Paper SAS1770-2015

Practical Applications of SAS® Simulation Studio

Ed Hughes and Emily Lada, SAS Institute Inc.

ABSTRACT

SAS® Simulation Studio, a component of SAS/OR® software for Microsoft Windows environments, provides powerful and versatile capabilities for building, executing, and analyzing discrete-event simulation models in a graphical environment. Its object-oriented, drag-and-drop modeling makes building and working with simulation models accessible to novice users, and its broad range of model configuration options and advanced capabilities makes SAS Simulation Studio suitable also for sophisticated, detailed simulation modeling and analysis. Although the number of modeling blocks in SAS Simulation Studio is small enough to be manageable, the number of ways in which they can be combined and connected is almost limitless. This paper explores some of the modeling methods and constructs that have proven most useful in practical modeling with SAS Simulation Studio. SAS has worked with customers who have applied SAS Simulation Studio to measure, predict, and improve system performance in many different industries, including banking, public utilities, pharmaceuticals, manufacturing, prisons, hospitals, and insurance. This paper looks at some discrete-event simulation modeling needs that arise in specific settings and some that have broader applicability, and it considers the ways in which SAS Simulation Studio modeling can meet those needs.

INTRODUCTION

SAS Simulation Studio provides a graphical, object-oriented environment for discrete-event simulation modeling and analysis, with a goal of making its capabilities easily and directly accessible. However, like any technologically sophisticated product, it has a number of useful features, and some are more obvious than others. It's not surprising that some elements elude even ardent and diligent users.

This paper highlights a number of useful and productive SAS Simulation Studio features, tips, and modeling techniques. Topics that are explored include choices in entity modeling, working with data, and managing the execution of a model. As a reader, it's advisable for you to have prior experience in discrete-event simulation modeling, and it's preferred that you have direct experience with SAS Simulation Studio.

ENTITY MODELING CHOICES

Entities, which represent the individuals or objects that move through the workflow structure of a system, are critical elements of simulation models. How you choose to use entities affects the clarity and accuracy of your models, and also influences how efficiently your models can be executed.

CHOOSING BETWEEN REGULAR ENTITIES AND RESOURCE ENTITIES

In SAS Simulation Studio you can use both regular entities and resource entities in models, and in some cases it's relatively easy to see which class of entity best represents a specific type of individual or object in the system you are modeling. At other times, the choice is more subtly suggested. In a practical example, resource entities are the best choice, but the advantages of using them—though substantial—are not immediately apparent.

In a model of a large and complex supply chain that was built for a prominent telecommunications manufacturer, a SAS consultant needed to represent inventories for each of hundreds of electronic components that were then assembled into dozens of finished components. For each component, inventory could easily total thousands of items. Although it would certainly be acceptable in theory to represent each item of inventory by a separate regular entity, this approach would result in potentially hundreds of thousands or even millions of entities being active in the model simultaneously. Because the time that you require to run a simulation model correlates directly with the number of active entities in the model, this is a situation to avoid.

The consultant took the much more scalable approach of modeling the inventory of each component as a single resource entity. Each resource entity carries an attribute that denotes the current inventory level of the item it represents. All these resource entities can be stored in Resource Pool blocks. They are seized by a regular entity that represents an in-progress final product. A resource entity is seized, its inventory attribute is decremented by the required number of units of its represented component, and the entity is then released back to its Resource Pool block. This modeling approach results in a model that contains far fewer entities and in which even fewer entities are active at any one time, thus considerably shortening the model's run time.

MANAGING MULTIPLE ENTITY TYPES WITH ONE ENTITY GENERATOR

Whether you're working with regular entities, resource entities, or a mix of both classes, in SAS Simulation Studio it's easy to create and use many types of entities in the same model. You can select any defined entity type in the block properties dialog box of an Entity Generator block, so that the selected Entity Generator block creates entities of the selected type. Multiple entity types could represent different manufactured products in a factory model, different types of merchandise orders in a supply chain model, or other distinctions among objects or individuals in a system that you are modeling.

In a model with just a few entity types, it's natural to use a separate Entity Generator block for each type. It's tempting to continue to use this paradigm as the number of entity types grows, but doing so can create problems. For a large number of entities, your model would have a corresponding number of Entity Generator blocks, eventually making the model unwieldy and difficult to maintain. There is a better, more efficient alternative: you can generate one entity for each type by using a single Entity Generator block, and recursively model arrivals of an unlimited number of entity types. The key to this approach is to create generic entities and let an attribute designate the type of each entity, rather than creating multiple entity types. The two alternatives are equivalent on a technical basis, but using a single Entity Generator block is much more scalable, and this approach has been used successfully to build very large simulation models.

