extendsim®

USER REFERENCE

License, Software Copyright, Trademark, and Other Information

The software described in this manual is furnished under a separate license and warranty agreement. The software may be used or copied only in accordance with the terms of that agreement. Please note the following:

> ExtendSim blocks and components (including but not limited to icons, dialogs, and block code) are copyright © by ANDRITZ Inc. and/or its Licensors. ExtendSim blocks and components contain proprietary and/or trademark information. If you build blocks, and you use all or any portion of the blocks from the ExtendSim libraries in your blocks, or you include those ExtendSim blocks (or any of the code from those blocks) in your libraries, your right to sell, give away, or otherwise distribute your blocks and libraries is limited. In that case, you may only sell, give, or distribute such a block or library if the recipient has a valid license for the ExtendSim product from which you have derived your block(s) or block code. For more information, contact ANDRITZ at Info.ExtendSim@Andritz.com or Support.ExtendSim@Andritz.com.

Acknowledgments

Extend was created in 1987 by Bob Diamond; it was re-branded as ExtendSim in 2007.

The contents of this document are the result of years of work by software architects, simulation engineers, and technical writers and editors of ExtendSim products.

ANDRITZ Inc • 13560 Morris Road, Suite 1250 • Alpharetta, GA 30004 USA
770.640.2500 • Info.ExtendSim@Andritz.com
www.ExtendSim.com

Table of Contents

Table of Contents

Simulation

About ExtendSim

How To

DISCRETE EVENT MODELING

DISCRETE RATE MODELING

APPENDIX

INDEX

Preface

Why I Created ExtendSim

ExtendSim's architect
talks about simulation

"What we experience of nature is in models,
and all of nature's models are so beautiful."
— R. Buckminster Fuller

Dedicated to the pleasure of finding things out

Simulation is defined as the act of imitation. Even a word processor simulates pen and paper, but how do you get the computer to behave like the stock market, or an electronic circuit, or even a car, and how can you communicate this power to the user? My search for the answer began in the early days of the space race.

I was attending the Polytechnic Institute of Brooklyn when the head of the Electronics Engineering department told us that a new department was being formed... a combination of mathematics, computers, physics and engineering. Being into math, and curious about the large IBM mainframe lurking down the hall, I immediately joined and made a constant pest of myself at the computer center.

The bug bit hard, I guess, and I began to realize that I could use computers to duplicate the laboratory experiments in class so well, that I never really did them, I just simulated them on the computer. NASA then asked if I could develop a simulation of their new liquid fuel booster for something called Project Apollo. I came up with the Rocket-Drop simulation, a monstrously large program that only had one function: follow the path of a single droplet of fuel, from the shower heads (as the fuel sprayers at the top of the engine were called) to the rocket engine exhaust, via subsonic, supersonic, and hypersonic flow.

It hit me then that simulation was inaccessible, except to the select few who had the resources to put together an entire system dedicated to one function. A generalized simulation application would be a great and useful thing, if one could find the computer that was both powerful enough and widespread enough to support it. This was 1965, and Seymour Cray was still building his super fast (at the time!) computers by hand and graphic user interfaces were still decades in the future.

When I saw the graphical user interfaces (GUIs) on the Mac OS and Windows machines, I realized that I could use these tools to fulfill that long awaited dream. ExtendSim is built upon those roots. Imagine That Inc. was founded in 1987 to develop and market Extend and its successor ExtendSim, the first simulation applications allowing users of any discipline to use simulation and to develop their own libraries of customized simulation tools.

My joyous moments come from bringing the art, science, and fun of simulation to the desktop, in a form digestible and accessible by everyone. ExtendSim is the first user-extendible simulation package that meets those expectations.

Bob Diamond

"You see? That's why scientists persist in their investigations, why we struggle so desperately for every bit of knowledge, stay up nights seeking the answer to a problem, climb the steepest obstacles to the next fragment of understanding, to finally reach that joyous moment of the kick in the discovery, which is part of the pleasure of finding things out." attributed to Richard P. Feynman

Simulation

An Overview of Simulation

What simulation is and why it's important.

"'Begin at the beginning,' the King said, gravely,
'and go 'til you come to the end; then stop.'"
— Lewis Carroll

Why simulate?

The goal of every company, government agency, and educational institution should be to develop a strong and competitive organization. Cost reduction and quality improvement alone are not sufficient to achieve market share. Organizations must also be able to quickly develop and provide innovative new products and services. Invention, innovation, quality, productivity, and speed are the keys to making companies competitive.

One way to maximize competitiveness is to improve operational systems and processes by:

- Eliminating nonessential, non value-adding steps and operations
- Implementing and inserting technology where appropriate
- Managing the deployment and utilization of critical resources
- Identifying key cost drivers for reduction or elimination

In spite of recent and rapid advances in technology, many companies and institutions still suffer from outdated equipment and inefficient work practices. This is due in part to the prohibitive expense and time required to explore alternative methods of operation and try out new technologies on real systems and processes. Simulating a system or process provides a quick and cost-effective method for determining the impact, value, and cost of changes, thus validating proposed enhancements and reducing the resistance to change.

Simulation models allow for time compression, are not disruptive of the existing system, and are more flexible than real systems. They also provide metrics for meaningful analysis and strategic planning. Simulation can help organizations answer questions about how they do work: what they do, why they do it, how much it costs, how it can be changed, and what the effects of changes will be.

What you can do with simulation

Simulation allows you to see how a real-world activity will perform under different conditions and test various hypotheses or alternatives at a fraction of the cost of performing the actual activity. It also allows you to explore the effect of making modifications to systems that are inaccessible, for systems and processes that do not yet exist, or where making changes would be dangerous or prohibited.

One of the principal benefits of a model is that you can begin with a simple approximation of a process and gradually refine the model as your understanding of the process improves. This “stepwise refinement” enables you to achieve good approximations of very complex problems surprisingly quickly. As you add refinements, the model more closely imitates the real-life process.

An important aspect of strategy development, simulation helps you understand complex systems and produce better results faster because you will be able to:

- Predict the course and results of certain actions
- Gain insight and stimulate creative thinking
- Visualize your processes logically or in a virtual environment
- Identify problem areas before implementation
- Explore the potential effects of modifications
- Confirm that all variables are known

- Optimize your operations
- Evaluate ideas and identify inefficiencies
- Understand why observed events occur
- Communicate the integrity and feasibility of your plan

Systems, models, and simulation

All professions use models of one form or another. But the word "model" does not always have the same meaning to business professionals, managers, scientists, and engineers. Even within a specific discipline, such as manufacturing, modeling has many different definitions. The following discussion serves to clarify what "modeling" means as it relates to dynamic simulation tools such as ExtendSim.

Systems

The real world can be viewed as being composed of systems. A system is a set of interacting or interdependent components that form a whole. The components interact with each other based on the rules or operating policies of the system.

- Components are the internal parts of the system. They could be individual entities, generalized substances, or any other parts of the system that are involved in one or more of its processes.
- Operating policies are the external inputs to the system. These types of controls and the availability of resources govern how the system operates and thus how the components of the system interact.

Over time, the activities and interactions of components cause changes to the state of the system; this is called system behavior or dynamics. Systems can be mathematically straightforward, such as a flower growing in the soil and turning towards the sun to maximize photosynthesis. Or they can be more complex, such as supply chain operations composed of planning, selling, distribution, production, and sourcing subsystems.

Models

A model is an abstracted and simplified representation of a system at one point in time. Models are an abstraction because they attempt to capture the realism of the system. They are a simplification because, for efficiency, reliability, and ease of analysis, a model should capture only the most important aspects of the real system.

Most models can be classified into four basic types:

1) A scaled representation of a physical object, such as a 1:18 diecast model of a Tesla Model X, a clay model of a proposed packaging bottle, or a scale model of the solar system.

2) A graphical or symbolic visualization, such as a flow chart of office procedures or an architect's plans for a building.

3) An analytical or mathematical formula that yields a static, quantitative solution. For example, an analytic model might consist of several independent sample observations that have been transformed according to the rules of the model. Common examples of analytic models are spreadsheet models or linear programming models.

4) A mathematical description that incorporates data and assumptions to logically describe the behavior of a system. This type of model is typically dynamic—it has a time component and shows how the system evolves over time. ExtendSim products are tools for building mathematically-based, dynamic models of systems.

Dynamic modeling is the foundation for computer modeling. For purposes of this manual, the word "model" will be used to mean a description of the dynamic behavior of a system or process.

Simulation

The Merriam-Webster OnLine Dictionary defines simulation as "the imitative representation of the functioning of one system or process by the functioning of another." Thus to determine how an actual system functions, you would build a model of the system and see how the model functions.

Simulations run in simulation time, an abstraction of real time. As the simulation clock advances, the model determines if there have been changes, recalculates its values, and outputs the results. If the model is valid, the outputs of the simulation will be reflective of the performance or behavior of the real system.

Simulation with ExtendSim means that instead of interacting with a real system you create a logical model that corresponds to the real system in certain aspects. You simulate the operations or dynamics of the system, then analyze one or more areas of interest. You do this in order to reduce risk and uncertainty so that you can make informed, timely decisions.

In summary

Simulation is the act of creating a mathematical model of a system then causing the model to replicate over time to represent the operation of the system. The modeled system can be any real world or proposed problem, situation, or process that you want to explore. Thus simulation is:

- An analysis tool, used to predict the effect of changes on existing systems
- A design tool, for predicting the behavior or performance of potential new systems

Additional modeling terminology

These are terms you're going to need to know, so it's good to get them straight from the beginning.

Modeling methodologies

A modeling methodology (continuous, discrete event, discrete rate, and so forth) is how you model a system. See "Methods for specifying systems" on page 10 for more information.

Model parameters, inputs, and outputs

A parameter is any numerical characteristic of a model or system. Parameters describe something about the model and are known or can be estimated.

- An input parameter is a value that is required as part of the model specification.
- An output parameter is a value determined by the input parameters and the operation of the system. They specify some measure of the system's performance or dynamics.

Constants and randomness

Input parameters specify the settings for a model and can be constant or random. Constant values never change; random values are based on distributions and change each time they are used. Models that have no random input parameters are referred to as deterministic models. Models that are based on one or more variables that are random are said to be stochastic, as discussed below:

- *Deterministic* models contain only non-random, fixed parameters. No matter how many times a deterministic model is run, unless some parameter is changed there is no uncertainty and the output will be exactly the same. Thus the behavior of the model is "determined" once the inputs have been defined.

  The advantage of a deterministic model is that only one run is necessary, since it produces an exact measurement of the model's performance. It is also helpful when initially building a model since you can be assured that changes in results will be due to changes made to the model and not to randomness. The disadvantage is that these types of models can only accurately be used to model a few types of processes, since real-world systems typically contain some element of randomness.

- Adding randomness to one or more inputs to a deterministic model changes it to a *stochastic* or *Monte Carlo* model. Stochastic models should be run repeatedly and then analyzed statistically to determine a likely outcome. Notice that the occurrence of randomness does not mean that the behavior of a process is undefinable or even that it is unpredictable. Random variables vary statistically as defined by a distribution. This means that their range and possibility of values is predictable.

  While stochastic models can be applied to very complex systems, a disadvantage is that the output is itself random—the average of multiple simulation runs provides only an estimate of the model's true behavior.

☞ ExtendSim provides several methods for including randomness in models. For instance, the Random Number block (Value library) allows you to select a random distribution or enter a table of values to specify an empirical distribution of probabilities. For more detailed information about random number capabilities, see "Random numbers" on page 197.

Simulation

Modeling Methodologies

Methods for specifying systems

As mentioned in an earlier chapter, a system is the problem, process, situation, or entity that you want to model. The method you use to specify the system is termed a modeling methodology.

Main modeling methodologies

The three main modeling methodologies are:

- *Continuous process* modeling (sometimes just called process modeling) describes a time-based system where state variables, which describe the system at any point in time, change continuously as time advances.
- *Discrete event* models simulate event-based systems and processes that usually involve queues. In a discrete event model, nothing happens between points in time unless an event occurs.
- *Discrete rate* models share some aspects of both continuous and discrete event modeling.

These methodologies are described, compared, and contrasted in the later sections in this chapter.

Other modeling approaches

In addition to the main modeling methodologies listed above, other modeling approaches are useful and will be discussed in this chapter. These approaches are often based on one of the three main methods and include:

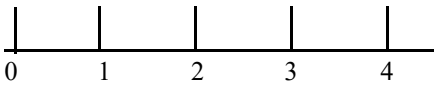
- Monte Carlo
- Agent-based
- State/Action
- Reliability Block Diagramming (RBD)

These will be discussed in “Other modeling approaches” on page 13.

How time is treated in the main modeling methodologies

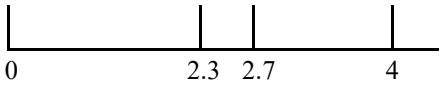
Time is an element of continuous process, discrete event, and discrete rate simulations. The differences between them mainly involve how time is treated and what causes the state of the model to change.

- In *continuous process* models, the time step is fixed at the beginning of the simulation, time advances in equal increments, and values change based directly on changes in time. For a continuous model, values reflect the state of the modeled system at any particular time and simulated time advances evenly from one time step to the next. For example, an airplane flying on autopilot represents a continuous system since its state (such as position) changes continuously with respect to time. Continuous simulations are analogous to a constant stream of fluid passing through a pipe. The volume may increase or decrease at each time step, but the flow is continuous.



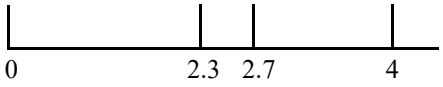
Time line for continuous simulation; step = 1

- In *discrete event* models, the system changes state as events occur and only when those events occur; the mere passing of time has no direct effect on the model. Unlike a continuous model, simulated time advances from one event to the next and it is unlikely that the time between events will be equal. A factory that assembles parts is a good example of a discrete event system. The individual entities (parts) are assembled based on events (receipt or anticipation of orders). Accordingly, a discrete event model would not be a good choice for simulating fluid flow.



Time line for discrete event simulation

- *Discrete rate* simulations are a hybrid type, combining aspects of continuous and discrete event modeling. Like continuous models they simulate the flow of stuff rather than items; like discrete event models they recalculate rates and values whenever events occur. Using the pipe analogy for a discrete rate simulation, there is a constant stream of fluid passing through the pipe. But the rates of flow and the routing change when an event occurs, not at set points in time.



Time line for discrete rate simulation

☞ In some branches of engineering, the term *discrete* is used to describe a system with periodic or constant time steps. This definition of discrete indicates a continuous model; it does not have the same meaning as discrete event or discrete rate. Continuous models in ExtendSim are stepped using constant time intervals; discrete event and discrete rate models are not.

Table of continuous process, discrete event, and discrete rate differences

Although not definitive, the following table will help to determine which style to use when modeling a system.

Factor	Continuous Process	Discrete Event	Discrete Rate
What is modeled	Generic “stuff”, represented by values that change when the model changes state.	Distinct and identifiable entities (“items” or “things”) that can be individually tracked.	Streams of homogeneous “stuff”. Or bulk or high speed flows of otherwise distinct entities where sorting or separating is neither necessary nor wanted.
What causes a change in state	A time change	An event	An event
Time steps (see “How time is treated in the main modeling methodologies” on page 10)	Interval between time steps is constant. Model recalculations are sequential and time-dependent.	Interval between events is dependent on when events occur. Model only recalculates when events occur.	Interval between events is dependent on when events occur. Model only recalculates when events occur.

Factor	Continuous Process	Discrete Event	Discrete Rate
Characteristics of what is modeled	Track characteristics in a database or assume the flow is homogeneous.	Use attributes to assign unique characteristics to items, which can then be tracked throughout the model.	Use attributes to assign unique characteristics to the flow, so it can be tracked throughout the model. Or assume the flow is homogeneous.
Order method	N/A	Items can move in FIFO, LIFO, Priority, or a custom order based on Item Attributes	Flow moves in FIFO, LIFO, or a custom order based on Flow Level Attributes
Routing	Values need to be explicitly routed by being turned off at one branch and turned on at the other (values can go to multiple places at the same time.).	By default, items are automatically routed to the first available branch (an item can only be in one place at a time.)	Flow is routed based on constraint rates and rules that are defined in the model (flow can be merged or divided into multiple branches.)
Statistical detail	General statistics about the system: amount, efficiency, etc.	In addition to general statistics, each item can be individually tracked: count, utilization, cycle time.	In addition to general statistics, effective rates, cumulative amount.
Typical uses	Scientific (biology, chemistry, physics), engineering (electronics, control systems), finance and economics, environmental impacts, System Dynamics.	Manufacturing, service industries, business operations, networks, systems engineering, transportation, logistics, call centers, emergency departments.	Manufacturing of powders, fluids, and high speed, high volume processes. Chemical processes, oil and gas, mining, ATM transactions. supply chains.
Examples	Processes: chemical, biological, economic, electronics, water and waste.	Things and Information: traffic, equipment, work product, people, data, messages, and network protocols at the packet level.	Rate-based flows: homogeneous products, high speed or high volume production, data feeds and streams, fluids and gases, minerals.
Main ExtendSim library	Value library Electronics library Custom blocks in custom libraries	Item library	Rate library

Which one do you choose?

☞ The types of models that can be built in ExtendSim depend on the product that was purchased.

"The" model of a system

As you might expect, you can use different methods to model different aspects of real-world systems. For example, at a chemical plant you could model the chemical reactions as a continuous process, the control logic of the chemical process using discrete event modeling, and the tanks, valves, and flow of the production process with discrete rate.

It is also good to note that there is no such thing as "the" model of a system: a system can be modeled in any number of different ways, depending on what it is you want to accomplish. In general, how you model the system depends on the purpose of the model: what type, level, and fidelity of information you want to gather and the intended amount of detail, or level of abstraction or granularity, of the model. Once that has been determined, you can intelligently choose which type of model to build.

Mixing it up

Some systems, especially when a portion of the flow has a delay or wait time, can be modeled using any of the three styles. In this case, you would generally choose how to model the system based on the level of detail required. Discrete event models provide much more detail about the workings of these types of systems than continuous models. Continuous and discrete rate models, on the other hand, usually run faster than discrete event models.

Also, its very likely that you would want to combine blocks from different ExtendSim libraries within the same model.

- For example, it is quite common to use continuous blocks from the Value library along with discrete event blocks from the Item library. Note however that using blocks from the Item or Rate libraries means that the model will be event-driven (non-continuous).
- And because discrete event blocks (Item library) and discrete rate blocks (Rate library) are both event-based, it is very common to use them to simulate different aspects of the same model.

☞ If you use any discrete event or discrete rate blocks in a model, the timing will change to event driven (time steps will not be periodic) and it will not be a continuous model. That is because a continuous model cannot use an Executive block (Item library), but discrete event and discrete rate models require it.

One interface for all modeling needs

Even though ExtendSim manages these three modeling methodologies differently, the models have the same user interface and are built and used in the same manner. ExtendSim takes care of the internal differences, so you can easily transition between building a discrete rate, a continuous, or a discrete event model. This consistency and flexibility is why ExtendSim is the product of choice for modeling all aspects of a business, from top to bottom and everything in between.

Other modeling approaches

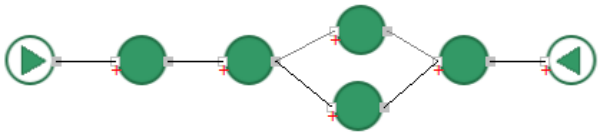
Although there are several other approaches to modeling, they usually fit within one of the three major categories (continuous, discrete event, or discrete rate) discussed above. For example, System Dynamics and Bond graphs are subsets of continuous modeling, and queueing theory models are subsets of discrete event modeling.

Because of their specialized use, four specific modeling approaches (Reliability Block Diagramming, Monte Carlo, State/Action, and Agent Based) are described below.

Reliability Block Diagramming

Reliability block diagramming (RBD) is an event-based methodology that graphically and statistically describes a system's resource availability over time. An RBD captures complex availability behavior, describing when scheduled and unscheduled downs occur for individual resources and specifying what impact that has on the availability of entire system.

