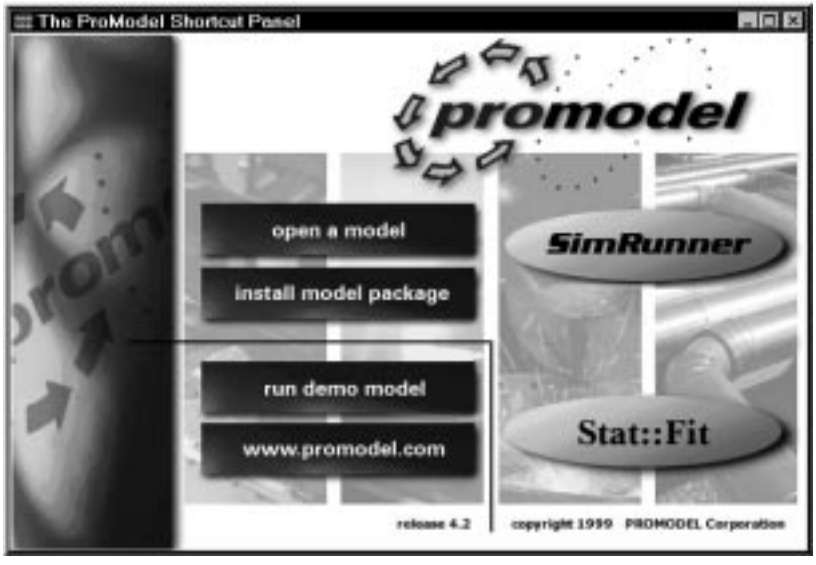
Chapter 3 Getting Started

CHAPTER CONTENTS

3.0.1 Getting Started

Using ProModel requires a basic understanding of simulation methodology. PROMODEL recommends that you read through this chapter before beginning to construct your models to get a good overview of the various modeling elements. From the ProModel shortcut panel seen here, you may run a demonstration model to see how ProModel uses these various elements.



Open a model Opens an existing model.

Install model package Loads an existing model package.

Run demo model Allows you to run a demonstration model.

www.promodel.com Immediately connects you with the ProModel support page on the PROMODEL web site.

SimRunner Activates SimRunner.

Stat::Fit Allows you to launch Stat::Fit.

3.1 Steps for Doing Simulation

3.1.1 Introduction

Doing simulation requires more than just knowing how to use a simulation product. A simulation study is, by its very nature, a project. Like any project, there are tasks to be completed and resources required to complete them. To be successful, a simulation project should be planned with an understanding of the requirements of each of the tasks involved. Many failures result from hastily jumping into a simulation without first taking time to consider the steps involved and developing a plan for proceeding.

Simulation modeling requires good analytical, statistical, communication, organizational, and engineering skills. The modeler must understand the system being investigated and be able to sort through complex cause-and-effect relationships that determine system performance. At least a basic foundation in statistics is needed to properly design experiments and correctly analyze and interpret input and output data. Ongoing communication with owners, stakeholders, and customers during a simulation study is also vital to ensure that a purposeful model is built and that everyone understands the objectives, assumptions, and results of the study.

3.1.2 General Procedure

A decision to do a simulation usually results from a perception that simulation can help resolve one or more issues associated with the design of a new system or the modification of an existing system. Before launching into a simulation project, one or more individuals should have been assigned to the study who have at least a basic knowledge of the system to be studied and the issues of concern. Enough background information should have been obtained about the nature of the problem to determine whether simulation is a suitable solution. If the simulation is being conducted by individuals inside the company, there may already be a basic knowledge of the operation. For outsiders or those unfamiliar with the operation, a brief description of the system and explanation of key issues should be provided. For an existing system, a facility walk-through is an excellent way of getting familiar with the operation.

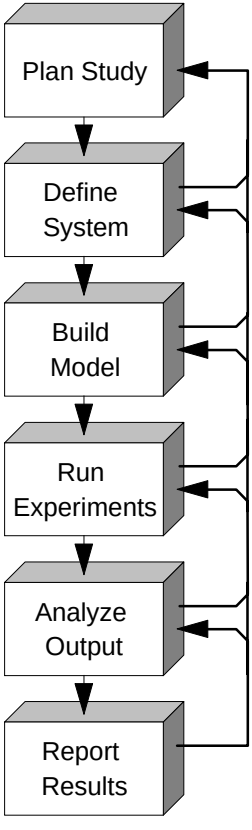
Once a suitable application or project has been identified as a candidate for simulation, decisions must be made about how to conduct the study. There are no strict rules on how to perform a simulation study, however, the following steps are generally recommended as a guideline (Shannon, 1975; Gordon, 1978; Law, 1991):

1. Plan the study
2. Define the system
3. Build the model
4. Run experiments

5. Analyze the output
6. Report results

Each step need not be completed in its entirety before moving on to the next step. The procedure for doing a simulation is an iterative one in which activities are refined and sometimes redefined with each iteration. Describing this iterative process, Pritsker and Pegden (1979) observe:

The iterative nature of this process is shown below:



Procedure for Conducting a Simulation Study

While the requirements for each step vary from simulation to simulation, the basic procedure is essentially the same. The primary value of adopting this systematic procedure, or one like it, is to ensure that the project is conducted in an organized, timely fashion with minimal waste of time and resources and maximum effectiveness in achieving the objectives.

3.1.3 Step 1: Planning the Study

Many simulation projects are doomed to failure from the outset due to poor planning. Undefined objectives, unrealistic expectations and a general lack of understanding of requirements frequently result in frustration and disappointment. If a simulation project is to be successful, a plan must be developed which is realistic, clearly communicated and closely followed. Planning a simulation study involves the following sub tasks:

- **Defining Objectives**
- **Identifying Constraints**
- **Preparing a Simulation Specification**
- **Developing a Budget and Schedule**

Each of these tasks is discussed below.

Defining Objectives

With a basic understanding of the system operation and an awareness of the issues of concern or interest, one or more objectives can be defined for the study. Simulation should only be used if an objective can be clearly stated and it is determined that simulation is the most suitable tool for achieving the objective. Defining an objective does not necessarily mean that there needs to be a problem to solve. A perfectly valid objective may be to see if there are, in fact, any unforeseen problems. Common types of objectives for a simulation study include the following:

- **Performance Analysis** How well does the system perform under a given set of circumstances in all measures of significance (utilization, throughput, waiting times, etc.)?
- **Capacity Analysis** What is the maximum processing or production capacity of the system?
- **Capability Analysis** Is the system capable of meeting specific performance requirements (throughput, waiting times, etc.) and, if not, what changes (added resources, improved methods etc.) are recommended for making it capable?
- **Comparison Study** How well does one system configuration or design variation perform compared to another?
- **Sensitivity Analysis** Which decision variables are the most influential on one or more performance measures, and how influential are they?
- **Optimization Study** What combination of feasible values for a given set of decision variables best achieves desired performance objectives?

- **Decision/Response Analysis** What are the relationships between the values of one or more decision variables and the system response to those changes?
- **Constraint Analysis** Where are the constraints or bottlenecks in the system and what are workable solutions for either reducing or eliminating those constraints?
- **Communication Effectiveness** What variables and graphic representations can be used to most effectively depict the dynamic behavior or operation of the system?

Defining the objective should take into account what the ultimate intended use of the model will be. Some models are built as "throw-away" models to be used only once and then discarded. Other models are built to be used on an ongoing basis for continued "what-if" analyses. Some models need only provide a quantitative answer. Others require realistic animation to convince a skeptical customer. Some models are intended for use by only the analyst. Other models are intended for use by managers with little simulation background and must be easy to use. Some models are used to make decisions of minor consequence. Other models are relied upon to make major financial decisions.

Realizing the objectives of a simulation should consider both the purpose as well as the intended use of the model, the following questions should be asked when defining the objectives of the study:

- Why is the simulation being performed?
- Who will be using the model?
- To whom will the results of the simulation be presented?
- What information is expected from the model?
- Is this a "throw-away" model?
- How important is the decision being made?

Identifying Constraints

Equally as important as defining objectives is identifying the constraints under which the study must be conducted. It does little good if simulation solves a problem if the time to do the simulation extends beyond the deadline for applying the solution, or if the cost to find the solution exceeds the benefit derived. Objectives need to be tempered by the constraints under which the project must be performed such as the budget, deadlines, resource availability, etc. It is not uncommon to begin a simulation project with aspirations of developing an impressively detailed model or of creating a stunningly realistic animation only to scramble at the last minute, throwing together a crude model that barely meets the deadline.

Constraints should not always be viewed as an impediment. If no deadlines or other constraints are established, there is a danger of getting too involved and detailed in the simulation study and run the risk of "paralysis from analysis." The scope of any project has a tendency to shrink or expand to fill the time allotted.

In identifying constraints, anything that could have a limiting effect on achieving the desired objectives should be considered. Specific questions to ask when identifying constraints for a simulation study include the following:

- What is the budget for doing the study?
- What is the deadline for making the decision?
- What are the skills of those doing the study?
- How accessible is the input data?
- What computer(s) will be used for the study?

Preparing a Simulation Specification

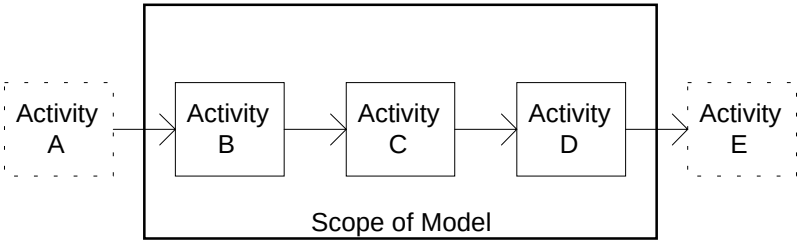
With clearly defined objectives and constraints, the simulation requirements can be specified. Defining a specification for the simulation is essential to projecting the time and cost needed to complete the study. It also guides the study and helps set expectations by clarifying to others exactly what the simulation will include or exclude. A specification is especially important if the simulation is being performed by an outside consultant so that you will know exactly what you are getting for your money. Aspects of the simulation project to be contained in the specification include the following:

- Scope
- Level of detail
- Degree of accuracy
- Type of experimentation
- Form of results

Each of these specification criteria will be discussed in the following pages.

Scope The scope refers to the breadth of the model or how much of the system the model will encompass. Determining the scope of the model should be based on how much bearing or impact a particular activity has on achieving the objectives of the simulation. A common tendency is to model the entire system, even when the problem area and all relevant variables are actually isolated within a smaller subsystem. If, for example, the objective is to find the number of operators required to meet a required production level for a machining cell, it is probably not necessary to model what happens to parts after leaving the cell.

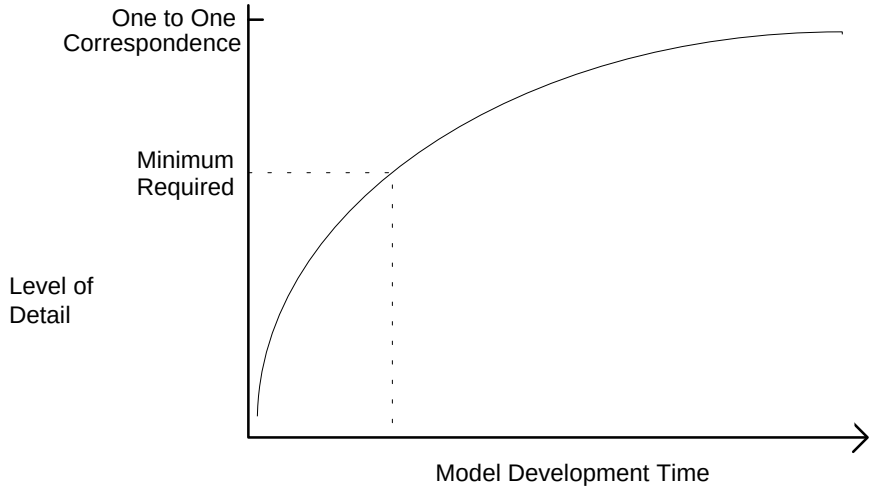
The following figure illustrates how the scope of the model should be confined to only those activities whose interactions have a direct bearing on the problem being studied. Upstream and downstream activities that do not impact the performance measure of interest should be omitted from the model. In the following figure, since the output rate from activity A is predictable, it can be modeled as simply an arrival rate to activity B. Since activity E never constrains output from activity D, it can also be ignored.



Confining the Scope to Impacting Activities

Level of Detail Project the level of detail defines the depth or resolution of the model. At one extreme, an entire factory can be modeled as a single "black box" operation with a random activity time. At the other extreme, every detailed motion of a machine could be modeled with a one-to-one correspondence depicting the entire machine operation.

Unlike the model scope which affects only the size of the model, the level of detail affects model complexity as well as model size. Determining the appropriate level of detail is an important decision. Too much detail makes it difficult and time consuming to develop a valid model. Too little detail may make the model too unrealistic by excluding critical variables. The figure below illustrates how the time to develop a model is affected by the level of detail. It also highlights the importance of including only enough detail to meet the objectives of the study.



Effect of Level of Detail on Model Development Time.

The level of detail is determined largely by the degree of precision required in the output. If only a rough estimate is being sought, it may be sufficient to model each activity by its time, rather than specific details making up the activity. If, on the other hand, details such as downtimes or move times have an appreciable effect on the outcome of the model, they should be included.

Degree of Accuracy The degree of accuracy pertains to the correctness of the data being used. For some models or activities, the information need not be as accurate or exact as it does for others. The required degree of accuracy is determined by the objectives of the study. If the decision is important or a comparison is close, greater accuracy may be required. Accuracy sometimes has to be sacrificed if reliable information is simply unavailable such as when modeling a completely new system.

The required degree of accuracy can have a significant impact on the time and effort required to gather data. It often has little impact, however, on the model building time since a model can be built with estimated values that can later be replaced with more accurate values. Output precision is often governed by the degree of accuracy of the model.

Type of Experimentation The number and nature of the alternative solutions to be evaluated should be planned from the outset in order to ensure that adequate time is allotted. This decision is often influenced by the deadline constraints of the study. Where alternatives with only slight differences are to be evaluated, a base model can be developed that requires only minor modifications to model each alternative. If alternative configurations are significantly different, it may require nearly as much effort modeling each configuration as it does developing the initial model.

For studies in which improvements to an existing system are being considered, it is often helpful and effective to model the current system as well as the proposed system. The basic premise is that you are not ready to make improvements to a system until you understand how the current system operates. Information on the current system is easier to obtain than information on areas of change. Once a model of the current system is built, it is often easier to visualize what changes need to be made for the modified system. Both systems may even be modeled together in the same simulation and made to run side by side. During the final presentation of the results, being able to show both "as is" and "to be" versions of the system effectively demonstrates the impact changes can have on system performance.

Form of Results The form in which the results are to be presented can significantly affect the time and effort involved in the simulation study. If detailed animation or an extensive report is expected, the project can easily stretch on for several weeks after the experimental phase has been completed. Many times the only result required is a simple verification of whether a system is capable of meeting a production requirement. In such cases, a simple answer will suffice.

Developing a Budget and Schedule

With objectives and constraints clearly defined and a specification prepared identifying the work to be performed, a budget and schedule should be developed projecting the expected cost and time for completing the simulation project. Obviously, the time to perform a simulation project will vary depending on the size and difficulty of the project. If data is not readily available, it may be necessary to add several additional weeks to the project. A small project can take two to four weeks while large projects can take two to four months. A simulation schedule should be based on realistic projections of the time requirements keeping in mind the following:

- Defining the system to be modeled can take up to 50% of the project time.
- Model building usually takes the least amount of time (10 to 20%).
- Once a base model is built, it can take several weeks to conduct all of the desired experiments, especially if alternative designs are being compared.

While it may have initially been determined that simulation is a suitable tool for achieving the objective, the decision to use simulation may need to be reconsidered in light of projected cost and time estimates. Simulation may be a good solution to the problem at hand, but if the time or the cost of doing the project outweighs the anticipated benefits, either an alternative solution may need to be explored or the objectives may need to be modified to cut down on the level of effort required.