Figure 1 shows a model that creates and uses regular entities of two types but can generalize to as many entity types as you want. In the compound block labeled "Initial Entities," an Entity Generator creates a single batch of two entities, one for each type. A Modifier block assigns the attributes InterArrivalMean (mean interarrival time) and Type to each entity, obtaining the data from an Observation Source block that reads a SAS data set holding this information.

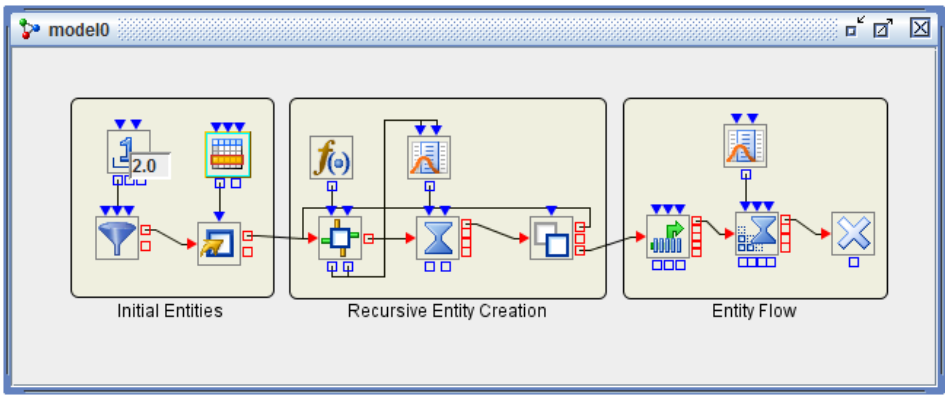


Figure 1. A Scalable Method of Creating Multiple Types of Entities

Next, the two entities enter the stage in which multiple instances of each entity are generated recursively. The entities first flow through a Gate block, which is used to dynamically set the mean of an exponential distribution according to each entity's InterArrivalMean attribute as follows. The first output value port on the Gate block supplies a text string to the InStreamPolicy port of a Numeric Source block. A Formula block reads the InterArrivalMean attribute value and constructs a text string to specify it as the mean of the distribution sampled in the Numeric Source block. The block properties dialog box for this Formula block appears in Figure 2. The second output value port on the Gate block sends a true Boolean value to

the InUpdate port of the Numeric Source block, signaling it to pull information about its probability distribution through its InStreamPolicy port.

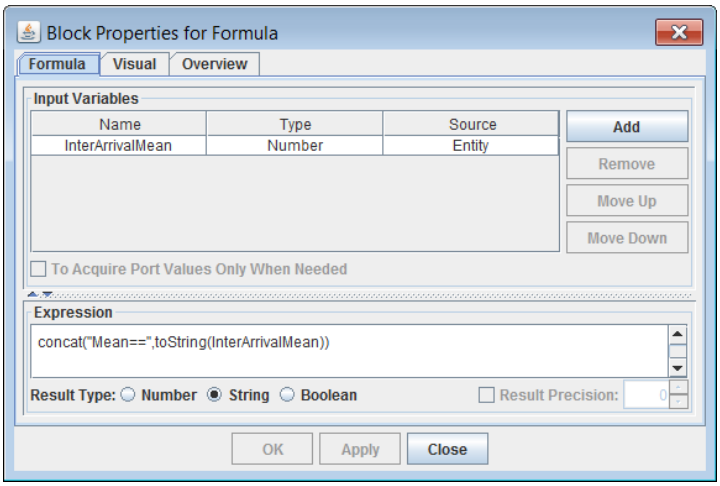


Figure 2. Specifying an Attribute Value as a Distribution Mean

The Numeric Source block supplies the delay time to a Delay block that the entities visit next. This Delay block represents the interarrival time for each type of entity that enters it. When this interval elapses, it's time to create a new entity of the same type. A Clone block creates one copy of the entity and sends it into the main flow of the model (the compound block labeled "Entity Flow"). The original entity travels back to the Gate block and repeats the process, generating another new entity that has the same Type attribute value after the next sampled interarrival time elapses.

You don't need to add any blocks to this model to change the number of entity types created. The only changes that you need to make are in the batch size specified for the Entity Generator block and in the number of observations in the SAS data set that holds the attribute information for the various types of entities. This approach is eminently scalable.