Like the ExtendSim Item and Rate libraries used for discrete event and discrete rate modeling, dedicated RBD tools have a simulation clock that moves forward in discrete chunks of time. The advantage that dedicated RBD tools have over those process simulation tools is that RBD's visually capture and validate complex reliability logic and the relationships between a system's resources.



Using with process simulation software

When used with process simulation software, such as the ExtendSim Item or Rate libraries, RBD tools gain additional capabilities such as:

- More accurate assessment of when wear-based failures occur.
- Detailed repair modeling.
- The ability to explore the relationship between resource availability and system performance metrics such as throughput, production costs, repair costs, utilization, inventory, service levels, and so forth.

⚠ The Reliability library of ExtendSim Pro is used for creating either standalone RBD's or combined process simulation and reliability RBD's. For a tutorial and additional information, see the separate document titled Reliability; it is located at Documents/ExtendSim/Documentation.

Monte Carlo modeling

Widely used to solve certain problems in statistics, Monte Carlo simulations provide a range of results rather than a single value. This approach can be applied to any ExtendSim model and used wherever uncertainty is a factor.

Monte Carlo modeling uses random numbers to vary input parameters for a series of calculations. These calculations are performed many times and the results from each individual calculation are recorded as an observation. The individual observations are statistically summarized, giving an indication of the likely result and the range of possible results. This not only tells what could happen in a given situation, but how likely it is that it will happen.

Creating a Monte Carlo model in ExtendSim

You build a Monte Carlo simulation in ExtendSim by incorporating random elements in a model and obtaining multiple observations. There are two ways to do this:

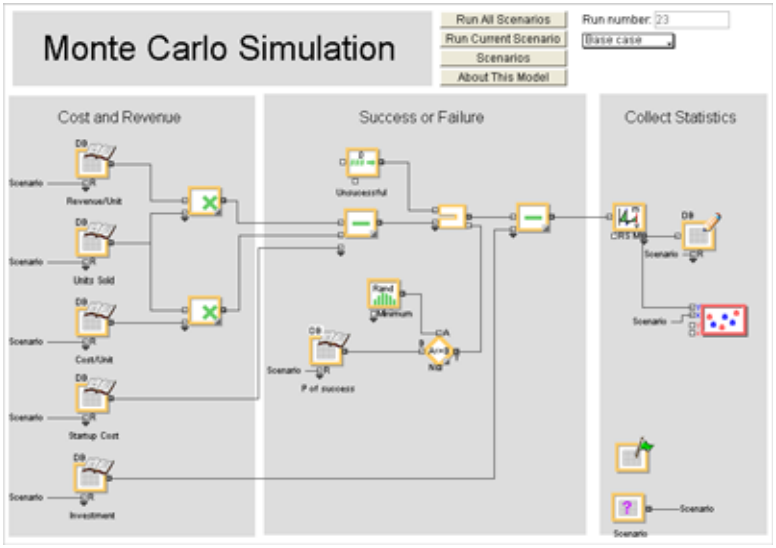
- The classical Monte Carlo method is to take a single mathematical equation or set of equations, then cause the equation to be calculated many times. In this type of simulation, time is not a factor. The entire model is run to completion and evaluated at each step; each subse-

quent step performs a new calculation. An example is the Monte Carlo model, discussed later in this section.

- An alternative Monte Carlo approach, typically applied in a discrete event model, is to either divide a single simulation run into multiple sections (*batch means*) or run the simulation many times (*multirun analysis*). Monte Carlo is incorporated by adding randomness to the model, running it many times, and analyzing the results. This method can be applied to any continuous, discrete event, or discrete rate model. It is shown in the Queue Statistics model, described later in this section. For more information about using the Statistics block (Report library) for performing batch means or multirun analysis, see “Statistics” on page 134.

Monte Carlo model

An example of the classical method is the Monte Carlo model. This model determines the expected revenue from a new product. It runs for 10,000 steps, from time 0 to time 9999, and each step results in an observation. This cycle is repeated 24 times, once for each of the cases. For scenario experimentation purposes, the inputs and outputs for this model are stored in an ExtendSim database.



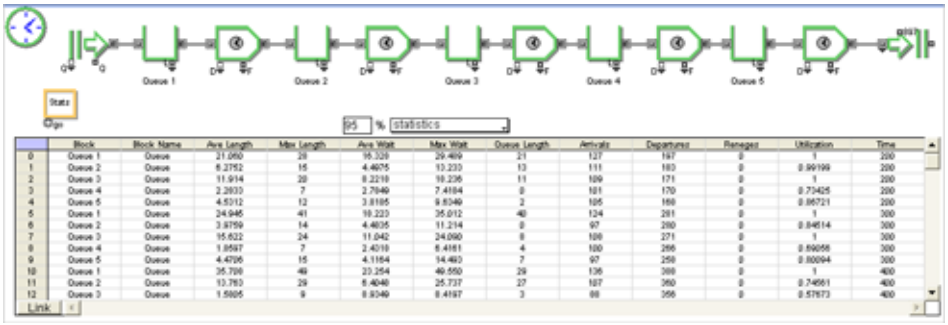
Monte Carlo model

☞ The Monte Carlo model is located at Documents/ExtendSim/Examples/Continuous/Standard Block Models.

Queue Statistics model

The Queue Statistics model is an example of an alternative Monte Carlo modeling approach. It applies batch means analysis to a discrete event model. The model uses the batch-means method to collect multiple observations of the queueing statistics. Every 100 time units a new set of observations are recorded. Information (such as the maximum and average queue

length, the number of arrivals and departures, and utilization) is stored and displayed in a table in the dialog of the Statistics block (Report library).



Queue Statistics model

The Queue Statistics model is located in the folder Documents/ExtendSim/Examples/Discrete Event/Statistics. It is not available with ExtendSim CP.

State/Action models

With *state/action* modeling a system is modeled as a collection of discrete states. Sometimes known as a state chart, a state/action model represents a system that responds to an event by transitioning to another state. The model is composed of a series of states where each state depends on a previous state. A state has an associated action and an event that will cause that state to change to another. The transition from one state to the next is not sequential; each state can lead to any other state.

There are rules that govern the communication and transition between the states:

- All states accept events.
- One or more states may create an event as a result of a transition by another state or group of states.
- A group of states can be set to transition conditionally, for instance to only change if another state or group of states achieve a specific stage. These are known as guard conditions.

State/action models are independent of any of the three modeling methodologies (continuous, discrete event, or discrete rate.) They are useful for specification and verification in many areas, from computer programs to business processes.

Creating a state/action model in ExtendSim

In ExtendSim, the three most common ways of creating state/action models are:

1) Define as an item. Define one or more discrete event items as objects with behavior that is determined by their states. The information about each state and its next state is stored in a Lookup Table block (Value library) or an ExtendSim database table. This uses ExtendSim's internal event queue and scheduling capabilities to signal and manage events for the item/objects within the system. This method can only be used with discrete event models and is illustrated in the State Action model later in this section.

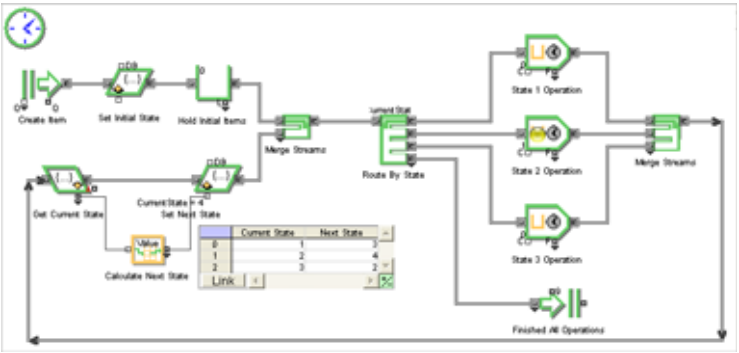
2) Define in a database table. Store each state, action, event, and next state for the system in records in an ExtendSim database table. This maps the states for the entire model into one

database and works with any type of model -continuous, discrete event, and discrete rate. It is illustrated in the “Markov Chain model” on page 17.

3) Program a block. Create new blocks that store their current state in a static variable and send messages to other blocks at appropriate state change events. To do this, use ExtendSim functions and its simulation development environment to create custom blocks and store them in a library. For more information about creating new blocks, see the Technical Reference.

State Action model

In the State Action model, items are created with attributes that determine the item’s state. The items are then routed to one of three operations depending on their state. After processing, the item’s state is changed based on entries in a Lookup Table block (Value library). The item continues to be routed to various processes until it reaches state 4, at which point it leaves the simulation.



State Action model

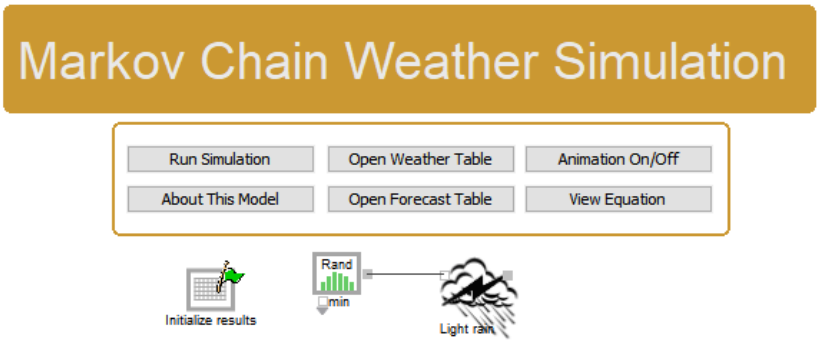
Initially, each item has a CurrentState attribute with a value of 1. The Lookup Table block causes each item with CurrentState 1 to be changed to CurrentState 3 after processing, then to CurrentState 2, and finally to CurrentState 4. The operations are represented by Workstation blocks, which can hold and process the items. After each operation, the item is examined and its state is transitioned accordingly.

Running the simulation with animation on shows the items changing from state 1 (green), to state 3 (red), then state 2 (yellow), and finally state 4 (blue).

The State Action model is located in the folder Documents/ExtendSim/Examples/Discrete Event/Routing, It is not available with ExtendSim CP.

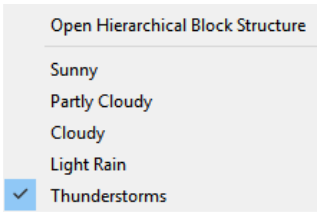
Markov Chain model

A Markov chain represents a transition from one state to another as defined by a table of probabilities. The Markov Chain Weather model simulates the weather based on a Markov chain. The states are the weather – sunny, cloudy, rainy, and so forth; they are stored in an ExtendSim database named “Weather”. The model runs for 365 days. Each day there is a probability of transitioning from one weather state to the next. For example, if today is sunny, the next day could be sunny (50%), partly cloudy (30%), cloudy (10%), light rain (5%), or rainy (5%). As the model runs, the states move through a probability table, changing the weather for each day.



Most of the calculation in this model is done by a single Equation block (Value library) inside the hierarchical "Weather Forecast" block. In the equation, a random input and the previous state (the output of the equation) are used to lookup a probability for the next day's weather in the Weather database. The number and percent of days at each weather state is also calculated and recorded in the database.

Additionally, if the model is run with animation on, the current weather state is animated on the icon of the Weather Forecast block. It does this by showing different icon views (Sunny, Cloudy, Thunderstorms, etc.) depending on the state; you can see the icon views by right-clicking on the icon. For information about icon views, see page 66.



☞ The Markov Chain Weather model is located in the folder ExtendSim/Examples/Continuous/Standard Block Models.

Agent-based models

Most of the models discussed in this document represent a system where the behavior of the system's components are known or can be estimated in advance. With agent-based modeling you usually do not know model dynamics in advance; instead, you obtain that information from the interaction of the agents in the model.

Agent-based models share the following characteristics:

- The identification of individual entities within the model
- A set of rules that govern individual behavior
- The premise that local entities affect each other's behavior

Agent-based modeling is concerned with individual entities (called "agents") that interact with other agents within their specified locality. All the agents have a set of rules to follow but they also have a degree of autonomy such that model dynamics cannot be predefined. This is because agents can have intelligence, memory, social interaction, contextual and spatial awareness, and the ability to learn.

Creating agent-based models

The agents used in agent-based modeling are programmed as ExtendSim blocks. Blocks and their enclosed data have unique searchable identities and locations within the model. The ExtendSim database and ExtendSim functions can find and send messages to blocks that have specific characteristics, locations, and values. This makes it easy to create intelligent behavior,

facilitate block-to-block interaction, and cause blocks to be moved in, added to, or removed from, a model.

A good method for communicating with agents is through the ExtendSim database. This can store all the data for the model and agents can post changes to the database and/or react through links to database changes.

A separate document—the Technical Reference—lists several categories of functions that are helpful when creating agents for agent-based modeling:

- Scripting functions are used to build a new model or to add or remove blocks from an existing model. They do this by creating, placing, and connecting blocks, then populating the blocks with specific data. These functions can be called from an ExtendSim block within the model or from an external application.
- Block and inter-block communication functions query the status of a block—its type, label, data, location, size, and connectivity with the rest of the model. They also get information about block dialog values and data table settings.
- Message sending functions can use the results of inter-block communications to send messages globally to unconnected blocks or blocks that are connected in specific ways.
- Database and linking functions can link the blocks to data in a database and dynamically react to any change made to the database.
- Animation functions provide a visual indication of block-to-block interaction, such as the influence of one block on another.

For example, in constructing an agent-based model of the robotic clean up of a chemical spill, you could use the inter-block communication functions in a "Controller" block to locate all of the "Robotic Clean Up" blocks in the model. The Controller could send messages to the robots asking them to move towards a spill and clean it up. The robots could send messages back to the Controller stating whether they were available or were currently being recharged, and whether they were too far from a chemical spill or close enough to be useful. The scripting and animation functions would show the robot blocks physically moving around within the model and the spill being removed.

The Game of Life

The Game of Life was devised by British mathematician John Conway in 1970 and published as an article in Scientific American. It is the most well known example of cellular automata (CA), a type of modeling studied in computability theory, mathematics, theoretical biology, and other fields.

A CA model represents a regular grid of finite state automata (cells) that sit in positional relationships to one another, with each cell exchanging information with the eight other cells to which it is horizontally, vertically or diagonally adjacent. A cell can be in one of a finite number of states and the state of a cell at time *t* is a function of the states of its neighboring cells at time *t-1*. Every cell has the same rule for updating; each time the rules are applied to the whole grid a new generation of cells is produced.

You interact with the Game of Life by specifying an initial configuration of effects and observing how the CA universe evolves. At each step in time, the following happens:

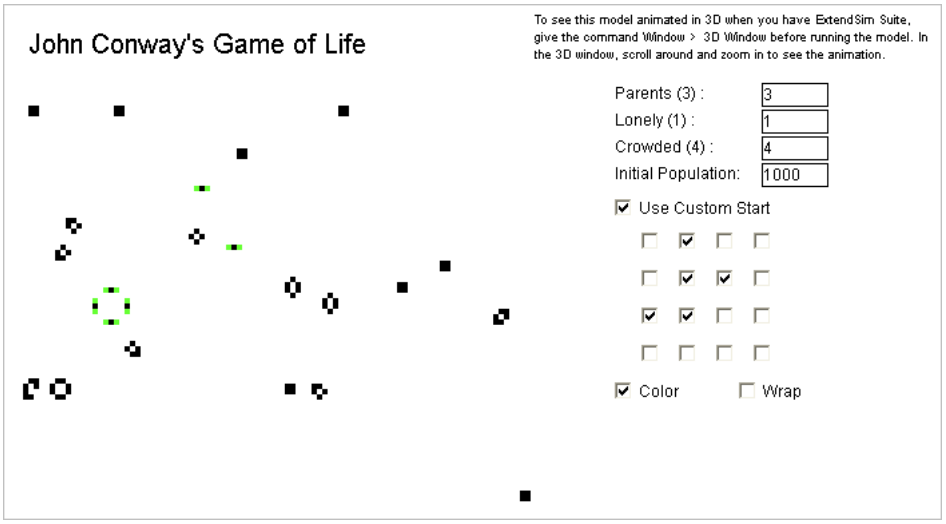
- A cell is born if it has a specified number of neighbors who act as parents.
- "Loneliness" causes any live cell with fewer than a specified number of neighbors to die.

- "Overcrowding" causes any live cell with more than a specified number of neighbors to die.

The initial pattern constitutes the first generation of the system. The second generation is created by applying the above rules simultaneously to every cell in the first generation. In other words, births and deaths happen simultaneously. The rules continue to be applied repeatedly to create further generations.

Life has a number of recognized patterns that emerge from particular starting positions, including static patterns ("still lifes" such as block and boat), repeating patterns ("oscillators" such as blinker and toad), and patterns that translate themselves across the board ("spaceships" such as gliders).

The Life model
The one-block Life model was created using the Life block (Custom Blocks library) that was specifically developed for this model. The code of the block contains the algorithm for Conway's Game of Life. The block's dialog has fields for specifying initial settings and rules; the dialog items have been cloned to the model worksheet for convenience.



Life model

The concept for this model is that each cell of the grid is defined as living or empty. On each generation, a given cell can give birth to a new life, survive, die, or remain empty. Using the default settings in the Life block, the model adheres to the following rules:

- Count the number of neighbors a given cell has (the maximum possible is 8).
- If an empty cell has 3 neighbors, it will produce a new life (birth).
- If a full cell has less then 1 or zero (loneliness), or 4 or more neighbors (overcrowding), it will die.

Changing the default rule values causes some interesting affects on the population.

There are two ways to set the starting population for the model:

- Define an initial number of cells (1000 is a reasonable starting population for the size of this block.) The cells will be populated randomly.

- Use the Custom Start grid to select up to 16 initially populated cells in specific locations. This is a quick way to begin with a recognized pattern, such as a glider or a blinker.

One feature of the Life block that is not specified in Conway's algorithm is that the color of the cells varies with the age of the cell - new cells are green and older cells vary from light gray to black as they age.

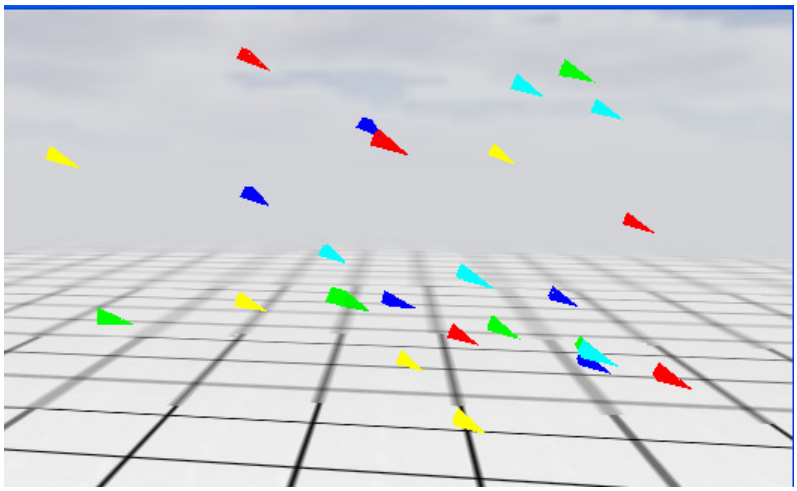
Variations
The Life block is open source so you have complete access to the dialog editor and block code. To see the underlying structure of the block, select it on the model worksheet and give the command Develop > Open Block Structure. The procedures that define cell birth, death, or survival are listed at the top of the block's structure window.

☞ The Life model is located in the folder Documents/ExtendSim/Examples?Agent Based. The Life block is located in the Libraries/Example Libraries/Custom Blocks library.

Boids

The Boids model is based on an artificial life program, developed by Craig Reynolds in 1986, that simulated the flocking behavior of birds. It can also be applied to schools of fish, herds of animals, or any other type of flocking behavior.

⚠ The Boids model requires 3D animation and is only available in release 9 of the ExtendSim Suite product. For release 9, the model is located in the folder \Examples\3D Animation.