3.1.4 Step 2: Defining the System

With clearly defined objectives and a well organized plan for the study, the system that will be simulated can begin to be defined in detail. This can be viewed as the development of a conceptual model on which the simulation model will be based. The process of gathering and validating system information can be overwhelming when faced with the stacks of uncorrelated data to sort through. Data is seldom available in a form that defines exactly how the system works. Many data gathering efforts end up with lots of data but very little useful information.

Data gathering should never be performed without a purpose. Rather than being haphazard, data gathering should be goal oriented with a focus on information that will achieve the objectives of the study. There are several guidelines to keep in mind when gathering data.

- **Identify cause-and-effect relationships** It is important to correctly identify the causes or conditions under which activities are performed. In gathering downtime data, for example, it is helpful to distinguish between downtimes due to failure, planned downtimes for breaks, tool change, etc., and downtimes that are actually idle periods due to unavailability of stock. Once the causes have been established and analyzed, activities can be properly categorized.
- **Look for key impact factors** Discrimination should be used when gathering data to avoid wasting time examining factors that have little or no impact on system performance. If, for example, an operator is dedicated to a particular machine and, therefore, is never a cause of production delay, there is no need to include the operator in the model. Likewise, extremely rare downtimes, negligible move times, on-the-fly inspections and other insignificant or irrelevant activities that have no appreciable effect on routine system performance may be safely ignored.
- **Distinguish between time and condition dependent activities** Time-dependent activities are those that take a predictable amount of time to complete, such as an inspection time. Condition-dependent activities can only be completed when certain defined conditions within the system are satisfied. Because condition-dependent activities are uncontrollable, they are unpredictable. An example of a condition-dependent activity might be filling a customer order or performing an assembly operation that requires component parts to become available.

  Many activities are partially time-dependent and partially condition-dependent. When gathering data on these activities, it is important to distinguish between the time actually required to perform the activity and the time spent waiting for resources to become available or other conditions to be met before the activity can be performed. If, for example, historical data is used to determine repair

times, the time spent doing the actual repair work should be used without including the time spent waiting for a repair person to become available.

- **Focus on essence rather than substance** A system definition for modeling purposes should capture the key cause-and-effect relationships and ignore incidental details. Using this "black box" approach to system definition, we are not concerned about the nature of the activity being performed, but only the impact that the activity has on the use of resources and the delay of entity flow. For example, the actual operation performed on a machine is not important, but only how long the operation takes and what resources, if any, are tied up during the operation. It is important for the modeler to be constantly thinking abstractly about the system operation in order to avoid getting too caught up in the incidental details.
- **Separate input variables from response variables** Input variables in a model define how the system works (e.g., activity times, routing sequences, etc.). Response variables describe how the system responds to a given set of input variables (e.g., work-in-process, idle times, resource utilization, etc.). Input variables should be the focus of data gathering since they are used to define the model. Response variables, on the other hand, are the output of a simulation. Consequently, response variables should only be gathered to help validate the model once it is built and run.

These guidelines should help ensure that the system is thought of in the proper light for simulation purposes.

To help organize the process of gathering data for defining the system, the following steps are recommended:

- Determine data requirements.
- Use appropriate data sources.
- Make assumptions where necessary.
- Convert data into a useful form.
- Document and approve the data.

Each of these steps is explained on the following pages.

Determining Data Requirements

The first step in gathering system data is to determine what data is required for building the model. This should be dictated primarily by the scope and level of detail required to achieve the model objectives as described earlier. It is best to go from general to specific in gathering system data. The initial focus should be on defining the overall process flow to provide a skeletal framework for attaching more detailed information. Detailed information can then be added gradually as it becomes available (e.g., resource requirements, processing times, etc.). Starting with the overall process flow not only provides an orderly approach to data gathering, but also enables

the model building process to get started which reduces the amount of time to build and debug the model later. Often, missing data becomes more apparent as the model is being built.

In defining the basic flow of entities through the system, a flow diagram can be useful as a way of documenting and visualizing the physical flow of entities from location to location. Once a flow diagram is made, a structured walk-through can be conducted with those familiar with the operation to ensure that the flow is correct and that nothing has been overlooked. The next step might be to define the detail of how entities move between locations and what resources are used for performing operations at each location. At this point it is appropriate to identify location capacities, move times, processing times, etc.

To direct data gathering efforts and ensure that meetings with others, on whom you depend for model information, are productive, it may be useful to prepare a specific list of questions that identify the data needed. A list of pertinent questions to be answered might include the following:

1. What types of entities are processed in the system and what attributes, if any, distinguish the way in which entities of the same type are processed or routed?
2. What are the route locations in the system (include all places where processing or queuing occurs, or where routing decisions are made) and what are their capacities (i.e., how many entities can each location accommodate or hold at one time)?
3. Besides route locations, what other types of resources (personnel, vehicles, etc.) are used in the system and how many units are there of each type (resources used interchangeably may be considered the same type)?
4. What is the routing sequence for each entity type in the system?
5. What activity, if any, takes place for each entity at each route location (define in terms of time required, resources used, number of entities involved and any other decision logic that takes place)?
6. Where, when and in what quantities do entities enter the system (define the schedule, interarrival time, cyclic arrival pattern, or condition which initiates each arrival)?
7. In what order do multiple entities depart from each location (First in, First out; Last in, First out; etc.)?
8. In situations where an output entity could be routed to one of several alternative locations, how is the routing decision made (e.g., most available capacity, first available location, probabilistic selection, etc.)?
9. How do entities move from one location to the next (define in terms of time and resources required)?

10. What triggers the movement of entities from one location to another (i.e., available capacity at the next location, a request from the downstream location, an external condition)?
11. How do resources move from location to location to perform tasks (define either in terms of speed and distance, or time)?
12. What do resources do when they finish performing a task and there are no other tasks waiting (e.g., stay put, move somewhere else, etc.)?
13. In situations where multiple entities could be waiting for the same location or resource when it becomes available, what method is used for making an entity selection (e.g., longest waiting entity, closest entity, highest priority, preemption, etc.)?
14. What is the schedule of availability for resources and locations (define in terms of shift and break schedules)?
15. What planned interruptions do resources and locations have (scheduled maintenance, setup, changeover, etc.)?
16. What random failures do resources and locations experience (define in terms of distributions describing time to failure and time to repair)?

Depending on the purpose of the simulation and level of detail needed, some of these questions may not be applicable. For very detailed models additional questions may need to be asked. Answers to these questions should provide nearly all of the information necessary to build a model.

Using Appropriate Data Sources

Having a specific set of questions for defining the system, we are now ready to search for the answers. Information seldom comes from a single source. It is usually the result of reviewing reports, conducting personal interviews, personal observation and making lots of assumptions. “It has been my experience,” notes Carson (1986), “that for large-scale real systems, there is seldom any one individual who understands how the system works in sufficient detail to build an accurate simulation model. The modeler must be willing to be a bit of a detective to ferret out the necessary knowledge.” Good sources of system data includes the following:

- Process Plans
- Time Studies
- Predetermined Time Standards
- Flow Charts
- Facility Layouts
- Market Forecasts
- Maintenance Reports
- Production Logs

- Equipment Manufacturers
- Managers
- Engineers
- Shop Floor Personnel
- Facility Walk-throughs
- Comparisons with Similar Operations

In deciding whether to use a particular source of data, it is important to consider the relevancy, reliability and accessibility of the source. If the information that a particular source can provide is irrelevant for the model being defined, then that source should not be consulted. What good is a maintenance report if it has already been decided that downtimes are not going to be included in the model? Reliability of the source will determine the validity of the model. A managers perception, for example, may not be as reliable as actual production logs. Finally, if the source is difficult to access, such as a visit to a similar facility in a remote site, it may have to be omitted.

Making Assumptions

Not long after data gathering has started, you may realize certain information is unavailable or perhaps unreliable. Complete, accurate, and up-to-date data for all the information needed is rarely obtainable, especially when modeling a new system about which very little is known. For system elements about which little is known, assumptions must be made. There is nothing wrong with assumptions as long as they can be agreed upon, and it is recognized that they are *only* assumptions. Any design effort must utilize assumptions where complete or accurate information is lacking.

Many assumptions are only temporary until correct information can be obtained or it is determined that more accurate information is necessary. Often *sensitivity analysis*, in which a range of values is tested for potential impact, can give an indication of just how accurate the data really needs to be. A decision can then be made to firm up the assumptions or to leave them alone. If, for example, the degree of variation in a particular activity time has little or no impact on system performance, then a constant activity time may be used. Otherwise, it may be important to define the exact distribution for the activity time.

Another approach in dealing with assumptions is to run three different scenarios showing a “best-case” using the most optimistic value, a “worst-case” using the most pessimistic value, and a “most-likely-case” using a best estimate value. This will help determine the amount of risk you want to take in assuming a particular value.

Converting Data to a Useful Form

Data is seldom in a form ready for use in a simulation model. Usually, some analysis and conversion needs to be performed for data to be useful as an input parameter to the simulation. Random phenomena must be fitted to some standard, theoretical distribution, such as a normal or exponential distribution (see Law and Kelton, 1991), or be input as a frequency distribution. Activities may need to be grouped together to simplify the description of the system operation.

Distribution Fitting To define a distribution using a theoretical distribution requires that the data, if available, be fit to an appropriate distribution that best describes the variable. Several distribution fitting packages (e.g., BestFit, UniFit II, StatFit) are available to assist in fitting sample data to a suitable theoretical distribution. The alternative to using a standard theoretical distribution is to summarize the data in the form of a frequency distribution that can be used directly in the model. A frequency distribution is sometimes referred to as an *empirical* or *user-defined* distribution.

Whether fitting data to a theoretical distribution, or using an empirical distribution, it is often useful to organize the data into a frequency distribution table. Defining a frequency distribution is done by grouping the data into intervals and stating the frequency of occurrence for each particular interval. To illustrate how this is done, the following frequency table tabulates the number and frequency of repairs for a particular machine that required a certain frequency of repairs for a particular machine requiring a certain range of time to perform.

Frequency Distributions of Repair Times

Repair Time (minutes)	Number of Observations	Percentage	Cumulative Percentage
0 - 1	25	16.5	16.5
1 - 2	33	21.7	38.2
2 - 3	30	19.7	57.9
3 - 4	22	14.5	72.4
4 - 5	14	9.2	81.6
5 - 6	10	6.6	88.2
6 - 7	7	4.6	92.8
7 - 8	5	3.3	96.1
8 - 9	4	2.6	98.7
9 - 10	2	1.3	100.0

Total Number of Observations = 152

While there are rules that have been proposed for determining the interval or cell size, the best approach is to make sure that enough cells are defined to show a gradual transition in values, yet not so many cells that groupings become obscured.

Note in the last column of the frequency table that the percentage for each interval may be expressed optionally as a cumulative percentage. This helps verify that all 100% of the possibilities are included.

When gathering samples from a static population, one can apply descriptive statistics and draw reasonable inferences about the population. When gathering data from a dynamic and possibly time varying system, however, one must be sensitive to trends, patterns and cycles that may occur with time. The samples drawn may not actually be homogenous samples and, therefore, unsuitable for applying simple descriptive techniques.

Activity Grouping Another consideration in converting data to a useful form is the way in which activities are grouped for modeling purposes. Often it is helpful to group activities together so long as important detail is not sacrificed. This makes models easier to define and more manageable to analyze. In grouping multiple activities into a single activity time for simplification, consideration needs to be given as to whether activities are performed in parallel or in series. If activities are done in parallel or with any overlap, the time during which overlapping occurs should not be additive.

If, for example, a part is loaded onto a machine at the same time a finished part is unloaded from the machine, only the longest of the two times needs to be included. Serial activities are always additive. For example, if a series of activities are performed on an entity at a location, rather than specifying the time for each activity, it may be possible to sum activity times and enter a single time or time distribution.

Documenting and Approving the Data

When it is felt that all relevant information has been gathered and organized into a usable form, it is advisable to document the information in the form of data tables, relational diagrams and assumption lists. Sources of data should also be noted. This document should then be reviewed by others who are in a position to evaluate the validity of the data and approve the assumptions made. This document will be helpful later if you need to make modifications to the model or look at why the actual system ends up working differently than what was modeled.

In addition to including those factors to be used in the model, the data document should also include those factors that are deliberately excluded from the model because they are deemed to be either insignificant or irrelevant. If, for example, break times are not identified in the system description, a statement of explanation should be made explaining why. Stating why certain factors are being excluded from the system description will help resolve later concerns that may question why the factors were omitted.

Validating system data can be a time-consuming and difficult task, especially when so many assumptions are made. In practice, data validation ends up being more of a consensus or agreement confirming the information is good enough for the purposes of the model. While this approved data document provides the basis for building the model, it often changes as model building and experimentation get under way.

3.1.5 Step 3: Building the Model

Once sufficient information has been compiled to define the basic system operation, the model building activity can begin. While starting to build a model too early can be a wasted exercise, waiting until all of the information is completely gathered and validated may unnecessarily postpone the building of the model. Getting the model started before the data is completely gathered may even help identify missing information needed to proceed.

The goal of model building is to provide a valid representation of the defined system operation. Additionally, the model must be able to provide any other statistical or graphical representation needed to satisfy the objectives of the study. A model is neither true nor false, but rather useful or not useful. Once validated, a model is useful when it provides the needed information to meet the objectives of the simulation.

Progressive Refinement

One nice feature of simulation is that models do not have to include all of the final detail before they will run. This allows a progressive refinement strategy to be used in which detail is added to the model in stages rather than all at once. Not only do models get built and running quicker this way, but it also makes models easier to debug. In the initial stages of a model, for example, attractive graphics are not very useful and, since they are likely to be changed anyway, should not be added until later when preparing for the final model presentation.

The complexity of model building should never be underestimated and it is always better to begin simple and add complexity rather than create an entire complex model at once. It is also easier to add detail to a model than it is to remove it from a model. Building a model in stages enables bugs to be more readily identified and corrected. Emphasizing the importance of applying progressive refinement to model building, Law and Kelton (1991) have advised:

> Although there are few firm rules on how one should go about the modeling process, one point on which most authors agree is that it is always a good idea to start with a simple model which can later be made more sophisticated if necessary. A model should contain only enough detail to capture the essence of the system for the purposes for which the model is intended: it is not necessary to have a one-to-one correspondence between elements of the model and elements of the system. A model with excessive detail may be too expensive to program and to execute.

Incremental Expansion

In addition to adding complexity to a model in stages, models that have a broad scope are sometimes easier to build in phases where additional sections are added incrementally to the model. This method of "eating the elephant one bite at a time" allows a portion of the model to be built, tested and debugged before adding new sections and makes a large task more manageable.

For unusually large models, it may be useful to identify definable boundaries within a model to permit *model partitioning*. Model partitioning is the process of subdividing a model into two or more modules that represent physically separate sections within the system. The purpose of model partitioning is to allow model sections to be built and debugged, possibly even by separate individuals, independently of each other. Once sections are finished, they can be merged together to create the overall model. This "divide-and-conquer" method of model building can greatly reduce the time and difficulty in building and debugging large models.

Model Verification

Once a model is defined using a selected software tool, the model must generally be debugged to ensure that it works correctly. The process of demonstrating that a model works as intended is referred to in simulation literature as model *verification.* It is much easier to debug a model built in stages and with minimal detail than to debug a large and complex model. Eliminating bugs in a program model can take a considerable amount of time, especially if a general purpose programming language (e.g., C++) in which frequent coding errors occur is used. Most simulation languages provide a trace capability in the form of audit trails, screen messages, graphic animation, or some combination of all three. A trace enables the user to look inside of the simulation to see if the simulation is performing the way it should. Good simulation products provide interactive debugging capability which further facilitates the debugging process. A thorough "walk-through" of the model input is always advisable.

Model Validation

During the process of model building, the modeler must be constantly concerned with how closely the model reflects the system definition. The process of determining the degree to which the model corresponds to the real system, or at least accurately represents the model specification document, is referred to as model *validation*. Proving absolute validity is a non attainable goal. As Neelamkavil (1987) explains, "True validation is a philosophical impossibility and all we can do is either invalidate or fail to invalidate." For this reason, what we actually seek to establish is a high degree of *face validity.* Face validity means that, from all outward indications, the model appears to be an accurate representation of the system. From this standpoint, validating a model is the process of substantiating that the model, within its domain of applicability, is sufficiently accurate for the intended application (Schlesinger, 1979).