A Note about Probability Distributions and Experimental Design

In this example, a Gate block supplies a Numeric Source block with a text string that specifies the mean of a probability distribution, so that the distribution in the Numeric Source block is entity-specific. Such a text string can specify any supported probability distribution or its parameters. For example, to specify a normal distribution with a mean of 1 and a standard deviation of 2, the text string value is

```
Class==Normal;Mean==1;Std Dev==2
```

You can also specify a text string like this as a factor and control the choice of distribution from the Experiment window.

WORKING WITH DATA

Data play a critical role in simulation modeling, as both input and output. This section reviews some data-related SAS Simulation Studio features and modeling approaches that have proven to be useful in practice.

USING INPUT DATA

Every good simulation model is grounded in realistic data, which are often collected from the system being modeled or a similar system. SAS Simulation Studio provides a number of means for bringing data into your model. The Numeric Source block reads values of a specified variable (column) from a SAS® data set or JMP® table to use as direct input to your model, or it can coordinate with JMP to fit a distribution to the data and sample from the distribution in your model. As an alternative, you can use the Observation Source block to read an entire observation (row) from a data set. Both of these blocks read one value or observation at a time from the source data, advancing to the next row each time they are

signaled to read. The Observation Source block can also read an entire data set and create a data model object that holds the contents of the input data set. Often, the resulting data model object is passed from the Observation Source block's OutData port to the InData port of a Dataset Holder block.

The Dataset Holder block stores a data model object whose contents resemble those of a SAS data set or a JMP table, and keeps the data freely accessible throughout the execution of the simulation model. You can configure custom query output ports for the Dataset Holder block to allow either data cell values or observation objects to be pulled from the block. You define these queries by specifying the cell or cells from the data set that you want to access and use in your model. You can use these queried data, for example, to control the routing of entities in a model, to assign values to entity attributes, or for other purposes. Without the Dataset Holder block, the most logical alternative would be to store the needed data as entity attributes. But this would needlessly duplicate the data many times over, as each entity is created and assigned attribute values, and the duplicate reading of external data (compared with the single reading needed to populate the Dataset Holder block) could lengthen the simulation model run time considerably.

A single Dataset Holder block can serve as a central data repository for many other blocks in the simulation model, again avoiding duplication of data and effort. In the Dataset Holder block, you can define multiple queries, each designating the row and/or column of interest, either by specifying explicit row and/or column values in the query or by using the InRow and/or InColumn data input ports. Using the input ports enables you to dynamically change the row or column designation as needed during the simulation model run. For each query that you define, the Dataset Holder block creates a data output port that you can use to communicate the specified data to the appropriate block.