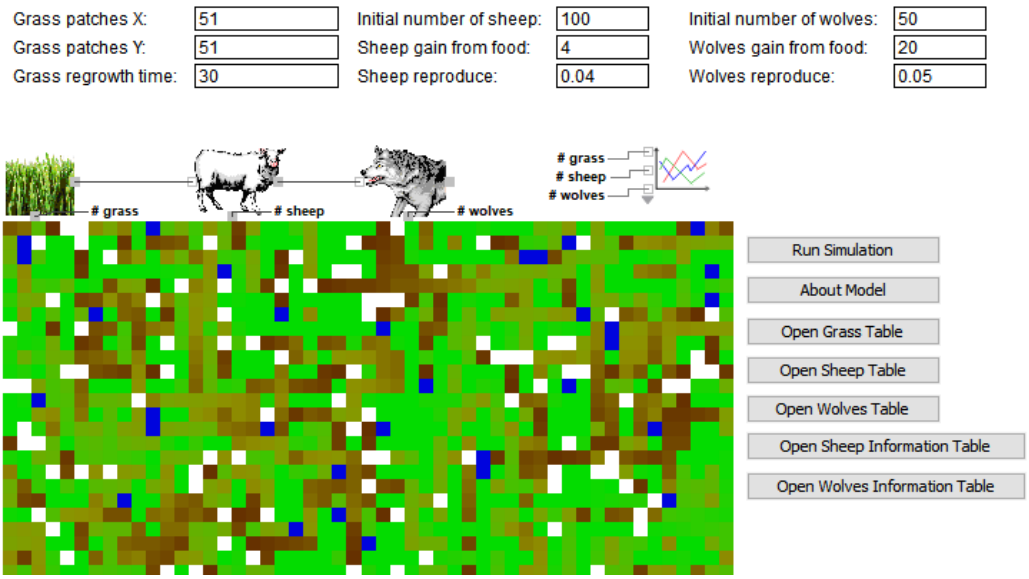
Boids model in E3D window

In the model, each bird is an individual agent that interacts with other local agents based on a set of rules:

- Separation – birds steer to avoid crowding their local flock mates.
- Alignment – each bird steers towards the average heading of its local flock mates.
- Cohesion – birds steer toward the average position of their local flock mates.

Other agent-based models

Additional agent-based models, including “Sheep and Wolves Agents” (shown below) and “Breakout” are located in the folder \Examples\Agent Based.


Grass patches X: 51
Grass patches Y: 51
Grass regrowth time: 30
Initial number of sheep: 100
Sheep gain from food: 4
Sheep reproduce: 0.04
Initial number of wolves: 50
Wolves gain from food: 20
Wolves reproduce: 0.05
grass
sheep
wolves
grass
sheep
wolves
Run Simulation
About Model
Open Grass Table
Open Sheep Table
Open Wolves Table
Open Sheep Information Table
Open Wolves Information Table

Simulation

There’s a Process To It

There’s a right way and then there are all the other ways.

“It must be remembered that there is nothing more difficult to plan, more doubtful of success, nor more dangerous to manage, than the creation of a new system.”
— Niccolo Machiavelli

The modeling process

A simulation project involves creating a logical model of the system, running the simulation, analyzing the data, optimizing the solutions, and interpreting and presenting the results.

Goals of modeling

As stated in *Modeling Tools for Environmental Engineers and Scientists* (N. Nirmalakhandan, CRC Press), the "...goals and objectives of modeling are two-fold: research oriented and management oriented. Specific goals of modeling efforts can be one or more of the following: to interpret the system, analyze its behavior, manage, operate or control it to achieve desired outcomes; to design methods to improve or modify it, to test hypotheses about the system, or to forecast its response under varying conditions."

The simulation process

Like all tasks, you can start modeling simply by jumping into it. Also like all other tasks, it is usually better to have a plan before starting. The basic steps to creating a model are:

1) *Formulate the problem*. Define the problem and state the objectives of the model. For more about this step, see "Before you build a model" on page 25.

2) *Describe the flow of information*. Determine where information flows from one part of the model to the next and which parts need information.

3) *Acquire input data*. Identify, specify, and collect the data you need for the model. This is usually the most time-consuming step. It includes finding not only numerical data values but also mathematical formulas such as distributions for random events. In the absence of real data, distributions can be used to mimic the patterns that would be expected given the actual data.

4) *Build and test the model*. Start small, test as you build, and enhance as needed. See also "Refining models" on page 25.

5) *Determine where and how to store and manage output data*. This includes storing data in an internal database, representing data in charts and graphs, and exporting to other applications.

6) *Run the model*. Determine how long you want to simulate and the granularity of results, then run your model. NOTE: since most models contain statistical distributions, the models should be run enough times to generate statistically reliable results.

7) *Verify the simulation results*. Compare the model results to what you intended and expected. See "Model verification" on page 26 for more information.

8) *Validate the model*. Compare the model to the real system, if available. Or have system and subject matter experts evaluate the model and its results. See "Model validation" on page 26 for more information.

9) *Analyze your results*. Draw inferences from the model's results and make recommendations on how the system can change.

10) *Conduct experiments*. Implement and test recommended changes in the model.

11) *Document*. State the model's purpose, assumptions, techniques, modeling approaches, data requirements, and results.

12) *Implement your decisions*. Use the results in the real world.

Before you build a model

Remember that building a model is an iterative process and each step in the process will require comparing the model to the existing system, analyzing the results, and refining the model. A natural inclination is to immediately start building the model. However, you will end up with more useful models if you begin the model-building process by asking a few basic questions, such as:

- *What is the goal of the model?* It is important to determine the purpose of a model. This will indicate the levels of detail required and will help keep you focused.
- *What are the boundaries of the model and what level of detail should be included?* The model goal should dictate what to include in the model and what to leave out.
- *Where is the required data?* It is useful to start collecting data early in the model building process because it can often take a while to obtain all of the necessary information. You will also need to know if the input data is an absolute value or if the data is from a statistical distribution. Additional data requirements may surface once the model building process has begun. Your model may, for example, lead you to explore alternatives that had not been considered before.
- *How shall the model be conceptualized?* Before even launching ExtendSim, think about what the various components of the system represent. Roughly determine the time delays, resource constraints, flows through the system, and any logical actions that occur in the model. This will help you determine how to build the model.
- *What alternatives will be investigated?* Although the model may lead you into new, unexpected directions, try to think ahead so that the model can be easily changed from one alternative to the next.

☞ It is common to use a constant or a uniform (integer or real) distribution in the early stages of model building so that modeling problems and variations can be more easily detected. After the model is verified, you can easily change the distributions to correspond to real-world processes.

Refining models

It is important to remember that models may not give you a single "correct" answer. Instead, they make you more aware of gaps in your thought process. These gaps may originate from over-simplification in the model, false assumptions when creating the model, or missing connections between parts of a model. Refining your model step by step helps eliminate these and other pitfalls.

Every model can be made more complex by adding assumptions and interconnections. The model-building process commonly begins with the creation of a simple model. After analyzing the simple model, complexity is added, followed by further analysis, the addition of more complexity, and so on. The complexity takes one of two forms:

- Taking one block (a process) and turning it into many blocks (a more complex process)
- Adding a connection between two previously unrelated blocks, usually through a mathematical operation (finding an interconnection between two processes)

At each step, look at your results and make sure they make sense relative to the data. If you can, verify the results in the real world. If one result is way off, check the output from each step to determine where the process went awry.

Model verification and validation

These are terms you're going to hear a lot, and for good reason! Gathering information and building a model are fine, but it's important that you make sure the model you've created represents the system and works as you intend it to.

Model verification

The process of debugging a model to ensure that every portion operates as expected is called model verification. One way to accomplish part of this verification process is to build the model in stages and with minimal detail, then run it at each stage to observe the results.

Another common verification technique could be termed *reductio-ad-absurdum* (reducing to the absurd), which means reducing a complex model, or a part of a model, to an aggressively simple case so that you can easily predict what the outcome will be. Some examples of "reducing to the absurd" are:

- Remove all variability from the model, making it deterministic
- Run the deterministic model twice to make sure you get the same results
- Output detailed reports or traces to see if the results meet your expectations
- Run a schedule of only one product line as opposed to several
- Reduce the number of workers to 1 or 0 to see what happens
- Uncouple parts of the model that interact to see how they run on their own
- Run very few or very many items through the model to determine if the model responds properly and that all items are properly accounted for.

☞ ExtendSim ships with tools for model verification. See "Automated test environment (Windows only)" on page 217.

Model validation

Once the model is verified you need to validate it to determine that it accurately represents the real system. Notice that this does not mean that the model should conform to the real system in every respect. Instead, a valid model is a reasonably accurate representation based on the model's intended purpose. When validating, it is important to make sure that you know what to compare to and that you verify that measures are calculated in the same manner.

For validation, your model should accurately represent the data that was gathered and the assumptions that were made regarding how the system operates. In addition, the underlying structure of the model should correspond to the actual system and the output statistics should appear reasonable. While you would normally compare critical performance measures, it is also sometimes helpful to compare nonessential results that may be symptomatic and therefore show the character of the system.

One of the best validation measures is "Does the model make sense?" Other methods involve obtaining approval of the results by those familiar with the actual process (those "working in the trenches" as well as subject matter experts) and comparing simulation results with historical data. For example, when validating model performance compared to historical data, try to simulate the past. If you have sufficient historical data, break the actual system performance into various windows of time, where all of the input conditions correspond to the input conditions for multiple runs of your model.

☞ Also see the chapter "Debugging Tools" on page 205 and especially the model verification techniques discussed in "Automated test environment (Windows only)" on page 217.

About ExtendSim

Capabilities, Products, and Licensing

Your questions have answers.

ExtendSim is a powerful, leading edge simulation tool. Using ExtendSim, you can develop dynamic models of real-life or proposed processes in a wide variety of fields. Use ExtendSim to create models from building blocks, explore the processes involved, and see how they relate. Then change assumptions to arrive at an optimum solution. ExtendSim and your imagination are all you need to create professional models that meet your business, industrial, and academic needs.

What ExtendSim can do

ExtendSim allows you to simulate any system or process by creating a logical representation in an easy-to-use format.

Modeling capabilities

With ExtendSim, you get powerful modeling capabilities, including:

- **Multi-purpose simulation**. ExtendSim is a multi-domain environment so you can dynamically model continuous, discrete event, discrete rate, agent-based, linear, non-linear, and mixed-mode systems.
- A **complete simulation framework** with graphical model building, smart connections, internal relational data management, multiple notebooks, and unlimited hierarchical layers.
- A **state-of-the-art UI** with docking palettes, tear-off tabs in dialogs and worksheets, multiple undo, object grouping, and scaled text. Create **custom front end interfaces** for your models with buttons, check boxes, hyperlinks, cloned dialog items, pop-up menus, and more.
- **Libraries of pre-built components** for continuous process, discrete event, discrete rate, and reliability modeling plus **equation editors** for compiled logical statements and mini-programs that access 35 distributions and more than a thousand internal functions and procedures.
- **Data and process visualization** as the model runs, with on-screen results, graphs, reports, and integrated animation.
- **Analysis power** including an Evolutionary Optimizer, Scenario Analysis with export to JMP and Minitab, Sensitivity Analysis, distribution-fitting with Stat::Fit, Gantt charts, quantile and interval statistical analysis, warm-up periods with statistical clearing, and confidence intervals.
- **Robust data connectivity** before, during, and after the run between ExtendSim and external spreadsheets, databases, and analysis tools with import/export, COM, Automation, ADO, and more. You can even interactively link block dialog parameters to internal and external databases.

Simulation architecture

A robust architecture adds advanced features to make ExtendSim the most scalable simulation system available:

- An integrated development environment (IDE) with a **compiled language that is optimized for simulation**. Use **equation blocks** to create logical statements or mini programs by accessing the ExtendSim IDE and thousands of internal functions and procedures. Or use the ExtendSim API, dialog editor, and programming tools to **modify or create new blocks** with custom code, dialogs, and icons. Take advantage of programming tools such as include files,

extensions, conditional compilation, a source code debugger, code completion, and external source code capabilities. Save custom blocks in libraries for reuse in other models.

- **Scripting support**. Build and run models remotely, either from an ExtendSim block or from an outside application.
- **Integrated support for other programming languages**. Use ExtendSim's built-in APIs to access code created in C++, Visual Basic, C#, etc.
- **Thousands of functions**. Directly access functions for integration, statistics, queueing, animation, IEEE math, matrix, sounds, arrays, FFT, debugging, ADO, COM, Automation, ActiveX, string and bit manipulation, I/O, and so on; you can also define your own functions.
- **Message sending**. Blocks can send messages to other blocks interactively for subprocessing.
- **Sophisticated data-passing capabilities**. Pass values, arrays, or structures composed of arrays.
- **Full support for a wide range of data types and structures**. Arrays, linked-lists, and integers, real, and string data types are built in.

Levels of use

You can use ExtendSim on many levels:

- Run pre-assembled models and explore alternatives by changing the data. If you work in a group environment, one or more authors can create models for others to run for experimentation. The author can also build a custom user interface to facilitate user interaction with the model. Model developers can distribute ExtendSim models to colleagues who don't own ExtendSim.

- Create your own models from the blocks that come with ExtendSim. ExtendSim includes libraries of blocks to handle most modeling needs. To build a model, pull blocks from libraries and link connectors on the blocks. You can also use pre-built templates (Templates library) or assemble your own hierarchical blocks of subsystems and save them in libraries. This saves starting from scratch when you're building a model of a process that has elements in common with a previous model.

- Use the integrated development environment to create new blocks that conform to the ExtendSim modeling architecture. The development environment is optimized for simulation and allows you to create blocks with custom code, dialogs, and icons, then use them in your models just as you would other ExtendSim blocks. You can also modify the blocks that come with ExtendSim to work with your specific needs.

- Develop your own modeling architecture, conventions, and features. With the ExtendSim development environment, you can create a custom set of blocks with unique interfaces, communication protocols, and behaviors. This new architecture can be continuous, discrete event, discrete rate, agent-based, or an entirely new type of simulation.

- Automate your model building using the scripting functions to build wizards, or by using COM. You can use COM or block-based wizards to cause models to be automatically created or modified. Models can also be programmatically created from a user input form or

data file. This allows the modeling environment to be utilized indirectly by end-users who have little or no simulation experience.

While ExtendSim models can be freely distributed without license or cost, an ExtendSim Cloud license is required to provide Internet or intranet access to ExtendSim or to its functionality. Contact Andritz Inc. for more information.

ExtendSim products

Simulation capabilities

ExtendSim is a multi-domain environment, so you can dynamically model the continuous, discrete event, discrete rate, agent-based, Monte Carlo, state/action, linear, non-linear, and mixed-mode systems discussed on page 10. ExtendSim also includes features and components for animating discrete event processes and systems.

Core features in ExtendSim products

ExtendSim is the family name for a line of powerful simulation products and components. Adopted throughout the world by corporations, academia, and governmental entities, ExtendSim is used to simulate processes and systems in business, engineering, and science.

All ExtendSim products contain a **core set of features**, including:

+ **Pre-built blocks** for data access (read/write/import/export), math, equations and logical expressions, optimization, random number generation, scenario analysis, routing, statistics, charting, and output reports. (Additional libraries of blocks are included, depending on the product purchased.)
+ An **equation editor** for specifying the simplest to the most complex of logical statements, writing compound conditions, and even full programming segments. Use the same ExtendSim tools that programmers would use to create custom blocks.
+ A **Chart** library with blocks for graphically displaying outputs
+ A **Report** library with a Reports Manager, Statistics, and other blocks for output analysis.
+ **Hierarchy** for unlimited layers of submodels with customizable icons. Store hierarchical blocks in libraries for reuse in other models.
+ **Cloning of dialog items**. Copy clones of parameter values to the model worksheet or to notebooks for quick access to inputs and outputs.
+ An authoring environment for creating **custom user interfaces** and dashboards, including buttons, popups, check boxes etc.
+ **Interactive** simulation runs - change parameters while the simulation is running.
+ Organize and manage data with **internal relational databases**, plus an Excel Add-In for transferring data between Excel and an ExtendSim database.
+ Find an optimal solution using the **Evolutionary Optimizer** that can be added to any model.
+ Run experiments and perform analysis using Sensitivity Analysis
+ Systemically and strategically examine the outcome of different model configurations using the **Scenario Manager** with export to JMP and MiniTab.

+ Data connectivity (**ADO database support, COM, etc**) with Excel and external databases such as Microsoft Access .
+ **Activity-based costing** for financial outcomes.
+ Integrated and customizable **animation**.
+ The ExtendSim IDE, described on page 30.
+ Extensive documentation with example models showing how to use every aspect of ExtendSim and how to build professional models in every field.
+ And much more!

Model Developer Editions

The following products are used to create. run, and use models. They are available as Individual, Node-Locked, or Floating licenses.

☞ See "Licensing options" on page 36 for information about license types.

1) ExtendSim CP

- For simulating continuous, time-based processes, typically in the fields of business, science, or engineering. Also used for creating custom libraries of blocks for specific market segments.
- Build models using the pre-built blocks or create libraries of custom blocks using the ExtendSim integrated development environment (IDE).
- ExtendSim CP is used to model control systems, environmental impact analysis, pulp and paper processes, weather pattern prediction, electronic systems, waste treatment, hydrologic systems, and more.
- ExtendSim CP includes all the core ExtendSim features mentioned on page 32
- For more information, see the **Continuous Process Modeling QSG** (Quick Start Guide) **located at Documents/ExtendSim/Documentation**.

2) ExtendSim DE

- Robust discrete event capabilities for modeling any system where time advances when events occur and tracking of model entities is important.
- ExtendSim DE is used to model production lines, supply chains, call centers, capacity planning projects, emergency departments, logistics, service performance, transportation systems, and much more.
- ExtendSim DE includes:
  - All the core ExtendSim features discussed on page 32.
  - All the capabilities of ExtendSim CP discussed above.
  - An Item library with pre-built blocks for:

+ Quickly building discrete event models: item generation, queues, activities and delays, batching, routing, shutdown and shift, and more.
+ Specifying item properties and attributes, warm-up periods, item logging, history, etc.

o Advanced Resource Management (ARM)—a comprehensive system for systematically dealing with multiple types of resources and complex resource requirements in simulation models.

o Query Equation block—intelligently ranks the records in an ExtendSim database table and, based on a ranking rule that you set, selects one record.

- For more information, see the **Discrete Event Simulation QSG** (Quick Start Guide) **located at Documents/ExtendSim/Documentation**.

3) **ExtendSim Pro**

- This package offers advanced technology that goes beyond the typical continuous or discrete event modeling capabilities of other simulation applications.

- ExtendSim Pro includes:
  - o All the core ExtendSim features listed on page 32
  - o All the capabilities of ExtendSim CP on page 33
  - o All the capabilities of ExtendSim DE (above)
  - **o Rate module**
    + Model any system or process that involves tanks, levels, and valves and the storage and rate-based movement of system components.
    + Simulate rate-based flows: data feeds and streams; liquids, minerals, and gases; food and beverage, pharmaceuticals, paper: and any rate-based movement of otherwise distinct entities where uniquely identifying individual entities is either unneeded or unwanted.
    + Model high volume, high speed, or bulk processes: oil and gas production, refining or mining operations, power systems and grids, petrochemical and chemical processing, high speed packaging and production lines, water treatment plants, and more.
    + Combine with discrete event and reliability modeling to represent entire systems and supply chains.
    + Merge and diverge flow using rule-based options such as distributional, priority, and proportional. Customize flow attributes to provide information about the flow and organize quantities or volumes of flow into layers.
    + Set indicators (set points) to report when a tank's flow level is within a specified range. Convey Flow block has multiple modes (accumulating, non-accumulating) and sensors for communicating the density of flow over time at points along the conveyor.