There is no simple test to establish the validity of a model. Validation is an inductive process through which the modeler draws conclusions about the accuracy of the model based on the evidence available. Gathering evidence to determine model validity is largely accomplished by examining the model structure (i.e., the algorithms and relationships) to see how closely it corresponds to the actual system definition. For models having complex control logic, graphic animation can be used effectively as a validation tool. Finally, the output results should be analyzed to see if the results appear reasonable. If circumstances permit, the model may even be compared to that actual system to see how they correspond. If these procedures are performed without encountering a discrepancy between the real system and the model, the model is said to have face validity.

3.1.6 Step 4: Conducting Experiments

The fourth step in a simulation study is to conduct simulation experiments with the model. Simulation is basically an application of the scientific method. In simulation, one begins with a theory of why certain design rules or management strategies are better than others. Based on these theories the designer develops a hypothesis which he tests through simulation. Based on the results of the simulation the designer draws conclusions about the validity of his hypothesis. In a simulation experiment there are input variables defining the model which are independent and may be manipulated or varied. The effects of this manipulation on other dependent or response variables are measured and correlated.

In some simulation experiments we are interested in the *steady-state* behavior of the model. Steady-state behavior does not mean that the simulation produces a steady outcome, but rather the distribution or statistical variation in outcome does not change over time. For example, a mass production facility may fluctuate between 200 and 220 parts per hour under normal operating conditions. For many simulations we may only be interested in a particular time period, such as a day run in a job shop. For these studies, the simulation may never reach a steady state.

As with any experiment involving a system with random characteristics, the results of the simulation will also be random in nature. The results of a single simulation run represent only one of several possible outcomes. This requires that multiple *replications* be run to test the reproducibility of the results. Otherwise, a decision might be made based on a fluke outcome, or at least an outcome that is not representative of what would normally be expected. Since simulation utilizes a pseudo-random number generator for generating random numbers, running the simulation multiple times simply reproduces the same sample. In order to get an independent sample, the starting seed value for each random stream must be different for each replication, thus ensuring that the random numbers generated from replication to replication are independent.

Depending on the degree of precision required in the output, it may be desirable to determine a *confidence interval* for the output. A confidence interval is a range within

which we can have a certain level of confidence that the true mean falls. For a given confidence level or probability, say .90 or 90%, a confidence interval for the average throughput rate of a system might be determined to be between 45.5 and 50.8 units per hour. We would then be able to say that there is a .90 probability that the true mean throughput of the model (not of the actual system!) lies between 45.5 and 50.8 units per hour.

Fortunately, ProModel provides convenient facilities for conducting experiments, running multiple replications and automatically calculating confidence intervals. The modeler must still decide, however, what types of experimentation are appropriate. When conducting simulation experiments, the following questions should be asked:

- Am I interested in the steady-state behavior of the system or a specific period of operation?
- How can I eliminate start-up bias or getting the right initial condition for the model?
- What is the best method for obtaining sample observations that may be used to estimate the true expected behavior of the model?
- What is an appropriate run length for the simulation?
- How many replications should be made?
- How many different random streams should be used?
- How should initial seed values be controlled from replication to replication?

Answers to these questions will be determined largely by the following three factors:

- The nature of the simulation (terminating or nonterminating).
- The objective of the simulation (capacity analysis, alternative comparisons, etc.).
- The precision required (rough estimate versus confidence interval estimates).

Terminating Versus Non-terminating Simulations

As part of setting up the simulation experiment, one must decide what type of simulation to run. Simulations are usually distinguished as being one of two types: *terminating* or *non-terminating*. The difference between the two has to do with whether we are interested in the behavior of the system over a particular period of time or in the steady-state behavior of the system. It has nothing to do, necessarily, with whether the system itself terminates or is ongoing. The decision to perform a terminating or non-terminating simulation has less to do with the nature of the system than it does with the behavior of interest.

A terminating simulation is one in which the simulation starts at a defined state or time and ends when it reaches some other defined state or time. An initial state might be the number of parts in the system at the beginning of a work day. A terminating state or event might be when a particular number of jobs have been completed. Consider, for example, an aerospace manufacturer that receives an order to manufacture 200 airplanes of a particular model. The company might be interested in knowing how long it will take to produce the aircraft along with existing workloads. The simulation run starts with the system empty and is terminated when the 200th plane is completed since that covers the period of interest. A point in time which would bring a terminating simulation to an end might be the closing of shop at the end of a business day, or the completion of a weekly or monthly production period. It may be known, for example, that a production schedule for a particular item changes weekly. At the end of each 40 hour cycle, the system is "emptied" and a new production cycle begins. In this situation, a terminating simulation would be run in which the simulation run length would be 40 hours.

Terminating simulations are not intended to measure the steady-state behavior of a system. In a terminating simulation, average measures are of little meaning. Since a terminating simulation always contains transient periods that are part of the analysis, utilization figures have the most meaning if reported for successive time intervals during the simulation.

A non-terminating or steady-state simulation is one in which the steady-state behavior of the system is being analyzed. A non-terminating simulation does not mean that the simulation never ends, nor does it mean that the system being simulated has no eventual termination. It means only that the simulation could theoretically go on indefinitely with no statistical change in behavior. For non-terminating simulations, the modeler must determine a suitable length of time to run the model. An example of a non-terminating simulation is a model of a manufacturing operation in which oil filters are produced on a continual basis at the same pace. The operation runs two shifts with an hour break during each shift in which everything momentarily stops. Break and third shift times are excluded from the model since work always continues exactly as it left off before the break or end of shift. The length of the simulation is determined by how long it takes to get a representative steady-state reading of the model behavior.

Running Terminating Simulations

Experiments involving terminating simulations are usually conducted by making several simulation runs, or replications, of the period of interest using a different random seed for each run. This procedure enables statistically independent and unbiased observations to be made on the system response over the period simulated. Statistics are often gathered on performance measures for successive intervals of time during the period.

For terminating simulations, we are usually interested in final production counts and changing patterns of behavior over time rather than the overall average behavior. It would be absurd, for example, to conclude that because two machinists are only busy an average of 40% during the day that only one machinist is needed. This average measure reveals nothing about the utilization of the machinists during peak periods of the day. A more detailed report of waiting times during the entire work day may reveal that three machinists are needed to handle peak periods, whereas only one machinist is necessary during off-peak hours. In this regard, Hoover and Perry (1990) note:

> It is often suggested in the simulation literature that an overall performance be accumulated over the course of each replication of the simulation, ignoring the behavior of the systems at intermediate points in the simulation. We believe this is too simple an approach to collecting statistics when simulating a terminating system. It reminds us of the statistician who had his head in the refrigerator and feet in the oven, commenting that on the average he was quite comfortable.

For terminating simulations, the three important questions to answer in running the experiment are:

1. What should be the initial state of the model?
2. What is the terminating event or time?
3. How many replications to make?

Many systems operate on a daily cycle, or, if a pattern occurs over a weeks time, the cycle is weekly. Some cycles may vary monthly or even annually. Cycles need not be repeating to be considered a cycle. Some manufacturers, for example, may be interested in the ramp-up period of production during the introduction of a new product which is a one-time occurrence.

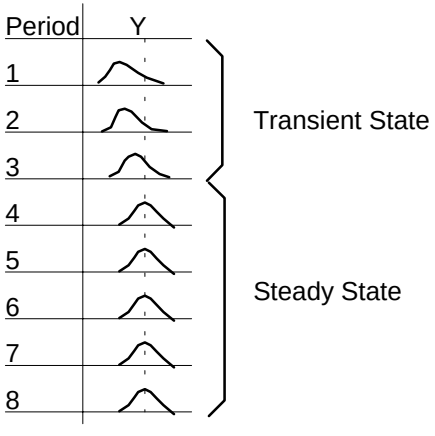
The number of replications should be determined by the precision required for the output. If only a rough estimate of performance is being sought, three to five replications are sufficient. For greater precision, more replications should be made until a confidence interval with which you feel comfortable is achieved.

Running Non-terminating Simulations

The issues associated with generating meaningful output statistics for terminating simulations are somewhat different that those associated with generating statistics for non-terminating systems. In steady-state simulations, we must deal with the following issues:

1. Determining the initial warm-up period.
2. Selecting among several alternative ways for obtaining sample observations.
3. Determining run length.

Determining the Warm-up Period In a steady-state simulation, we are interested in the steady-state behavior of the model. Since a model starts out empty, it usually takes some time before it reaches steady-state. In a steady-state condition, the response variables in the system (e.g., processing rates, utilization, etc.) exhibit statistical regularity (i.e., the distribution of these variables are approximately the same from one time period to the next). The figure below illustrates the typical behavior of a response variable, Y, as the simulation progresses through N periods of a simulation.

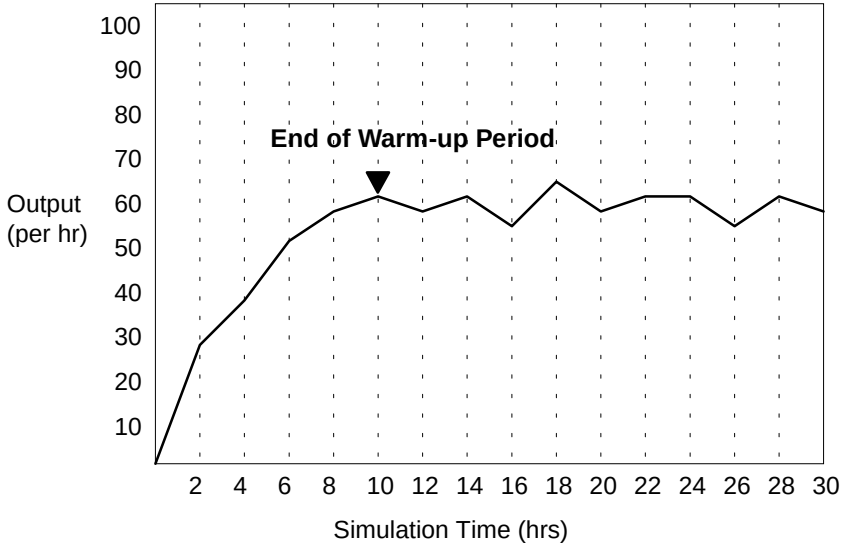


Behavior of Response Variable Y For Successive Periods During Simulation.

The time that it takes to reach steady-state is a function of the activity times and the amount of activity taking place. For some models, steady-state might be reached in a matter of a few hours of simulation time. For other models it may take several hundred hours to reach steady-state. In modeling steady-state behavior we have the problem of determining when a model reaches steady-state. This start-up period is usually referred to as the *warm-up period*. We want to wait until after the warm-up period before we start gathering any statistics. This way we eliminate any bias due to observations taken during the transient state of the model.

While several methods have been presented for determining warm-up time (Law and Kelton, 1991), the easiest and most straightforward approach, although not necessarily the most reliable, is to run a preliminary simulation of the system, preferably with several (3 to 5) replications, and observe at what time the system reaches statistical stability. The length of each replication should be relatively long and allow even rarely occurring events, such as infrequent downtimes, to occur at least two or three times. To determine a satisfactory warm-up period using this method, one or more key response variables should be monitored by period over time, like the average number of entities in a queue or the average utilization of a resource. This approach assumes that the *mean* value of the monitored response variable is the primary indicator of convergence rather than the *variance*, which often appears to be the case. If possible, it is preferable to reset the response variable after each period rather than track the

cumulative value of the variable, since cumulative plots tend to average out instability in data. Once these variables begin to exhibit steady-state, we can add a 20% to 30% safety factor and be reasonably safe in using that period as the warm-up period. This approach is simple, conservative and usually satisfactory. Remember, the danger is in underestimating the warm-up period, not overestimating it. Relatively little time and expense is needed to run the warm-up period longer than actually required. The figure below illustrates averaging the number of entities processed each hour for several replications. Since statistical stability is reached at about 10 hours, 12 to 15 hours is probably a safe warm-up period to use for the simulation.



Plot of Hourly Entity Output to Identify Start of Steady-State.

Obtaining Sample Observations In a terminating simulation, sample observations are simply made by running multiple replications. For steady-state simulations, we have several options for obtaining sample observations. Two widely used approaches are running multiple replications and interval batching. The method supported in ProModel is running multiple replications.

Running multiple replications for non-terminating simulations is very similar to running terminating simulations. The only difference is that (1) the initial warm-up period must be determined, and (2) an appropriate run length must be determined. Once the replications are made, confidence intervals can be computed as described earlier in this chapter. One advantage of running independent replications is that samples are independent. On the negative side, running through the warm-up phase for each replication slightly extends the length of time to perform the replications. Furthermore, there is a possibility that the length of the warm-up period is underestimated, causing biased results.

Interval batching (also referred to as the *batch means technique*) is a method in which a single, long run is made with statistics being reset at specified time intervals. This allows statistics to be gathered for each time interval with a mean calculated for each interval batch. Since each interval is correlated to both the previous and the next intervals (called serial correlation or autocorrelation), the batches are not completely independent. The way to gain greater independence is to use large batch sizes and to use the mean values for each batch. When using interval batching, confidence interval calculations can be performed. The number of batch intervals to create should be at least 5 to 10 and possibly more depending on the desired confidence interval.

Determining Run Length Determining run length for terminating simulations is quite simple since there is a natural event or time point that defines it for us. Determining the run length for a steady-state simulation is more difficult since the simulation can be run indefinitely. The benefit of this, however, is that we can produce good representative samples. Obviously, running extremely long simulations is impractical, so the issue is to determine an appropriate run length that ensures a sufficiently representative sample of the steady-state response of the system is taken.

The recommended length of the simulation run for a steady-state simulation is dependent upon (1) the interval between the least frequently occurring event and (2) the type of sampling method (replication or interval batching) used. If running independent replications, it is usually a good idea to run the simulation enough times to let every type of event (including rare ones) happen at least a few times if not several hundred. Remember, the longer the model is run, the more confident you can become that the results represent a steady-state behavior. If collecting batch mean observations, it is recommended that run times be as large as possible to include at least 1000 occurrences of each type of event (Thesen and Travis, 1992).

Comparing Alternative Systems

Simulations are often performed to compare two or more alternative designs. This comparison may be based on one or more decision variables such as buffer capacity, work schedule, resource availability, etc. Comparing alternative designs requires careful analysis to ensure that differences being observed are attributable to actual differences in performance and not to statistical variation. This is where running multiple replications may again be helpful. Suppose, for example, that method A for deploying resources yields a throughput of 100 entities for a given time period while method B results in 110 entities for the same time period. Is it valid to conclude that method B is better than method A, or might additional replications actually lead the opposite conclusion?

Evaluating alternative configurations or operating policies can sometimes be performed by comparing the average result of several replications. Where outcomes are close or where the decision requires greater precision, a method referred to as *hypothesis testing* should be used. In hypothesis testing, first a hypothesis is formulated (e.g., that methods A and B both result in the same throughput) and then a test is made to

see whether the results of the simulation lead us to reject the hypothesis. The outcome of the simulation runs may cause us to reject the hypothesis that methods A and B both result in equal throughput capabilities and conclude that the throughput does indeed depend on which method is used.

Sometimes there may be insufficient evidence to reject the stated hypothesis and thus the analysis proves to be inconclusive. This failure to obtain sufficient evidence to reject the hypothesis may be due to the fact that there really is no difference in performance, or it may be a result of the variance in the observed outcomes being too high given the number of replications to be conclusive. At this point, either additional (perhaps time consuming) replications may be run or one of several variance reduction techniques might be employed (see Law and Kelton, 1991).

Factorial Design

In simulation experiments we are often interested in finding out how different input variable settings impact the response of the system. Rather than run hundreds of experiments for every possible variable setting, experimental design techniques can be used as a "short-cut" to finding those input variables of greatest significance. Using experimental-design terminology, input variables are referred to as *factors*, and the output measures are referred to as *responses*. Once the response of interest has been identified and the factors that are suspected of having an influence on this response defined, we can use a *factorial design* method which prescribes how many runs to make and what level or value to be used for each factor. As in all simulation experiments, it is still desirable to run multiple replications for each factor level and use confidence intervals to assess the statistical significance of the results.

