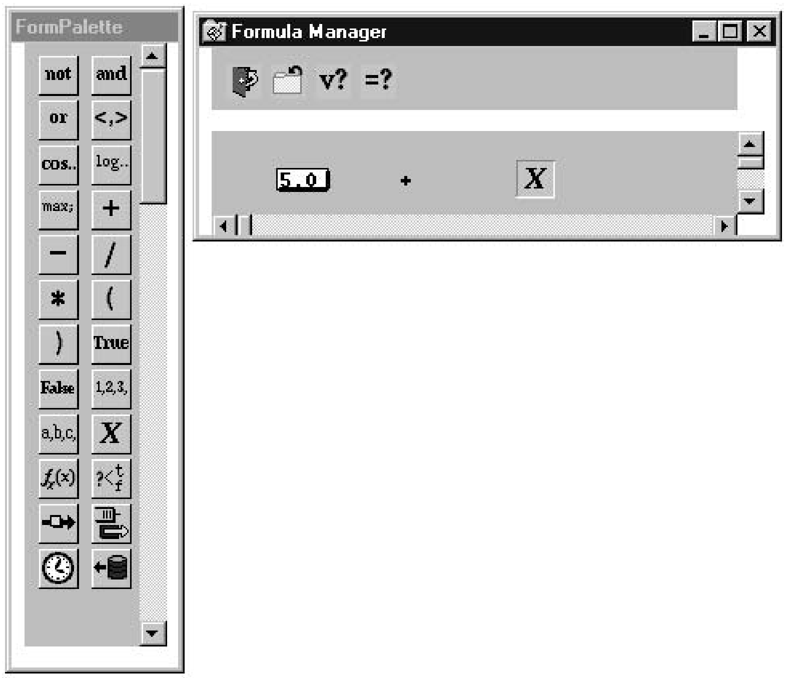


Figure 5.1. Formula Manager Window

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.

Icon	Description
f(x)	another formula
⇥	a transaction attribute
🕒	the simulation clock
▤	a component in the model
⇤	an observation of a variable in a SAS data set
1,2,3,	a number constant
a,b,c,	a string constant

As with components, each of these elements has a pop-up menu associated with it. For example, the random variable icon *X* shown in Figure 7.6 has a pop-up menu, shown in Figure 5.2.

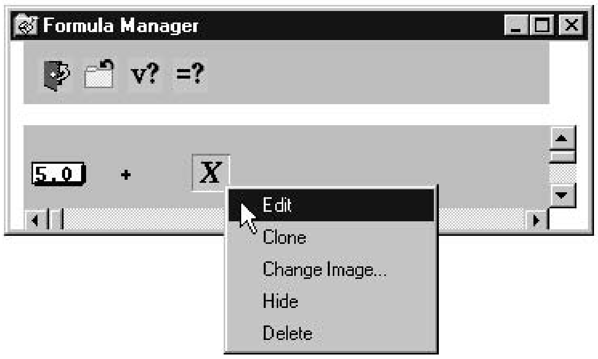


Figure 5.2. Pop-up Menu on the Random Variable Formula Element

If you select **Edit**, the window shown in Figure 5.3 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.

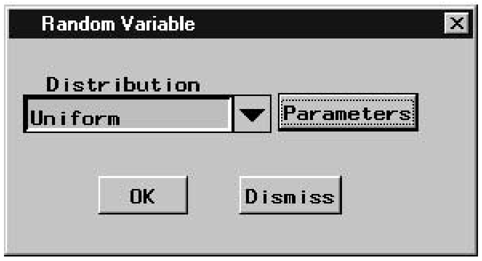


Figure 5.3. Control Panel for the Random Variable Formula Element

Another important formula element is the model element, . Recall the example discussed in the section "Switch Component" on page 33. There, the model in Figure 3.11 on page 34 routes transactions to the shortest queue. The control panel on the Switch shown in Figure 3.10 on page 34 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.

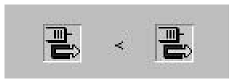


Figure 5.4. Formula for Comparing the Size of Two Queues

Figure 5.4 shows the formula for accomplishing this. The edit window for the left-hand model element is shown in Figure 5.5.

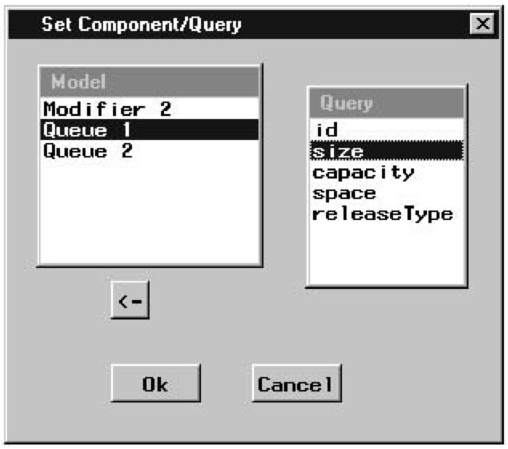


Figure 5.5. The Model Element Edit Window

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 5.4 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 5.6.

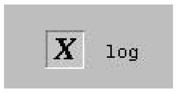


Figure 5.6. A Formula for the Log of a Random Variable

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 5.7.

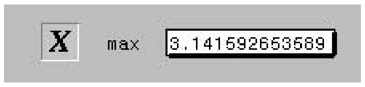


Figure 5.7. A Formula for the Maximum of a Random Variable and π

Chapter 6
Building a Model with Compound Components

Chapter Contents

Chapter 6 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.

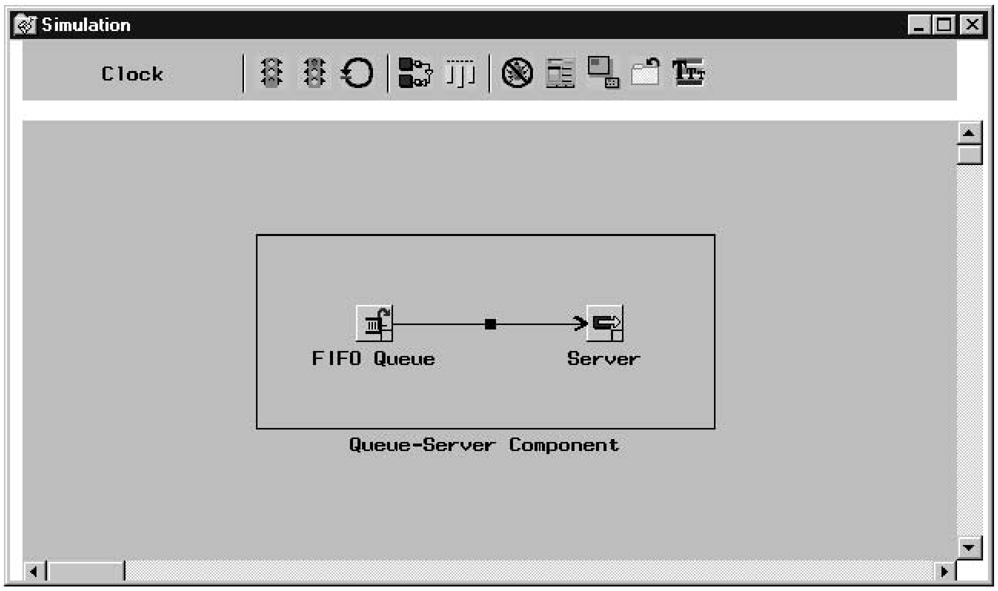


Figure 6.1. A Compound Component with Queue and a Server

Assembling Components into Compound Components

The pop-up menu on the Simulation window (see Figure 6.2) 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.

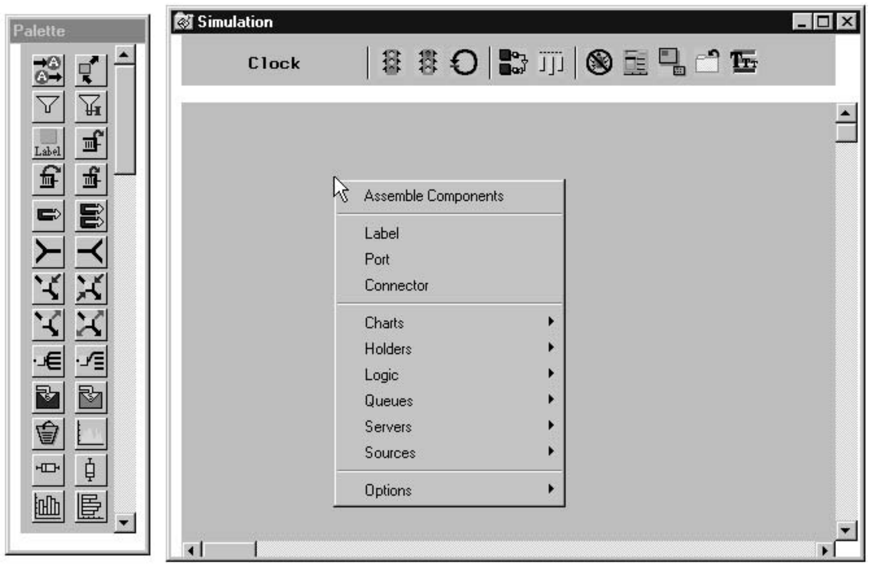


Figure 6.2. The Simulation Window Pop-up Menu

Figure 6.1 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 6.3, which includes **Assemble Components** and **Disassemble Components** for assembling additional compound components within it and for removing the encapsulation.

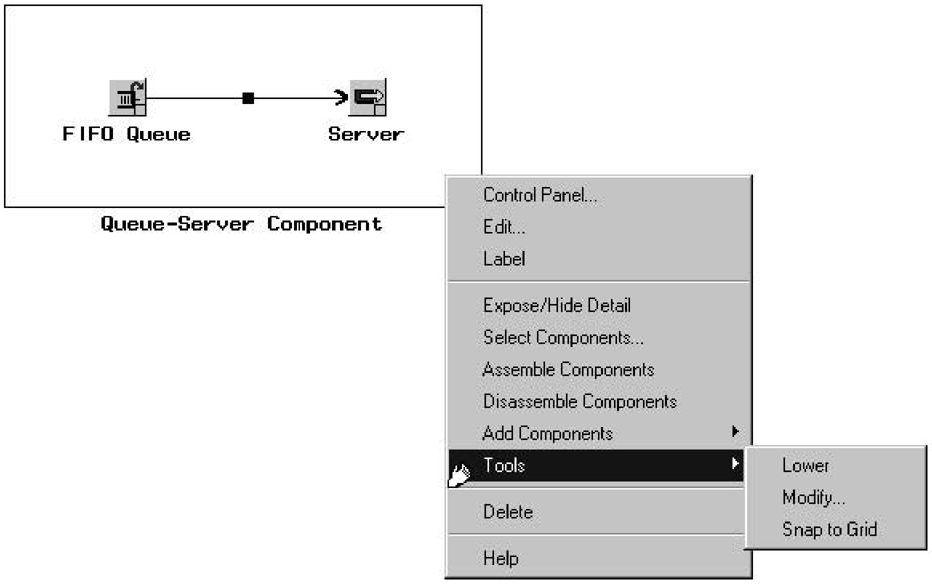


Figure 6.3. The Compound Component Pop-up Menu

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 9.2 on page 88).

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 6.4.