    + For tutorials showing how to model using the Rate module, see the **Discrete Rate QSG** (Quick Start Guide) **located at Documents/ExtendSim/Documentation**.
  - **Reliability module**
    + Adds a Reliability Block Diagramming (RBD) capability to ExtendSim.
    + Graphically and statistically describe when scheduled downs (maintenance and off-shifting) and unscheduled downs (failures) occur for individual resources and determine what impact availability has on the entire system.
    + Use as a standalone RBD tool to determine resource and component availability. Capture complex availability behavior of both individual resources and entire systems.
    + Or, use with ExtendSim event-based process simulation capabilities to explore the impact of resource availability on key process metrics such as throughput, production costs, repair costs, utilization, inventory, service levels, and so forth.
    + To learn how to use the Reliability module, see the document named **Reliability** at Documents/ExtendSim/Documentation.
    + Includes up to 100 Event Cycles per model. Additional Event Cycles can be purchased separately.

RunTime and specialized products

In addition to the Model Developer Editions above, other ExtendSim products include:

- ExtendSim **Analysis RunTime**
  - A cost-effective way for you or others to access ExtendSim analysis functionality without needing to purchase additional model-developer licenses.
  - Distribute the Analysis RunTime along with your models as part of the response to a Request for Proposal, showcase your consulting capabilities to potential clients, or provide models to your company's sales staff so they can show customers the outcomes of various equipment or service options.
  - Also useful for off-loading intensive analysis runs, such as Scenario Analysis or Optimization, to other computers. With programming, you can also remotely cause the model to run and transfer data between computers.
  - Model files and libraries must have been developed in a Model Developer Edition of ExtendSim.
- ExtendSim **Player RunTime**
  - A free method for distributing your models to end users who only need to run models (not save changes or perform analysis).
  - Your colleagues can view and interactively run a simulation model you've developed – change dialog parameters and settings, see animation, view results, and more
  - No limit on the number of blocks or size of model

- Limitations: models and blocks cannot be built or saved in the Player RunTime; runs are single-run only and cannot be done remotely; import/export of data is not allowed; and Stat::Fit, Optimization, and Scenario Manager are not included.
- Model files must have been developed in a Model Developer Edition of ExtendSim
- Available as a free download only from www.ExtendSim.com

- ExtendSim **Cloud**
  - Use this self-hosted cloud-based ExtendSim product to provide ExtendSim models in the cloud for use by your internal or external end-user "clients". Cloud deployment allows a company's clients, such as its employees or customers, to have remote access to ExtendSim simulations without installing the ExtendSim application on their devices.
  - Once a company has set up the ExtendSim Cloud framework on a server, its clients submit credentials to get access to ExtendSim models. Using a customized GUI front end (a web page in a browser or a standalone application) on their devices, the clients can send inputs through the cloud to the models, run the simulations, and receive the results on their computer, tablet, or smart phone.
  - The Cloud architecture has been implemented in a framework that consists of a back-end which responds to requests for Cloud services and a licensee-customized frontend from which these requests are made.
  - The ExtendSim Cloud product has the advantages of remote access via the web, centralized feature updating, and lower or even non-existent learning curves for non-modelers.
- ExtendSim **OEM**
  - For system builders who want to embed the simulation power of ExtendSim within their own applications
  - Provides the customized right to use and/or sell your product bundled with ExtendSim or with ExtendSim technologies under your brand name
  - Contact Andritz Inc. for more information

There are also ExtendSim licenses at reduced pricing for educators and students at accredited universities. For more information about ExtendSim products, or about licenses for academic use, go to www.ExtendSim.com.

☞ Individual licenses are available from the ExtendSim website and store. For Node-Locked or Floating licenses, contact Imagine That Inc or your local distributor.

Licensing options

Most ExtendSim products are available as Individual, Node-Locked, or Floating licenses.

Individual licenses for a single user

- Licensed for use by one Authorized User.
- Activation of ExtendSim is required.
- An annual Maintenance Plan for technical support and upgrades is required. When purchasing a new license, the first year's maintenance and support is included in the purchase price.

Maintenance must be renewed annually after year one to avoid triggering our Lapsed Maintenance Policy.

Node-Locked licenses for sequential users

- Licensed for installation on a single computer for use by multiple individuals in succession.
- Activation of ExtendSim is required and is tied to the device's ID.
- An annual Maintenance Plan for technical support and upgrades is required. When purchasing a new license, the first year's maintenance and support is included in the purchase price. Maintenance must be renewed annually after year one to avoid triggering our Lapsed Maintenance Policy.

Floating licenses for concurrent users

- Licensed for installation on multiple computers for use by multiple individuals simultaneously.
- Limited to a maximum number of concurrent users and, unless a global license is purchased, limited to use within a single continent.
- License Management software (included) must be installed on a central server to monitor and control usage. Activation of the License Management Software is required.
- Licenses may be checked out (Roam) for off-network use.
- An annual Maintenance Plan for technical support and upgrades is required. When purchasing a new license, the first year's maintenance and support is included in the purchase price. Maintenance must be renewed annually after year one to avoid triggering our Lapsed Maintenance Policy.

☞ For information about the annual Maintenance Plan, see "Technical support" on page 41.

About ExtendSim

Support: Documentation, Training, and Upgrades

Just in case you need some help.

ExtendSim written documentation

The documentation for ExtendSim is separated into several electronic books.

Quick Start Guides (QSG's)

The guides are focused on specific simulation methodologies and associated technologies—*Continuous Process Modeling*, *Discrete Event Simulation*, and *Discrete Rate Simulation*. Each guide describes the methodology or technology, provides examples of use, and has short tutorials, terminology explanations, and more. Different guides are installed in the Documents/ExtendSim/Documentation folder depending on the ExtendSim product purchased and the type of models that can be built with that package.

Tutorial/Reference Books

The ExtendSim Tutorial & Reference books are eBooks describing specific ExtendSim features, such as:

- The ExtendSim internal relational database
- The Reliability module for reliability block diagramming (RBD)
- ExtendSim Advanced Resource Management (ARM) capability

These eBooks explain the feature and contain tutorials showing how to use it, supported by example models. Like the Quick Start Guides, these eBooks are installed in the Documents/ExtendSim/Documentation folder.

User Reference

A general purpose manual for using ExtendSim; it is divided into five sections:

1) *Simulation*—overview, process, methodologies.
2) *About ExtendSim*—capabilities, products, licensing, support, and more
3) *How To*—a general purpose section showing how to build, run, and use ExtendSim models, create a model dashboard, import and export data, and much more.
4) *Reference* sections for:
   - Discrete Event Modeling (see also the Discrete Event Quick Start Guide)
   - Discrete Rate Modeling (see also the Discrete Rate Quick Start Guide)
5) *Appendices* describing the menu commands and the blocks in the ExtendSim libraries.

Technical Reference

This is helpful for two types of ExtendSim users:

- Modelers who use the equation-based blocks
- Programmers who want to create custom blocks or effects for specific purposes

The Technical Reference describes the integrated development environment (IDE). It lists all the ExtendSim functions and message handlers and shows how to use programming tools such as include files, conditional compilation, external source code control, and the source code debugger.

☞ All documentation is located in the Documents/ExtendSim/Documentation folder. They may also be accessed by going to the ExtendSim > Help command or through the Quick Start interface that launches when you launch ExtendSim.

Videos and additional resources

Andritz Inc. provides several resources to support your simulation experience.

Quick Start interface

The Quick Start interface (Getting Started model) opens by default when you launch ExtendSim. Use this interface to learn about ExtendSim, explore sample models, view tutorials, and more.

Videos

Each of the Quick Start Guides has a corresponding set of videos that get installed with your ExtendSim product. There are additional video tutorials, techniques, Master Classes, and more at the ExtendSim Video Channel (https://www.youtube.com/user/ExtendSim).

Online help

- Access the electronic manuals by selecting the menu command or press F1 on your keyboard.
- Use tooltips to identify interface elements; see the Edit > Options > Model tab for settings.
- Get the complete definition of how a block works, including descriptions of its dialog items and connectors, by clicking the Help button in a block's dialog.

Web Advice

FAQs are available at www.ExtendSim.com/support/advice/faq.html.

Networking

Links for ExtendSim user forums, networks, webinars, and blogs are at www.ExtendSim.com.

Technical support

An annual Maintenance Plan for technical support and upgrades is required for most products. When purchasing a new license, maintenance and support is included in the purchase price for the first year; it must be renewed annually after year one to avoid triggering our Lapsed Maintenance Policy.

If the Maintenance and Support Plan is current, support includes:

- Technical assistance for installation issues, basic usage questions, and troubleshooting.
- Updates and upgrades released during the Maintenance period.


You must be a registered user to receive technical support. Support for models used in limited editions of ExtendSim is supplied by other parties—for the Student version, see your professor; for the Analysis RunTime version, see the model developer.

Contacting Technical Support


Support is complimentary for the first year after purchase. After that, you must either subscribe to the ExtendSim Maintenance Plan or purchase per-incident support.

To contact our support representatives, go to our website at www.ExtendSim.com, click the Contact Us link, and fill out a support ticket.


Be sure you are using the current version of ExtendSim. Software updates are available at our web site (www.extendsim.com). Upgrades to newer versions may be purchased separately if your license is not covered by a support plan.

An ExtendSim *upgrade* is a newer, more fully-featured version of your current software. Upgrades are denoted by a change in release number; for example, ExtendSim 10 is an upgrade from ExtendSim 9. See http://www.ExtendSim.com/Products/Purchase/Upgrades.html for more information.

An ExtendSim *update* improves the quality and stability of a major release. Updates tend to be mostly bug fixes but often also include new features. An update will append a release number with ".x". During your maintenance period, updates are made available by download from our website at www.ExtendSim.com/Support/Downloads/Updates.html.

Authorized ExtendSim trainers provide training several times a year as public courses at various locations; training customized to your needs is also available at your site. See http://www.ExtendSim.com/Support/Training.html for more information.

How To

The Big Picture

Overview

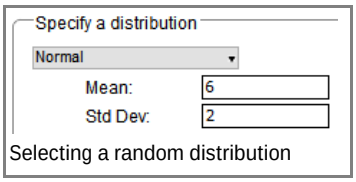
This chapter summarizes most of the information in the “How To” chapters that follow.

⚠ Depending on the ExtendSim product you have, some features, libraries, and blocks might not be available.

Model-building to represent the dynamics of the system

Use these features and capabilities to build a model that represents the dynamic behavior of any system.

- Quick model building
  - Use pre-built modeling components from the libraries shipped with ExtendSim to represent system behavior. For example, place a Queue block (Item library) on the worksheet, or select one of 35 built-in distributions using the Random Number block (Value library).



Selecting a random distribution

  - See the list of libraries and blocks in the appendices that start on page 575.
- Equation blocks
  - The Equation block (Value library) directly accesses the ExtendSim API, has its own code editor, debugger, and compiler, and supports include directives.
  - For more information, see “Equation-based blocks” on page 189.

```
real reservoirDepth;          // define a n
reservoirDepth = 50.0;     // set its valu

if (contents >= reservoirDepth)     // if
    overflow = contents - reservoirDepth;
else
    overflow = 0.0;     // If the contents
```

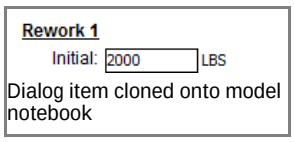
Code in Equation block

- Hierarchy
  - Encapsulate sections of a model into hierarchical blocks for scalability. Create generic model structures and store in libraries for reusability. Customize and animate the icons.
  - For more information, see “Hierarchy” on page 78.
- C-based IDE
  - Since the Item and Rate libraries are so extensive and complete, it is unlikely that you will need to build a custom discrete event or discrete rate block. However, especially if you do continuous process modeling, you may want to build a custom value-type block using the ExtendSim integrated development environment (IDE).
  - See the Technical Reference located at Documents/ExtendSim/Documentation.

Creating a user interface for the model

Use these ExtendSim features to create a custom graphical user interface, control panel, or dashboard for the model, simplifying and localizing data entry and displays.

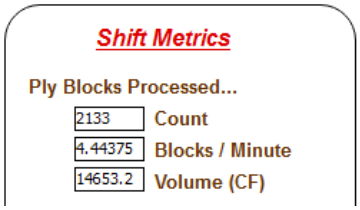
- Clones



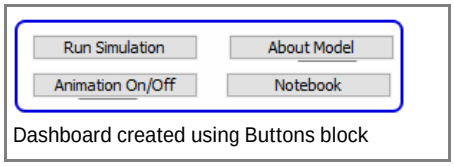
Dialog item cloned onto model notebook

- Exact copies (similar to a shortcut or alias) of dialog items, tables, or graphs which behave identically to the original; no programming required.
- Place on the model worksheet or notebook to create a centralized location for making changes or displaying inputs and results.
- Go to “Cloning” on page 72 to see how it’s done.

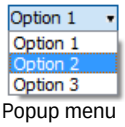
- Notebooks
  - Customizable windows for organizing and managing a model’s information—control model parameters, report simulation results, and document the model.
  - Clone objects onto a notebook, insert text and graphics, and paste pictures, as shown to the right and on page 72.
  - Notebooks are discussed on page 79



- Control panel/dashboard interface
  - Use blocks from the Utilities library to create a dashboard or control panel for the model without programming:
    - Buttons block—set up pre-built buttons or create your own for a custom interface
    - Meter, Slider, and Switch blocks—customizable controls for the model
    - Model Interface block—create a button interface for running multiple trials for experimentation
    - Popups block—create popup menus with multiple custom options. Can also be used to create buttons for accessing database tables.
    - Pause Sim and Run Model blocks—place a button to run or pause the simulation at specified times



Dashboard created using Buttons block



Popup menu

  - Create custom buttons using the Buttons block’s equation editor or custom interfaces using the ExtendSim IDE or an external application such as VB.net.
  - For more information, see “Creating a dashboard interface” on page 74 and “Control blocks” on page 76.

- Animation
  - Many blocks, such as the Tank (Rate library), automatically show 2D animation on their icons if 2D Animation is on during the run.
  - The Animate Value block (Animation library) displays text, a graphic object, or pictures on the worksheet plus can be used to animate a hierarchical block’s icon.

- Use the Equation block (Value library) or the ExtendSim IDE to programmatically create custom animation behavior in blocks you create.
- See “Animation” on page 110.

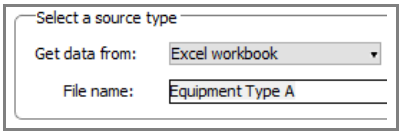
- The Notify block (Value library) plays a sound, stops the simulation, or prompts for an output value when its input is true
- Also see...
  - The ExtendSim database as a tool for presenting data, as discussed on page 47
  - Text, graphic objects, and pictures: ExtendSim supplies tools for, and supports the standard methods of, handling text and graphics. Pictures placed directly on the model worksheet are automatically placed in the background.
  - Interactivity: click a button or change parameters during a simulation run.
  - Named connections: text labels that are used to represent one output at multiple locations in the model, without using connection lines. See page 118.
  - The Throw and Catch blocks (Value, Item, and Rate libraries): use these blocks to transfer values remotely, even between hierarchical layers, without connection lines.
  - The Navigator tool: this explorer-like window is essential for navigating through the hierarchical structure of a model. It is discussed on page 110.

Getting data into the model

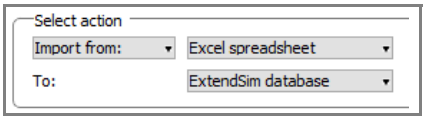
Bring data into the appropriate parts of a model for use before or during the run.

⚠ The preferred method is to get all the data into the model at the beginning of a run; importing data during a run could unnecessarily and dramatically slow the simulation.

- Enter data into model components before the run starts:
  - Type or use the Copy and Paste commands or tools to enter data into fields or tables in block dialogs
  - Enter data into parameter fields and tables that have been cloned from block dialogs into notebooks (mentioned in page 45) or onto the worksheet
- Use a Read block (Value library) or Read(I) block (Item library) to get model data, at the start or during a run, from an Excel worksheet, a text file, local table, global array, or ExtendSim internal database.



- Create an ExtendSim database, discussed on page 47, to manage data; enter inputs directly into its tables
- Import data into a model’s internal relational database or (less common) global array; both are discussed on page 47:
  - Data Import Export block (Value library)—transfer data before, during, or after a run from an external source (Excel, ODBC database, FTP file, text file, ADO database such as Access or MySQL) to an internal target (ExtendSim database or global array).



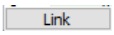
  - Read blocks, discussed above.

  - Database > Import New Database command—import into a model a database text file that was created in another model or was generated by Excel using the Excel DB Add-In (optional add-on for some ExtendSim products).
- Fit raw data to a distribution with Stat::Fit, as mentioned on page 48
- For more information, see the chapter "How To: Data Management" that starts on page 221, especially the section for "User interfaces for data exchange" on page 222.

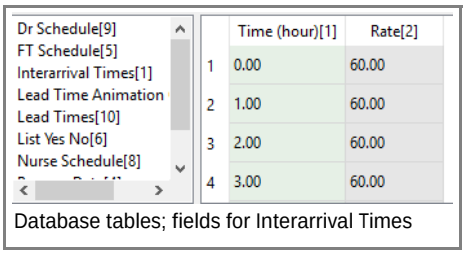
Managing data for use in the model

Localize and manage data so it is available before, during, and after a simulation run.

- Create an ExtendSim internal relational database—a centralized repository for data—and exchange data between the database and the model:
  - Many blocks have a built-in capability to exchange data with the database during the run:
    - •Equation, Read, Write, Query Equation, and Data Specs blocks (Value library)
    - •Read(I), Write(I), and Query Equation(I) blocks (Item library)
    - •Popups block (Utilities library)
  - The Data Import Export block (Value library; see page 46) can access data before, during, or after the run
  - Dynamic data linking (DDL) links to data in an ExtendSim database during the run
    - •Dynamically link dialog items (parameters and data tables) to internal data structures—internal databases, global arrays, and (programmatically) dynamic arrays.
    - •See the Link button on a block's data table or right-click a dialog parameter and choose Create/Edit Dynamic Link



  - Create custom interfaces using the ExtendSim IDE
  - For more information, go to "ExtendSim databases for internal data storage" on page 232.
- Use another data management method as described more on page 233 of the User Reference:
  - Global array—a two-dimensional array of data accessible by any block in a model.
  - Dynamic array—a multi-dimensional internal data structure only accessible through programming.
  - Linked list—an internal data structure that allows the construction and manipulation of complex lists of data.
- Much more information can be found in the chapter "How To: Data Management" that starts on page 221.

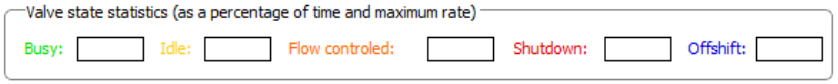


Database tables; fields for Interarrival Times

Analyzing data and assessing results

Perform calculations, analysis, experimentation, and optimization using information generated by the simulation model.

- Verification and validation
  - Use ExtendSim debugging tools and methods to verify that models operate as expected and to validate that they accurately represent the system. See more information starting on page 206.
  - The Model Compare block (Utilities library) helps determine if changes to the model affect results. It is also helpful when validating models between ExtendSim releases. Go to “Automated test environment (Windows only)” on page 217.
- Value library blocks:
  - Mean & Variance block—calculate the mean, variation, and standard deviation of the values input during a simulation run; specify the confidence interval.
- Report library blocks:
  - Statistics block—accumulate data on certain types of blocks and report statistics in a single table using a statistical method; specify the confidence interval.
  - DB Statistics block—calculate statistics for all the records in a field of an ExtendSim internal database table.
- Rate library block dialogs:
  - Blocks in the Rate library have a Results tab that automatically calculates important statistics such as effective inflow and outflow rates, utilization, and so forth.



  - History(R) block—reports flow history, including times, rate, and quantity
- Distribution fitting (Windows only)
  - Determine which distributions, if any, offer a good fit for the underlying raw data.
  - In the Random Number block (Value library) choose Stat::Fit or another distribution fitting software package to perform the analysis. Stat::Fit is an optional add-on for some ExtendSim products.
  - See “Stat::Fit (Windows only)” on page 172.
- Charts and data tables
  - Blocks from the Charts library display a graphical representation of the numbers fed to them as well as a table of the numerical values.
  - See the descriptions of the Chart blocks in “Chart library” on page 174.
- Evolutionary Optimizer
  - Use this Value library block to automatically determine ideal values for parameters, then have it populate the model with the optimized parameter values.
  - See “Optimization” on page 158.
- Scenario Manager
  - Systematically and strategically explore the outcome of different model configurations and analyzes alternatives.