One's natural inclination when experimenting with multiple factors is to test the impact that each individual factor has on system response. This is a simple and straightforward approach, but it gives the experimenter no knowledge of how factors interact with each other. It should be obvious that experimenting with two or more factors together can affect system response differently than experimenting with only one factor at a time and keeping all other factors the same.

One type of experiment that looks at the combined effect of multiple factors on system response is referred to as a *two-level, full-factorial design*. In this type of experiment, we simply define a high and low level setting for each factor and, since it is a full-factorial experiment, we try every combination of factor settings. This means that if there are five factors and we are testing two different levels for each factor, we would test each of the $2^5 = 32$ possible combinations of high and low factor levels. For factors that have no range of values from which a high and low can be chosen, the high and low levels are arbitrarily selected. For example, if one of the factors being investigated is an operating policy for doing work (e.g., first come, first served; or last come, last served), we arbitrarily select one of the alternative policies as the high level setting and a different one as the low level setting.

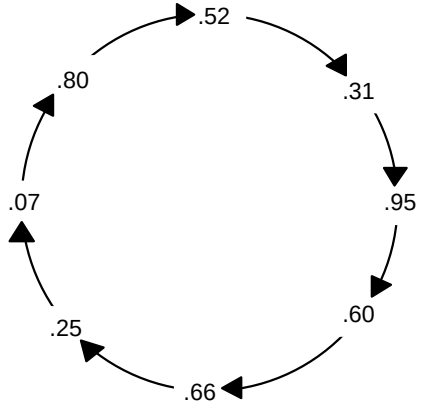
For experiments in which a large number of factors are being considered, a two-level full-factorial design would result in an extremely large number of combinations to test. In this type of situation, a *fractional-factorial design* is used to strategically select a subset of combinations to test in order to "screen out" factors with little or no impact on system performance. With the remaining reduced number of factors, more detailed experimentation such as a full-factorial experiment can be conducted in a more manageable fashion.

After fractional-factorial experiments and even two-level, full-factorial experiments have been performed to identify the most significant factor level combinations, it is often desirable to conduct more detailed experiments, perhaps over the entire range of values, for those factors that have been identified as being the most significant. This provides more precise information for making decisions regarding the best factor values or variable settings for the system. For a more concise explanation of the use of factorial design in simulation experimentation see Law and Kelton (1991).

Use of Random Streams

One of the most valuable characteristics of simulation is the ability to reproduce and randomize replications of a particular model. Simulation allows probabilistic phenomena within a system to be controlled or randomized as desired for conducting controlled experiments. This control is made available through the use of *random streams*.

A stream is a sequence of independently cycling, unique random numbers that are uniformly distributed between 0 and 1 (see the figure on next page). Random number streams are used to generate additional random numbers from other probability distributions (Normal, Beta, Gamma, etc.). After sequencing through all of the random numbers in the cycle, the cycle starts over again with the same sequence. The length of the cycle before it repeats is called the *cycle period* and is usually very long.



Example of a Random Stream Cycle With a Very Short Period.

A random stream is generated using a random number generator or equation. The random number generator begins with an initial seed value after which, each successive value uses the previous value as input to the generator. Each stream used in a simulation has its own independent seed and tracks its own values for subsequent input to the generator. Where the sequence begins in the cycle depends on the initial seed value used by the generator.

Any time a particular number seeds a stream, the same sequence of values will be repeated every time the same seed is used to initialize the stream. This means that various elements within a model can be held constant with respect to their performance while other elements vary freely. Simply specify one random number stream for one set of activities and another random number stream for all other activities.

Because the same seed produces the same sequence of values every time it is used, completely independent functions within a model must have their own streams from the start. For example, arrival distributions should generally have a random number stream used nowhere else in the entire model. That way, activities added to a model that sample from a random number stream will not inadvertently alter the arrival pattern because they do not affect the sample values generated from the arrival distribution.

To show an example of how multiple streams can be useful, consider two machines, Mach1 and Mach2, which go down approximately every 4 hours for servicing. To model this, the frequency or time between failures is defined by a normal distribution with a mean value of 240 minutes and a standard deviation of 15 minutes, N(240,15). The time to repair is 10 minutes. If no stream is specified in the normal distribution, the same stream will be used to generate sample values for both machines. So, if the next two numbers in the stream number are .21837 and .86469, Mach1 will get a sample value from the normal distribution that is different from Mach2. Therefore, the two machines will go down at different times.

Suppose, however, that the resource servicing the machines must service them both at the same time, so we would like to have the machines go down at the same time. Using the same stream to determine both downtimes will not bring them down at the same time, because a different random number will be returned from the stream with each call to generate a random normal variate. Using two different streams, each dedicated to a machine's downtime and each having the same initial seed, will ensure that both machines go down at the same time every time. The two streams have the same starting seed value so they will produce exactly the same sequence of random numbers.

3.1.7 Step 5: Analyzing the Output

Output analysis deals with drawing inferences about the actual system based on the simulation output. When conducting simulation experiments, extreme caution should be used when interpreting the simulation results. Since the results of a simulation experiment are random (given the probabilistic nature of the inputs), an accurate measurement of the statistical significance of the output is necessary.

People doing simulation in academia are often accused of working with contrived and often oversimplified assumptions, yet are extremely careful about ensuring the statistical significance of the model results. Simulation practitioners in industry, on the other hand, are usually careful to obtain valid model data, only to ignore the statistical issues associated with simulation output. Maintaining a proper balance between establishing model validation and establishing the statistical significance of simulation output is an important part of achieving useful results.

The most valuable benefit from simulation is to gain insight, not necessarily to find absolute answers. With this in mind, one should be careful about getting too pedantic about the precision of simulation output. With more than 60 combined years of experience in doing simulation modeling, Conway, Maxwell and Worona (1986) caution that attaching a statistical significance to simulation output can create a delusion that the output results are either more or less significant than they really are. They emphasize the practical, intuitive reading of simulation results. Their guideline is, “If you can't see it with the naked eye, forget it.”

The goal of conducting experiments is not just to find out how well a particular system operates, but hopefully to gain enough insight to be able to improve the system. Unfortunately, simulation output rarely identifies causes of problems, but only reports the symptomatic behavior of problems. Bottleneck activities, for example, are usually identified by looking for locations or queues that are nearly always full which feed into one or more locations that are sometimes empty. Detecting the *source* of the bottleneck is sometimes a bit trickier than identifying the bottleneck itself. Bottlenecks may be caused by excessive operation times, prolonged delays due to the unavailability of resources, or an inordinate amount of downtime. The ability to draw correct inferences from the results is essential to making system improvements.

3.1.8 Step 6: Reporting the Results

The last step in the simulation procedure is to make recommendations for improvement in the actual system based on the results of the simulated model. These recommendations should be supported and clearly presented so that an informed decision can be made. Documentation of the data used, the model(s) developed and the experiments performed should all be included as part of a final simulation report.

A simulation has failed if it produces evidence to support a particular change which is not implemented; especially if it is economically justified. The process of selling sim-

ulation results is largely a process of establishing the credibility of the model. It is not enough for the model to be valid, the client or management must also be *convinced* of its validity if it is to be used as an aid in decision making. Finally, the results must be presented in terms that are easy to understand and evaluate. Reducing the results to economic factors always produces a compelling case for making changes to a system.

In presenting results it is important to be sensitive to the way in which recommendations are made. It helps to find out whether recommendations are being sought or whether a simple summary of the results is wanted. It is generally wise to present alternative solutions and their implications for system performance without suggesting one alternative over another, particularly when personnel changes or cuts are involved. In fact, where there may be careers on the line, it is best to caution the decision maker that your simulation study looks only at the logistical aspects of the system and that it does not take into account the potential reactions or potential difficulties employees may have in accepting a particular solution.

Animation and output charts have become an extremely useful aid in communicating the results of a simulation study. This usually requires that some touch-up work be done to create the right effect in visualizing the model being simulated. In preparing the results, it is often necessary to add a few touch-ups to the model (like a full dress-rehearsal) so the presentation effectively and convincingly presents the results of the simulation study.

After the presentation is finished and there is no further analysis to be conducted (the final presentation always seems to elicit further suggestions for trying this or that with the model), the model recommendations, if approved, are ready to be implemented. If the simulation has been adequately documented, it should provide a good functional specification for the implementation team.

3.1.9 Pitfalls in Simulation

If the steps that have been outlined are followed, the chances of performing a successful simulation project are very good. Typical reasons why simulation projects fail include the following:

- Failure to state clear objectives at the outset.
- Failure to involve individuals affected by outcome.
- Overrunning budget and time constraints.
- Failure to document and get a consensus on input data.
- Including more detail than is needed.
- Including variables that have little or no impact on system behavior.
- Failure to verify and validate the model.
- Basing decisions on a single run observation.
- Basing decisions on average statistics when the output is actually cyclical.
- Being too technical and detailed in presenting the results to management.

3.1.10 Summary

A simulation project has distinct phases that must be understood and followed in order to be successful. Simulation requires careful planning with realistic goals and expectations. Steps to performing a simulation study include planning the study, defining the system, building the model, conducting experiments, analyzing the output, and presenting the results. Systematically following these steps will help avoid the pitfalls that frequently occur when conducting a simulation study.

3.2 Modeling Environment

When you open a model or select New from the File menu, your screen appears as shown below with a menu bar across the top of the screen and a layout window. For now, let's look briefly at the menus accessible from the menu bar.

3.2.1 Menu Bar

All of the tools necessary to build and run a model and view the corresponding output are accessed through the menu bar. The menu bar is located just beneath the ProModel caption bar and contains the selections listed on the following page. These selections access other menus with selections related to the menu heading.

File The File menu allows you to open new models, save current models, merge two or more models into one, and load models created with earlier versions of ProModel. It also allows you to view a text version of the model and print either the model text file or the graphic layout of the model.

Edit The Edit menu contains selections for editing the contents of edit tables and logic windows. The selections available from this menu will change according to the module from which the Edit menu is selected. They also vary according to the currently selected window.

View The View menu lets you control the ProModel's appearance. From this menu you can control layout settings, hide or view hidden paths, operate the zoom controls, and more.

Build The Build menu contains all of the modules for creating and editing a model. This includes the basic modules such as Locations, Entities, Arrivals and Processing, and the optional modeling elements such as Variables, Attributes, Arrays and Subroutines.

Simulation The Simulation menu controls the execution of a simulation and contains options for running a model, defining model parameters, and defining and running scenarios.

Output The Output selection starts the ProModel Output Program for viewing model output. It also allows you to view the trace which was generated during runtime.

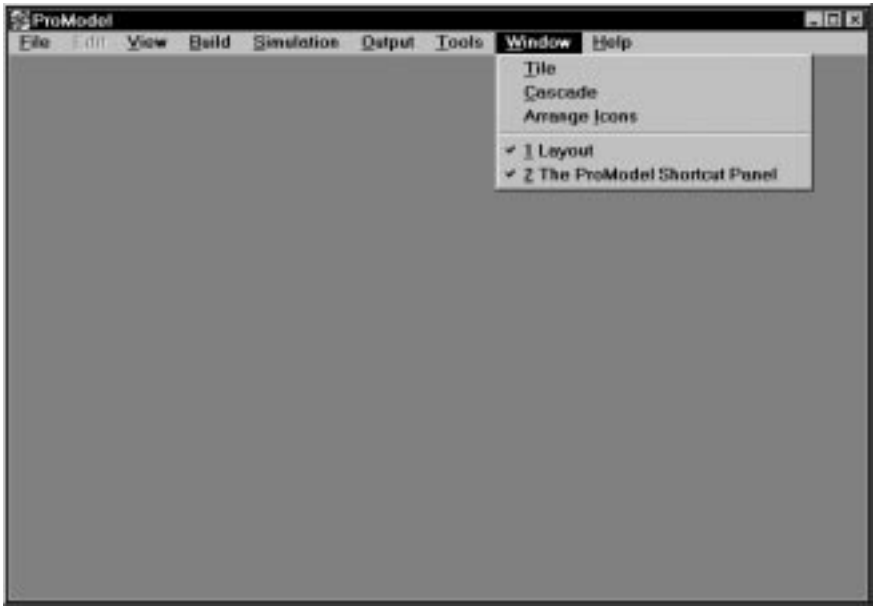
Tools The Tools menu contains various utilities including AutoBuild, the Graphics Editor for creating and modifying graphic icons, and a search and replace feature for finding or replacing expressions throughout a model.

Window The Window menu allows you to arrange the windows (or iconized windows) that are currently displayed on the screen such that all windows are visible at once. It also allows you to bring any individual window to the forefront of the display.

Help The Help menu accesses the ProModel On-line Help System.

3.2.2 Window Menu

The Window menu allows you to rearrange windows and icons and select the active window. These functions are standard to all Windows™ applications.



Tile Causes all open windows to fit in the available screen space. Windows that may be hidden behind other windows become visible.

Cascade Causes all open windows to overlap such that the title bar of each window is visible.

Arrange Icons Causes all icons representing iconized applications to be arranged neatly along the bottom of the screen.

Note Below **Arrange Icons** is a list of the open windows. The window with the check next to it is the active window.

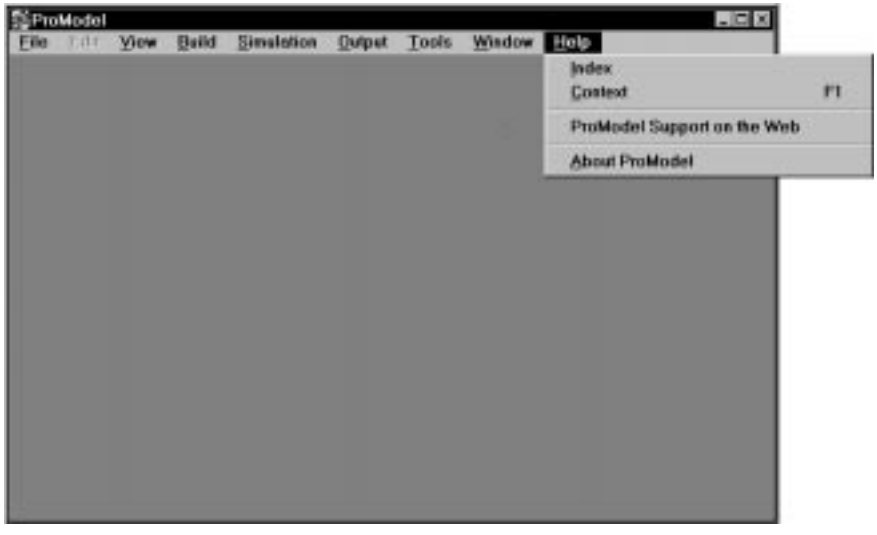
How To

Reset the windows to their default positions:

- Choose **Reset Window Positions** from the **View** menu.

3.2.3 Help Menu

The ProModel Help menu is a convenient, quick way to look up information about a task you are performing, a feature you would like to know more about, or a command you want to use. ProModel Help is available whenever you see a Help command button, or Help as an item on a menu bar.



The selections from the Help Menu are as follows:

Index Brings up the Main Help Index.

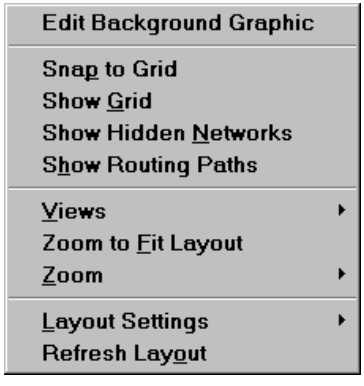
Context Opens the help system to the topic related to the currently active window.

ProModel Support on the Web When you select this option from the help menu, ProModel automatically connects you with the ProModel customer service page on the PROMODEL web site.

About ProModel Displays a message containing information about the product.

3.2.4 Right-Click Menu

To simplify many of the steps required to perform common modeling operations, ProModel now includes a variety of right-click menus. From these menus, you can access context-sensitive options and settings for variables, locations, processing, path networks, resources, and other components. The following menu appears when you right click on the layout.



The right-click layout menu includes the following options:

Edit Background Graphic Only when you have a background graphic defined, this option appears and lets you edit the graphic.

Snap to Grid Selecting this option snaps all graphics to the grid.

Show Grid When you select this option, ProModel displays the background grid.

Show Hidden Networks Displays all *hidden* networks.