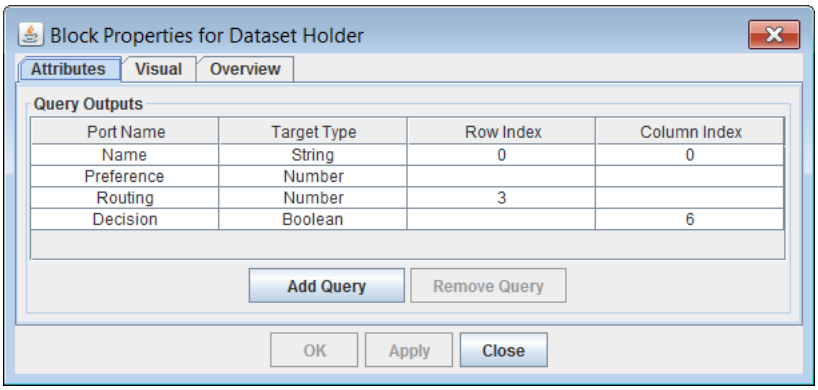


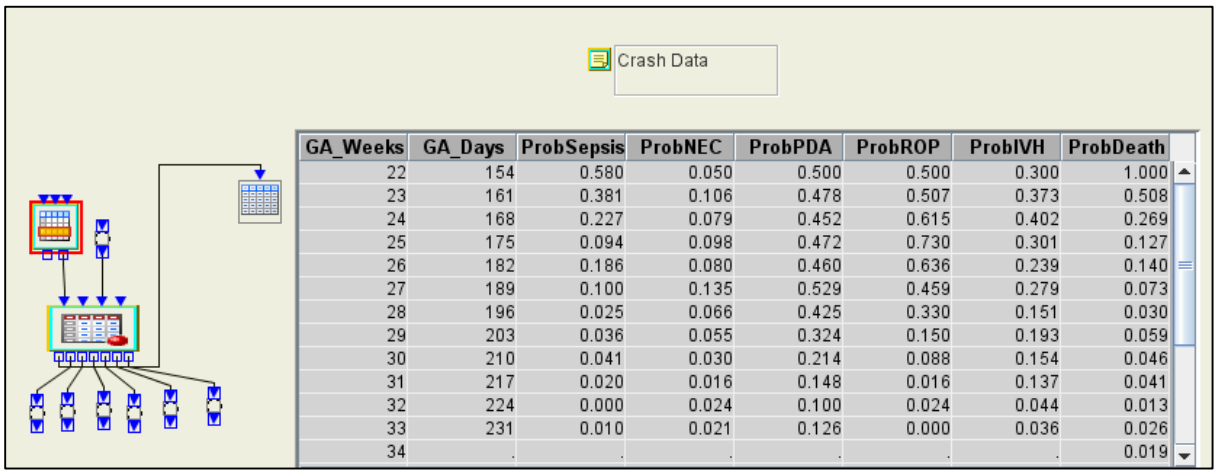
Figure 3. Block Properties Dialog Box for the Dataset Holder Block

Figure 3 shows the block properties dialog box for a sample Dataset Holder block with four queries specified. The first query reads a Name value (a character string) from the cell in the first row and first column (the indexing is zero-based). The second query reads a Preference value (numeric) from a row and column specified by the InRow and InColumn data input ports; this is designated by the missing values in the Row Index and Column Index columns. The third query reads a Routing value (numeric) from the fourth row, with the current column value supplied by the InColumn port. The fourth query reads a Decision value (Boolean) from the seventh column, with the InRow port supplying the current row value.

An example illustrates the practical importance of the Dataset Holder block. SAS worked in partnership with Duke University Medical Center to develop a simulation model of the hospital's neonatal intensive care unit (NICU) to predict the number of nurses needed per shift and to study the effects of various nurse staffing policies (DiRienzo, Tanaka, Lada, and Meanor 2014). Entities represent the infants receiving care in the NICU, and each entity carries multiple attributes that record the infant's gestational age, days since birth, and other important medical information.

In this model, five major morbidities (long-term medical problems) can affect the length of stay of NICU patients. The probabilities of developing each of these conditions according to gestational age (GA) are stored in a SAS data set and read into the model by using an Observation Source block. The Observation Source block passes the resulting data model to a Dataset Holder block, as shown in Figure 4. After each entity (representing an infant) is generated, the entity uses its GA attribute to query a specific row in the

Dataset Holder block. The probabilities from the queried row are used in calculations to determine whether that entity will develop each of the five major morbidities. When you store the morbidity probabilities in a Dataset Holder block, hundreds of entities in each run of the model can access the same information.



GA_Weeks	GA_Days	ProbSepsis	ProbNEC	ProbPDA	ProbROP	ProbIVH	ProbDeath
22	154	0.580	0.050	0.500	0.500	0.300	1.000
23	161	0.381	0.106	0.478	0.507	0.373	0.508
24	168	0.227	0.079	0.452	0.615	0.402	0.269
25	175	0.094	0.098	0.472	0.730	0.301	0.127
26	182	0.186	0.080	0.460	0.636	0.239	0.140
27	189	0.100	0.135	0.529	0.459	0.279	0.073
28	196	0.025	0.066	0.425	0.330	0.151	0.030
29	203	0.036	0.055	0.324	0.150	0.193	0.059
30	210	0.041	0.030	0.214	0.088	0.154	0.046
31	217	0.020	0.016	0.148	0.016	0.137	0.041
32	224	0.000	0.024	0.100	0.024	0.044	0.013
33	231	0.010	0.021	0.126	0.000	0.036	0.026
34	.	.	.	.	.	.	0.019

Figure 4. Dataset Holder Storing Infant Morbidity Probabilities

SAVING OUTPUT DATA

One of the primary reasons for creating a simulation model is to amass simulated data that describe the performance of the modeled system. SAS Simulation Studio provides several options for saving different levels of data created during the run of a simulation model. The Experiment window displays the data for included responses, and you can save these data as a SAS data set or a JMP table by selecting **Save Design** from the Experiment window's pop-up menu.

The data that you save from the Experiment window, however, are intended to measure system behavior for the entire run of a simulation model, taken as a whole. The Experiment window reports data about each replication of each design point, for the responses that you choose to include. If you need to save data in greater detail, you can do so at the block level. In SAS Simulation Studio, several blocks can collect and save data. This section points out two ways to control how blocks in your model save data.