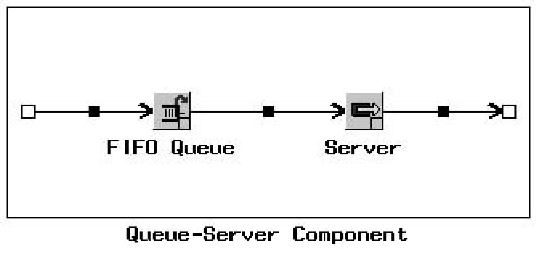


Figure 6.4. A Multi-Level Compound Component with Queue, Server, and Ports

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 6.4 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 6.4 looks like Figure 6.5.

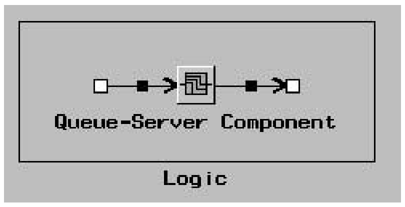


Figure 6.5. A Multi-Level Compound Component with Ports

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 6.5 and you obtain the example shown in Figure 6.6.

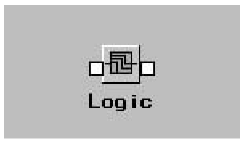


Figure 6.6. A Multi-Level Compound Component with Details Hidden

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 6.7.

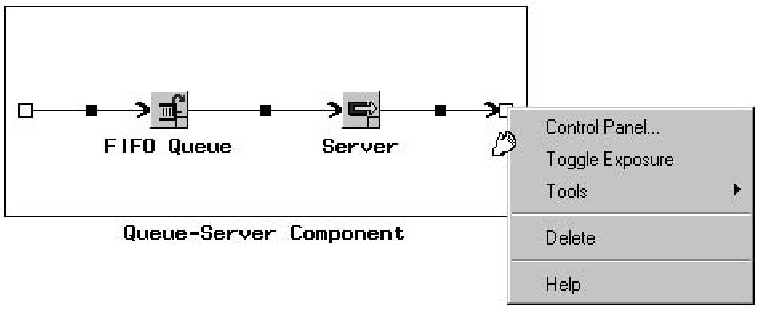


Figure 6.7. A Multi-Level Compound Component with Queue, Server, and Ports

The pop-up menu on compound components in Figure 6.3 shows the selection **Select Components...**. When this is selected for the component labeled Logic, the window in Figure 6.8 is displayed. In this window, you can select those components that you want exposed when queries are made on the Logic compound component.

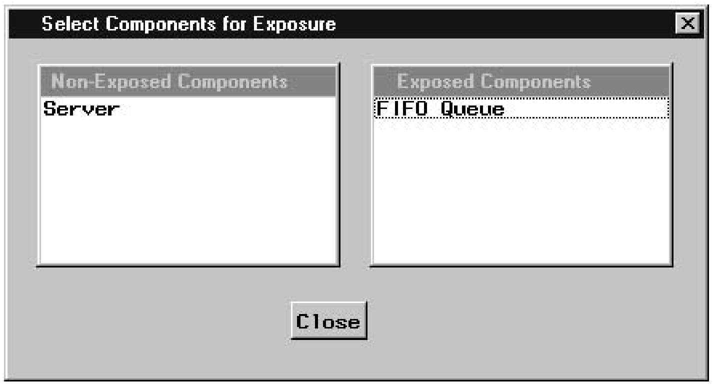


Figure 6.8. A Compound Component Control Panel

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 6.9.

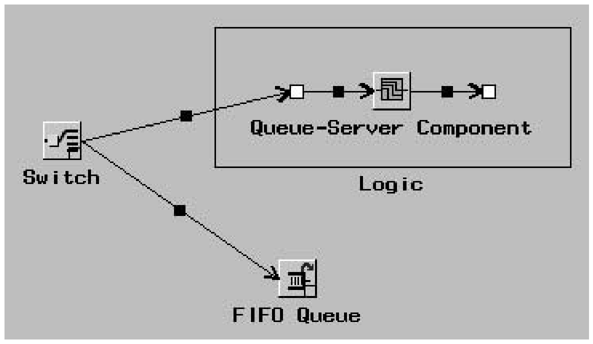


Figure 6.9. A Model with Switch and Compound Component

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 6.10.

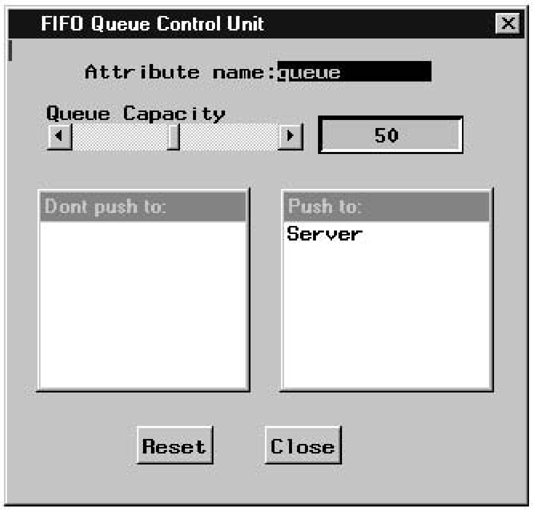


Figure 6.10. Control Panel for the Formula Model Element

This is because you had previously (in Figure 6.8) selected only that element to be exposed.

Editing Components

Suppose that you want to edit the Logic component in the model in Figure 6.9. 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 6.3). Figure 6.11 shows such an edit window for the Logic component.

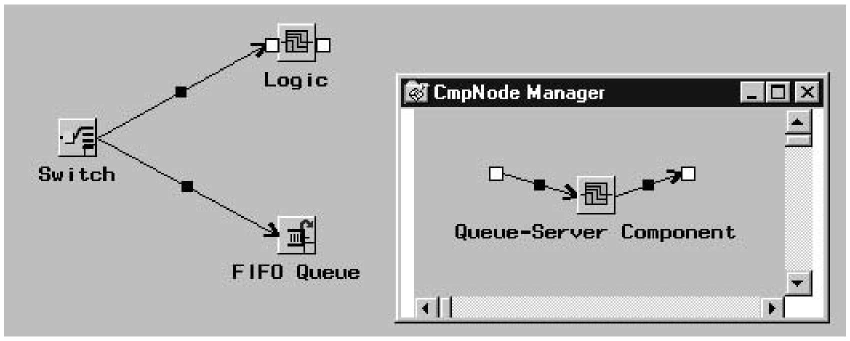


Figure 6.11. Editing the Logic Compound Component

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 6.12. Since this is in the palette and is a template, it can be replicated.

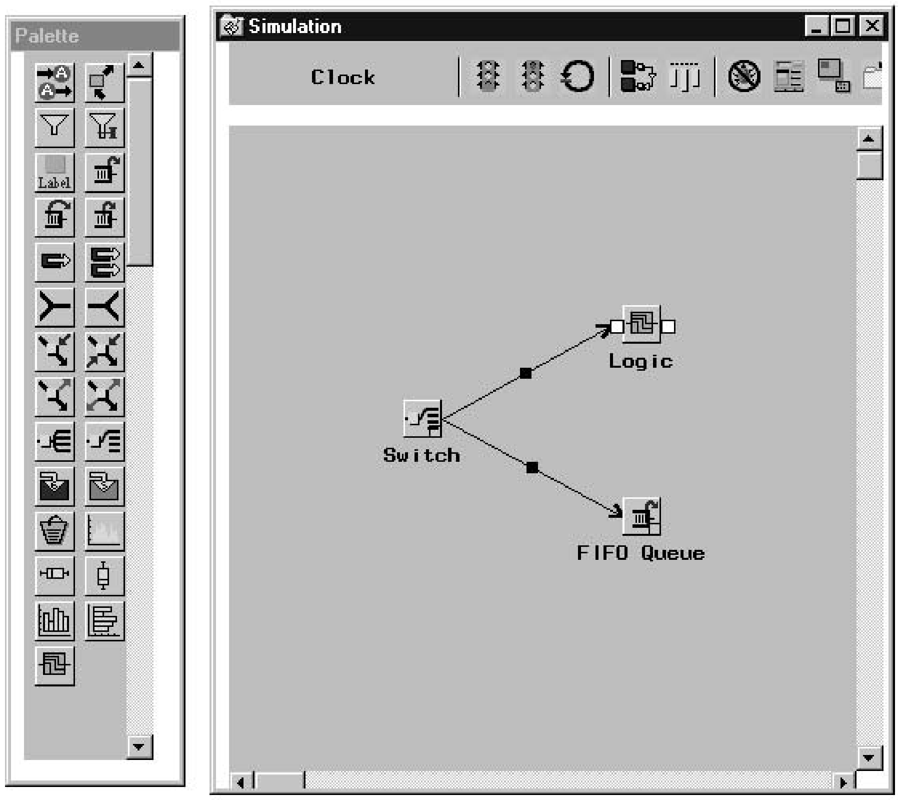


Figure 6.12. The Toolbar with the New "Logic" Button

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 8, "Saving and Restoring," for details.

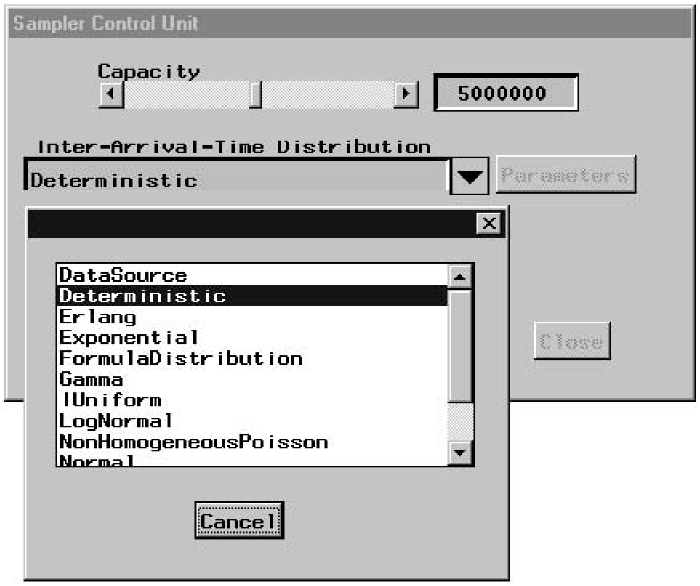
Note that it is possible to create compound components and, as a result, templates, that reference components not in the compound component and template itself. For example, you can have a Trigger as a template that references a queue not in that template. When the template is replicated, the Trigger 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 7
Random and Exogenous Variation in the Model

Chapter Contents

Chapter 7
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 7.1, 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.


Sampler Control Unit

Capacity: 5000000

Inter-Arrival-Time Distribution: Deterministic

DataSource
Deterministic
Erlang
Exponential
FormulaDistribution
Gamma
IUniform
LogNormal
NonHomogeneousPoisson
Normal

Figure 7.1. Sources of Variation

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.

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 λ and density function

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

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

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

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

  $$\lambda^* \equiv \sup_{t \geq 0} \lambda(t)$$

- Gamma, with parameters λ and n and density function

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

- Erlang, with parameters λ and n and density function

  $$f(x) = \frac{x^{n-1} \exp^{-x/\lambda}}{\lambda^n \Gamma(n)}, \text{ for } x \geq 0, \lambda > 0, n \epsilon Z^+$$

- Uniform, with parameter U and density function

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

- IUniform, with parameter U and density function

  $$f(x) = \frac{1}{\lfloor U \rfloor}, \text{ for } U \geq 1, 0 \leq x \leq \lfloor U \rfloor, x \epsilon Z^+$$

- Deterministic, with parameter U and density function

  $$f(x) = 1, \text{ for } x = U$$

Each of these generators has a control panel for setting parameters. For example, if you pick Exponential from the list box in Figure 7.1 and then click the **Parameters** button, the exponential control panel in Figure 7.2 will be displayed. In this window, you set the initial seed value and the mean $1/\lambda$.