Show Routing Paths Displays all routing paths used in the model.

Views From this option, you may select from the views defined in the model.

Zoom to Fit Layout This option resizes the model to fit the entire image in the layout window.

Zoom Allows you to select the zoom percentage.

Layout Settings From here, you may select and define the grid settings, background color, and routing color.

Refresh Layout This option refreshes the image to reflect recent changes.

Note For information about the right-click menus for model elements (e.g., locations, variables, routings, resources, path networks, queues, conveyors, etc.), see *Defining General Elements* on page 197.

3.3 Building a Model

ProModel gives you the flexibility to create a model in several ways. The easiest method is to use the graphical point and click approach to first define locations in the system. Once locations have been defined, entities (parts) are defined and scheduled to arrive at locations in the system. Then you define any optional model elements such as attributes, variables or arrays that will be used in the processing. Finally, the processing of entities at each location is specified in the processing logic.

3.3.1 Using the AutoBuild Feature

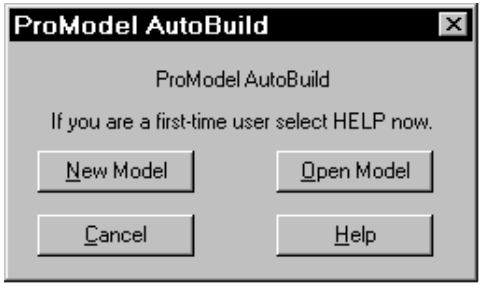
If you are new to simulation or are unsure of which elements to include in a model, ProModel offers a structured environment called the AutoBuild feature to guide you through the required and optional modeling elements. Since we will use the AutoBuild feature to build our demonstration model, let's examine how it works.

How To

Use AutoBuild:

- Choose **Autobuild** from the **Tools** menu.

The start of your AutoBuild session begins with the following dialog box.

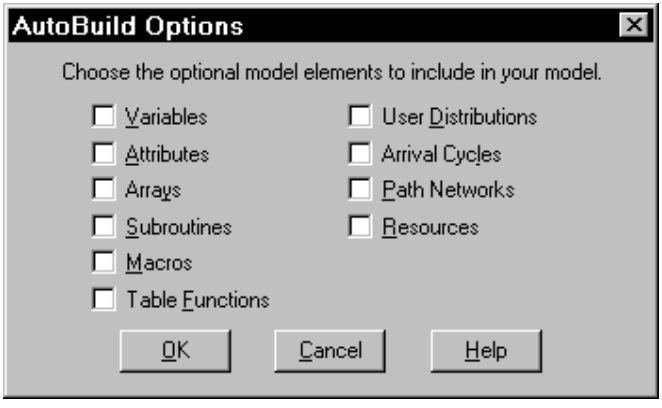


You would select **New Model** to begin building a model from scratch or **Open Model** if you wish to modify an existing model.

Certain elements must be included in every model. The required elements are:

- Locations
- Entities
- Processing
- Arrivals

Once you are in AutoBuild mode, you are prompted to enter information in the General Information dialog. ProModel then prompts you to select any optional elements to include in the model with the following dialog:



Modules are completed by filling out information in an edit table specific to each module. When you finish defining the information in a module, click on the upper left corner of the edit table to close the module. You will then be prompted to go to the next module.

AutoBuild is flexible enough to allow you to jump back and forth between build modules. Once the AutoBuild session has begun, you may enter any other module to make an addition or change and then return to AutoBuild where you left off. To do this, simply select another module and make the desired change or addition. When completed, exit the module by closing the module window. The AutoBuild feature will then take over and return you to the next uncompleted module. For example, if the Entities module was not complete, AutoBuild would return you to that module, otherwise AutoBuild would take you to the next uncompleted module.

3.3.2 Modeling Scenario

Before we actually begin building a model, let's look at a fictitious scenario for our model building session.

Cogswell Cogs has just secured a contract to produce a new cog for production of the X-95C Family Space Cruiser. The current capacity of the Cogswell facility is not adequate to handle any additional work load while continuing to fill existing orders. Therefore, Mr. Cogswell has ordered the I.E. department to simulate the design of a new workcell dedicated to the production of the new cog.

The process consists of loading a cast blank onto an NC mill for milling of the outside splines. Once the splines have been cut, the cog must be degreased, inspected and loaded with an inner bearing. All operations, including inspection, are to be performed by a single operator.

Model Elements

In building this model we must define all of the basic modeling elements and a few of the optional elements.

Locations

We need some type of receiving location to hold incoming entities. We also need processing locations where entities have value added to them. For the given production rate, Cogswell's engineers have determined that the workcell will require two NC_300 series numerically controlled mills, a degreasing machine, and an inspection/assembly station.

Entities (Parts)

The entity types in this system include Pallets, each carrying six cast Blanks. Blanks become Cogs after processing, and Bearings are loaded at the Inspect station. If a Cog fails the inspection it will be called a Reject.

Arrivals

Cogswell's engineers have determined that Pallets should arrive at the rate of 1 Pallet every 45 minutes.

Processing

The operation at each mill requires an operator to load the Blank, which takes a normally distributed amount of time with a mean of 3 minutes and standard deviation of .2 minutes (i.e., N(3,.2)). After a blank has been loaded, the machining time is a constant 5.7 minutes.

Cogs are then removed from the mill and placed in the degreasing machine. The degreasing machine has capacity for 2 Cogs, and has a cycle time of 5 minutes.

Once the Cogs have been degreased, they are inspected for proper spline depth, and a bearing is installed in the center hole. This process requires the cell operator, and takes U(3.2,.3) minutes for the inspection and U(1.5,.2) minutes for the Bearing to be installed. If the Cog fails inspection, no Bearing is installed.

Resources

A single operator, CellOp, performs all manual operations.

Path Networks

In order to make CellOp a mobile resource, we must define a path network. We'll call it CellNet.

Attributes

An attribute is simply a “numeric tag” attached to either an entity (entity attribute) or a location (location attribute). Since each Cog is inspected for proper spline depth, we attach an attribute called Test to each Cog, specifying the Pass/Fail status of the Cog.

User Distributions

We will sample from a user defined distribution and set the Test attribute to either 1 (for pass) or 0 (for fail). Ninety six percent of the Cogs pass inspection and have their Test attribute set to one. Four percent fail the inspection and have their Test attribute set to zero.

3.3.3 Phased Modeling Approach

Instead of trying to build the model all at once, you may want to implement a phased modeling approach where you build the model in stages. This will help you to understand the basic modeling elements before moving on to more complex ones like attributes and if-then logic. The following pages walk you through this phase approach step by step.

Phase 1: Basic Model Elements

In the first phase you input all of the basic model elements: General Information, Locations, Entities, Arrivals, and Processing. You also import a background graphic to help in placing the locations in the layout window. Upon completion of this phase you have a fully working model, ready to animate and collect output.

Phase 2: Adding Resources & Variability

The second phase consists of adding an operator (resource) and corresponding path network to move entities from location to location. This operator is also used to load and unload the machines, and perform the inspection of the splines. Because human operators introduce variability into the system you will change some of the processing times from constants to distributions to reflect this variance.

Phase 3: Additional Operations

The final phase consists of adding the assembly operation at the inspection station. Cogs that pass inspection are loaded with a Bearing in the center hole. In doing this you will use IF...THEN logic to test whether or not the Cog has passed or failed the inspection. If the Cog passes inspection, a JOIN command is used to accomplish the assembly.

The final task in this phase is to add downtimes for machine failure at each of the two milling machines. You will base the time between failures on the total amount of processing time incurred by each machine. When a machine fails, you will require the operator to service the machine.

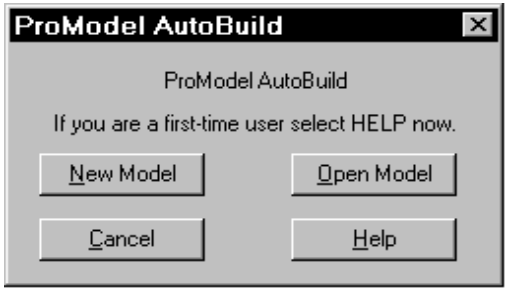
3.3.4 Phase 1: Basic Model Elements

The first step in building the model is to define the model's basic elements. To do this, select New Model from the initial AutoBuild dialog box.

How To

To use AutoBuild:

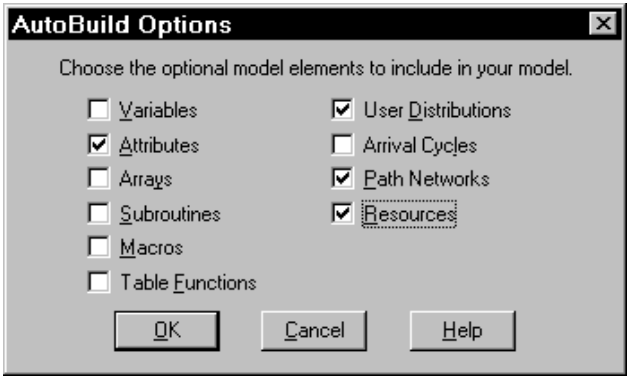
- Choose **AutoBuild** from the **Tools** menu. The AutoBuild selection dialog is displayed.



Selecting **New Model** displays the AutoBuild Options dialog.

AutoBuild Options

From the AutoBuild Options dialog, choose optional model elements to include in your model. For this example, select Attributes, User Distributions, Path Networks, and Resources.

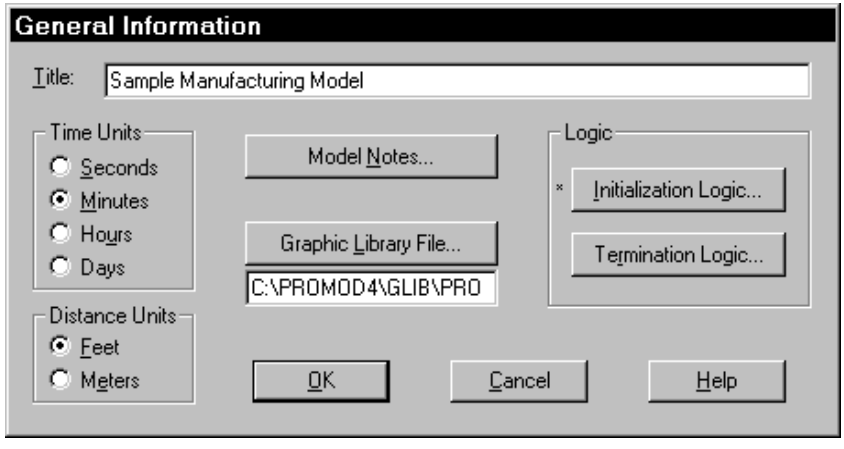


After clicking **OK** from the AutoBuild Options dialog box, AutoBuild takes you back to the General Information dialog and allows you to edit your selections. The Auto-Build feature is flexible enough to allow you to move to another module, perform some action, and then return to where you left off.

For example, suppose you want to import the background graphic of a workcell before defining locations. First, import the graphic into the background, then return to the AutoBuild procedure by closing the Background Graphics module and selecting AutoBuild from the Tools menu.

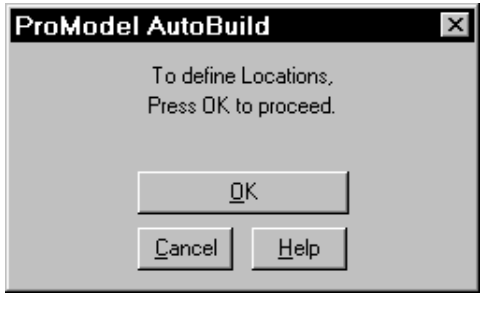
General Information

The General Information dialog box allows you to name your model and specify default information such as time and distance units. You also specify the name of the graphics library to use. In this case it is PROMOD4.GLB.



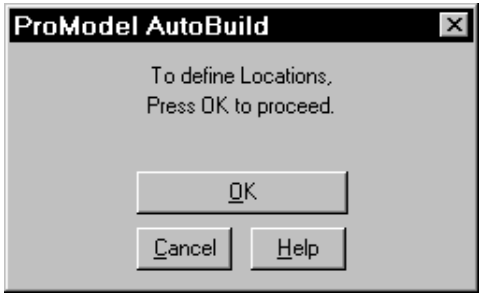
When you click **OK** in the General Information dialog, AutoBuild takes you first through each of the required modules and then through the optional modules you selected.

Note AutoBuild prompts you with dialog boxes like the following one each time you complete and close a module. For simplicity, we will not show the dialog boxes for each module.



How To **To move to another module and return to AutoBuild:**

1. At the ProModel AutoBuild dialog shown below, click **Cancel**.



2. Select the menu item for the other module and perform the action(s).
3. Click **AutoBuild** from the **Tools** menu. The above dialog reappears.

Importing a Background Graphic

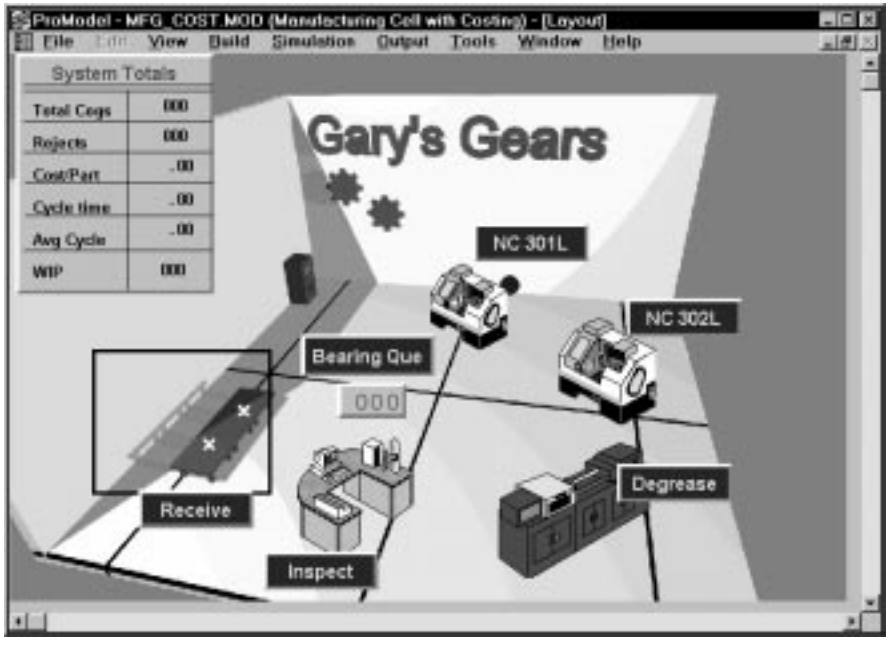
Simple background graphics such as the one below are imported easily through the Background Graphics Editor. ProModel also allows you to import complex graphics files such as AutoCAD™ drawings to use as the background for your simulations. For more information on how to import a background graphic, see *Background Graphics Editor* on page 404.

Often, importing a background graphic makes the process of placing locations easier, or altogether eliminates the need to create graphic icons for locations.

Defining Locations

Locations are defined easily by selecting the desired icon and placing it in the layout window. Each time a location is placed in the layout window, a corresponding record is entered in the Location edit table. This table lists each location along with location parameters such as the capacity, number of units, and downtime information. For more information on how to define a location, see *Locations* on page 201.

Layout Window (*maximized*)



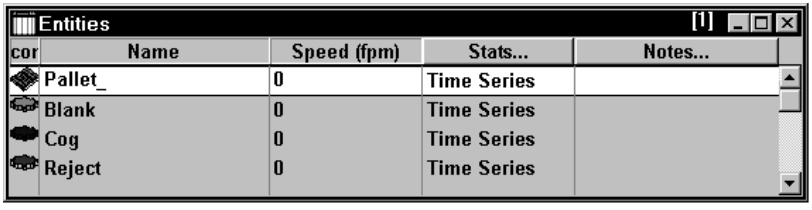
Location Edit Table

Locations [1]

Icon	Name	Cap.	Units	DTs...	Stats...	Rules...	Notes...
	Receive	2	1	None	Time Se	Oldest, FIFO	
	NC_301L	1	1	Usage,	Basic	Oldest, FIFO	
	NC_302L	1	1	Usage,	Basic	Oldest, FIFO	
	Degrease	2	1	None	Basic	Oldest, FIFO	

Defining Entities

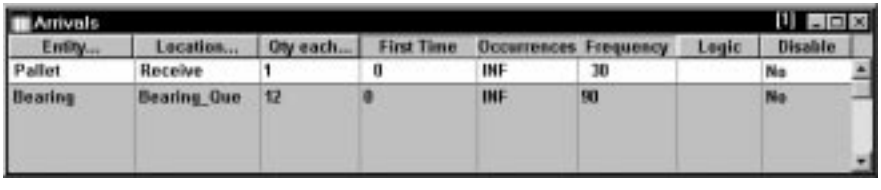
Once all locations have been defined, we define entities in a similar way by selecting an icon for each entity type. As we do this, a record is created in the Entity edit table for each entity type. For more information on how to define entities, see *Entities* on page 237.