- Use pre-built experimental designs (DOE) or export to Excel, JMP, or Minitab for further analysis.
- See “Scenario analysis” on page 143.

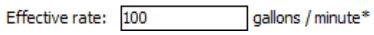
- Sensitivity Analysis
  - Conduct controlled experiments to explore how much of an impact one or more parameters have on model results.
  - See “Sensitivity analysis” on page 138.
- Also see:
  - Decision, Clear Statistics, and Max & Min blocks in the Value library

Reporting results and exporting data

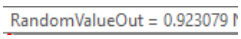
Monitor and/or export intermediate information and simulation results

⚠ The preferred method is to output the data at the end of a run; exporting data during a run could unnecessarily and dramatically slow the simulation.

- Get data from the following components of your model:
  - Fields and tables in block dialogs, especially on the Results tabs in blocks such as the Activity (Item library) or Valve (Rate library)

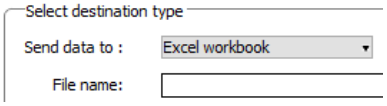


  - Output parameter fields and tables that have been cloned from block dialogs into notebooks (mentioned on page 45) or onto the worksheet
  - Animation, as mentioned on page 45
  - Tooltips, which display current data each time the cursor is placed over an output connector

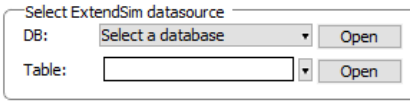


- Connect outputs to blocks in the Chart library (page 174) to see graphs and tables of data during or after the simulation run
- Add blocks to the model to obtain information:
  - Blocks from the Report library
    - Reports Manager block—the primary interface in ExtendSim for creating reports from simulation runs for blocks, events, items, and resources. See “Model reporting” on page 184.
    - Item Log Manager block—collects customized information about how item states change over time. See “Using the Item Log Manager to get item information” on page 399.
  - Blocks from the Rate or Item library
    - History(R) block (Rate library) records information about flow attributes as well as the rate and its duration
    - History block (Item library) reports item statistics and history
    - Write(I) block (Item library) writes data to a database when an item arrives
    - Item Log Manager (Report library) creates a log of information found in certain blocks

- Blocks from the Utilities library:
  - •Pause Sim block, Pause tool, or Run > Debugging commands to pause and step through the simulation to monitor data.
  - •Record Message block (Utilities library) to the model to capture the messages between connectors
- Blocks from the Value library:
  - •Use a Write block to write model data, during or at the end of a run, to an Excel worksheet, a text file, local table, global array, or an ExtendSim internal database.



  - •Notify block sends alerts based on its settings and input
  - •Display Value block displays the input value and can pause between updates
- Create an ExtendSim database as discussed on page 47 to manage data, and see results in its tables
- Export data from the model's internal relational database or (less common) from a global array; both are discussed on page 47.
  - Data Import Export block (Value library)—transfer data during or after a run from an internal source (ExtendSim database or global array) to an external target (Excel, ODBC database, FTP file, text file, or an ADO database such as Access or MySQL).



  - Write block, discussed above.
  - Database > Export Database command—export an ExtendSim database as a text file so it can be used in another model or can be imported into Excel using the ExtendSim DB Add-In for Excel. To export specific tables, see the Database > Export Selected Tables command.
- Use the Scenario Manager (Value library) to export data to Excel, Minitab, or JMP for design of experiments. See page 48.
- For more information, see "User interfaces for data exchange" on page 222 and "Blocks for data management and exchange" on page 245.

Communicating with external applications and devices

Methods and technologies for data sharing, communication, and control between ExtendSim and external applications and equipment.

⚠ Some functionality is Windows only

- Pre-built blocks in the Value library:
  - Data Import Export block—exchange data between ExtendSim (internal database or global array) and outside applications and files (Excel, ODBC database, FTP file, text file, ADO databases such as Microsoft Access, SQL Server, and MySQL) at the start, during, or end of a run

    - Read and Write blocks—exchange data between an ExtendSim database, Excel workbook, global array, text file, or local table at the start, during, or end of a run
    - Equation and Query Equation blocks—see "Equation-based blocks" on page 189
    - Scenario Manager block—export scenario results to Excel, JMP, or Minitab for further analysis. Mentioned on page 48.
    - Random Number block—use this block to communicate with JMP, BestFit, ExpertFit, and Stat::Fit to determine the appropriate distribution to use given the raw data
- ExtendSim DB Add-In for Excel (Windows only)
    - Create, edit, and export files between Excel and the ExtendSim internal database
    - Can also use Excel as the master for documenting the data and database structure
    - See "Excel Add-In for ExtendSim databases" on page 233.
- ADO (ActiveX Data Objects) for databases (Windows only)
    - Communicate between ExtendSim and external databases—Microsoft Access, SQL Server, and MySQL—using the Data Import Export block (Value library)
    - For programmatic control, the include file "ADO_DBFunctions.h" stores user-defined functions for ADO
    - See the Technical Reference, User-defined functions for ADO
- ExtendSim COM Object Model
    - Primary method for ExtendSim to be controlled and communicated with as an Automation Server
    - Allows external languages to control ExtendSim in many ways, as discussed in "Scripting functions" on page 51
    - See "ActiveX/COM/OLE (Windows only)" on page 242 and the Technical Reference, ExtendSim as an Automation Server
- ActiveX Automation
    - Primary method for ExtendSim to serve as an Automation Client
    - Access and manipulate (set properties of or call methods on) automation objects that have been developed in and exported by other applications
    - See the Object Mapper block (Utilities library), the section "ActiveX/COM/OLE (Windows only)" on page 242, as well as the OLE/COM functions and the section on OLE and ActiveX Automation in the Technical Reference
- DLL functions
    - Provide custom operations from within a block's code by calling Dynamic Link Libraries (DLLs) of routines or functions written in an external language
    - Primary method for ExtendSim to communicate with DLLs created by other applications or by equipment
    - See the Technical Reference.
- Scripting functions

- Automate the process of building or making changes to models, either from within ExtendSim or by external applications
- Create custom wizards to simplify tasks or interact with modelers; develop self-modifying models
- See the Technical Reference, Scripting functions and Scripting technique

- Internet access
  - Access and manipulate data and files using Internet protocols
  - See the FTP block (ModL Tips library) or program using functions
  - See the Technical Reference, Internet access
- Mailslot functions (Windows only)
  - Communicate and control other computers in a network domain using this messaging functionality supported by the Windows API
  - Send unidirectional messages from a given machine to a specified mailslot on other machines
  - See the Technical Reference, Mailslot
- See also:
  - Command block (Value library)
  - Additional blocks in the ModL tips library: ActiveX Control, DLL Add Block, FTP, ODBC, and Read Device
  - The blocks in the ModL Tips library; the path is Documents/ExtendSim/Libraries/Example Libraries.
  - File I/O, Interprocess Communication (IPC), and OLE/COM (Windows only) functions in the Technical Reference

☞ Also see the ExtendSim white paper titled “Interprocess Communication”.

How To

Libraries and Blocks

A description of ExtendSim libraries and blocks
and the many ways to use them

“The first thing to have in a library is a shelf.
From time to time, this can be decorated with literature.
But the shelf is the main thing.”
— Finley Peter Dunne

Overview

Working efficiently with libraries and blocks is critical to simulation modeling.

The libraries that ship with ExtendSim

ExtendSim libraries are repositories for the blocks that you use to build models. ExtendSim ships with several libraries, each one containing blocks that are used for modeling specific situations or types of systems. You can also develop your own libraries of custom blocks, or create libraries to save and organize hierarchical blocks in a way that best works for your modeling needs.

Libraries are located in the ExtendSim/Documents/Libraries folder.

☞ Some libraries are included in all products while others are only included with specific products, depending on the type of modeling the product is used for.

Item library (not available in ExtendSim CP)

The blocks in the Item library are used primarily in discrete event modeling, although they can also be included in discrete rate models. The Appendix that starts with "Item Library Blocks" on page 583 has a complete list and brief description of the blocks in this library.

Chart library

The Chart library holds all the common types of blocks used to graph and record data for models. All Chart library blocks are described in detail in "Chart library" on page 174.

☞ The Chart library replaces the Plotter library of previous releases. The Plotter library is now a "legacy library", included only so older models can be run.

Report library

The blocks in the Report library are also useful for any type of model. They gather information as the model runs and display and calculate statistics on the results. All the Report library blocks are described in detail in "Model reporting" on page 184.

Rate library (only available in ExtendSim Pro)

The Rate library is used for discrete rate simulations — modeling the flow of product or material according to some rate-based calculations. Discrete rate models also often use blocks from the Value and Item libraries. A complete list and brief description of the blocks in the Rate library starts on page 592.

Utilities library

The Utilities library contains a collection of blocks that are helpful for performing various tasks such as adding active buttons to models, fitting data to a curve, and synchronizing the model to real time. A complete list and brief description of the blocks in this library starts on page 595.

Value library

The Value library contains blocks that are used for continuous modeling and play a critical role in other types of models as well. A complete list and brief description of the blocks in this library starts on page 576.

Animation library

The Animation library lets you add custom animation to models and hierarchical block icons. The Animate Item and Animate Value blocks in this library are used to add 2D animation to

models and hierarchical blocks, as described in “Blocks for customized animation” on page 112.

Electronics library

The blocks in the Electronics library are used to simulate system level design of analog, digital, signal processing, and control systems. For example the Electronics blocks are used in the example model Noisy FM System.

Example Libraries folder

The following libraries are located in the ExtendSim/Documents/Libraries/Example Libraries folder.

Custom Blocks library

These blocks have been created for very specific purposes, such as to illustrate a concept or hard-code certain behavior. For instance, the Planet block from this library is used in the continuous model Planet Dance.

ModL Tips library

This library contains blocks that illustrate the techniques described in the ExtendSim Technical Reference.

Templates library (discrete event blocks are not available in ExtendSim CP)

Contains hierarchical blocks that serve as templates for specific behaviors, such as setting the arrival time based on the time of day.

Tutorial library

Contains blocks built in conjunction with tutorials in the Technical Reference.

Legacy folder

Legacy libraries have been replaced by newer libraries and/or ExtendSim functionality and are no longer supported. They are included with ExtendSim for a few years after decommissioning to allow customers to run older models while transitioning them to use the new libraries.

Plotter library

In ExtendSim 10, the Chart library replaced the Plotter library which has now been relegated to Legacy status. For the next few versions customers can continue running older models that use the Plotter library but are encouraged to use the blocks in the Chart library for new models.

Legacy libraries from ExtendSim 7

☞ The following libraries are only available in ExtendSim releases prior to 10.

Certain libraries were transferred to Legacy status as of ExtendSim 7. These libraries were only included in releases 7-9 so that customers could continue running older models while transitioning to new libraries. Depending on which ExtendSim product you had, those Legacy libraries were:

- Animation (v6 and prior)
- BPR—replaced by the Item library
- Discrete Event—replaced by the Item library
- Flow—replaced by the Rate library
- Generic—replaced by the Value library

- Items (DB)—not needed; the ExtendSim database is fully integrated with ExtendSim blocks
- Mfg (Manufacturing)—replaced by the Item library
- Quick Blocks—replaced by the Templates library
- SDI Tools—replaced by ExtendSim functionality

⚠ Do not use these legacy libraries to build new models. They are not supported and will not be included in future releases.

Opening, closing, and searching for libraries

Libraries are the repositories for the blocks used to build models and the entire definition for a block (its program, icon, dialog, and so on) is stored in the library. When you place a block in a model, the block itself is not copied to the model. Instead, a reference to the block is included in and stored with the model. On the other hand, any data you enter in the block's dialog is stored within the model, not within the library.

There are many advantages to this method of using references to libraries instead of actual blocks in models. If you change the definition of a block in a library, all models that use that block are automatically updated. Also, block definitions are quite large, so storing just a reference to the library saves memory and reduces processing time.

When you save a model, ExtendSim saves the names of the blocks as well as the locations of the libraries that store the blocks. The next time you open the model, ExtendSim automatically opens the libraries the model uses.

Opening a library

Whenever you open a model, ExtendSim automatically opens the libraries that are used in the model.You can also open a library directly by:

- Choosing Library > Open Library. This opens a window for selecting which library file(s) to open. If an "Alternate path" has been entered in the Edit > Options > Libraries tab, this command will open that folder. Otherwise (or if can't find the library in the alternate location), it opens the Documents/ExtendSim/Libraries folder.
- Double-clicking the library file or drop-launching it.

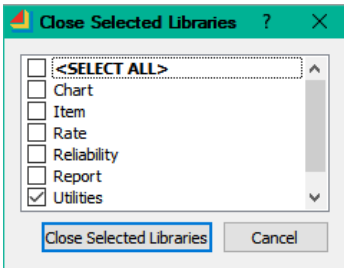
☞ The command Edit > Options > Libraries tab allows several libraries to load automatically whenever ExtendSim is launched. By default, the most common libraries for your ExtendSim product will already be in that list. See "Libraries tab of Options command" on page 547 for more information

By default, when a library opens the ExtendSim application:

- Opens the library window for that library. (See see "Library windows" on page 58.)
- Adds the library name, in alphabetical order, to the bottom of the Library menu.

☞ By default, blocks in the Library menu are listed by category. To view them alphabetically, go to the Edit > Options > Libraries tab and uncheck *List blocks in Library menu by category*.

Closing a library



Unless you close ExtendSim, libraries stay open whether they are used in a model or not. To close libraries that are not used in a model, choose Library > Close Library and select the libraries you want to close. It is unlikely that you will need to close libraries often since open libraries do not take up much memory and it is usually convenient to leave all your commonly-used libraries open.

Closing a library also closes its library window. Note however that closing a library window does not close the library. Library windows are discussed on page 58.

☞ ExtendSim will not let you close a library that is being used by an open model.

Searching for libraries and blocks

When you open a model, ExtendSim opens the libraries that were last used in the model. Then it tries to find the model's blocks within those libraries. The search methods are detailed below.

☞ These searches are based on the *name* of the library or block. If you've changed the name, also see "Substituting one library for another" on page 68.

Library searches

ExtendSim searches for libraries based on the name of the library. Unless you've unchecked the *Automatic search* option in the Edit > Options > Libraries tab, the search order ExtendSim uses to locate the needed libraries is:

1) The folder containing the model (but not any sub-folders within that folder)
2) The *Alternate path*, if one is specified in the Edit > Options dialog
3) The Libraries folder within the Documents/ExtendSim folder, including any sub-folders within the Libraries folder

You can manually stop this search process at any time by pressing Ctrl+period (Windows) or Command+period (Mac OS).

⚠ To avoid extensive searches, libraries should be kept in the same folder as the model that uses them or in the Documents/ExtendSim/Libraries folder.

If ExtendSim can't find a library when the model opens, or if the Automatic search option is unchecked, it will put up a file selection dialog that asks "Where is the library xxx?". Your choices are:

- Cancel the library search operation. ExtendSim will then try to find the missing blocks, as described in "Block searches" below.
- Find and open the library manually. Since ExtendSim only needs to know where the blocks for the model are, you could also use this choice to substitute a library that has a different name (but the same named blocks) as the search library. See "Substituting one library for another" on page 68 for more information.

When the model file is subsequently saved, any new library name and/or location will be saved as well, so searching will not be necessary the next time the model file is loaded.

☞ To force ExtendSim to ask for all the libraries to open when opening a model, uncheck the Automatic search option in Edit > Options > Libraries tab.

Block searches

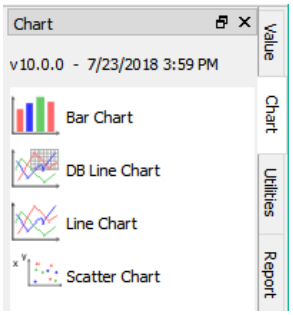
ExtendSim automatically searches for each block that the model uses. It first searches for the block in the library in which the block last resided. If you have renamed the blocks or removed them from the library, ExtendSim may not be able to find them. In that case, ExtendSim prompts you to locate any missing blocks. The dialog offers two choices:

- Choose *Open* to locate and open another library where the block resides. If ExtendSim is unable to locate the block in this library, the *Substitute Block* dialog will open so you can substitute an alternative block from any open library. When the model file is subsequently saved, any new block name and/or location will be saved as well, so searching will not be necessary the next time the model file is loaded.

☞ Any substituted block must be substantially the same as the searched-for block, or ExtendSim will report error messages.

- Select *Cancel* if the block no longer exists or if you can't substitute another block. In this case, ExtendSim notifies you that it will put text in the place of the block.

Library windows



.A library window is a separate window that shows all the blocks inside the library in alphabetical order. Library windows are one of the main methods for adding blocks to models, so by default they open whenever a library is opened. If you program blocks, you can also use library windows to manage blocks, as discussed in "Managing blocks" on page 69, and to access a block's structure as discussed in the Technical Reference.

Opening library windows

Library windows open automatically based on choices in the Edit > Options > Libraries tab. By default ExtendSim opens the most commonly used libraries upon launch and also opens their library windows. As discussed below, library windows stack on the right side of the application window and each library window has its own tab on the right side, as shown here.

☞ The order of the stacked library windows depends on the order in which the libraries are listed in the Edit > Options > Libraries tab. If there are no libraries listed, or if an unlisted library is subsequently opened, the stacked list of libraries reflects the order in which the libraries were opened.

To open a library window manually, go to the Library menu and choose Open Library Window, the first choice in the list of blocks for each library.

The top portion of the library window gives the library name. Below the library's name is its version and the date last modified. If selected in the Edit > Options > Libraries tab, this section also shows the path to the location where the library resides.

☞ Hovering over the library's name will also show the path to the library.

The blocks in the library are listed with pictures of their icons and, if the *Show block modified and compiled dates* option is selected in the Edit > Options > Libraries tab, their modified dates.

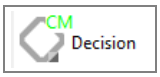
Docking and stacking

Library windows are docking windows that by default stack on the right side of the application window.

To have a library window be a floating window, click its tab to bring it to the front. Then click in its title bar and tear it off from the dock. To return a library window to a docking position, double-click its title bar or drag it into the docking area such that the area above it contains no other blocks.

☞ Double-clicking the library window's title bar performs the tear off action if the library window is docked, or returns the window to its docking area if it has been torn off.

Code management and debugging designations



- If a block has been compiled with external source code, it will be listed in the library window with the designation *CM* (code management) on the right side of its icon. The CM designation does not appear when the block is placed on the model worksheet.

- If a block has been compiled with debugging code, its icon in the library window will have a red border around it. The border also appears if the block is placed on the model worksheet.

- Blocks that have been saved in a library but not compiled will appear in the library window with their names in red italics.

Compiling options are used by block developers and are discussed in the Technical Reference.

Closing library windows

Close an individual library window by clicking its red close box.

To close all open library windows, use the Library > Close All Library Windows command or the corresponding button in the toolbar at the top of the windows. The Open All Library Windows command and button re-opens the library window for every open library.

☞ Closing a library window only closes that window, not the library itself.

Working with blocks

ExtendSim blocks have a user interface (their icon, dialog, and help) and internal, integrated ModL code that determines how the blocks behave.

As discussed on page 56, blocks are stored in repositories called libraries and the entire definition for a block (its program, icon, dialog, and so on) is stored in the library. When you place a block in a model, the block itself is not copied to the model. Instead, a reference to the block is included in and stored with the model. On the other hand, any data you enter in the block's dialog is stored within the model, not within the library.

Placing blocks on the model worksheet

As discussed below, there are three methods for placing blocks on a model worksheet without programming: Point and Click, Smart Block technique, and Bump Connect.

The Smart Block and Bump Connect methods not only place the blocks but connect them to other blocks. In addition, scripting allows you to place and connect blocks on the worksheet programmatically (see the Technical Reference).

Point and Click

Click once on the block in the Library menu or library window, then click on the model worksheet wherever you want the block to be placed. Unlike the other methods, this works for every type of block in every library.