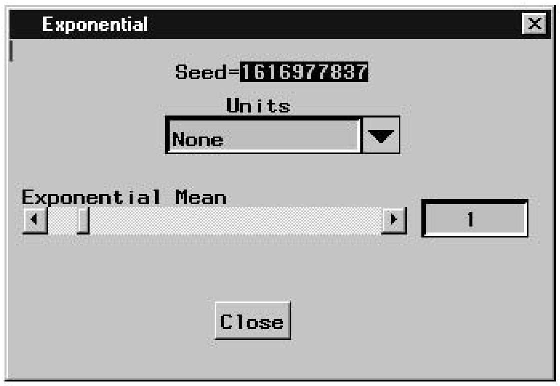


Figure 7.2. The Exponential Control Panel

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 7.2, 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 7.2 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 7.3) on the Simulation window.

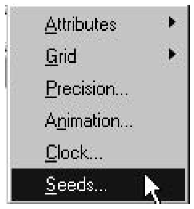


Figure 7.3. The Options Pull-down Menu

In the **Seed Preferences** window, shown in Figure 7.4, 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.

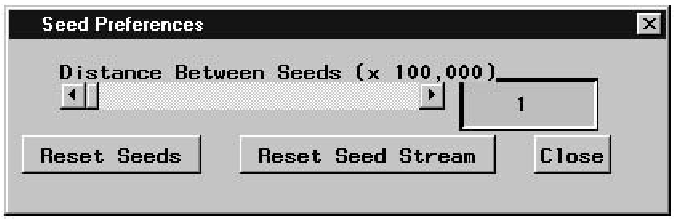


Figure 7.4. The Seed Control Panel

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 data set 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 7.2 for an example. With this you can identify a value for the time units. Then, with the Clock Options window, (see Figure 7.5) accessed by selecting **Options ➤ Clock Options...** from the Simulation window (Figure 7.3), you can assign time units to be used for the simulation clock.

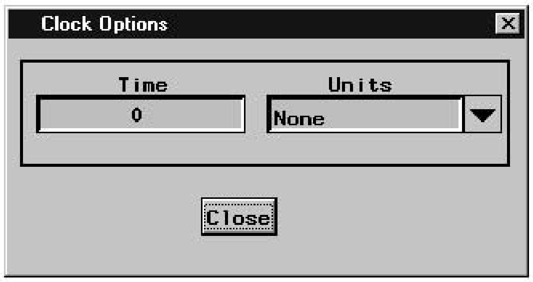


Figure 7.5. The Clock Control Panel

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 7.1. After you select it and then click the **Parameters** button, a Formula Manager window (as shown in Figure 5.1 on page 55) 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

$$f(x) = .5f_1(x) + .5f_2(x)$$

where $f_1(x)$ is exponential with parameter λ, and $f_2(x)$ is uniform. Figure 7.6 shows a portion of the four windows needed to express this.

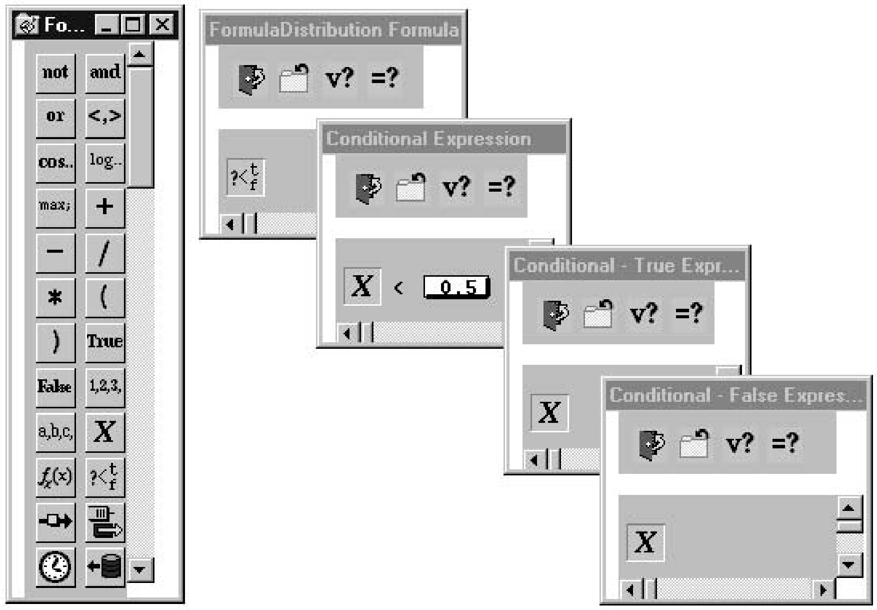
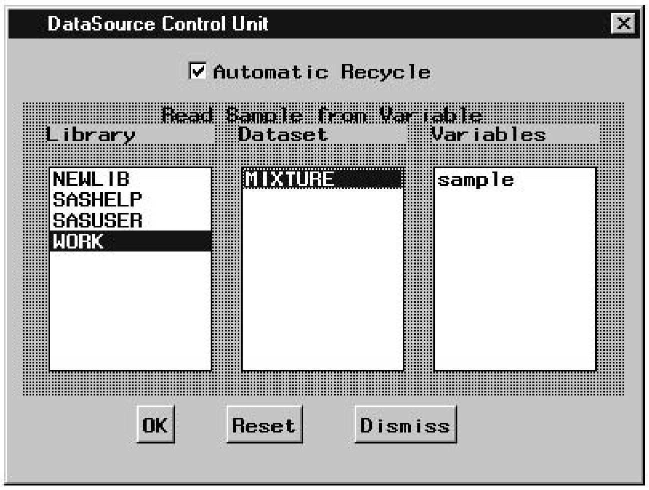


Figure 7.6. An Example of a Mixture Distribution

The bottom window is the FormulaDistribution Formula window, and it contains the conditional element ?<^t_f . 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 7.1. 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 7.7.



[✓ Automatic Recycle]

Library	Dataset	Variables
NEWLIB	MIXTURE	sample
SASHELP		
SASUSER		
WORK		

Figure 7.7. The DataSource Control

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 7.7 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 8
Saving and Restoring

Chapter Contents

Chapter 8
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.

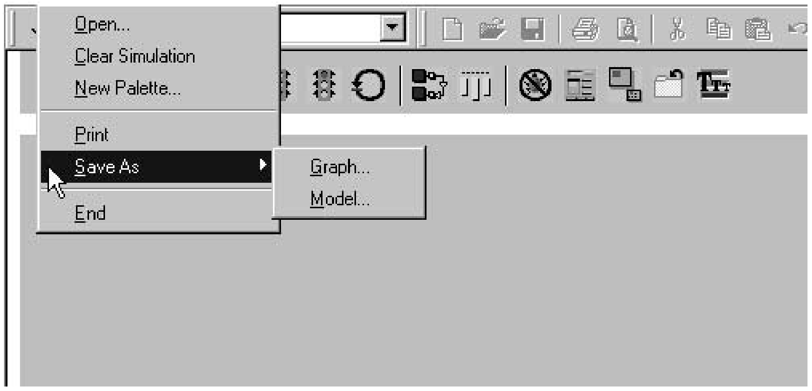
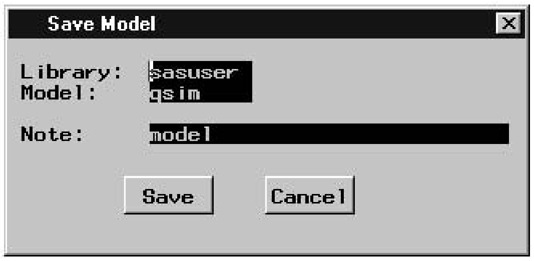


Figure 8.1. The Save As Selection on the Simulation Window

Figure 8.1 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 8.1, the Save Model window in Figure 8.2 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 8.1.



Library: sasuser
Model: qsim
Note: model

Figure 8.2. Save the Simulation Model

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 3, "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.

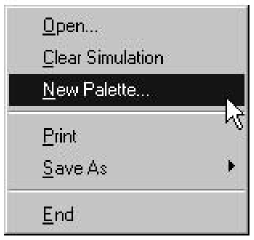


Figure 8.3. The Simulation File Pull-down Menu

Figure 8.3 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 8.4, 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.

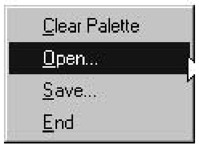
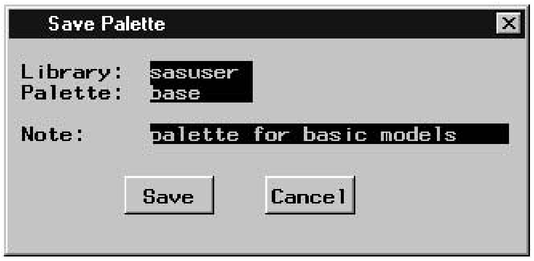


Figure 8.4. The Palette File Pull-down Menu

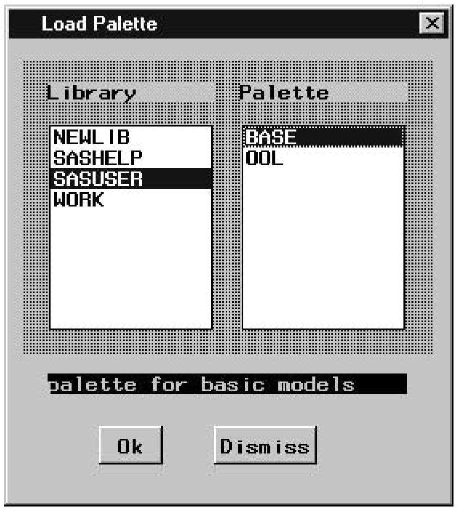
The palette **File** pull-down menu also has an **Open...** and **Save...**. With these selections, you can save all the templates in a palette into a SAS data set which can then, in another invocation of the QSIM Application, be opened and loaded into a palette.



Library: sasuser
Palette: base
Note: palette for basic models

Figure 8.5. Save Palette Window

In Figure 8.5, the palette is saved into a data set named SASUSER.BASE.



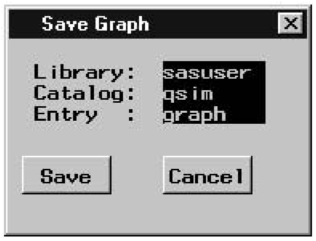
Library: NEWLIB, SASHELP, SASUSER, WORK
Palette: BASE, OOL
palette for basic models

Figure 8.6. Load Palettes Window

In Figure 8.6, 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.

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 8.1, the Save Graph window opens.



```
Library: sasuser
Catalog: qsim
Entry  : graph
```

Figure 8.7. 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 8.8 shows a printout of the data set for a simple M/M/1 model.

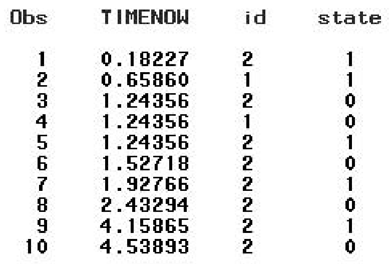
```
Obs     cmponent       id

 1     Sampler  0      0
 2     FifoQueue  1    1
 3     Server  2       2
```

Figure 8.8. 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 8.9 shows an example. Notice that the ID corresponds to either the server or the queue as given in the dictionary in Figure 8.8.