In this model the Speed (fpm) column is irrelevant since all entities move according to the definition of the mobile resource CellOp. Also, the Stats column shows that we desire detailed statistics for all entity types. “Time series” statistics include throughput history of the entity. “Basic” statistics include only the total exits of each entity type from the system and the final quantity of each entity type in the system.

Defining Arrivals

Of the four entity types, only one needs to be scheduled to arrive in the system. Every 45 minutes one Pallet arrives at the Receiving location. The word “INFINITE” in the Occurrences column means that one pallet continues to arrive every 30 minutes as long as the simulation runs. For more information on defining arrivals, see *Arrivals* on page 315.



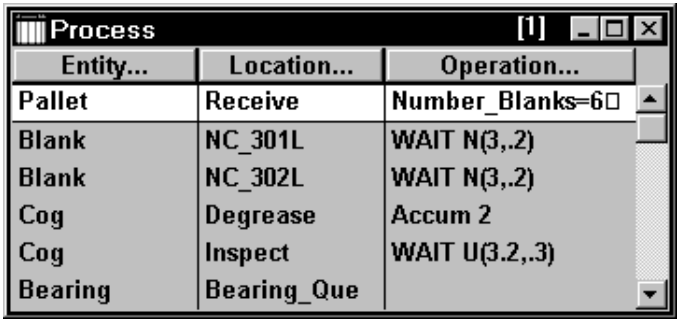
Defining Process Logic

The last step in defining Phase 1 of our model is to define the processing of entities at each location. ProModel simplifies this task by allowing you to select an entity type and then use the mouse to click on the locations in the order in which they will process the entity. Each time you click on a location, a new processing record is added to the Processing edit table, defining the process for that entity type at that location. For more information on defining process logic, see *Processing* on page 295, and *Operation Logic* on page 499.

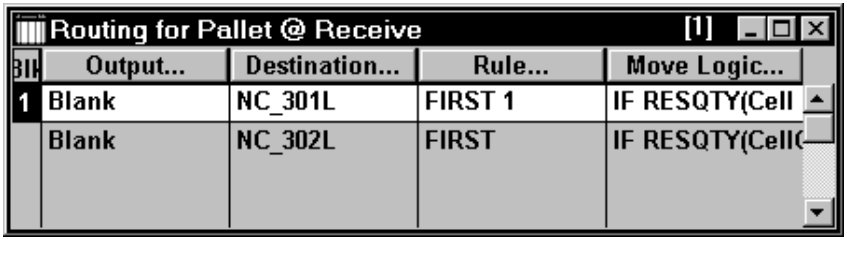
Once the basic entity flow has been defined using the point and click method, operation statements are added to the processing logic. The processing logic can be as simple as a constant operation time or as complex as a nested IF...THEN...ELSE statement.

Process editing actually involves two edit tables that normally appear side by side. The Process edit table specifies what happens to an entity when it arrives at a location, and the Routing edit table specifies where an entity is to be sent once processing is complete.

Process Edit Table



Routing Edit Table



Process and Routing Logic

The entire process and routing tables for the Phase 1 model are shown below. The table reads as follows:

1. When an entity called Pallet arrives at location Receive there is no operation time or processing logic (it's just a storage location). The resulting output is six entities called Blank that are routed to the FIRST available destination of either NC_301L or NC_302L.
2. When Blanks arrive at NC_301L or NC_302L, the processing time is a normal distribution with a mean of 3 and a standard deviation of .2 minutes. The name of the entity is now changed to Cog, and the Cog is sent to the Degrease location (FIRST is the default routing rule).
3. Two Cogs are accumulated at Degrease and processed for 5 minutes. When the degrease cycle is complete, Cogs are routed to location Inspect.
4. The inspection time is a uniform distribution with a mean of 3.2 and a half range of .3 minutes. Ninety six percent of the Cogs pass inspection and exit the system, while four percent of the Cogs fail inspection and become Rejects.

Process Table

Entity	Location	Operation (min)
Pallet	Receive	
Blank	NC_301L	WAIT N(3,.2)
Blank	NC_302L	WAIT N(3,.2)
Cog	Degrease	ACCUM 2 WAIT 5
Cog	Inspect	WAIT U(3.2,.3)

Routing Table

Blk	Output	Destination	Rule	Move Logic
1	Blank	NC_301L	FIRST 6	MOVE FOR .5
	Blank	NC_302L	FIRST	MOVE FOR .5
1	Cog	Degrease	FIRST 1	MOVE FOR .5
1	Cog	Degrease	FIRST 1	MOVE FOR .5
1	Cog	Inspect	FIRST 1	MOVE FOR .5
1	Cog	EXIT	0.960 1	
	Reject	EXIT	0.040	

3.3.5 Phase 2: Adding Resources & Variability

In this phase we wish to add the operator, CellOp, to move entities from location to location, and to perform the loading, unloading, and inspection operations. The easiest way to add the necessary model elements is to once again allow AutoBuild to guide us through the necessary steps. ProModel requires that all dynamic resources travel on a path network. Therefore, for Phase 2, we need to define resources and path networks.

Defining Path Networks

Path Networks consist of nodes and path segments which connect nodes to other nodes. Each location where a resource may stop to pick up, drop off, or process entities must interface with a path node.

We define path networks through the Path Network edit table similar to the previous edit tables. For each network we specify the nodes and path segments connecting the nodes. Some of the heading buttons, such as Paths and Interfaces, bring up other edit tables such as the Path Segment Edit table shown below. For more information on defining path networks, see *Path Networks* on page 243.

Path Network Edit Table

Path Networks [1]

Graphic	Name	Type	T/S	Paths...	nterfaces...	Mapping...	Nodes
	CellNet	Non-Passin	Time	7	5	0	5

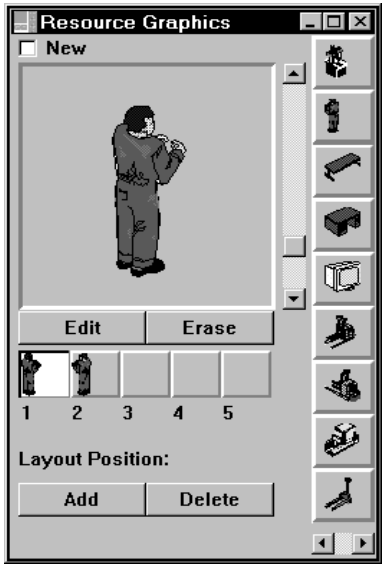
Path Segment Edit Table

Paths [1]

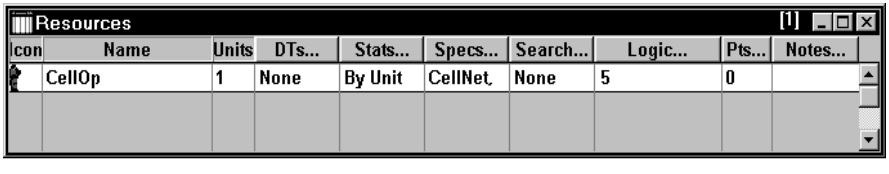
From	To	BI	Time
N1	N2	Bi	0.14
N2	N3	Bi	0.10
N3	N4	Bi	0.18
N4	N5	Bi	0.18
N5	N1	Bi	0.20
N2	N4	Bi	0.29
N1	N3	Bi	0.22

Defining Resources

Resources are defined in much the same way as entities. When in the Resource module we simply select an icon to represent the resource and then specify the characteristics of the resource in the Resource edit table. For more information about defining resources, see *Resources* on page 261.



The Resource edit table shown below contains fields for specifying the name and number of units of a resource. It also has fields for specifying resource downtimes (DTs...), the level of statistics to collect (Stats...), which path network used for travel (Specs...), and any work and park search routines (Search...). Clicking the mouse on any of these buttons brings up separate edit tables for specifying this data.



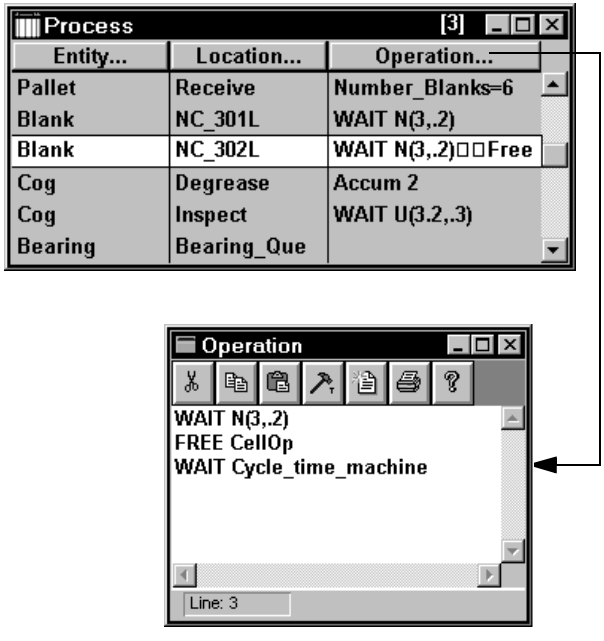
Process Editing

Now that we have defined a resource, we must specify how and when that resource is used in the processing logic.

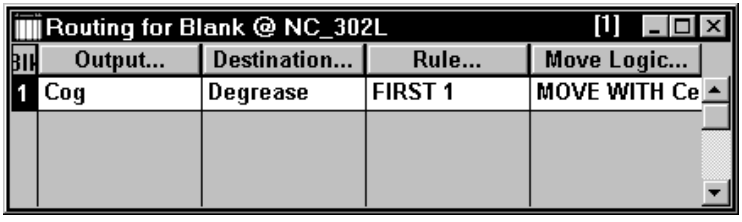
In the Phase 1 model we used only constant processing times. Now, due to variability associated with the operator, we must represent the loading and inspection times as distributions.

In the example below, CellOp loads the blank at mill NC_302L and is then FREEd to perform other operations. When the Blank has finished processing, the entity is moved with the CellOp to the degreasing machine.

Process Edit Table and corresponding operation logic



Routing Edit Table



Process and Routing Logic

The complete process and routing logic is shown below, with CellOp used to perform the loading operations at each mill and inspect the Cogs at the Inspect location. CellOp is also used to transport entities from location to location. All additions or changes to the Phase 1 model are shown in bold type.

Process Table

Entity	Location	Operation (min)
Pallet	Receive	
Blank	NC_301L	**WAIT N(3,.2)** **FREE CellOp** WAIT 5.7
Blank	NC_302L	**WAIT N(3,.2)** **FREE CellOp** WAIT 5.7
Cog	Degrease	ACCUM 2 WAIT 5
Cog	Inspect	**WAIT U(3.2,.3)** **FREE CellOp**

Routing Table

Blk	Output	Destination	Rule	Move Logic
1	Blank	NC_301L	FIRST 6	MOVE **WITH CellOp**
	Blank	NC_302L	FIRST	MOVE **WITH CellOp**
1	Cog	Degrease	FIRST 1	MOVE **WITH CellOp**
1	Cog	Degrease	FIRST 1	MOVE **WITH CellOp** **THEN FREE**
1	Cog	Inspect	FIRST 1	MOVE **WITH CellOp**
1	Cog	EXIT	0.960 1	
	Reject	EXIT	0.040	

With the new processing now defined, we have specified all of the necessary modeling elements. We are now ready for the model execution phase. Once again, we shall defer discussion of model execution until we have finished the final phase of the model.

3.3.6 Phase 3: Additional Operations

In the final phase of our modeling session we want to demonstrate an assembly operation by using the operator to install a Bearing into the center hole of the Cog if (and only if) the Cog passes inspection.

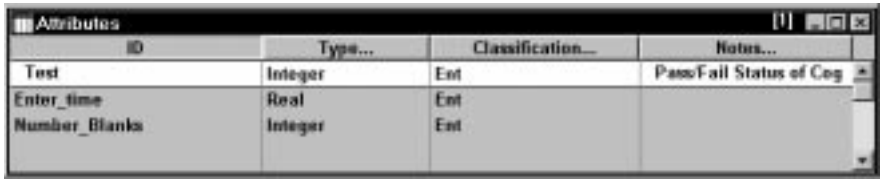
Once we have defined the attribute and distribution table, we must return to the Locations, Entities and Arrivals modules to define a new location called Bearing_Que, a new entity called Bearing, and an arrival schedule for the Bearings.

In addition, we also need to specify a usage based downtime for mills NC_301L and NC_302L from the Location module.

The final step in completing this phase of the model is to edit the processing logic to include the assembly. We will use the built-in JOIN construct to accomplish the assembly.

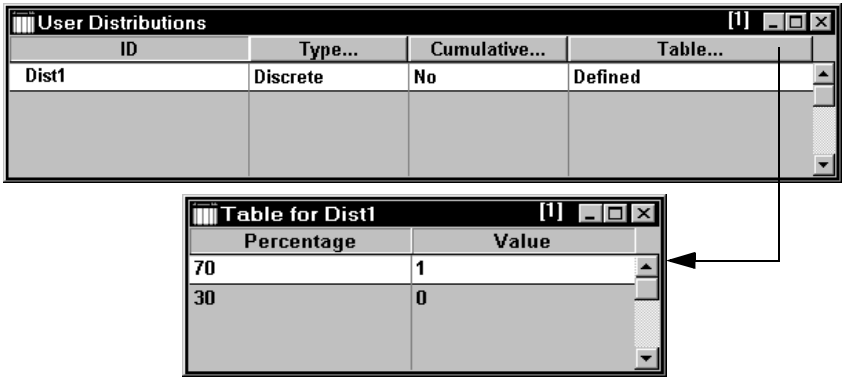
Defining Attributes

AutoBuild takes us to the Attribute module where we enter the Attribute edit table to define an Entity Attribute called Test that holds integer values. We set this attribute to one if the Cog passes the inspection, or zero if it fails the inspection. For more information about defining attributes, see *Attributes* on page 413.



Defining a Distribution

In order to determine if an entity passes or fails the inspection, you sample from a user-defined distribution called Dist1 (alternately, you could use the RAND() function). To define the distribution, simply click the mouse on the Table... button and fill in the distribution parameters. In this case, 96% of the entities pass inspection and 4% fail.

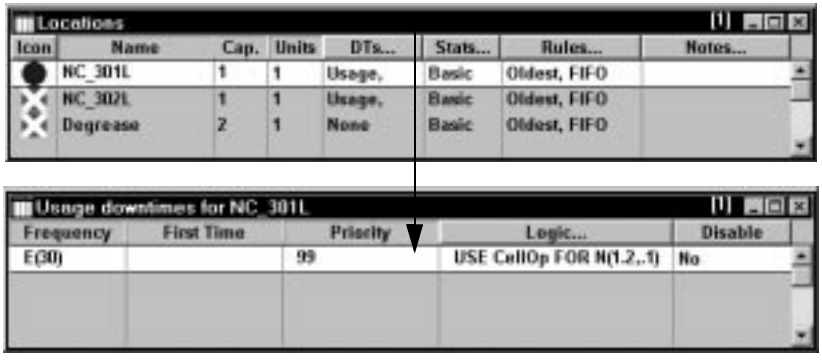


New Location, Entity, and Arrival

Before we can assemble the Cog at the Inspect location we must first define the new entity type called Bearing in the Entities module. We must also define a new location, Bearing_Que, to hold the Bearings, and an arrival schedule for the Bearings. To do this we simply open the appropriate module as in Phase 1 and supply the information. For simplicity we'll skip the details and move on to downtime specification.