☞ To place multiple copies of the same block on a worksheet, hold down the Alt or Option key while clicking on the worksheet. Use the Escape key if you have selected the wrong block.

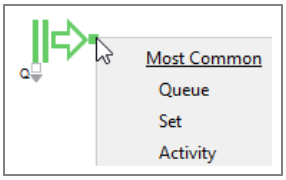
Smart Block technique

Blocks in most libraries of the libraries that ship with ExtendSim (Item, Value, etc) have an easier method to using Point and Click to add blocks to the model, and this method also automatically connects the blocks.

⚠ The Smart Block technique involves right-clicking on a connector, so an ExtendSim library block (other than the Executive) must already be on the worksheet. Smart Block does not work with hierarchical blocks.

To connect a block at a connector

Right-click on one of the block's input or output connectors. The window that appears lists at the top the blocks that are most commonly connected to that connector; below are the libraries so you can access the entire list of blocks in a library.



Select the block you want from the list and it will automatically be placed on the worksheet and connected to your first block. For example, if you right-click on the value input connector on a Set block (Item library), you can quickly connect a Random Number block (Value library). Repeat as desired for the second and subsequent blocks.

To connect a second block at a connector

To add and connect a block at a connector where there is already a block connected, hold down the Ctrl/Cmd key while right-clicking on the connector. Select the desired new block from the list.

To insert a block between two connected blocks

To insert a block between two connected blocks, move the two blocks apart to provide room for the new block. Then right-click on an input or output connector of one of the existing blocks and select the desired block.

The Smart Block technique uses scripting functionality and is database driven; the list of common blocks adapts depending on which block you are connecting to.

☞ The Smart Blocks feature is implemented for blocks in the libraries that ship with ExtendSim. You can add the functionality to your custom blocks by having their code call the relevant include files.

Bump Connect

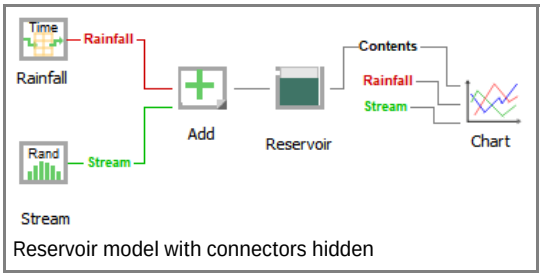
To add an Item or Rate library block to a model using Bump Connect, click on the block in the library menu or library window, then place the block in the model by bumping its itemIn or inFlow connector to the corresponding output of a block on the worksheet. This adds and connects the new block to the existing block using whichever connection line choices you've made.

☞ This technique is only available for blocks from the Item and Rate libraries. To disable this feature, go to the Edit > Options > Model tab.

Connectors

Connectors are used to input and output values, items, or flow for each block.

The Model menu has commands that let you show or hide the connectors in the model workspace. Hiding connectors can improve the appearance of the model, especially for complex models with many blocks.



Reservoir model with connectors hidden

Connector types

Connectors on a block are visually different, depending on how they're used. This table describes the different types of connectors that can be seen on or (if you program) added to blocks in ExtendSim.

Type	Input	Output	How they're used
Value			Continuous blocks use value input and output connectors to pass values from one block to another. Blocks in the Item and Rate libraries may also use value connectors for the same purpose.
Item			Discrete event blocks use itemIn and itemOut connectors to pass discrete items from block to block. Blocks in the Rate library may also use item connectors for the same purpose.
Flow			Discrete rate blocks use inflow and outflow connectors to pass information about the effective rate from one block to another. You can also connect from a flow connector (input or output) to a value input connector.
Reliability			Blocks in the Reliability use Reliability connectors to pass information between nodes.
Universal			Universal connectors are usually used as inputs. Value, Item, Flow, and User-Defined output connectors can connect to Universal inputs.
Array			Array connectors are for passing arrays of information from one block to another.
User Defined			User Defined connectors can be programmed to behave in any manner the developer wants.
Box			A user defined connector shaped like a box. It can be programmed to behave in any manner the developer wants.

☞ Other than some specific instances, you cannot connect from one type of connector to another. For example, attempting to draw a connection from an itemOut connector to a Value input will

result in an error message. The two exceptions are that Value, Item, Flow, and User-Defined outputs can be connected to Universal inputs, and Value outputs can be connected to Flow inputs.

Variable connectors

Regardless of type, each connector can be single or variable depending on how the block is constructed. Variable connectors act like a row of single connectors, where the row can be expanded or contracted to provide a required number of connectors. Some blocks have only single connectors, some have only variable connectors, and some have a mix. Variable connectors are usually designated by a black arrow.

The Constant block has two single input connectors and one single output connector:

A Math block in Add mode has a variable connector exposing two inputs as well as a single output connector:

The Math block's variable connector expanded with dragging cursor to expose three inputs:

Each block's functionality determines whether or not it has variable connectors, where they are located, and what each connector represents. For example, a variable connector could have connectors captioned *minimum* and *maximum* placed on the bottom of the block for downward expansion.

Variable connectors can be expanded or contracted using the dragging cursor—an up/down or left/right arrow as seen above; they can also be collapsed. As explained more below, you expand or contract a variable connector to provide the desired number of inputs or outputs. You collapse a variable connector to improve model appearance.

Expanding or contracting a variable connector

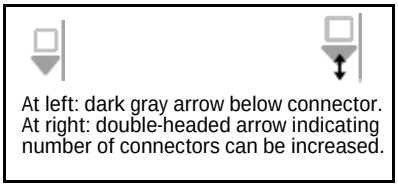
Wherever possible a block will anticipate its usage and provide the required number of connectors. For example, the Holding Tank block (Value library) has a variable connector that can be expanded to provide three inputs: the amount wanted from the tank, a trigger to reset the tank's settings, and the amount of initial contents. You might use one, two, or all three of those inputs, but the block won't allow more than three inputs.

For other blocks, the number of connectors you might need depends on choices you make in the dialog or how you use the block in a model. There are two primary methods for causing a variable connector to change the number of available connectors:

- Drag the variable connector (as discussed below) until the desired number of connectors is achieved. This is available if the block allows unlimited inputs or outputs.
- Change some setting in a dialog, causing an increase or decrease in the number of connectors allowed. An example of this is the Random Number block (Value library), where you select a distribution from a popup. Depending on the distribution chosen, the variable connector will provide from two to four inputs for specifying the distribution's arguments. Since the number of arguments for each distribution is fixed, you will not be able to expand the number of connectors beyond the proper number. You can, however, contract the variable connector as discussed below. Note that some blocks with variable connectors, such as the Equation blocks, do not allow you to increase or decrease the number of connectors except through dialog settings.

To increase the number of connectors when there is a variable connector:

- Hover over the dark gray arrow that is below the input or output connector. (Don't hover over or click on the connector, just hover over the gray arrow.) If you are in the correct spot, the cursor will turn into a dragging cursor—a double-headed up/down or left/right arrow—indicating that the array of connectors can be resized.


At left: dark gray arrow below connector.
At right: double-headed arrow indicating number of connectors can be increased.

- Once the cursor turns into the double-headed arrow, click and drag in the direction of the double-headed arrow until the desired number of connectors are exposed, as shown here on the left side of an Activity block. As you drag, more connectors appear; in most cases, the number of connectors is displayed.

☞ You will not be able to expand a variable connector beyond what is reasonable for the block, given its usage and dialog settings. And some blocks, for example the Equation blocks, do not provide an arrow for expanding the number of connectors; the number of connectors in those blocks is only controlled through the dialog.

To decrease the number of connectors, hover the cursor over the dark gray arrow at the end of the last connector until it becomes a dragging (double-headed) cursor. Then click and drag back towards the variable connector's starting point.

☞ The number and location of connections to the variable connector affect its ability to be contracted. You can only contract the variable connector to the point of its outermost connection. In this case, to reduce its size you need to collapse it, as discussed below.

Collapsing a variable connector

As discussed above, if you have made connections to a variable connector you might not be allowed to contract it back to its unexpanded position. This is a safety mechanism, so that it is clear what connections have been made. However, to simplify the appearance of the model, you can collapse a variable connector to an unexpanded state even if it has many connections to it.

To collapse a variable connector, place the cursor over the black arrow until it becomes a dragging cursor, then double-click. The connector now appears with a red + sign in place of the black arrow, as shown on the right.

To reverse the process, place the cursor over the red + sign until it becomes a dragging cursor, then double-click.

Connecting to different connector types

As you saw in earlier chapters, you can use continuous blocks (such as those from the Value library) in your discrete event and discrete rate models, which means you can have blocks in a single model that use different types of connectors (some blocks have multiple types of connectors on them as well).

You can connect value connectors together, item connectors together, and flow connectors together. You can also connect value and flow connectors to each other, but you cannot connect value and flow connectors to item connectors. This is because value and flow connectors only pass values while item connectors pass unique items.

☞ Any type of connector—value, flow, or item—can be connected to universal input connectors.

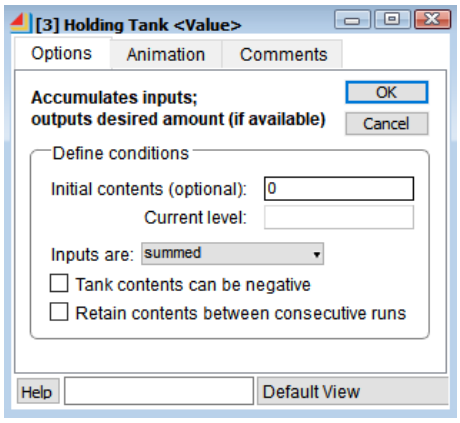
Also note that if you create blocks that have user defined connectors, those connectors can only be attached to other user defined connectors and to universal connectors.

Dialogs

Most block dialogs let you change or view the settings before, during, and after a simulation run. Some of the things you can do using a dialog include:

- Choose from multiple functions (e.g. the Math block)
- Enter initial values
- View simulation results
- Add comments to help document your models
- Set animation criteria
- Choose notification options

To open a block's dialog, double-click it or right-click it and select Open Dialog. For example, if you double-click the Holding Tank's icon, the dialog at right opens.

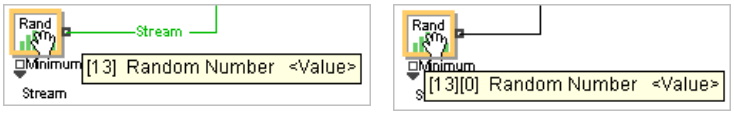


In the dialog's title bar is the block's global object ID, its name, and, in braces, the library it resides in.

☞ When there are multiple open models and dialogs, it can be difficult to remember which block comes from which model. To see which model a block is in, hover over the block's name in the dialog's title bar.

Every object on the worksheet has an ObjectID. As each block or other object is added to a model it is assigned a unique and sequential global object ID from 0 to n-1. This sequence does not change. If a block or any other object is deleted, its number becomes an "unused slot" which is available when another object is added to the model window.

Each object within a hierarchical block has a second, local, ID that reflects its relationship to the other objects in the hierarchical submodel.



Left: Random number block on worksheet, global number is 13
Right: Inside hierarchical block, global number is still 13, local number is 0

At the bottom of every dialog is a Help button that provides more information, such as the block's purpose and use, what each connector does, the meaning of each dialog item, and so on. Beside the button is a text box where you can enter a label for the block. When available, a View popup lets you change how the icon for the selected block is presented; icon views are discussed later in this section.

Some dialogs also calculate and display values that are generated as the model runs, so if you leave a dialog open during the simulation you can watch the impact on different variables. You can even change some of the settings in a dialog as you run the simulation, such as choosing different buttons or typing new values.

☞ When you click a button while the simulation is running, the block gets that changed value on the next step. However, if you type text or enter numbers into a field, the model pauses while you are typing in order to get your entire input.

Animating blocks

ExtendSim provides built-in animation for many blocks and you can also create custom animation for any block. To learn more, see "Animation" on page 110.

Customizing block icons

The blocks in the libraries that come with ExtendSim (such as the Value and Chart libraries) are displayed on the screen as icons that depict their function. Once you place a copy of these blocks in your model, you might want to make its icon more closely indicate its role in your particular model. However, using programming to change a block's icon in one model will change it in every model using that block, since blocks and their internal descriptions reside in libraries, not in models. In addition, since some blocks are animated, it is not easy to change a block's icon without having to modify its ModL code. For these reasons, you should not change the icon on the blocks in the libraries that come with ExtendSim.

If you build your own blocks, you can give them any icon you want. And there are two better ways to customize blocks in your model without having to directly modify icons:

- You can paste pictures on the worksheet to customize the model. Pictures automatically go behind blocks and text. For example, you can place a map of a region behind your model, or show the layout of a plant. This is described in more detail in "Working with pictures" on page 109.
- You can also add a picture to a block without affecting the original one in the library by making the block hierarchical. This technique is described in more detail in "Modifying hierarchical blocks" on page 128.

Icon views

At the bottom right of many dialogs is a popup for icon views. Some blocks have multiple icon views; some have none. If a block has icon views, the choice of view will determine how the block looks on the model worksheet.

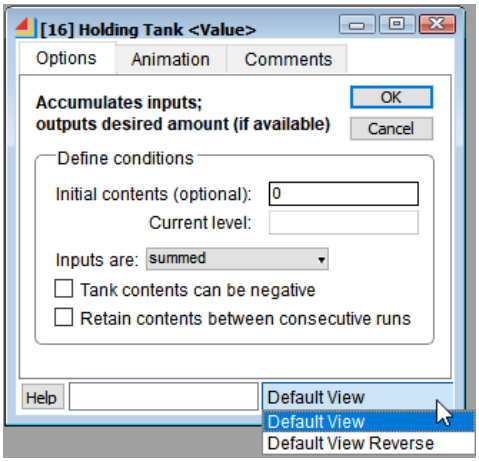
For example, an icon view could provide:

- Alternate connector positions, such as an input connector on the right side of the block. This helps avoid awkward connection line placements when building models.
- Choices of icon size, such as regular and reduced size. An example is the History block (Item library) where the *Status* view is smaller than the *Default* view.

History icon: Default and Status views

- Different icons depending on choices that have been made in the block's dialog. For example, a block that represents a flow could be pointed forward in the view labeled *Forward* and pointed backward in the view labeled *Backward*.
- Alternate icons depending on what the block represents in a model. This is often used for hierarchical blocks that represent a specific process in one part of the model but a different process, with the same functionality, in a different part of the model. For instance, in the Markov Chain Weather model discussed on page 17, the hierarchical block "Weather Forecast" uses views to indicate the current state for a state/action model.

Blocks without icon views will not have any choices in the Views popup menu at the bottom of the block's dialog. If additional views are present, they can be selected by right-clicking on a block's icon or by using the Views popup menu. They can also be called by functions.



ExtendSim facilitates the creation of different icon views while developing or editing a block's structure. For more information see the Technical Reference.

Hierarchical blocks

It's not unusual for models to have thousands of blocks, which can make it difficult to understand what is happening in the model. Hierarchy helps solve this problem by grouping multiple blocks into one hierarchical block that represents a portion of the process being modeled. A hierarchical block can contain submodels, text, graphics and clones of dialog items and tables. Hierarchical blocks can even contain other hierarchical blocks, so there are multiple levels of hierarchy.

☞ The Navigator is a useful tool for quickly drilling down through the different levels of hierarchical blocks to find the one you're looking for.

Because hierarchy is mainly used to enhance or simplify model appearance, it is discussed in further detail in "Hierarchy" on page 120.

Creating and maintaining libraries

☞ This section is only for those who build custom blocks or want to save hierarchical blocks in libraries.

Creating a new library

To start a new library, choose Library > Library Tools > New Library. A dialog appears for selecting a location for the new library and giving it a name. Note that libraries should be saved in the Documents/ExtendSim/Libraries folder or in the same folder as the model; otherwise ExtendSim will not be able to find them.

⚠ **Whether you build new blocks or modify ExtendSim blocks, give them a new name and put them in a new library.** Do not name them the same as the ExtendSim blocks and do not put them in the libraries that come with ExtendSim.

Saving and compiling libraries

Every time you make a change to a library, such as adding a new block or moving a block from one library to another, ExtendSim automatically saves the library. If you build a block and save it, both the block and the library are saved.

When you create or make changes to blocks using the ModL programming language, the blocks need to be compiled to machine code so they can be used to build models. If you make a change to an Include file that is used by more than one block, its usually simplest to recompile the whole library so the appropriate blocks incorporate the updated Include file.

☞ Unless you program your own blocks or move a library from one operating system to another (for example, if you move a library from Windows to Mac OS) you do not need to compile. The libraries and blocks that are packaged with ExtendSim are already compiled.

To compile a block or a library, use the compile commands in the Develop and Library menus:

- The Library > Library Tools > Compile Selected Blocks command causes the blocks selected in the library window to be recompiled. Uncompiled blocks show in the library window with their names in red italics.
- The Develop > Compile Block command compiles the block to machine code without saving or closing the block. This is useful for testing new code for syntax errors when you are building a block. The command is only enabled if a block's structure is open.
- The Library > Library Tools > Compile Libraries command compiles the libraries selected in the popup window and saves changes.

Compiling a library or block with debugging information allows you to set breakpoints, watch points, and so forth. Compiling with external source code causes ExtendSim to generate an external file that contains the source code; this is useful for version control.

☞ Blocks with external source code will show in the library window with *CM* (code management) next to their icons. Blocks with debugging code show in the library window and on the model worksheet with a red border around their icon. Uncompiled blocks show in the library window with their names in red italics.

For more information about saving and compiling libraries, see the Technical Reference.

Substituting one library for another

If you create your own blocks, you might want a model to use the blocks from a library with a different name. For example, if you have developed a newer version of a library and you want the model to use those blocks. Normally ExtendSim opens the library too fast for you to stop it.

In order to bypass this process, move the original library so that it is *not* in either the Documents/ExtendSim/Libraries folder nor in the folder that contains the model. Then make sure your new library *is* in the Libraries folder.

If you do this before opening the model, ExtendSim will stop and request the location of the library, as discussed in “Searching for libraries and blocks” on page 57. You can then use the dialog to find and open the libraries you want the model to use. This will work even if the library name is different than the one the model originally used, because what is important to ExtendSim is the block name within the library, rather than just the library name.

Once all the required libraries are open, save the model so that the new libraries get saved with the model.

☞ To force ExtendSim to ask for all the libraries to open when opening a model, uncheck the Automatic search option in Edit > Options > Libraries tab.

Arranging blocks in libraries

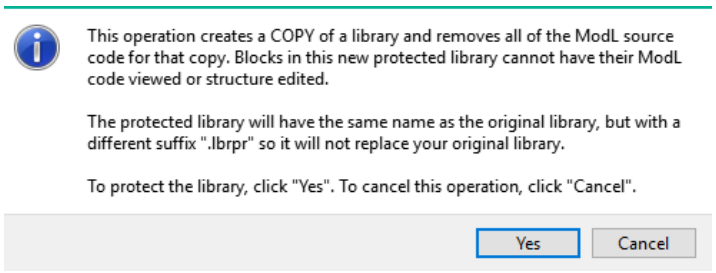
If you build your own blocks, you could put all of the blocks you use in one huge library. That way, you would never have to try to remember which block was where or remember where your libraries are on your hard drive. However, this arrangement could make it difficult to maintain your blocks.

Simulations run neither faster nor slower when you group your blocks in a single or multiple libraries. The only performance consideration is that it initially takes more time to open multiple libraries than it does a single library.

⚠ Do not add blocks, even hierarchical blocks, to the libraries that come with ExtendSim or move blocks from those libraries. If you add blocks to an ExtendSim library, your work will be lost when you update. If you move blocks from an ExtendSim library to a library you create, the blocks in your library will not be updated when the ExtendSim library is updated.

Protecting the code of library blocks

If you build your own blocks, you may not want others to have access to your code. As discussed in the Technical Reference, ExtendSim blocks contain ModL code and can also reference Include files.