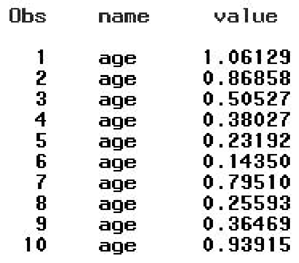
```
Obs    TIMENOW    id    state

  1    0.18227    2       1
  2    0.65860    1       1
  3    1.24356    2       0
  4    1.24356    1       0
  5    1.24356    2       1
  6    1.52718    2       0
  7    1.92766    2       1
  8    2.43294    2       0
  9    4.15865    2       1
 10    4.53893    2       0
```

Figure 8.9. 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 2.5 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 9.8 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.



```
Obs    name    value

  1    age    1.06129
  2    age    0.86858
  3    age    0.50527
  4    age    0.38027
  5    age    0.23192
  6    age    0.14350
  7    age    0.79510
  8    age    0.25593
  9    age    0.36469
 10    age    0.93915
```

Figure 8.10. An M/M/1 Transaction State Sample Path Data Set

Chapter 9
Analyzing the Sample Path

Chapter Contents

Chapter 9 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 8, "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 9.1.

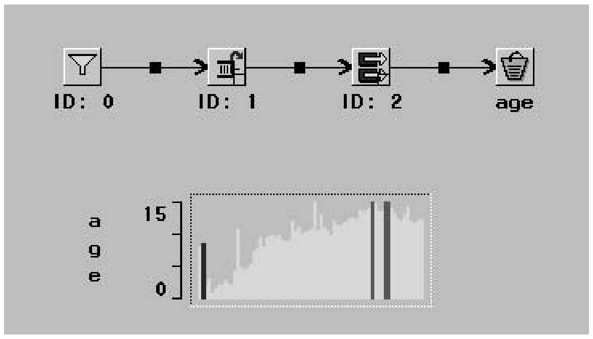


Figure 9.1. A Single Queue 3-Server Model

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 at many airports.

Running a Simulation

Suppose that you have built the model in Figure 9.1. 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.

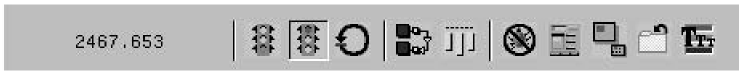
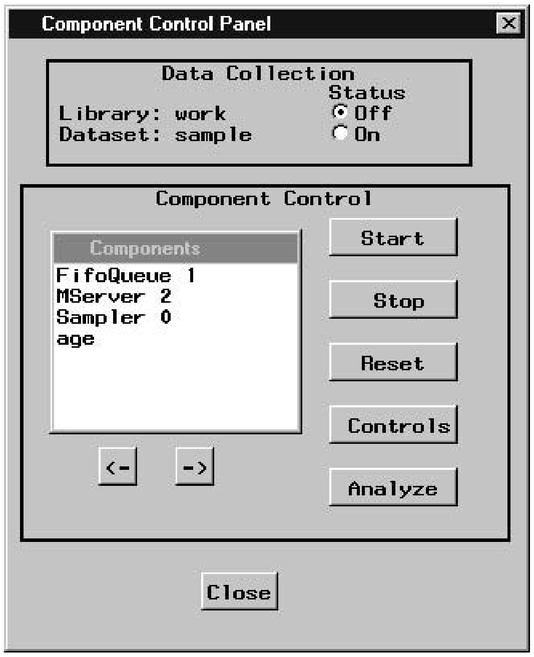


Figure 9.2. The Command Panel and Buttons

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 9.3.


Component Control Panel

Data Collection

Library: work
Dataset: sample

Status
[✓ Off]
On

Component Control

Components
```
FifoQueue 1
MServer 2
Sampler 0
age
```

Figure 9.3. 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.

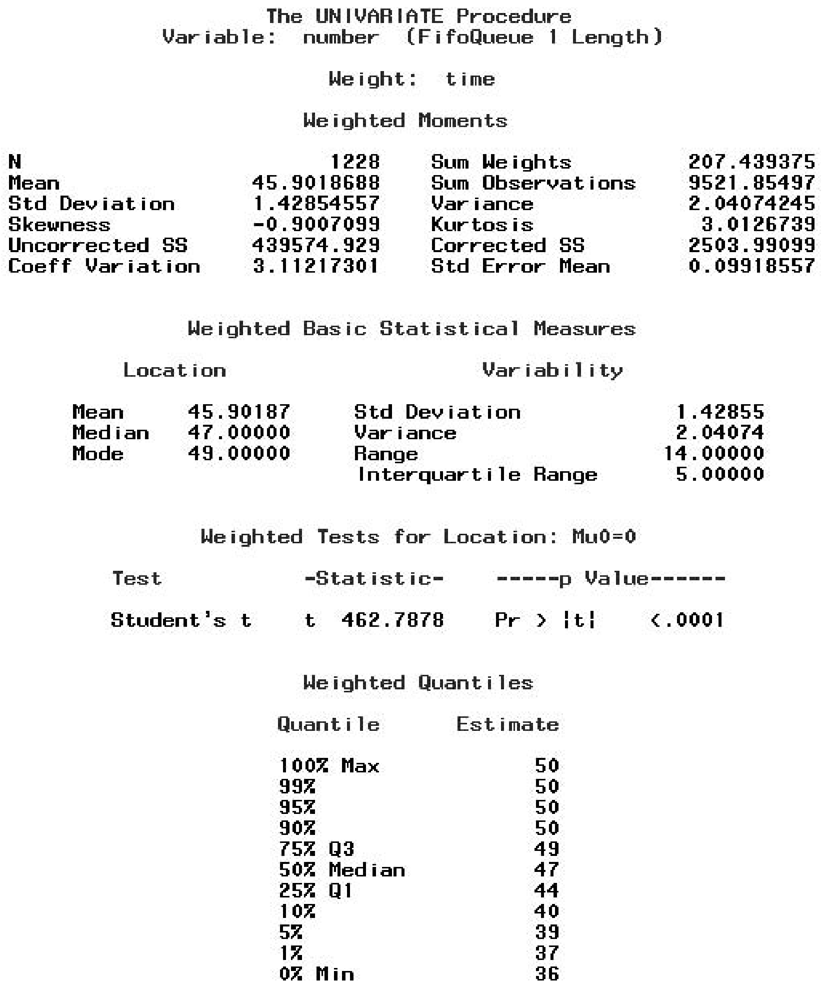
Statistics on Component State

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 section "Statistics on Transaction State" on page 97.

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 9.3. 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 9.1 by selecting FIFOQueue 1 in the Simulation Control panel and then by pressing **Analyze**.



```
                   The UNIVARIATE Procedure
          Variable:  number  (FifoQueue 1 Length)

                        Weight:  time

                      Weighted Moments

N                          1228    Sum Weights        207.439375
Mean                 45.9018688    Sum Observations   9521.85497
Std Deviation        1.42854557    Variance           2.04074245
Skewness             -0.9007099    Kurtosis            3.0126739
Uncorrected SS       439574.929    Corrected SS       2503.99099
Coeff Variation      3.11217301    Std Error Mean     0.09918557


             Weighted Basic Statistical Measures

        Location                    Variability

     Mean     45.90187     Std Deviation            1.42855
     Median   47.00000     Variance                 2.04074
     Mode     49.00000     Range                   14.00000
                            Interquartile Range      5.00000


              Weighted Tests for Location: Mu0=0

        Test           -Statistic-    -----p Value------

        Student's t    t  462.7878    Pr > |t|    <.0001


                      Weighted Quantiles

                    Quantile      Estimate

                    100% Max            50
                    99%                 50
                    95%                 50
                    90%                 50
                    75% Q3              49
                    50% Median          47
                    25% Q1              44
                    10%                 40
                    5%                  39
                    1%                  37
                    0% Min              36
```

Figure 9.4. Statistics on Queue Length

This generates the report shown in Figure 9.4 in the SAS Output Window. Figure 9.4 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 9.5. 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.

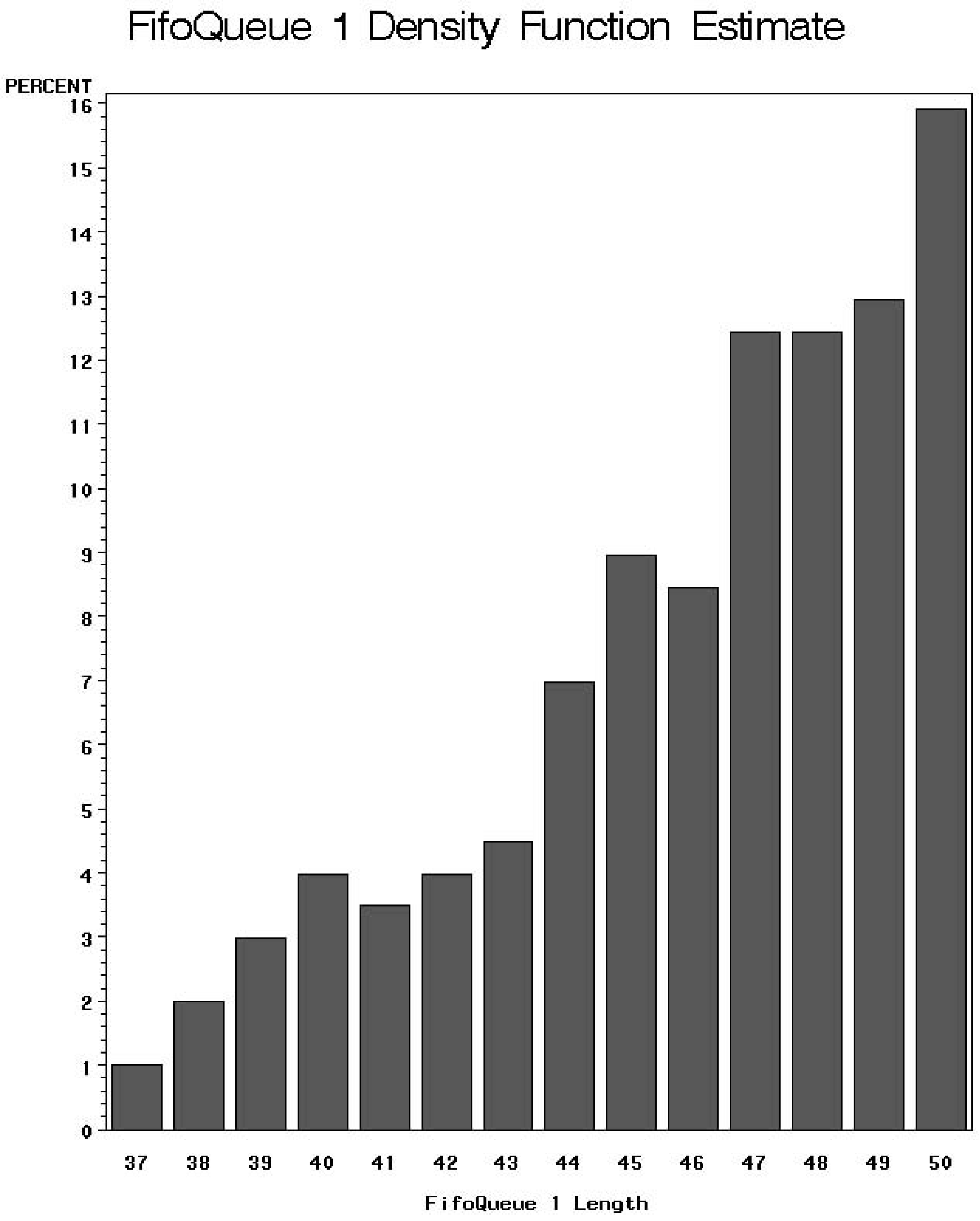
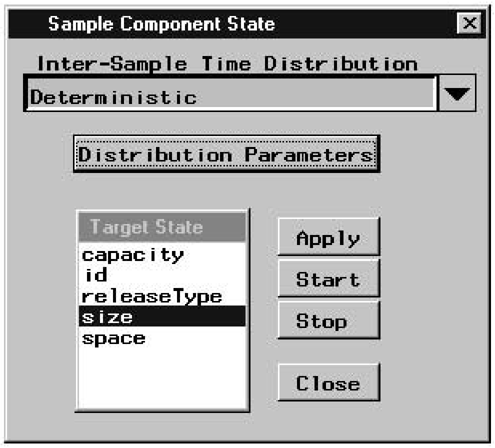