Defining Location Downtimes

In order to represent machine failure times for the two mills, NC_301L and NC_302L, we click on the DT... button in the Location edit table shown below. This brings up another edit table for specifying a downtime based on machine usage. For more information about defining location downtimes, see *Locations* on page 201.



In the example above, we have defined failures to occur according to an exponential distribution with a mean of 30 minutes. When a machine fails, resource CellOp is required to service the machine.

Process and Routing Logic

The process and routing table below shows all of the changes and additions to the Phase 2 model in bold text.

Process Table **Routing Table**

Entity	Location	Operation (min)	Blk	Output	Destination	Rule	Move Logic
Pallet	Receive		1	Blank	NC_301L	FIRST 6	MOVE WITH CellOp
				Blank	NC_302L	FIRST	MOVE WITH CellOp
Blank	NC_301L	WAIT N(3,.2) FREE CellOp WAIT 5.7	1	Cog	Degrease	FIRST 1	MOVE WITH CellOp
Blank	NC_302L	WAIT N(3,.2) FREE CellOp WAIT 5.7	1	Cog	Degrease	FIRST 1	MOVE WITH CellOp THEN FREE
Cog	Degrease	ACCUM 2 WAIT 5	1	Cog	Inspect	FIRST 1	MOVE WITH CellOp THEN FREE

Cog	Inspect	WAIT U(3.2,.3)
		Test = Dist1()
		IF Test = 1 THEN
		BEGIN
		JOIN 1 Bearing
		WAIT U(1.2,.2)
		FREE CellOp
		ROUTE 1
		END
		ELSE
		BEGIN
		FREE CellOp
		ROUTE 2
		END
			1	Cog	EXIT	**FIRST 1**	

			2	Reject	EXIT	**FIRST 1**	
Bearing	**Bearing_Que**		**1**	**Bearing**	**Inspect**	**JOIN** 1	**MOVE FOR .05**

This concludes the final phase of our model building session. We now turn our focus to running the model.

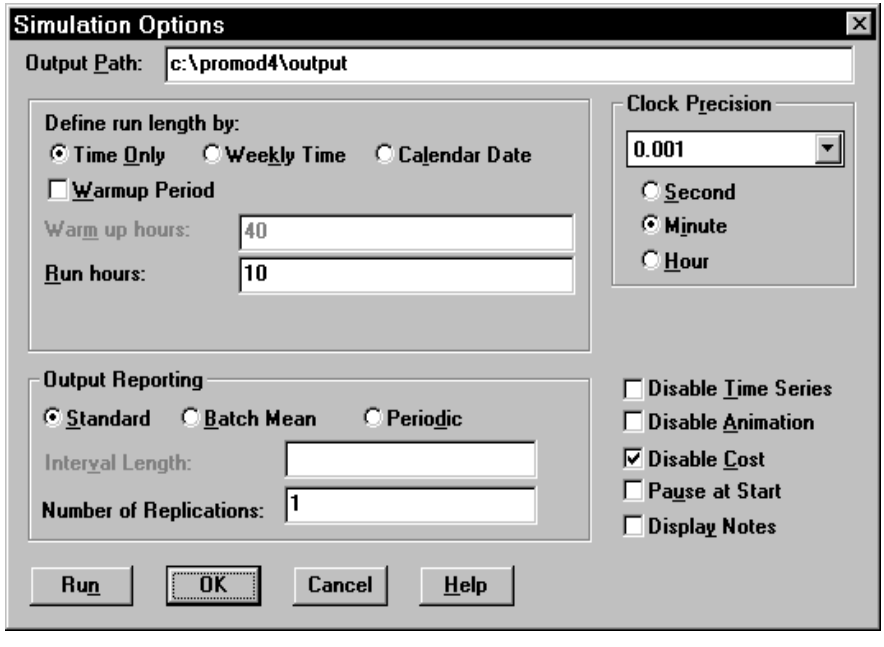
3.4 Running a Model

Running a model is a fun and easy process. Models are compiled automatically at run time, keeping you apart from any complex compilation process. If your model contains any errors, a detailed message explains the nature of the error and points to the module and line number where the error occurred. In most cases you are permitted to make changes on the fly.

ProModel uses concurrent animation, which means that the animation occurs while the simulation is running. Concurrent animation has many advantages over post-simulation animation. By eliminating the two-step process of running the simulation and then animating it, you save valuable time. Concurrent animation immediately allows you to see if a model is working properly.

3.4.1 Simulation Options

When you select Options from the Simulation menu, ProModel displays the Simulation Options dialog. This dialog contains several options for controlling the simulation, such as the run length, warm-up period, clock precision, and the name of the output file. You can also set the number of replications and the level of detail to be collected for the statistics. For more information about Simulation options, see *Simulation Options* on page 569.

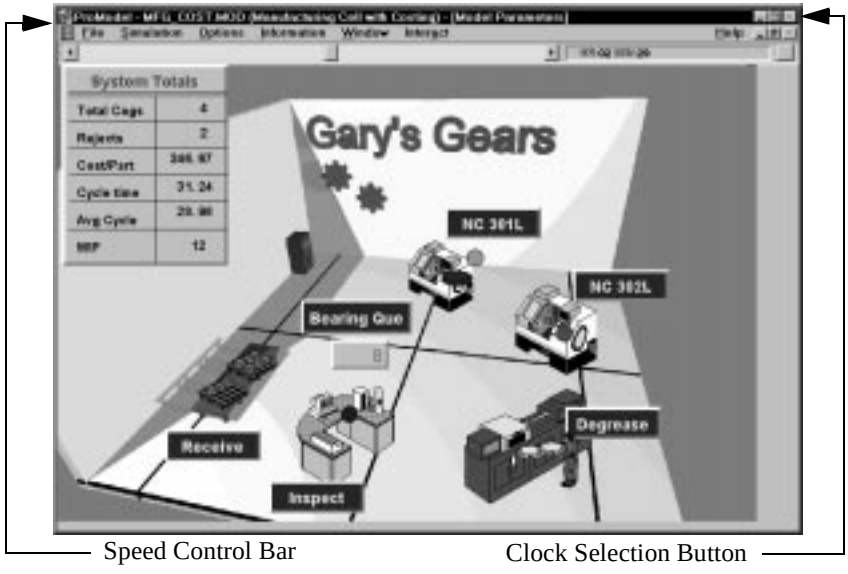


The maximum run length depends on the clock precision and the time unit selected as shown in the following table.

	CLOCK PRECISION			
Time Unit	**.01**	**.001**	**.0001**	**.00001**
Seconds (sec)	11,930 hrs	1,193 hrs	119 hrs	11 hrs
Minutes (min)	715,827 hrs	71,582 hrs	7,158 hrs	715 hrs
Hours (hr)	42,949,672 hrs	4, 294,967 hrs	429,496 hrs	42,949 hrs

3.4.2 Animation Screen

The ProModel animation screen has a menu of its own, with selections for controlling many simulation parameters (such as run speed). You can also control the animation through panning, zooming, and pausing.



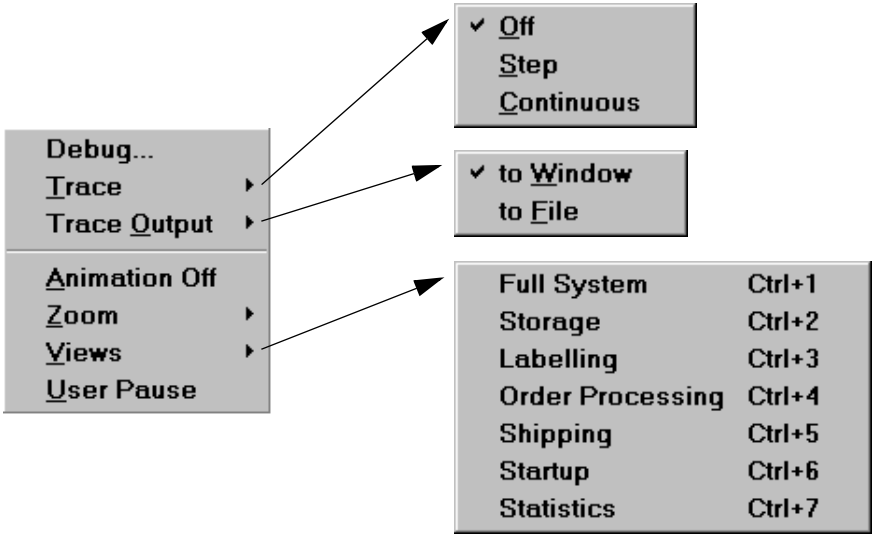
The screen above shows the speed control bar, along with the clock selection button for controlling the format of the clock readout. In addition, each machine or station has a status light which changes throughout the simulation to reflect the current operational state of each machine or station.

The File menu includes an option for viewing a text file of the model as the model is running. This is an excellent way of checking to make sure the model is doing what it is supposed to do!

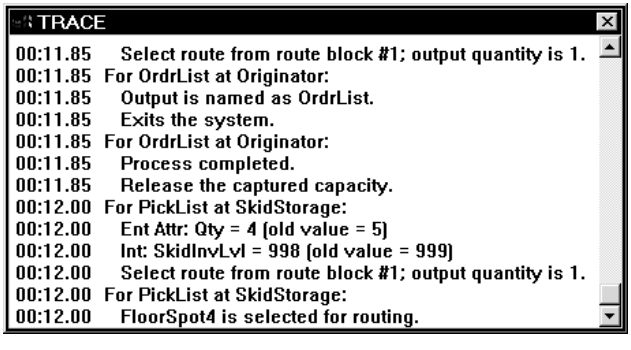
Next we'll take a look at two of the other menu items: Options and Information.

3.4.3 Options Menu

The Options menu contains several options that allow you to track events in the system as they occur. The Debugger is a convenient and efficient way to test or follow the processing of any logic defined in your model. The debugger is used to step through logic one statement at a time and examine variables and attributes while a model is running. A Step Trace allows you to step through the system events one at a time by clicking on the left mouse button. A Continuous Trace allows you to step through system events continuously without clicking the mouse.



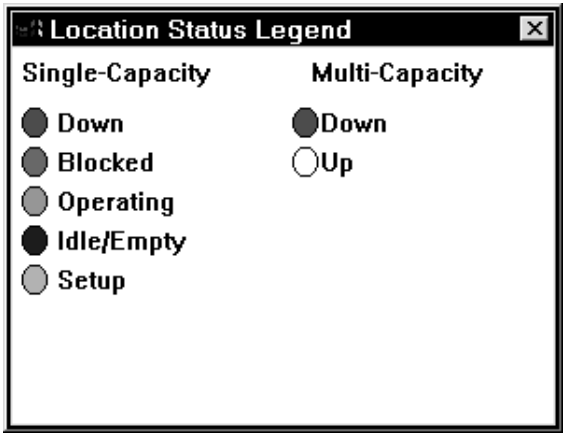
The following Trace window shows system events as they occur in the animation. The number in the left hand column represents the simulation time when the event occurred, while the text describes the event. Stepping through the events is an excellent way to verify and debug a model.



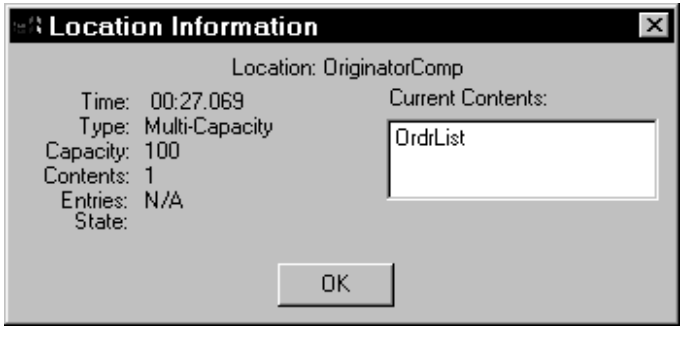
Other options include: Animation Off, which makes the simulation run considerably faster; Zoom, which allows you to zoom in or out to any degree on the animation; Views, which allows you to quickly and easily access specific areas of the model Layout window (see *Model Parameters & Scenarios* on page 575); and User Pause, which allows you to specify the time of the next simulation pause.

3.4.4 Information Menu

The Information menu contains selections for obtaining system information during the run. Select Status Light to bring up a Location Status Legend that defines the colors of the status light on each machine.



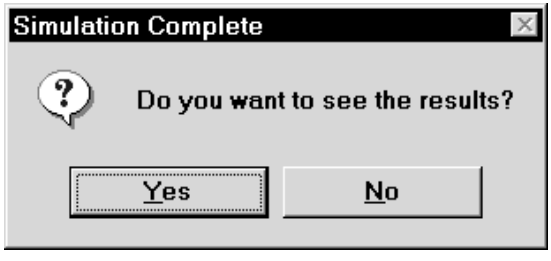
For up-to-the-minute location information, such as current location contents and total number of entries, select Locations from the Information Menu. The following information is for one location, OriginatorComp.



3.5 Viewing Model Statistics & Reports

The purpose of any simulation model is to gain a deeper understanding of the system under study. ProModel's output generator helps you to see the interactions between various system elements through tabular and graphical representation of system parameters such as resource utilization, throughput history, cycle time, and work in process levels.

After each simulation run you are prompted to view the model output. You can select yes to view the results immediately, or select no to continue with some other task. Selecting yes opens the Statistics Viewer and automatically loads the output of the most recent model run.



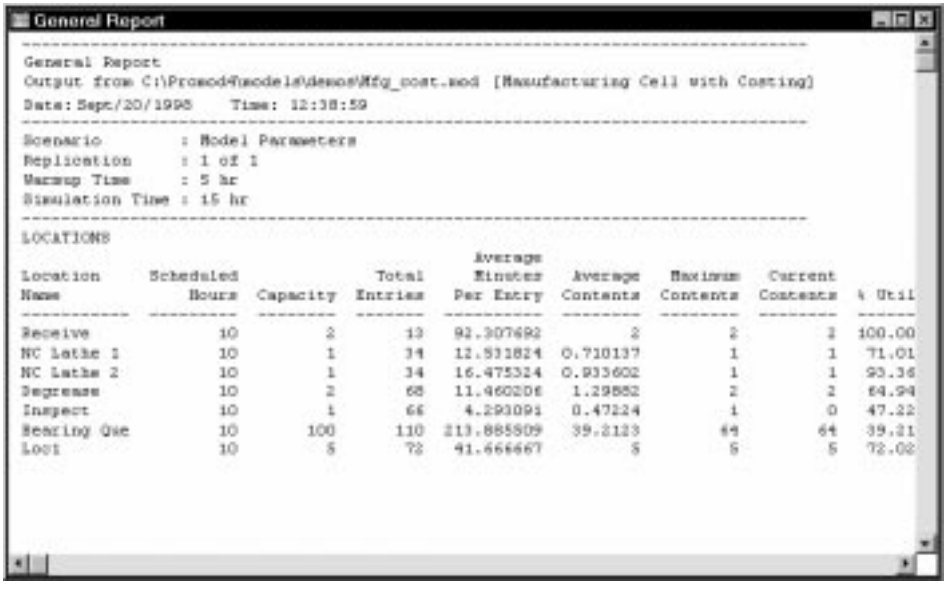
The STAT module can be run directly from within ProModel, or as an independent application separate from ProModel. You can load a single results file or several results files from different models for comparison of selected statistics.

Model output is written to several output files according to the type of data being collected. The main output file contains information of a summary nature such as overall location utilization and number of entries at each location. Other files keep track of information such as location contents over time and the duration of each entity at each location.

In order to see the power and flexibility of the ProModel output generator, we look next at some of the tabular and graphical output. For more information about viewing a simulation's output, see *Reports & Graphs* on page 619.

3.5.1 General Statistics Report

The General Stats report summarizes information for each location, entity, and resource contained in the output file <*model name*>.RDB.