Periodic Data Collection

In SAS Simulation Studio, nine different blocks can accumulate detailed data during the run of a simulation model and save these data at the end of the run. The Number Holder and String Holder blocks are in the "Standard" block template, and the Bucket, Dataset Writer, Probe, Stats Collector, Queue Stats Collector, Server Stats Collector, and Resource Stats Collector blocks are in the "Data and Display" block template. This section focuses on the Bucket and Dataset Writer blocks.

The Bucket block extracts attribute values from entities that pass through it, and it can be very useful in monitoring the status of entities. The Bucket block can extract, collect, save, and output data about any entity attributes that you select. It's easy to see how placing Bucket blocks at key points in your simulation model enables you to record, display, and save attribute values for entities that pass through those points.

A slightly more subtle feature of this block enables you to save entity attribute data not only at chosen points but also for the periods of time that you designate. The InClearData input port on the Bucket block accepts a Boolean signal; a true value clears all data currently collected by the block. Thus, any data that are displayed or saved represent only the simulation time that has elapsed since the most recent true value arrived at the InClearData port. By controlling when your model sends a true Boolean signal to the InClearData port, you can control the periods of time for which you collect data. You might want to discard transient, unrealistic data associated with the startup period of the model, or you might want to segment collected data in order to compile performance statistics that are specific to different time periods.

If you want to collect data for more than one period, then you need to save the data that correspond to the preceding period before you clear them and begin collecting data for the current period. The Dataset Writer block is well suited for this purpose because it (unlike the other data-saving blocks) is able to save data during a simulation run, not just at the end of the run. This block receives data and, when directed to do so, saves the data to a specified or default SAS data set or JMP table. The InSaveNow port on the Dataset Writer block receives a Boolean signal, with a true signal directing the block to save its collected data. You can use the InPolicy port on the Dataset Writer block to specify a location where you want to save data.

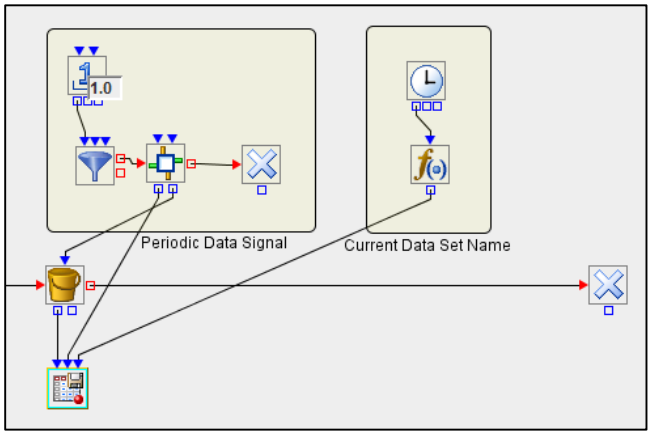


Figure 5. Saving Periodic Data with the Bucket and Dataset Writer Blocks

Figure 5 illustrates one way to use the Bucket and Dataset Writer blocks together to save periodic data. The Gate and Formula blocks also play important roles. Figure 5 depicts the last section of a model's workflow, which ends with entities exiting the model at a Disposer block. A Bucket block collects attribute data from passing entities and provides these data to a Dataset Writer block. Meanwhile, an auxiliary section of the model shown in the compound block "Periodic Data Signal" creates an entity every 1 time unit, sends it through a Gate block, and directs it to a Disposer block. In the Gate block, the passing entity first sends a true Boolean value to the Dataset Writer block's InSaveNow port, causing the currently collected data to be saved. The Gate block next sends a true Boolean value to the InClearData port of the Bucket block, causing it to clear its collected data. This sequence repeats for each integer value of the simulation clock, so that the SAS data sets that are saved represent the time intervals [0,1), [1,2), and so on until the end of the simulation run.

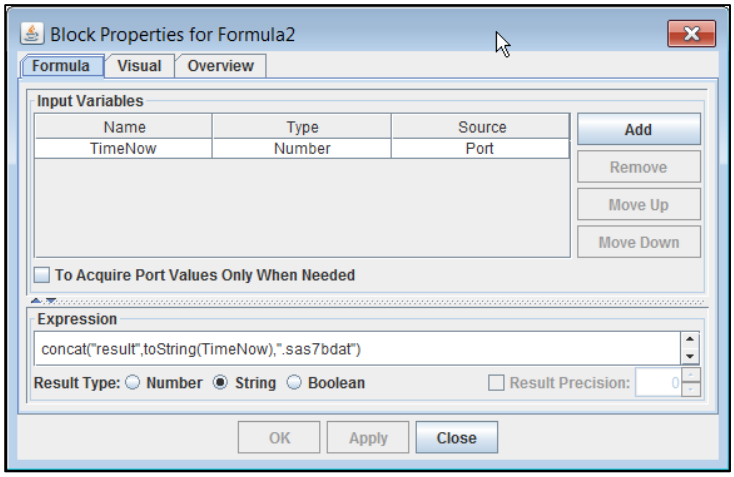


Figure 6. Naming Periodic Data Sets in a Formula Block

The logic in the compound block "Current Data Set Name" is responsible for naming the saved data sets to match the time intervals that they represent. Each data set is named according to the end of the time