Figure 9.5. Queue Length Histogram

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 9.6 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

Inter-Sample Time Distribution: Deterministic

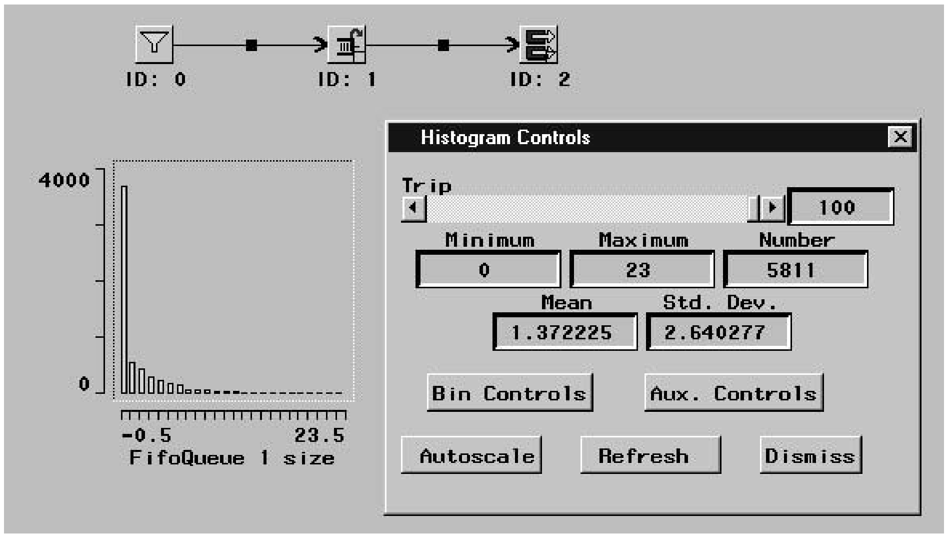
Distribution Parameters

Target State:
- capacity
- id
- releaseType
- size (selected)
- space

Apply | Start | Stop | Close

Figure 9.6. Sample Component State Control Panel

This control panel is opened when the queue labeled "ID: 1" in Figure 3.17 on page 43 is dropped on the histogram.



Histogram Controls

Trip: 100

Minimum	Maximum	Number
0	23	5811

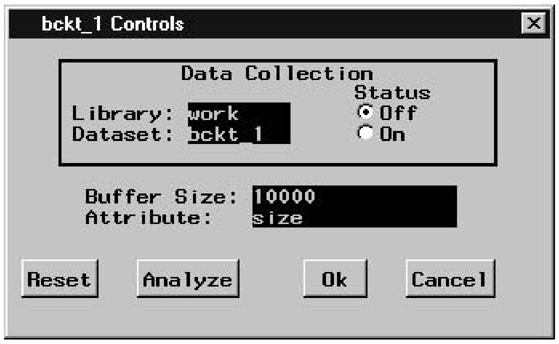
Mean	Std. Dev.
1.372225	2.640277

Bin Controls | Aux. Controls

Autoscale | Refresh | Dismiss

Figure 9.7. Vertical Histogram of the FIFO Queue length

Figure 3.17 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 3.15 on page 42.


bckt_1 Controls

Data Collection

Library: work
Dataset: bckt_1

Status
[✓ Off]
On

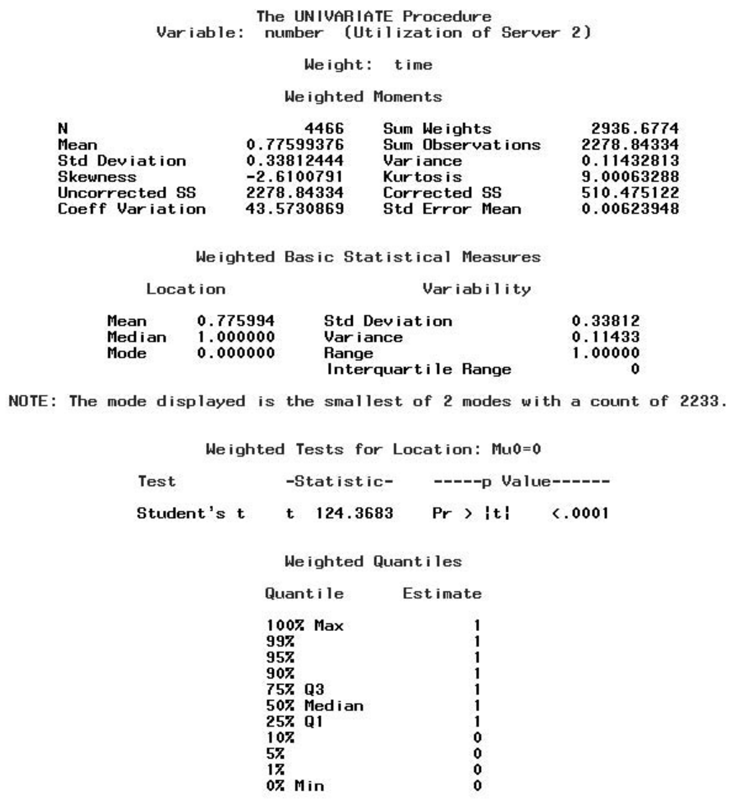
Buffer Size: 10000
Attribute: size

Figure 9.8. 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 9.9.



```
                          The UNIVARIATE Procedure
                 Variable:  number  (Utilization of Server 2)

                               Weight:  time

                             Weighted Moments

      N                        4466    Sum Weights          2936.6774
      Mean               0.77599376    Sum Observations    2278.84334
      Std Deviation      0.33812444    Variance            0.11432813
      Skewness           -2.6100791    Kurtosis            9.00063288
      Uncorrected SS     2278.84334    Corrected SS        510.475122
      Coeff Variation    43.5730869    Std Error Mean      0.00623948

                   Weighted Basic Statistical Measures

             Location                    Variability

          Mean     0.775994     Std Deviation            0.33812
          Median   1.000000     Variance                 0.11433
          Mode     0.000000     Range                    1.00000
                                 Interquartile Range           0

  NOTE: The mode displayed is the smallest of 2 modes with a count of 2233.

                     Weighted Tests for Location: Mu0=0

            Test           -Statistic-    -----p Value------

            Student's t    t  124.3683    Pr > |t|    <.0001

                           Weighted Quantiles

                         Quantile      Estimate

                         100% Max             1
                         99%                  1
                         95%                  1
                         90%                  1
                         75% Q3               1
                         50% Median           1
                         25% Q1               1
                         10%                  0
                         5%                   0
                         1%                   0
                         0% Min               0
```

Figure 9.9. Statistics on Server Utilization

The mean shown is approximately .77. This is an estimate of the probability that the server is busy.

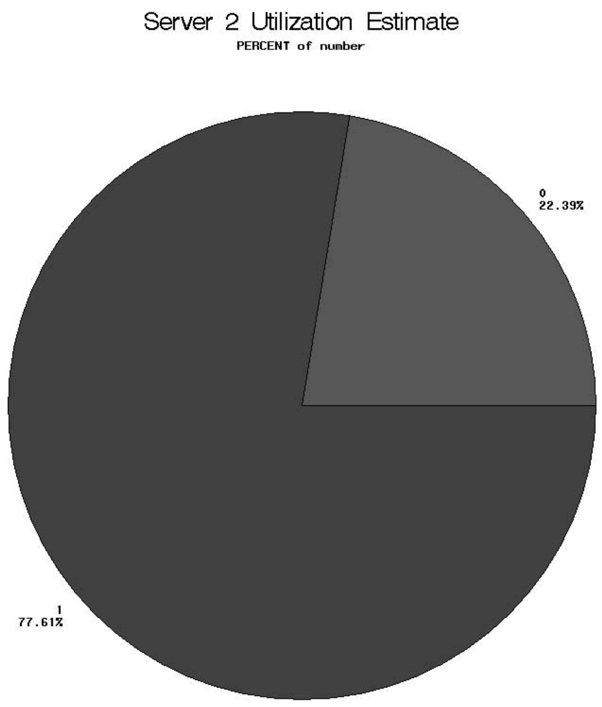
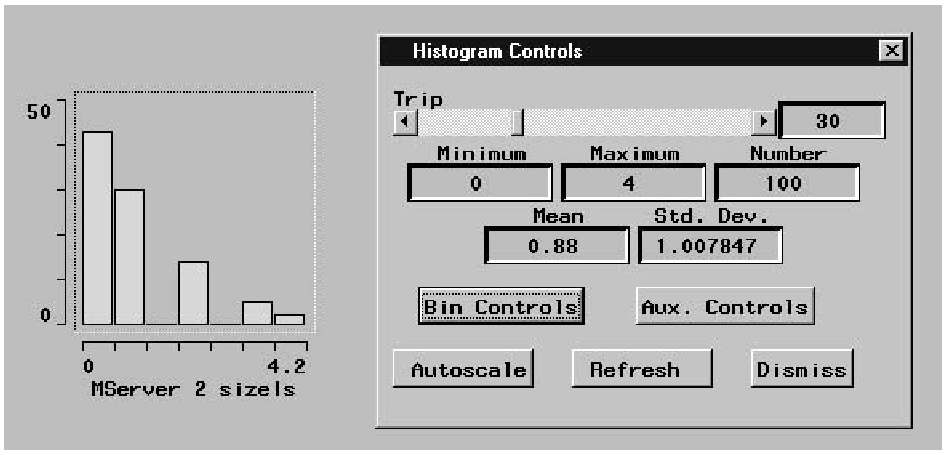


Figure 9.10. Server Utilization

The pie chart in Figure 9.10 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 9.11 shows a VHistogram and a Histogram control panel. This histogram had an MServer dropped on it and is monitoring the number of busy servers.



Trip: 30
Minimum: 0
Maximum: 4
Number: 100
Mean: 0.88
Std. Dev.: 1.007847

Figure 9.11. Histogram of Server Utilization

DATA Step Program for Extracting Information

When you click the **Analyze** button in the Simulation control panel (see Figure 9.3), 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:

```
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 9.5.

```
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 9.10 and Figure 9.9.

```
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;
```

```
proc gchart;
   pie busy / sumvar=time discrete percent=outside;
run;
```

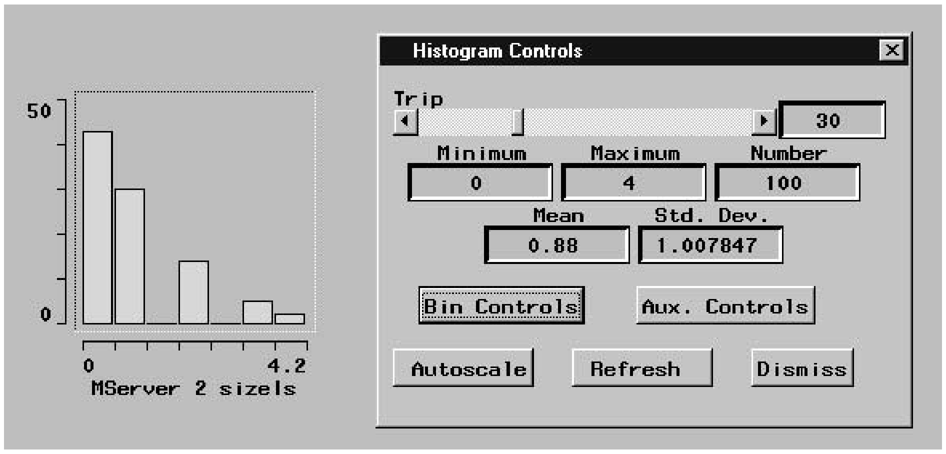
Statistics on Transaction State

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 3.15 on page 42.