General Report

```
General Report
Output from C:\Promod4\models\demos\Mfg_cost.mod [Manufacturing Cell with Costing]
Date: Sept/20/1995    Time: 12:38:59
-------------------------------------------------------------------------------
Scenario        : Model Parameters
Replication     : 1 of 1
Warmup Time     : 5 hr
Simulation Time : 15 hr
-------------------------------------------------------------------------------
LOCATIONS

Location      Scheduled                  Total     Average Minutes   Average    Maximum    Current
Name          Hours     Capacity         Entries   Per Entry         Contents   Contents   Contents   % Util
------------  --------  --------  -------  ----------  --------  --------  --------  ------
Receive         10          2          13    92.307692         2         2         2    100.00
NC Lathe 1      10          1          34    12.511824  0.710137         1         1     71.01
NC Lathe 2      10          1          34    16.475324  0.933602         1         1     93.36
Degrease        10          2          68    11.460206   1.29882         2         2     64.94
Inspect         10          1          66     4.293091   0.47224         1         0     47.22
Bearing Que     10        100         110   213.885509   39.2123        64        64     39.21
Lost            10          5          72    41.666667         5         5         5     72.02
```

Although the entire file is not visible in this window, scroll bars on the bottom and right hand side of the window allow you to scroll through the entire file.

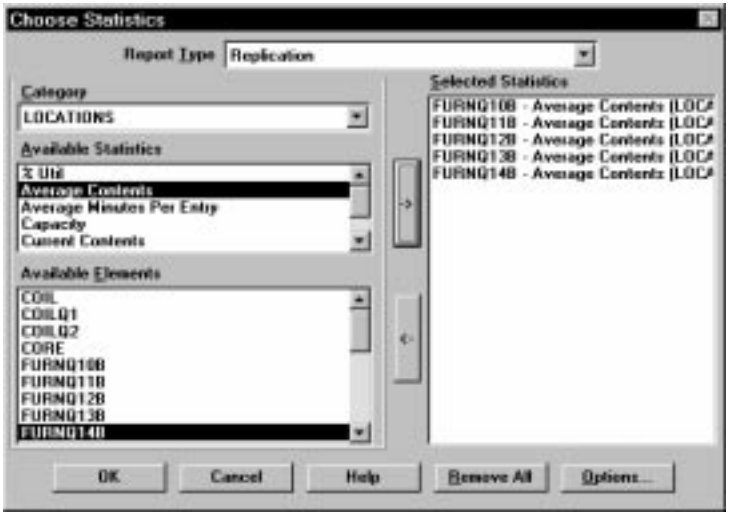
Summary information for locations is divided into two categories, one for single capacity locations and one for multi-capacity locations. This is due to the fact that many of the fields in the multi-capacity section, such as Max Contents, do not apply to single capacity locations.

For more information about a simulation's tabular output, see *Creating Reports* on page 631.

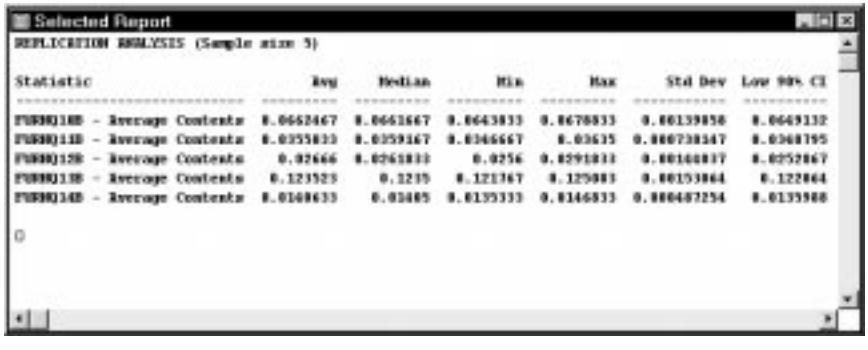
The following graphs are derived from the data in the file above and represent just a small sample of the many possible graphical representations of the data.

3.5.2 Selected Statistics Report

The Selected Stats report is an optional report when multiple replications, batch means, or periods have been defined. Information for specific selections you make is displayed in the following dialog.



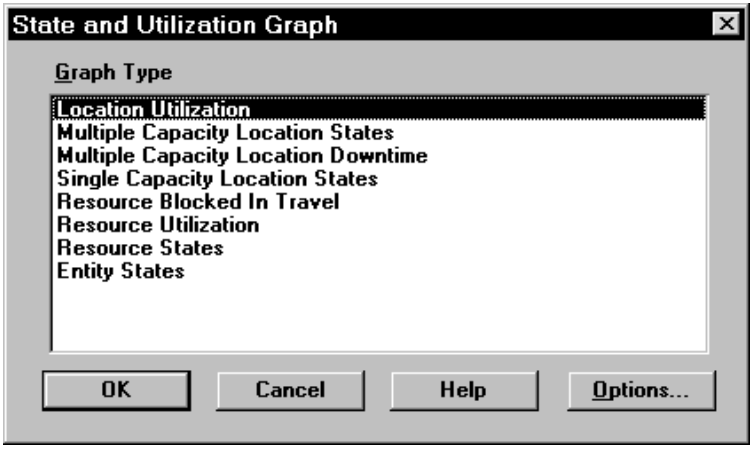
The selections in the dialog above produce the following report:



Although the entire file is not visible in this window, scroll bars on the bottom and right side of the window allow you to scroll through the entire file. For more information about a simulation's tabular output, see *Creating Reports* on page 631.

3.5.3 State & Utilization Graphs

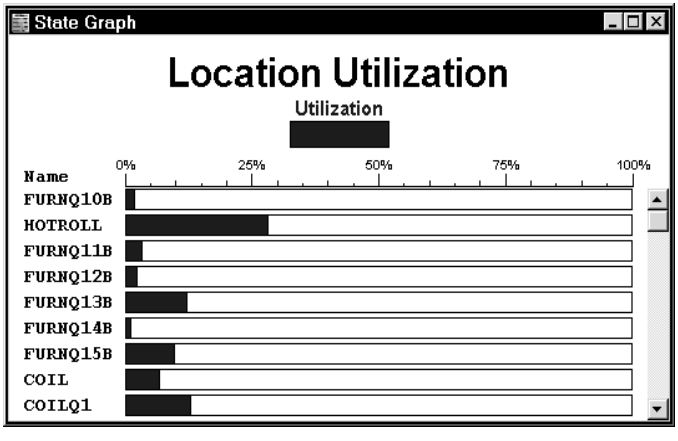
The Output Program allows you to create different types of state and utilization graphs to illustrate the percentage of time that locations, resources, and entities are in a particular state: operation, waiting, blocked, down, etc. For more information about state and utilization graphs, see *Creating Graphs* on page 647.



Location Utilization Graph

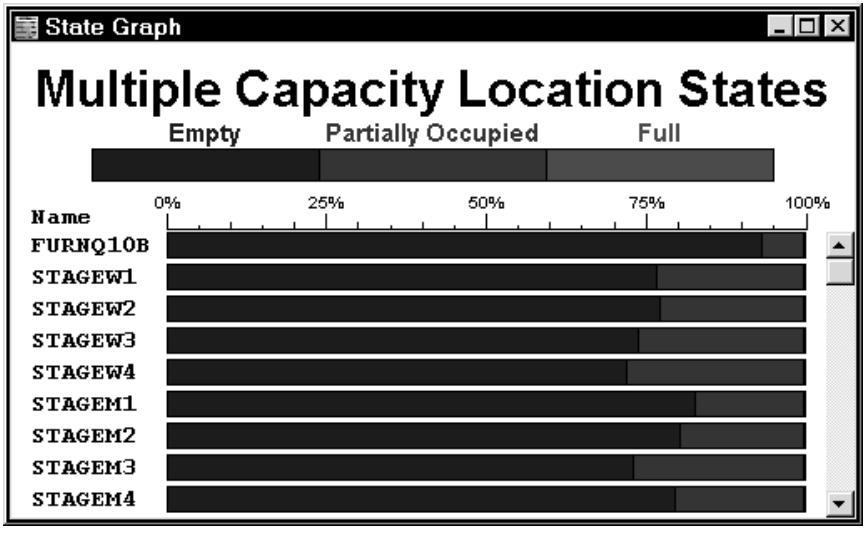
Location Utilization graphs show the percentage of time each location in the system was utilized. ProModel allows you to see this information for any or all of the locations in the system.

This location utilization graph displays all single capacity locations in the model.

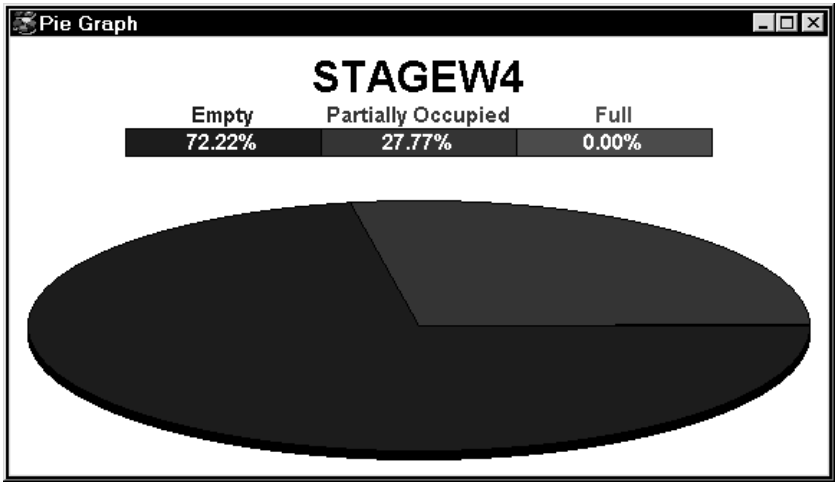


Location State Graphs

Single and Multiple Capacity Location State graphs show the percentage of time each location in the system was in a particular state, e.g., idle, in operation, waiting for arrivals, blocked or down for single capacity locations. ProModel allows you to see this information for all locations in the system at once.



For an alternative graphical view of one of the locations, simply click the mouse on any one of the bar graphs to create a pie chart for that location as shown below.



Multiple Capacity Location Downtime Graph

Multiple Capacity Location Downtime graphs show the percentage of time each multi-capacity location in the system was down. A pie chart can be created for any one of the locations.

Resource Blocked In Travel Graph

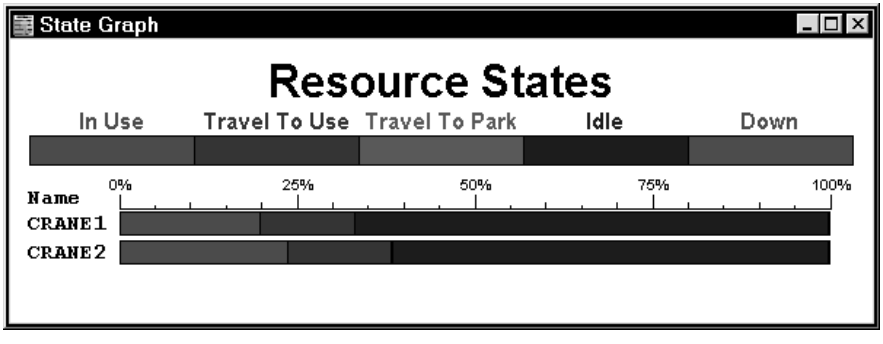
Resource Blocked in Travel graphs show the percentage of time a resource was blocked. A resource is blocked if it is unable to move to a destination because the next path node along the route of travel was blocked (occupied).

Resource Utilization Graph

Resource Utilization graphs show the percentage of time each resource in the system was utilized. A resource is considered to be utilized whenever it is *not* idle, traveling to park, or down. ProModel allows you to see this information for all resources in the system at once.

Resource State Graph

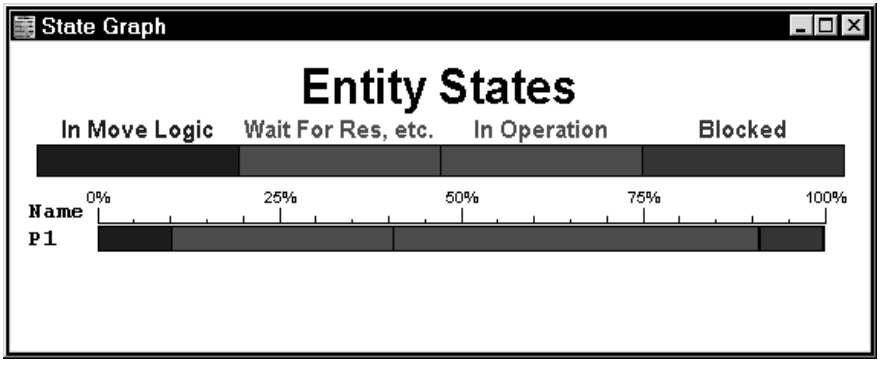
Similar to location state graphs are resource state graphs which show the relative amount of time each resource spent in a particular state. The graph below shows the state of the single resource **Crane1** and **Crane2**. From this graph we see immediately that **Crane2** is only 25% utilized.



Once again, a pie chart for any resource is just a mouse click away! Click on the bar of the graph and the Output Program displays a pie chart for the resource.

Entity State Graph

Entity State graphs show the percentage of time an entity is in a particular state, e.g., in transit, waiting for a resource, in operation, or blocked. Click directly on the desired bar of the graph to display a pie chart for that entity.



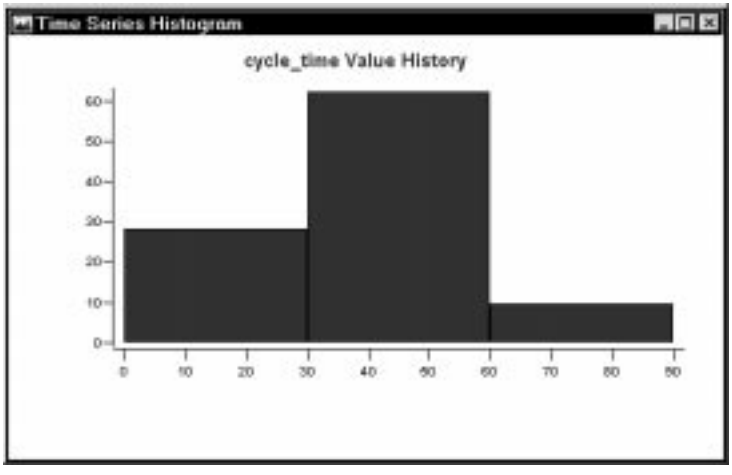
3.5.4 Time Series Plots & Histograms

In previous versions of ProModel, time series plots and histograms were known as throughput, content, value, and duration plots and histograms. These charts can now be combined on one chart for improved comparisons. Time series statistics can be collected for locations, entities, resources, and variables. See *Defining General Elements* on page 197 and *Defining Advanced Elements* on page 411 for more information on selecting the type of statistics to be collected for these elements.

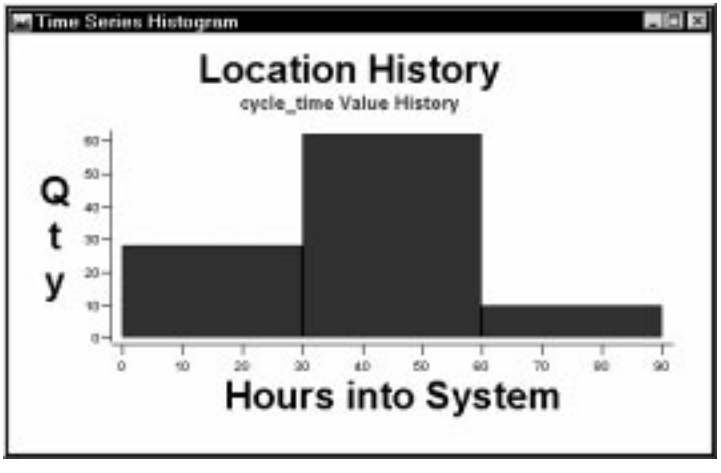
When you select one of the Time Series menu item choices, a dialog appears, allowing you to select the values to graph and the options to customize their appearance. For more information about time series graphs, see *Creating Graphs* on page 647.

Time Series Histogram

The graph below shows a content histogram for the **cycle_time** variable. The selected time interval is 90 minutes.



Selecting **Edit** from the **Options** menu displays the Graph Options dialog that allows you to select time units, bar width, and other options to modify the look of your graph. The following graph represents the same information as the one above, but helps you identify the information and understand it better.



Time Series Plots

Time Series Plots give you a different perspective from histograms. You can track global variables such as total time accumulated and work-in-process value history. The following plot shows the **total_time_accum** Value History and the **WIP** (work in progress) **Value History** over a 90 minute period.