You can prevent others from accessing the code in your libraries by either

- Not giving them the Include files, and/or
- Removing the ModL code of the blocks in a library

To remove ModL code from your blocks, use the Library > Library Tools > Protect Library command. This process creates a duplicate of the library with all the source code removed. Keep the original library in a safe place, since there is no way to recover the ModL source code from the protected library.

A protected library can be used in the same way as any other library except that the ModL code cannot be altered or viewed. This means that someone using the library has all the functionality of the blocks in that library but no ability to see how the blocks work.

☞ To protect a hierarchical block's structure and prevent a user from double-clicking the hierarchical block to see the underlying submodel, use the Model > Protect Model command discussed at "Protecting the model" on page 255.

Converting libraries to RunTime format

The ExtendSim Analysis and Player RunTime versions are useful for distributing models to those who do not have a full ExtendSim license. To have your custom blocks be available to run models in a RunTime version, but not be available for building models, convert the blocks to RunTime format. Converting libraries to RunTime format also prevents them from being used in the full version of ExtendSim. The conversion process is described in "Convert Library to RunTime Format" on page 553.

Managing blocks

☞ This section is only for those who build custom blocks or want to save hierarchical blocks in libraries.

⚠ Do not add blocks to the libraries that come with ExtendSim or move blocks from those libraries. If you add blocks to an ExtendSim library, your work will be lost when you update. If you move blocks from an ExtendSim library to a library you create, the blocks in your library will not be updated when the ExtendSim library is updated.

Copying blocks

To make a copy of a block in the *same* library, select the block in the library window and right-click to Duplicate Blocks. This copies the block into the current library and renames it with the block name followed by a sequential number. This is common practice when you want to use the ModL code in one block as the template for a different block.

To copy a block from one library to another, open the library windows for each of the libraries. Right-click to Copy Selected Blocks in the source library window, then right-click to Paste Blocks into the target library window.

Changing a block's name

To change the name of a block, select the block in the library window and right-click to Rename Block. In the window that opens, enter the new name.

If you change the name of a block, models that use that block will not be able to find it because they are expecting the original name. ExtendSim will then ask you where the block is located and present a dialog box for searching, as discussed in "Block searches" on page 58.

Removing blocks

It is rare that you will want to remove a block from a library but, if you do, select the block in the library window and right-click to Delete Block or press the Delete or Backspace key. ExtendSim will not let you remove a block that is in use in an open model. If you remove from a library a block that is used by a model and later open that model, ExtendSim will present an

error message and give you the opportunity to indicate the location of the missing block; if it can't find the block, it will put a placeholder in the model window.

Corrupted blocks

On rare occasions, you may get a message as you open a library that indicates that a block has been corrupted or is bad. The corrupted block will appear in the library window as **BAD*Blockname*. To save the rest of the library, copy the uncorrupted blocks to a new library and discard the old library. Then copy a backup copy of the block (hopefully you have one) into this new library. This is a good reason to always back up your work! (If your block is corrupted, contact Andritz Inc. and we may be able to fix the block.)

How To

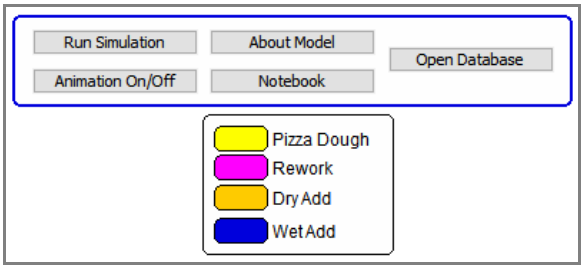
Customizing the User Interface

Personalize the ExtendSim
modeling environment

“If you can dream it, you can do it.”
— Walt Disney

Overview

Every modeler's needs are different. ExtendSim offers a variety of methods to customize the model interface so you, or those you give models to, can work in the most convenient and effective manner. Most of these methods use non-programming techniques.



Cloning

In most programs, buttons and dialog parameters are found only in dialogs. The advantage of this is you always know where to find them. However, having all your choices in dialogs can be a disadvantage in large models. For instance, you may want easy access to parameters in blocks that are scattered in multiple hierarchical layers throughout the model. Or you might want to provide a more accessible interface for other users of the model. ExtendSim overcomes these problems by giving you freedom to *clone* dialog items, tables, and graphs and place them in a more convenient location, effectively creating a link with the original dialog item.

☞ A clone is an exact copy (similar to a shortcut or alias) of a dialog item, table, or graph that behaves identically to the original.

Cloning gives easier access to inputs and outputs when you want to change settings or monitor simulation results. Dialog parameter fields, tables, text, buttons, checkboxes, and radio buttons, as well as graphs and data tables from blocks in the Chart library can all be cloned. Cloned items can be placed in the model worksheet, notebook, or a hierarchical block's worksheet, and text labels can be used to make the cloned item easy to understand.

You can clone multiple items from the same location or clone the same item to more than one location. Every clone acts exactly like the original: if you change the original or any clone, all instances are updated immediately.

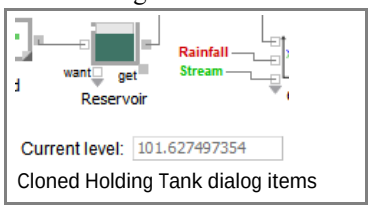
Using cloned dialog items puts you in direct control of your models. Clone the dialog items to a centralized location so you can easily change parameters. Label the clones so they are well-documented and use them as the simulation runs, such as clicking buttons or entering values in text entry boxes. For example, clone a Constant block's Constant value field to change the value between simulations, making it easier to test different assumptions without having to open the block's dialog. Or have an area of the model or a notebook that lets you monitor several numeric values at a time, rather than just using a chart to watch the simulation results.

⚠ Clones that display changes during a run will cause the simulation to run slower.

How to clone a dialog item

The Point and Click technique is used to clone an item onto the model or hierarchical block's worksheet or into a notebook.

- Open the Reservoir 1 model (located in the Documents\ExtendSim\Examples\Tutorials\Continuous folder)
- Open the dialog of the Holding Tank block, labeled Reservoir using the Block/Text cursor
- Select the Clone cursor either by right-clicking or from the toolbar
- Select both the Current level text and its parameter field by either holding down the Shift key as you click each item, or dragging a frame around the two items
- Then click on the model window (or Notebook) where you want the clones to be located. Clones can be moved, resize, and aligned after they are placed.
- Close the Holding Tank block's dialog.



Cloned Holding Tank dialog items

- Run the simulation. The cloned item will display the changing Reservoir level as the simulation runs.

☞ If the model runs too quickly to see the change, go to Run > Simulation Setup > Setup tab and change the end time to something much larger, such as 1000, then run the simulation again.

Using cloned items

Once you have a cloned item, you can use the Clone cursor to drag it almost anywhere you want. Many people prefer to have all the items together at one side of the model window, for example, or to clone all dialog items into a notebook or into a hierarchical block's worksheet.

⚠ You cannot move a cloned item from one hierarchical block to another unless the target hierarchical block contains the source hierarchical block.

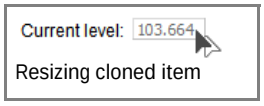
Moving

To move a cloned item, right-click or choose the Clone tool from the toolbar, click on the item, and drag it. To align two or more clones, select them and use the Alignment tools.

To move a cloned frame and all the dialog items it contains, use the Clone tool to frame- select the frame and everything within. Then click on the frame using the Clone tool, and drag to reposition it and its contents.

Resizing and deleting

To resize a cloned item, choose the Clone tool and click on the cloned item so that the resizing handle appears in the lower right side. Then click and drag the handle to change the size and shape of the clone.



Resizing cloned item

To remove cloned items from a model, simply select them (using the Clone tool) and press the Delete key or choose Edit > Clear.

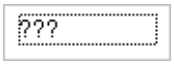
☞ If you delete the block from which you cloned a dialog item, the cloned item is automatically deleted.

Locating the original

To find and open the dialog from which an item was cloned, choose the Clone tool and double-click the cloned item.

Unlinked clones

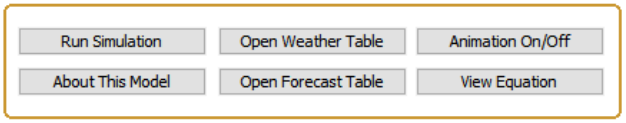
If a block's structure is modified by deleting a dialog item or changing the tab order of the dialog items, clones associated with that block may become *unlinked*. In this case, the clone will be grayed out and "???" will appear in place of the cloned dialog item.



Unlinked clone

Choosing the clone tool and double-clicking the unlinked clone will cause an alert message to appear and the dialog of the block from which the clone originally came will open. You may then select a replacement item to clone, or simply delete the clone if it is no longer needed.

Creating a dashboard interface



In addition to permitting cloned items in a notebook or model worksheet, ExtendSim lets you add customized buttons, popup menus, controls, and more to facilitate user interaction with a model. For instance, the model window shown on the right has a custom button (Run Simulation), a custom popup menu for the Inventory Strategy, and a cloned dialog item for the average WIP.

Buttons

The Buttons block (Utilities library) makes it easy to create buttons that will activate frequently-used commands. A number of predefined buttons can be selected from the popup menu in the block's dialog; you can also create custom buttons.

Predefined buttons

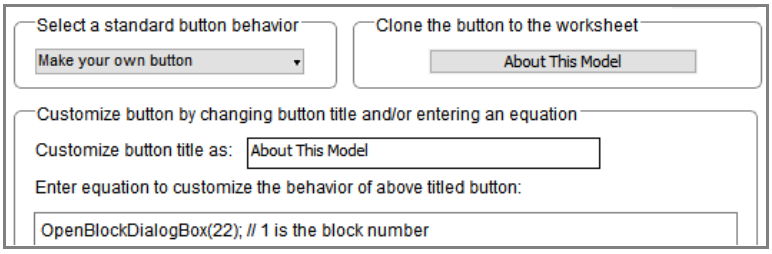
The predefined buttons are:

Animation On	Animation Off
Animation On/Off	Open Notebook
Run Simulation	Pause Simulation
Save Model	Open Block
Open H-block	Open Database
Open Database to Tab	Run Optimization or Scenarios

☞ The predefined Animation buttons only control 2D animation.

Custom buttons

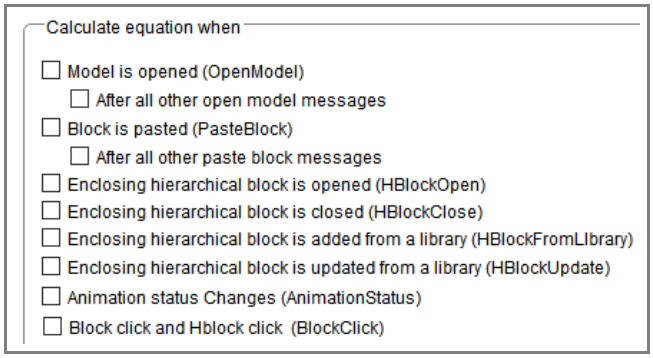
You can also create a custom button for a specific purpose by changing the label and/or its equation.



For example, the Buttons block dialog shown here has an *About This Model* button. After the button is cloned to a worksheet, clicking it opens a hierarchical block (objectID # 22) that has text and pictures describing the model, as shown below.

Using the Buttons block

To use a button, select or create the button in the Button block's dialog, then clone the button from the dialog to the model worksheet, notebook, or hierarchical block. Whenever it is clicked the button will execute the command during the enabled conditions on its Options tab, shown here.

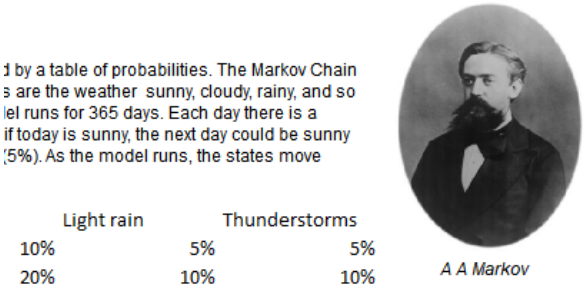


Each button requires a separate Buttons block. To store the block, place it anywhere on the model worksheet or within a hierarchical block. A common place is to store the Buttons block behind its cloned button.

Examples

The Buttons block is a favorite because there are so many things you can do with it. To see some of the ways it can be used, go to the Markov Chain Weather Simulation model located at Documents/ExtendSim/Examples/Continuous/Standard Block Models. On the Worksheet, scroll down until you see the 6 Buttons blocks. They provide the following functionality to the 6 buttons at the top of the model:

- Opening to a specific database table (Weather or Forecast)
- Causing the simulation to run
- Turning animation on and off
- Opening to the model's notebook, which contains a clone of the equation used in the model
- Opening a hierarchical block that gives information about the model as shown here

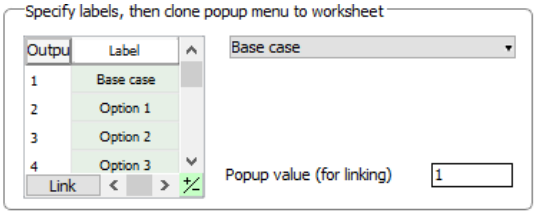


A A Markov

How To

Popup menus

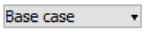
The Popups block (Utilities library) lets you create a popup menu with custom options that a user can select from to control model behavior.



To create a popup menu, enter a name for each option in the block's Label column. Then clone the menu item from the block's dialog to a model worksheet, hierarchical block's worksheet, or a notebook. Finally, connect the Popups block's output to the input of the block you want to control.

You can create a popup menu with as many options as you want. When the user selects an option from the menu, the Popups block outputs the corresponding numeric value from the first column in the dialog's table.

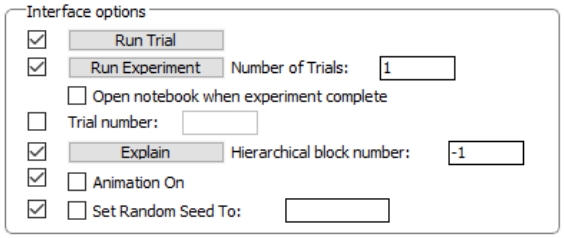
For an example where the Popups block is used, see the Monte Carlo model, located in the folder \Documents\ExtendSim\Examples\Continuous\Standard Block Models. The "Base case" popup menu in the upper right corner is cloned from a Popups block.



☞ You can also link the table in the Popups' dialog to an ExtendSim database table, creating a popup menu for selecting a database record.

Model Interface template

The Model Interface block (Utilities library) is a template for creating a custom interface to a model. As seen on the right, checkboxes in its dialog allow you to customize which interface elements will be cloned onto the model worksheet.



Run Model and Pause Sim

While the Run/Pause and Stop tools in the toolbar are most commonly used, the Run Model and Pause Sim blocks (Utilities library) give the user more customized control over model execution.

These blocks have settings in their dialogs that determine what happens when a model is run or paused. And you can clone dialog items to the model worksheet as part of creating a user interface. For more information, see "Blocks that control or monitor simulation runs" on page 91.

☞ See also the Notify block, discussed on page 77, which can be used to stop a simulation and alert the model user.

Control blocks

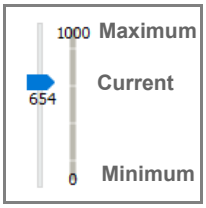
The Utilities library has three blocks—Meter, Slider, and Switch—that are used to control other blocks and show values as the simulation runs. The Slider and the Switch are used to set values in models. The Meter is used to see values as the model is running.

Meter

The Meter block (Utilities library) visually displays how a value varies between a specified maximum and minimum. Set the maximum and minimum values in the block's dialog or by connecting other blocks (such as Constant blocks from the Value library) to the top and bottom inputs. The clone from the block's dialog is shown here.

This is a good control to use if you want to see values while the simulation runs but you don't need to save them to a graph.

Slider



The dialog of a Slider block (Utilities library) lets you set minimum and maximum levels and slide an indicator along a scale to change the value of its output.

The current value is displayed on the block's icon and on its clone, and is output through the block's middle output connector. The block also has connectors to output the minimum and maximum values.

The Slider is useful to output a number within a range when the number does not need to be exact or to experiment with model parameters. For example, if you have an Activity block where you want to specify the delay as the simulation is running and a delay of 8 is slow but a delay of 2 is fast, set the Slider with 8 as the maximum and 2 as a minimum. Then clone the Slider onto the worksheet and move the indicator up and down as the simulation is running.

Switch

The Switch block (Utilities library) can be used like an On/Off switch to control some aspect of a model. The block's input connector is used to turn the Switch on and off; its output connector reports the status. A value of 0 (zero) indicates the Switch is off and the icon turns red; and a value of 1 (one) indicates it is on and the icon turns green. This block has several uses:

- Its dialog has a Switch that can be cloned to the model worksheet, notebook, and so forth
- A dialog parameter allows the Switch's status value (1 or 0) to be linked to a database, global array, or Excel spreadsheet
- In a discrete event model, the Switch block can send a message to a connected block to notify it of a status change

Interacting with the model user

Some ExtendSim blocks provide a convenient method for monitoring conditions in the model, requesting input from the user, and reporting changes. In addition, if you build your own blocks you can add customized alerts and prompts to display results and prompt for input data.

Notify block

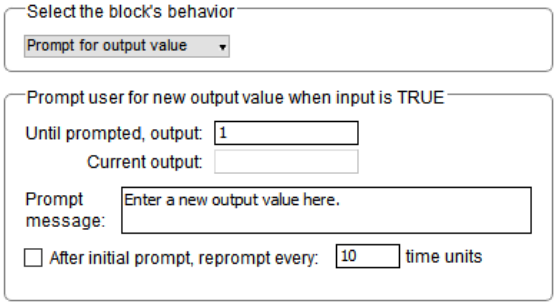
The Notify block (Value library) provides three options for sending messages to the user: Play a sound, Prompt for output value, and Stop the simulation. For all three options, the Notify block has an input connector that receives a True or False value that is used to determine whether or not to play a sound, stop the simulation, or issue a prompt. When the Prompt option is selected, the block also has an output connector to output the value the user enters after being prompted.

All three options define True as being ≥ 0.5.

Play a sound

This option plays a sound when its input connector gets a True value (alternately, you can choose to have it play only when the input value is True and animation is on). To use a sound, choose the Play a sound option and enter the sound name in the Notify block's dialog. You can enter the name (such as click or crack) for any sound located in the ExtendSim Extensions folder. To hear that sound, click the Play Sound button.

☞ Sound files you create and place in the Extensions folder must be in *.wav* format (Windows) or an *snd resource* (Mac OS). On Windows, you can also enter the name for any sound in Window's Sounds Control Panel.



Prompt for output value

This option is mainly used to prompt the user to change the block's output value. Until the input connector gets a True value, the block outputs the value specified in the dialog. When the input is ≥ 0.5, the block prompts the user to enter a different value to output.

You use this block to pause a simulation, request a value from the user, then continue the simulation using the new value. The output value can be any number, including 0. If the user clicks the Cancel button in the prompt dialog, the simulation stops.

Stop the simulation

This option is used to stop the simulation and alert the user if the monitored parameter gets a True value.

- You can customize the block to display any warning message you want, up to 255 characters. You can also set the message to include the block's number and the time the event occurred.

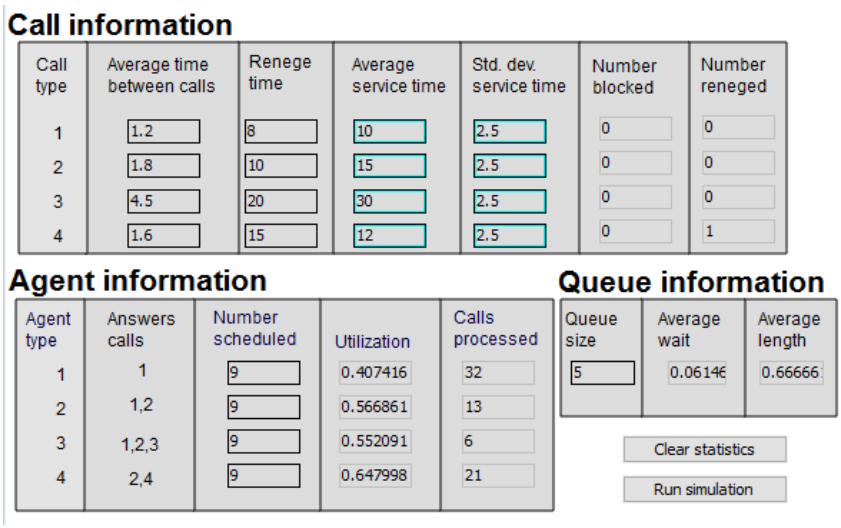
Hierarchy

Hierarchy is useful for customizing a model's interface. It reduces the number of blocks visible on the model worksheet, so models are easier to understand and navigate. It is also used to organize the model into easily-recognized components, separating the model into process elements that are more closely tied to reality.