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 3.15 on page 42 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.



Trip: 30
Minimum: 0
Maximum: 4
Number: 100
Mean: 0.88
Std. Dev.: 1.007847

Figure 9.12. 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 9.12.

Chapter 10
Selected Examples

Chapter Contents

Chapter 10
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 10.1 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 10.1.

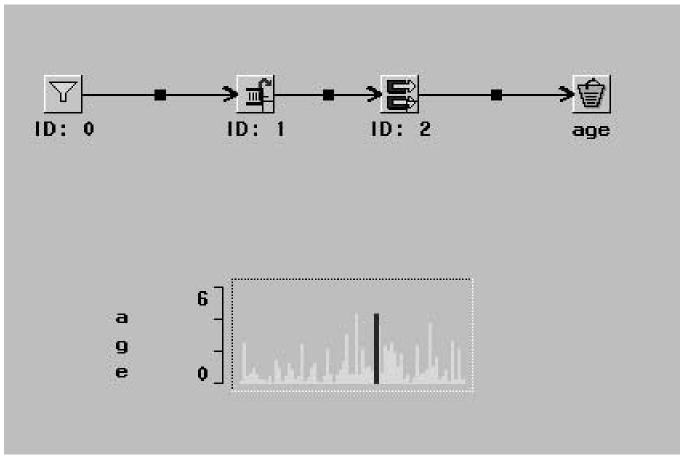


Figure 10.1. An M/M/c/K Model

This model is often compared to one with c parallel queues and servers, as shown in Figure 10.2.

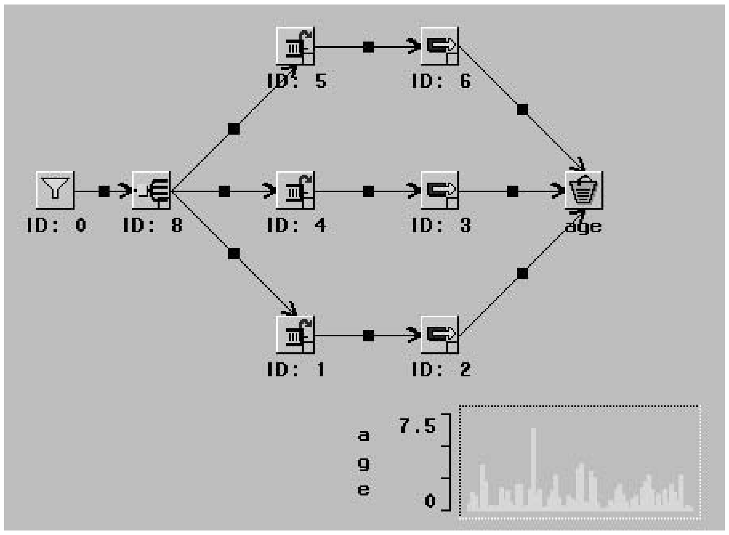


Figure 10.2. A 3-Queue 3-Server Model

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 3.11 on page 34.

Another variant on the parallel server models in Figure 10.1 and Figure 10.2 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 10.3.

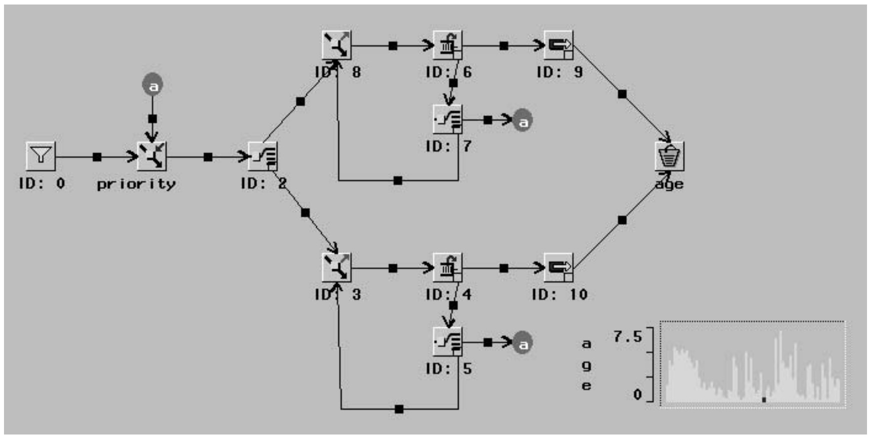


Figure 10.3. A 2-Queue 2-Server Model with Reneging

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.

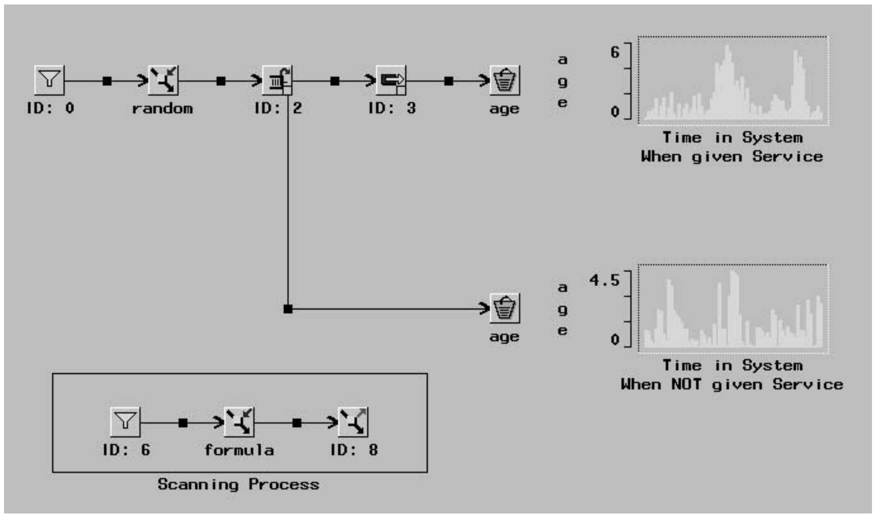


Figure 10.4. Scanning a Queue

Figure 10.4 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 < 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 *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 10.5 shows one such model having two classes.

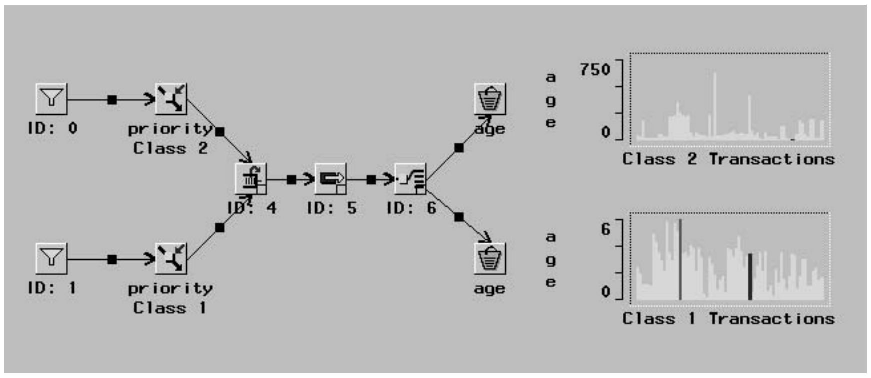


Figure 10.5. A Priority Queue Example with Two Transaction Classes

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 10.5 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 10.6 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.

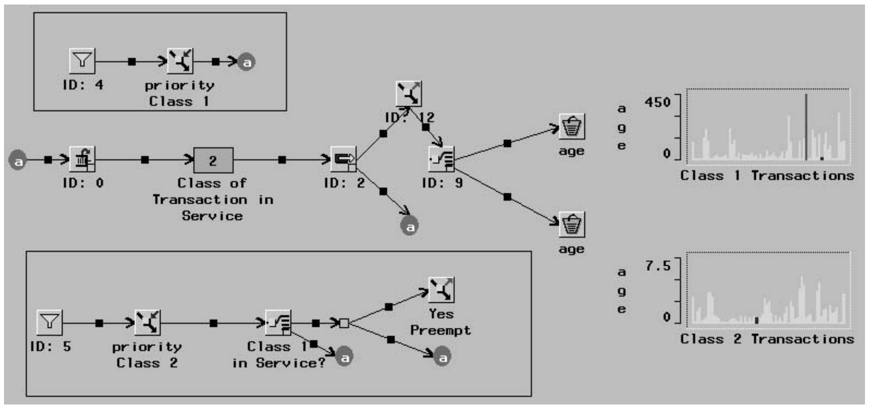


Figure 10.6. A Priority Queue Example with Two Transaction Classes and Preemption

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 10.7 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."

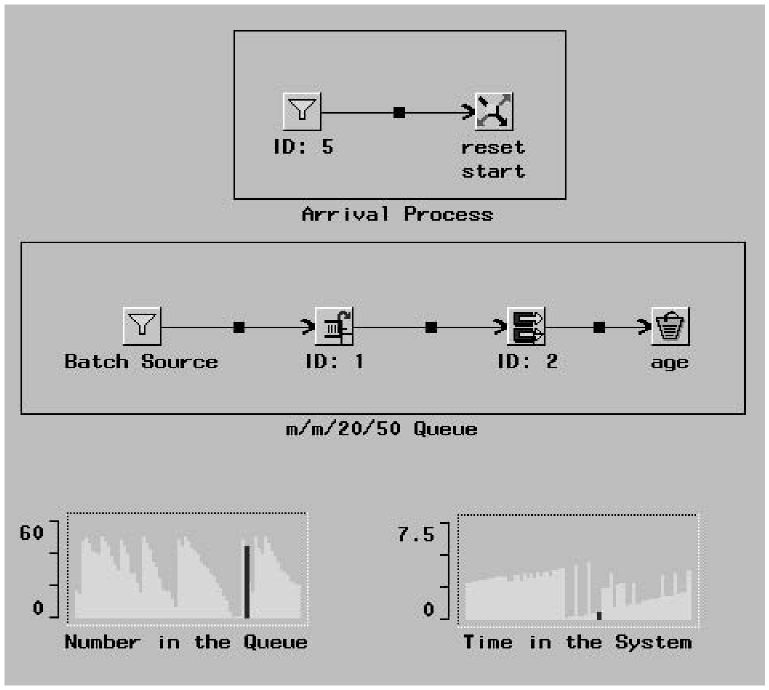


Figure 10.7. Batch Arrivals

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.

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.