period that its data cover, which is also the simulation clock time when the data set is created. The TimeNow block supplies the current simulation clock time, and the Formula block creates and outputs a text string that is sent to the InPolicy port of the Dataset Writer block to be used as the name of the data set. Figure 6 shows the block properties dialog box for this Formula block, including the expression that creates the text string. The name of each data set is “result,” followed by the current simulation time. Thus, the data set result1.sas7bdat covers the time period [0,1), result2.sas7bdat covers the time period [1,2), and so on.

A Note about Auxiliary Models and the Gate Block

This example not only illustrates how you can use the Bucket and Dataset Writer blocks together to save periodic simulated data but also points out one of many instances in which an auxiliary model, with its own separate flow of entities, can be useful. An auxiliary model can be useful whenever you need to perform a task on a periodic basis, such as monitoring the state of a system, running a SAS program, or, as in this case, writing out periodic data. The Gate block is often useful within these auxiliary models because its function is to pull in and push out values and signals when an entity passes through it. You are in full control of the values and signals that it pulls in and pushes out, and you choose their destinations—other blocks in the model. The Gate block adds tremendous flexibility not only to auxiliary models but to SAS Simulation Studio modeling in general.

Model Warm-Up Periods and the Data Trimmer Block

Not all data that you save from a simulation model are necessarily useful. In some cases, the state of your simulation model might not correspond to any realistic situation in the system you are modeling. Often this is true if you start a simulation model of a system in the empty and idle state (with no entities in the system) when in fact you are actually interested in the long-run behavior of the system in steady-state operation. For example, in the Duke NICU model discussed previously, the model is started empty and idle because it is difficult to prepopulate the model with the appropriate number of patients in various states. The model takes some period of simulated time to warm up before it reaches steady-state operation. Any data that are collected before the end of the warm-up period should be excluded from any analyses.

The Data Trimmer block provides one way to easily trim (delete) the data that are generated during what you assess as the startup or warm-up period of your model. Added in SAS Simulation Studio 13.2, the Data Trimmer block provides a central point from which to clear data from as many data collection blocks as you want. Furthermore, the data clearing does not need to be associated with model warm-up; you can clear data for any reason that you deem appropriate. The Data Trimmer block has a single input port labeled InTrimNow. When this port receives a true Boolean value, it signals all the blocks that you select to clear their saved data.

Figure 7 shows, on the left side, an excerpt of a model for a 24-hour incoming call center that uses a Data Trimmer block. The model runs for two days, and the data from the first day are discarded to account for model warm-up. The Value Generator block creates a true Boolean value at the end of the first day and sends it to the Data Trimmer block’s InTrimNow port. The Data Trimmer block then sends a “clear data” signal to every block selected on the **Targets** tab of its block properties dialog box, shown on the right side of Figure 7. The **Targets** tab displays every block in the model that can save data; you simply select the blocks from which you want to clear data. The signals are sent directly, without your having to create links between the Data Trimmer block and the InClearNow ports on the individual blocks. Moreover, because every data-saving block is listed on the **Targets** tab, it’s easy to ensure that no block has been overlooked.

In this model, data are trimmed only once because the Value Generator sends only one signal to the Data Trimmer. The mechanism that creates and sends the true Boolean signal can be as simple or as sophisticated as you want, and you can certainly configure it to send a signal more than once, if necessary. In addition to being eminently scalable, trimming simulated data with the Data Trimmer block is also quite flexible.