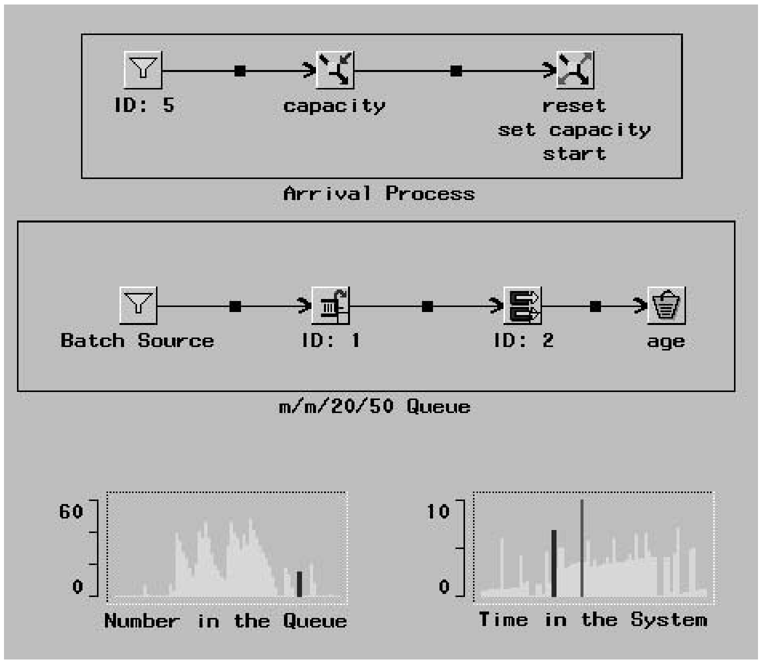


Figure 10.8. Batch Arrivals

Figure 10.8 shows two components added to the model of Figure 10.7: 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 10.9, the model has deterministic arrival rate and nonhomogeneous Poisson service times.

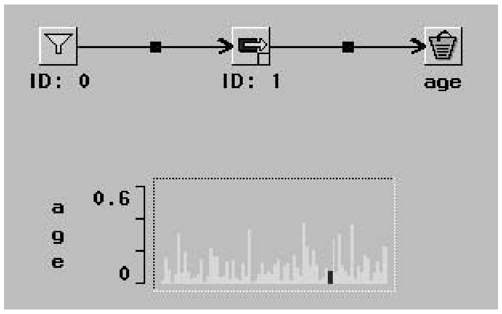
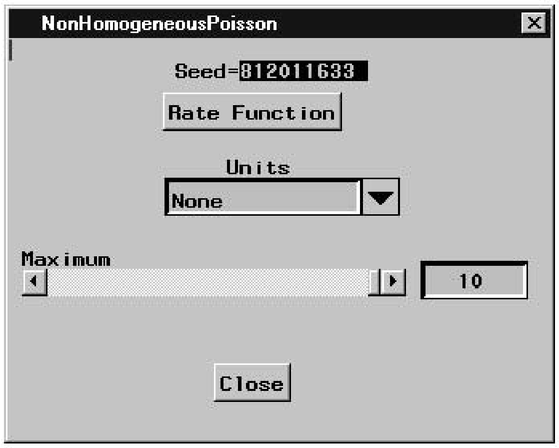


Figure 10.9. Nonhomogeneous Poisson Service

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 10.10.


NonHomogeneousPoisson

Seed=812011633

Units: None

Maximum: 10

Figure 10.10. 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.

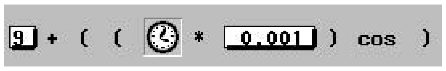

9 + ((⏲ * 0.001) cos)

Figure 10.11. Rate Function

Figure 10.11 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 10.9, 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.

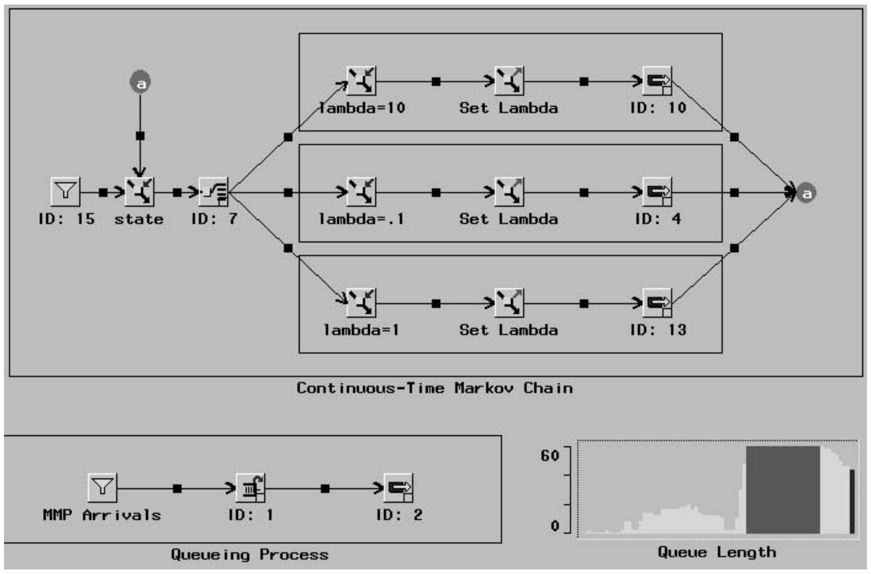


Figure 10.12. Markov-Modulated Poisson Arrivals

Figure 10.12 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 10.13, in which there are multiple classes of transactions to a single queue.

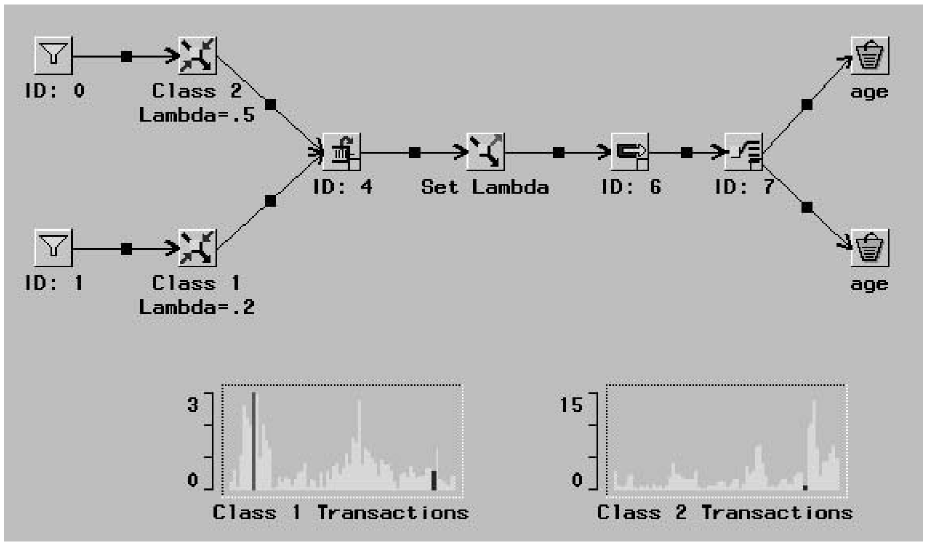


Figure 10.13. State Dependent Service

Figure 10.13 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.

Servers That Break Down I

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 10.14 shows such a model. The Server component labeled is the server that experiences down periods when it cannot service transactions.

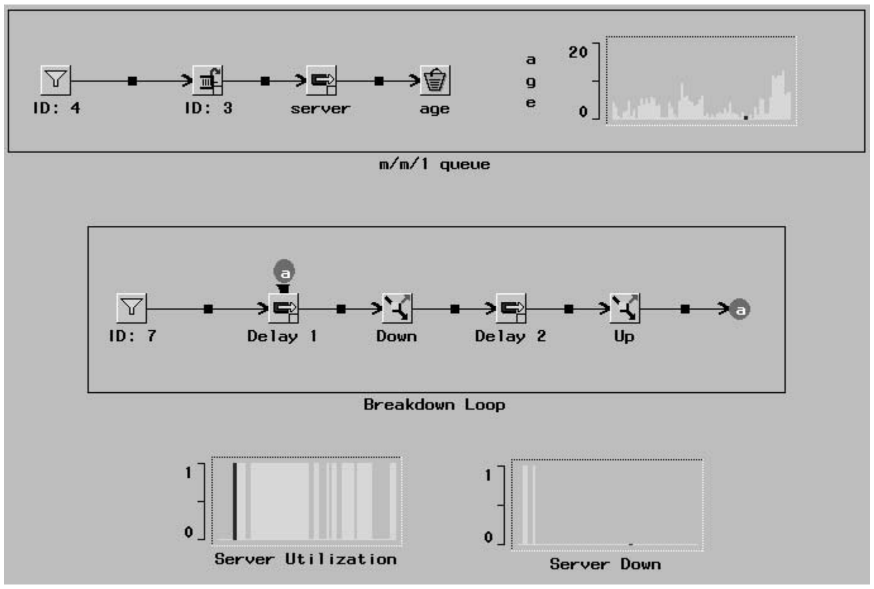


Figure 10.14. A Server That Breaks Down

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.

Servers That Break Down II

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 10.14, even though the server is stopped when it breaks, the transaction in service completes service. The model in Figure 10.15 adds the preemption of the transaction in service, which is routed back into the queue.

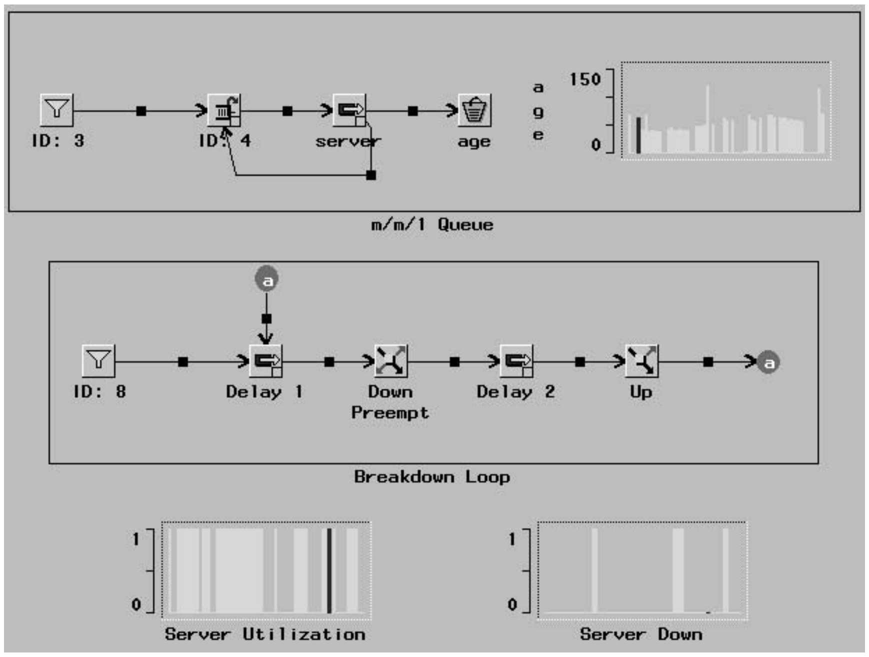


Figure 10.15. A Server That Breaks Down

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 3.5). See the preceding example on priority queues.

Batch Service I

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.

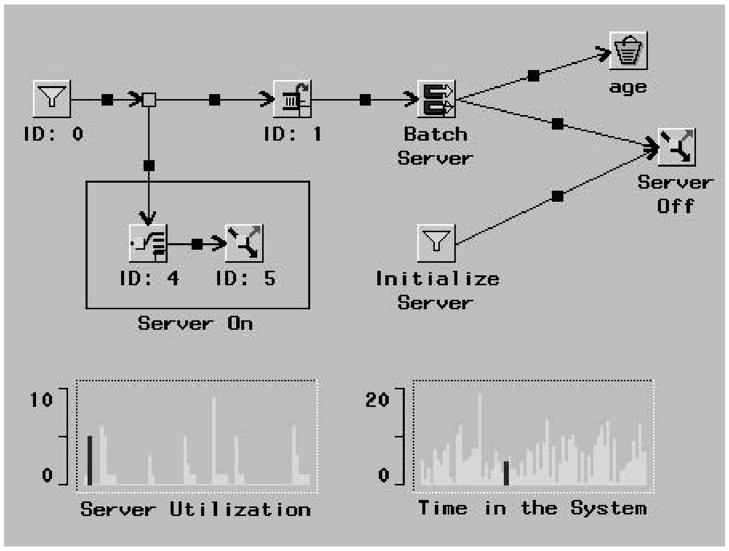


Figure 10.16. Batch Service

The model in Figure 10.16 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 10.17 accomplishes this.

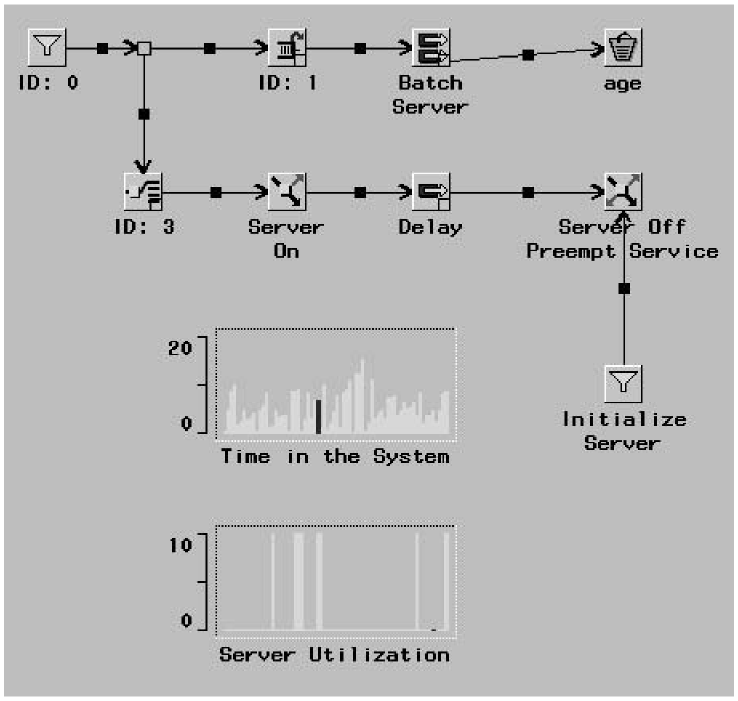


Figure 10.17. Batch Service

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.

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.

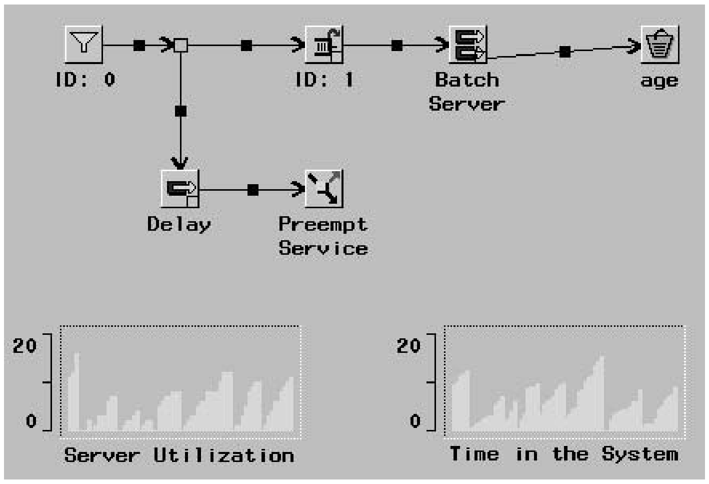


Figure 10.18. Batch Service

Figure 10.18 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.

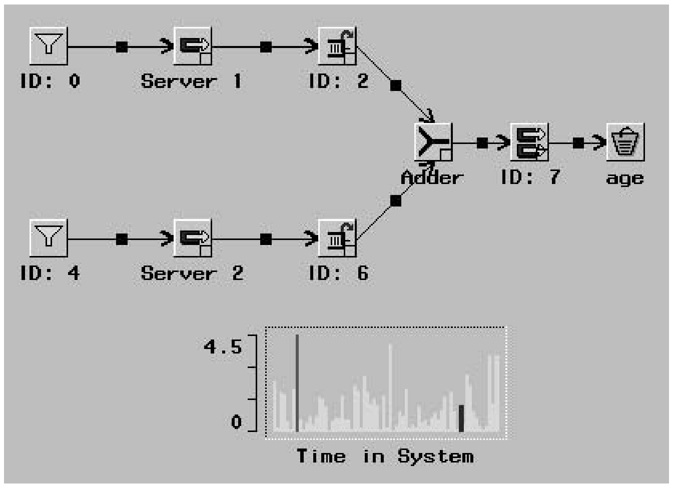


Figure 10.19. Assembly Unit

Figure 10.19 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.

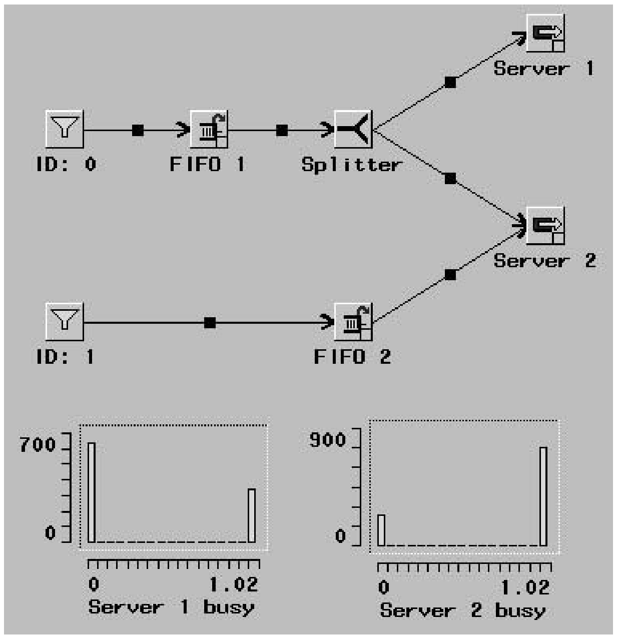


Figure 10.20. Servers as Resources

Figure 10.20 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).

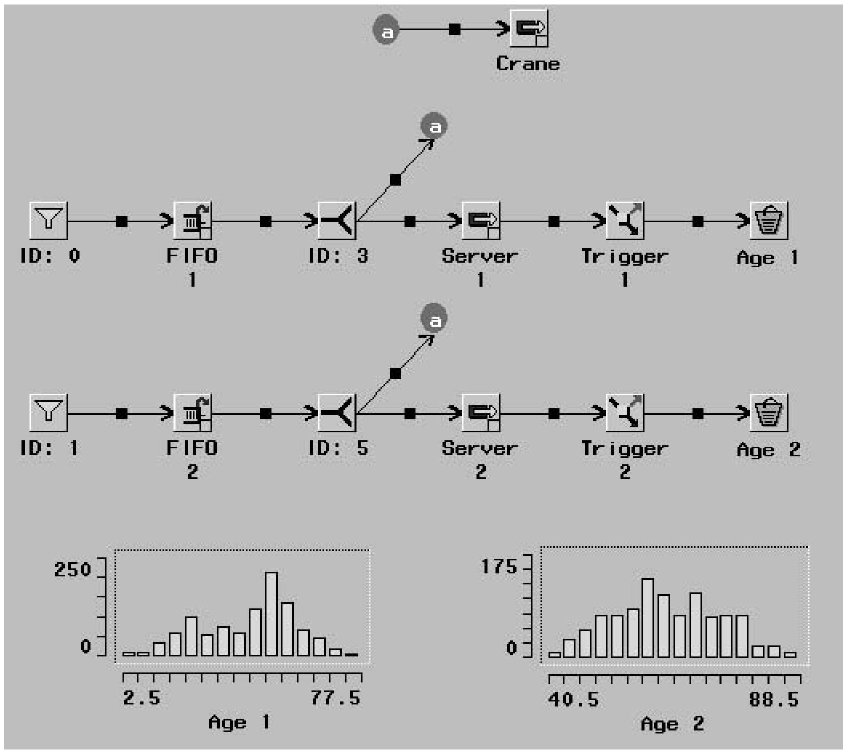


Figure 10.21. Servers as Resources

Figure 10.21 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 4.2 on page 52), 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.

Special Routing

There are other ways that independent streams can share resources. One is illustrated in Figure 10.22.

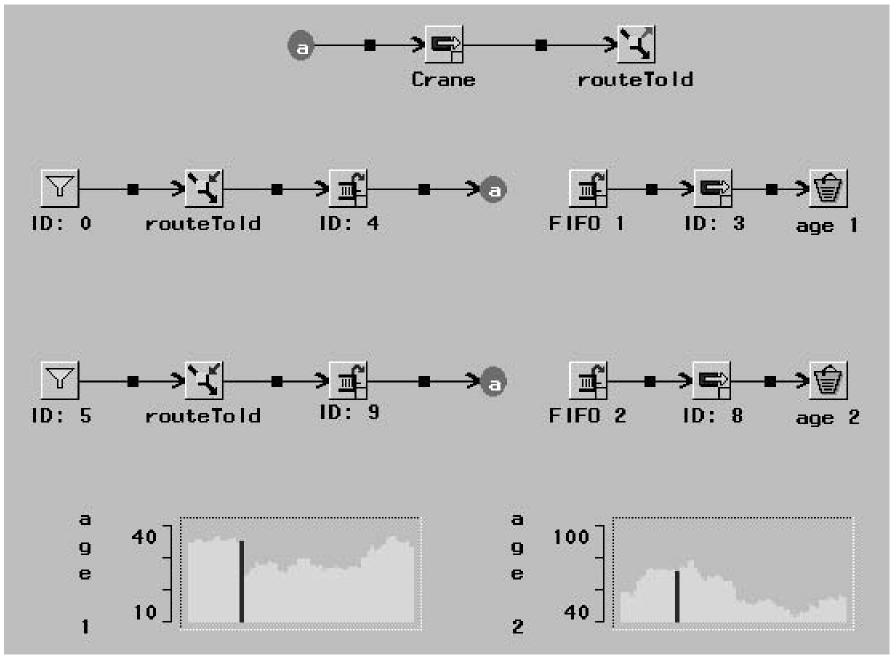


Figure 10.22. Special Routing

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

Bratley, P., Fox, B.L., and Schrage, L.E. (1983), *A Guide to Simulation*, New York: Springer-Verlag.

Fischer, W. and Meier-Hellstern K. (1992), "The Markov-modulated Poisson process (MMPP) cookbook," *Performance Evaluation*, 18, 149-171.

Fishman, G.S. (1978), *Principles of Discrete Event Simulation*, New York: John Wiley & Sons.

Gross, D. and Harris, C.M. (1985), *Fundamentals of Queueing Theory, Second Edition*, New York: John Wiley & Sons.

Kleijnen, J.P.C. (1987), *Statistical Tools for Simulation Practitioners*, New York: Marcel Dekker, Inc.

Kleijnen, J.P.C. (1974), *Statistical Techniques in Simulation - Part I*, New York: Marcel Dekker, Inc.

Kleijnen, J.P.C. (1975), *Statistical Techniques in Simulation - Part II*, New York: Marcel Dekker, Inc.

Law, A.M. and Kelton, W.E. (1982), *Simulation Modeling and Analysis*, New York: McGraw-Hill Book Company.

Rubinstein, R.Y. (1981), *Simulation and the Monte Carlo Method*, New York: John Wiley & Sons.

Index

Your Turn