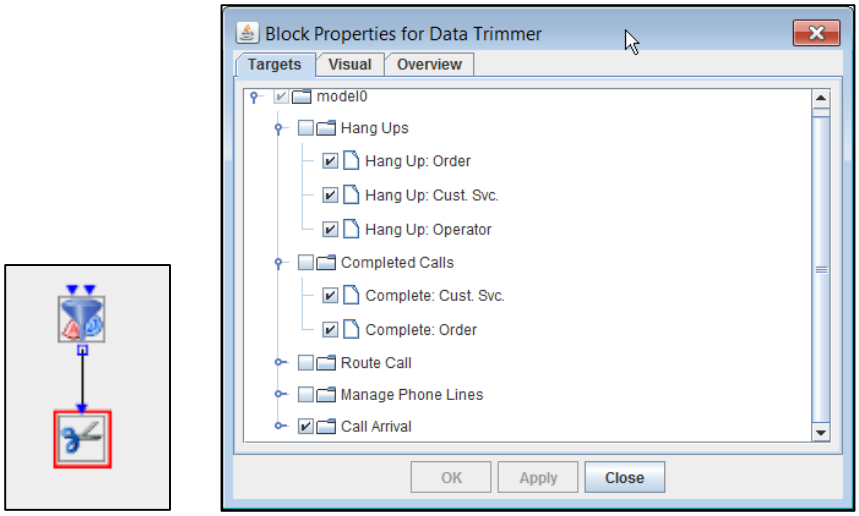


Figure 7. A Data Trimmer Block in a Model and Its Block Properties Dialog Box

ADDING CUSTOM CALCULATIONS TO SIMULATION MODELS

Whether you are working with data that you want to use as input to a simulation model or simulated data that have been created by a model, you might need to perform some calculations on the data before you can use it effectively. The data input ports on SAS Simulation Studio blocks give you a great deal of latitude in determining the sources for numeric, text, observation, and Boolean inputs. Often these calculations involve uses of the Formula block, which enables you to write expressions that synthesize multiple and disparate sources of data and produce the needed input.

One of the most prominent features that support custom calculations is the SAS Program block, which runs a SAS program or JMP script when signaled to do so—during or after the run of the simulation model. The SAS Program block's input port InSubmitCode receives a Boolean value, with a true value signaling the block to execute the code that is specified in its block properties dialog box. Alternatively, you can configure the SAS Program block to run its code automatically after each selected design point of the simulation model is executed.

The SAS program or JMP script that you specify in the SAS Program block can work with any input data; there are no special restrictions. Inputs can include external data and data that are collected and saved by other blocks in your model. The Dataset Writer block is especially useful for saving data during a simulation run to be used as input to a SAS program executed by a SAS Program block.

MANAGING MODEL EXECUTION

Simulation modeling doesn't just involve the model itself but also includes managing how the model and its experiment are executed. This can have implications for the design of both the model and the experiment. This section discusses model execution options that SAS Simulation Studio provides.

(STARTING AND) STOPPING A SIMULATION MODEL

Part of designing a simulation model and its accompanying experiment(s) is determining when the execution of each design point should start and stop. There are few issues surrounding the starting point, aside from the previously discussed frequent need to discard transient simulated data that are collected while the simulated system is being populated with entities. The issues surrounding the point at which the execution of a simulation model stops are more numerous and more nuanced. Accordingly, there are several methods that you can employ to halt the run of a simulation model.

The most basic approach is to set a fixed end time in the "EndTime" column of an Experiment window. You can define a different value for each design point or use the Experiment Properties dialog box to set a value of EndTime for all design points that you create. However, this approach isn't a realistic choice for some models. For example, suppose you are simulating the daily operation of a restaurant, a retail store,

or a commercial medical clinic, none of which are open 24 hours a day. At closing time, the front door is locked, but every customer or patient in the facility is permitted to complete service; no new customers or patients are admitted after this time. Simply stopping model execution at the EndTime of a design point is equivalent, for example, to ushering all customers—including those currently receiving service—out the door the moment closing time arrives.

You can make your model much more realistic by using a different method to stop the simulation run. First, you set the EndTime value in the Experiment window to infinity. Second, you set the End Time parameter of the Entity Generator block to the closing time so that no new entities are generated after closing time. This means that the model continues to run until all entities in the model at closing time complete service and exit the model. This corresponds directly to the business operations of the store, restaurant, or medical clinic and thus helps create more realistic simulated performance data.

In some cases, you might want to define systemwide criteria for stopping the execution of the model. Here, the Stopper block in SAS Simulation Studio is indispensable. The Stopper block, in the “Advanced” block template, operates very simply. If it receives a true Boolean signal via its InSignal input data port, it terminates execution of the current replication. The logic that evaluates the model status and produces the signal can be anything you design, as is true of any other SAS Simulation Studio block receiving a signal via an input port. You have complete flexibility in determining the stopping criteria, which can correlate to any specified set of conditions in the system that you are modeling. For example, in a set of clinical plan models of molecule development built in partnership with a prominent US pharmaceutical company, the Stopper block immediately halts all trials if toxic effects are detected in any of the model’s clinical trials.

Two output ports on the Stopper block provide additional functionality. Just before the Stopper block halts the current replication of the model, its OutWarnTime port outputs a numeric value that contains the current simulation clock time. You can convert this value to a Boolean value to trigger the model to collect data, make calculations, and perform other tasks that need to be done just before the replication stops. Next, the OutStopTime port outputs the simulation stop time. The order of execution for these two ports ensures that you can collect full data about the current replication and take any additional needed actions just before the replication stops.

PARALLEL EXECUTION OF DESIGN POINTS

For large models and especially for large experiments, full runs of an experiment can be time-consuming. To shorten run times for models, SAS Simulation Studio provides an optional parallel execution mode in which replications for the design points you choose to run are executed in parallel, using multiple computational cores on a single machine. To use parallel mode, you simply select the parallel mode icon (to the left of the animation icon) from the toolbar at the top of the SAS Simulation Studio interface, as shown in Figure 8.

Figure 8. Toolbar Before and After Selecting Parallel Mode

Notice that when you select the parallel mode icon, the animation icon disappears; thus animation is not available in parallel mode. Other features that are also associated with the currently running replication are likewise disabled in parallel mode. These features include the simulation clock, replication counter, interactive graphics displays in the model, and Trace messages. Parallel mode is not intended for model debugging (which is the main purpose of these features) but for runs of a debugged model. The **Start**, **Augment**, **Pause**, and **Reset** buttons on the toolbar and their corresponding options on the **Run** menu continue to function normally in parallel mode.

You specify the maximum number of cores to use in parallel mode in the SAS Simulation Studio Configuration dialog box, as shown in Figure 9. The minimum number of cores that you can specify is

two, the maximum is machine-dependent, and the default value is lower than the number of cores on your machine (to avoid interfering with other processes that might be running on your machine).

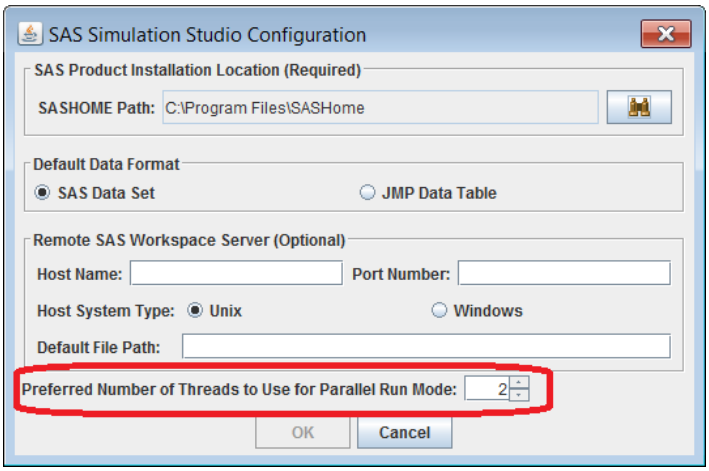


Figure 9. Configuration Dialog Box

CONCLUSION

SAS Simulation Studio's graphical interface provides a versatile and flexible environment for discrete-event simulation modeling and analysis. The simplicity of the interface shortens the learning curve, enabling you to start working with simulation models in fairly short order. As your expertise grows, you're able to create models that grow accordingly in sophistication, detail, and scale. As you build more complex models, you can collect more and better simulated data.

This paper highlights some of the methods that have proven to be productive in using SAS Simulation Studio to model and study systems in a range of practical settings. Some of these methods simply make use of documented features that warrant broader and more extensive use. Other methods build on the basic features of SAS Simulation Studio to provide you with approaches and constructs that you can use effectively in larger and more complex models. All the methods discussed here, applied appropriately, should make you more productive in your simulation modeling and analysis work, and should help you expand and improve your skills.

REFERENCE

DeRienzo, C., Tanaka, D., Lada, E., and Meanor, P. 2014. "Creating a SimNICU: Using SAS Simulation Studio to Model Staffing Needs in Clinical Environments." *Proceedings of the SAS Global Forum 2014 Conference.* Cary, NC: SAS Institute Inc.

RECOMMENDED READING

- *SAS Simulation Studio 13.2 User's Guide*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Ed Hughes
SAS Institute Inc.
Ed.Hughes@sas.com

Emily Lada
SAS Institute Inc.
Emily.Lada@sas.com