You can encapsulate portions of a model into hierarchical block then clone dialog items to the top of the hierarchical block's worksheet for easy access. Or add custom pictures as icons for groups of hierarchical blocks that all relate to a certain part of the process you're simulating. You could also create a hierarchical Help block for the model user, as discussed at "Help block" on page 80.

For more information about working with hierarchy and customizing and animating hierarchical blocks, see "Hierarchy" on page 120.

Notebooks



A notebook is a model window you can customize to help organize and manage model data. By default each model has at least one notebook which can contain clones of dialog items, tables, and graphs (see “Cloning” on page 72), as well as text, pictures, and drawing objects (see “Text and graphics” on page 80.

Notebooks can be renamed by double-clicking their title bar. For example, changing the names to Model Inputs and Model Results, as shown at the right.

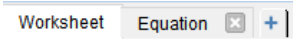
A typical use for notebooks is for collecting all of the items you might want to watch in one place. Since you can leave a notebook open as a simulation runs, use it as a central display for all of the important values in all the dialogs.

Another common use is as a control panel. Clone all the dialog items that you might want to change while the simulation is running to the notebook so you do not need to open the dialogs from the worksheet in order to make a change.

When you save a model with the notebook open, the notebook will automatically open with the same configuration the next time the file is opened. A notebook can be many pages long. With the notebook as the active window, choose File > Show Page Breaks to show the number of pages and how they would print.

You can copy the contents of a notebook as a picture which can be pasted into other applications such as a word processing document or a presentation. Simply select the items you want, then choose the Edit > Copy To Picture command to copy the picture into the Clipboard.

☞ Notebooks provide a conveniently accessible location for important information that is buried deep in hierarchy. For example, see the Equation notebook of the Markov Chain Weather Simulation model located at Documents/ExtendSim/Examples/Continuous/Standard Block Models.



Documenting models

Creating a user interface also involves adding information or graphics to clarify what is happening in a part of the model or to explain why a particular modeling approach was used. You do this by putting text, pictures, or draw objects on the worksheet or by creating a hierarchical “Help” block, as shown below.

Text and graphics

You can improve the organization of your model or Notebook by grouping elements together and using text and graphics to identify or highlight different sections. To learn more about adding text and graphics, including using colors and patterns to enhance them, see “Text” on page 106, “Graphic shapes, tools, and commands” on page 107, and “Working with pictures” on page 109.

Help block

To create a Help block, choose Model > New Hierarchical Block and name the hierarchical block “Help”. Then add explanatory text, pictures and/or draw objects to the hierarchical block's worksheet, close and save the block, and place it where you want on the model.

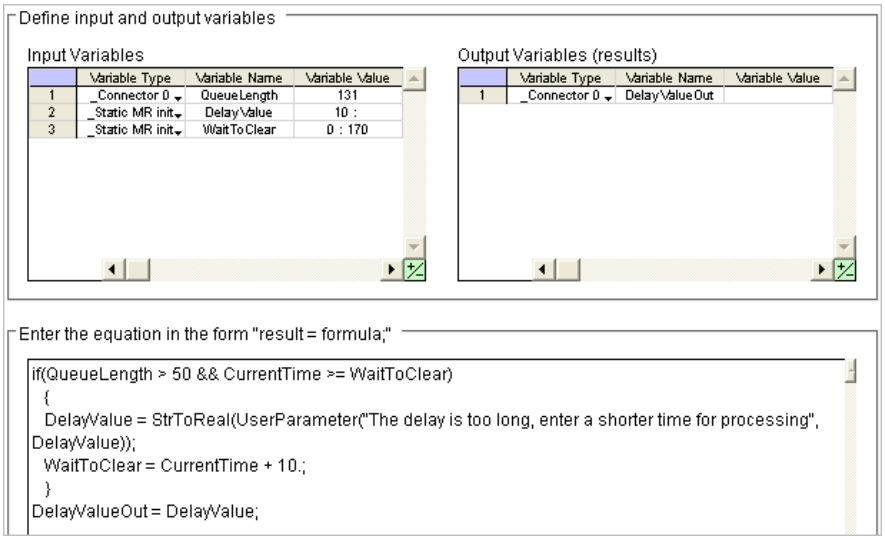
For more information about working with hierarchy and customizing hierarchical block icons, see “Hierarchy” on page 120.

Equation blocks

The Equation block (Value library) and Equation(I) block (Item library) provide flexibility and control for creating user interface elements. Most ModL functions can be called from an Equation block, including functions specifically for interacting with the model user. The blocks provide similar functionality, but the Equation block calculates its equation when it gets a value and the Equation(I) block calculates its equation when an item arrives. Equation blocks are useful when you want a more complex set of rules for the interaction than the Notify block provides.

For example, you could use an Equation block to prompt for a value if a certain condition exists and then change the result based on input from the user, like the Prompt block. But then add another condition, such as a delay between when the user is prompted.

The equation in the following screen shot causes a message to appear to the model user, asking for a new processing time if the length of the queue exceeds 50 and at least 10 time units have elapsed since the last value was requested.



☞ For more about using equation blocks, see “Equation-based blocks” on page 189.

Centralizing data in a database

Cloning is a quick and easy way to create a user interface and notebooks provide a convenient central location for data. However, for large models it may be too tedious to clone all the necessary items for easy access. ExtendSim databases are useful for organizing data for complex models into a central location.

You create databases to store parameters to be used in the model, model results, or a combination of the two. Use the linking technologies to dynamically link dialog parameters or tables with the database. Or use Read and Write blocks to dynamically access database data. The model user can easily access important data through buttons or popups placed on the model worksheet. Since databases are primarily data management systems, see further discussion at “ExtendSim databases for internal data storage” on page 232.

Additional interactive features if you program

If you develop your own libraries of blocks or want to modify existing blocks, ExtendSim provides additional capabilities for delivering messages and interacting with users. Here are some ideas from the ExtendSim Technical Reference:

- Change what is shown in a dialog depending on what occurs in the model or what the user selects or enters in the dialog. For example, you can change the text that is displayed in a block’s dialog depending on which button a user clicks.
- Some of the available functions include displaying a message, prompting the user to input a value, or making a sound. These functions can also be used for debugging the ExtendSim ModL code, although using the Source Code Debugger is the preferred approach.

- DLLs (Windows) and Shared Libraries (Mac OS) are especially handy where you want to add a feature or functionality that the ExtendSim language (ModL) does not support. For example, you could use a DLL or Shared Library to display a picture or graphic in a separate window when a user clicks a button or to create a customized sound resource based on numerical values from the model. DLLs and Shared Libraries are segments of code written in any language, such as Visual Basic or C++. These DLL and Shared Library functions allow you to call these code segment resources from within a block's ModL code and perform operations.
- COM/ActiveX Automation allows ExtendSim to communicate with other applications either as a server or a client. As a client, ExtendSim uses the external application's COM object model to communicate with it. As a server, ExtendSim responds to COM/ActiveX messages. For more information, see "ActiveX/COM/OLE (Windows only)" on page 242.

External applications as an interface

Other applications can serve as an interface to an ExtendSim model. This is accomplished by linking to ExtendSim models or blocks directly, using technologies such as ActiveX commands, or indirectly through text files.

Using an external application works much the same as using the ExtendSim database feature, since the application stores the parameters to be used in the model and the results of the simulation. For more information, see "Exchanging information with external applications (IPC)" on page 236.

How To

Model Execution

Tips for running simulations

“The future is something that everyone reaches at the rate of sixty minutes an hour.”
— C. S. Lewis

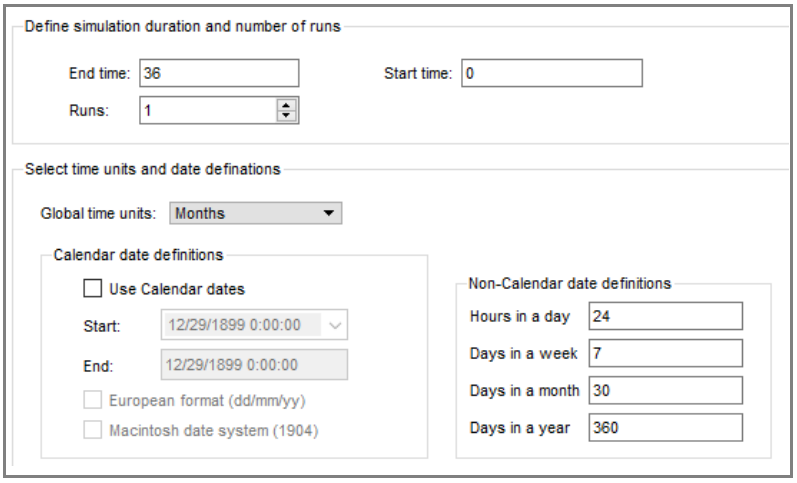
Overview

This chapter discusses how to setup and run a model, determine the number and lengths of simulation runs, use time units in models, and speed up or slow down the simulation run.

Simulation setup

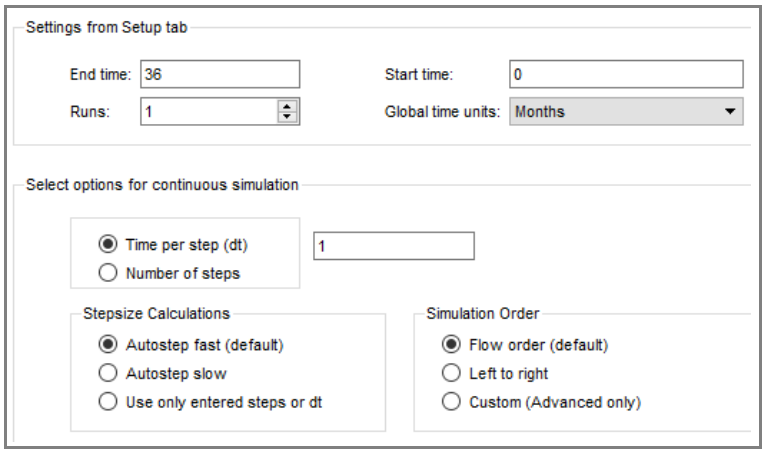
When creating a model, you will need to specify how long it will run and define other parameters. To access the Simulation Setup window, choose Run > Simulation Setup. The dialog has five tabs which are described in detail below.

Setup tab



Choice	Description
End time	The time that the simulation will end.
Start time	The current time at the start of the simulation. By default this is set to 0, since that is the most common starting point.
Runs	The number of consecutive times to run this simulation. Note: the Optimizer and Scenario Manager have their own methods of setting the number of runs; see “Length and number of runs” on page 96.
Global time units	Time unit for the entire model. Unless the global time unit is set to Generic, you can also define local time units. See “Time units” on page 93.
Calendar date definitions	For specifying calendar-based timing if *Use Calendar dates* is selected. Note: Models cannot use calendar dates if the global time unit is set to Generic, and Discrete Rate models have other restrictions. See “Calendar dates” on page 95.
Non-Calendar date definitions	Only available if *Use Calendar dates* is not selected. Used by ExtendSim to automatically convert local time units to the global, non-genereic, time unit. See “Time unit conversions (non-Calendar dates)” on page 96.

Continuous tab

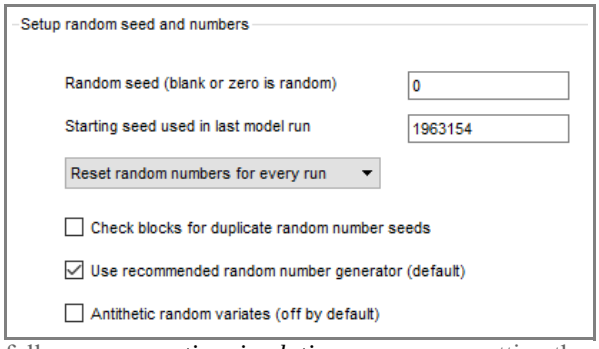


The Continuous tab is only used for changing the time step, stepsize calculation, or simulation order for a continuous model—models that do not contain an Executive block and where time advances in steps rather than when events occur.

Choice	Description
End time, Start time, Runs, Global time units	For the convenience of continuous modelers who want to change stepsize, these four settings from the Setup tab are repeated on the Continuous tab. Changing any of these settings on one tab will change them on the other. See their descriptions in “Setup tab” on page 84.
Time per step (dt)	Represents delta time (dt), or the length of time per step. This is the default choice for determining the granularity of the simulation run. For most purposes use the default setting of 1, meaning that each step will be one time unit long. A value for the number of steps is automatically calculated based on the dt entered. The value is computed as: floor(((EndTime-StartTime)/DeltaTime) + 1.5). To see this, select the *Number of steps* option after changing the *Time per step (dt)*. See the Continuous Process Modeling Quick Start guide for more information.
Number of steps	Another method for determining the granularity of time for the simulation run. In most cases, this would be a number equal to the duration (length of the simulation run) plus 1; the model calculates values once for each step and each step is one time unit long. A default value for delta time is automatically calculated based on the number of steps you enter. The value is computed as (EndTime-StartTime)/(NumSteps - 1). To see this, select the *Time per step (dt)* option after changing the *Number of steps*. See the Continuous Process Modeling Quick Start guide for more information.
Stepsize calculations	These are only used in continuous simulations that change the DeltaTime system variable, such as electronics models. Most models should use *Autostep fast*. If the blocks that you create change DeltaTime and demand more accuracy, use *Autostep slow* (which divides the calculated value for DeltaTime by 5). If your custom blocks change DeltaTime but you want to ignore the changes, select *Only use entered steps or dt*.

Choice	Description
Simulation order	*Flow order* should be used for most models. See the Continuous Process Modeling Quick Start manual for more information.

Random Numbers tab



☞ For the table that follows, ***consecutive simulation runs*** means setting the number of simulation runs in the Setup tab to something greater than 1.

Choice	Description
Random seed (blank or zero is random)	The “interface” for the random number generator. Leaving the field empty or entering a value of 0 uses a random seed. Any other value causes repeatable sequences of pseudo-random numbers and the word “Seed” will show in the model’s status bar during the run. Note: Each block that outputs random numbers will generate its own independent sequence. For more information see “Random seeds” on page 198.
Starting seed used in last model run	Reports the random seed from the previous run. To repeat the exact sequence of random numbers used during the previous run, copy that run’s starting seed and paste it into the field for this run’s random seed.
Reset random numbers for every run	When this option is selected from the popup menu, random numbers are initialized at the start of every simulation run. If the global random seed is blank or 0, a new randomized seed will be generated at the start of every consecutive simulation run. If the global random seed is a positive integer, the same seed value will be used for every consecutive run.
Continue random number sequence	When this option is selected from the popup menu, the sequence of seeds for each consecutive simulation run is continued from the previous run.

Choice	Description
Use database table__seed for values	When this option is selected from the popup menu, the starting seed for each consecutive run is read from the __Seed table in the selected ExtendSim database (note that Seed is preceded by 2 underscore characters). This option allows you to specify exactly which seeds will be used for each simulation run. It also is a convenient method for recording a set of seed values. To specify a set of random seeds, create a table in a database with one field and a number of records equal to the number of simulation runs. The model will access sequential records for each consecutive simulation run, and that record's value will be the seed value for the run. If you leave the records blank or enter zeros, ExtendSim will fill the table with seed values on the first run. It will then use those seed values for each subsequent set of runs.
Check blocks for duplicate random number seeds	At the start of the simulation, all blocks that use random numbers are checked to make sure that no two blocks are using the same seed values. This option is not necessary unless the *use block seed* option is selected in the individual blocks.
Use recommended random number generator (default)	When checked, ExtendSim will use the recommended *Minimum Standard* random number generator. When unchecked, ExtendSim will use the optional *Schrage random number generator* used in earlier versions (backwards compatibility). For details about the ExtendSim random number generator see "Random number generators" on page 197.
Antithetic random variates (off by default)	Generates antithetic variates for possible variance reduction using multiple runs. It can be controlled by the Optimizer block (Value library), or by custom blocks, to alternate on and off in alternate runs using the AntithetiCRandomVariates global variable. See the ExtendSim Technical Reference for more information. This setting should be off under normal use.

Comments tab

Use this tab to enter comments about the simulation run.

Running a simulation

In the Quick Start guides you learned the basic steps in running a model. Some additional points are discussed in the following sections.

Menu commands and toolbar buttons

Use the right-click menu, the Run buttons in the Model toolbar (as shown here), or the commands under the Run menu to configure a simulation run and to start, pause, resume, and stop a simulation.

Run modes

There are two modes toggled using the Run Mode toolbar button:

1) To run a single model as quickly as possible, even if you are running that model multiple times, use the default fastest run mode as shown here and in the Model toolbar above.
2) To run multiple different models at the same time, use the multi-threaded run mode . This takes advantage of the ExtendSim multi-threading capability—the user interface is processed in one thread and the processing for each model occurs in separate threads. In this mode, the Run Simulation button changes to .

☞ Whichever mode is active when the model is save, is saved with the model.

Run operations

There are two "Run" operations:

1) Run Simulation. Starts a simulation run, or the first of multiple simulation runs, for the active model window. Note that the toolbar's Run button:

   - Is a single green arrow if the Run Mode (discussed above) is set to fastest run. It has multiple arrows if the Run Mode is set to multi-threaded.
   - Changes to a Pause button (shown on the left) while the simulation is running and to a Resume button (shown on the right) while the simulation is paused.

2) Run Optimization or Scenarios. Runs an optimization or scenarios. This command is disabled unless there is an Optimizer or Scenario Manager block (Value library) in the model. See "Optimization" on page 158 and "Scenario analysis" on page 143. Also note that you cannot run optimization and scenarios at the same time. ExtendSim will warn if you try to do this with both an Optimizer and a Scenario Manager block in the model.

The Run menu commands are discussed at "Run menu" on page 565. Buttons to activate the Run commands are shown at "Model toolbar" on page 569.

Running a model multiple times

It is common that you will build a model and run it repeatedly. Running a model multiple times gives a range of values indicating possible outcomes and facilitates model analysis. For example, you would do this when the model has random inputs, for Monte Carlo simulations, or when performing sensitivity analysis.

☞ It is most common to use the Simulation Setup > Setup tab to set a model to run multiple consecutive times and to run the model with Run Mode set to Fastest.

- As discussed in "Constants and randomness" on page 7, models with one or more random inputs are called *stochastic* models. Stochastic models are run repeatedly and then analyzed statistically to determine a likely outcome.

- *Monte Carlo simulation* is commonly defined as applying random behavior to a static or dynamic model and evaluating the results over a number of trials or observations. Applying Monte Carlo simulation techniques to a dynamic model is also known as *stochastic* modeling, which is discussed above. You can build both static and dynamic models with ExtendSim. For more information, see "Monte Carlo modeling" on page 14.

- When you perform *Sensitivity Analysis* on a model, you select a variable for analysis and vary it to determine how much of an impact the variable has on the model as a whole. Sensitivity analysis is discussed in detail in "Sensitivity analysis" on page 138.

- Simulations are also run multiple times as part of *Scenario Analysis* (strategically examining the outcome of different model configurations) and for *Optimization* (finding the ideal value in a model). Note that for these technologies, the runs are setup based on settings in the block's dialog, not based on the Simulation Setup command. Scenario analysis is discussed on page 143; Optimization is discussed on page 158.

You should plan on running a simulation at least 3-5 times to estimate the variability in output whenever a model contains random variables. Then you can use statistical sample-size calculations (discussed in "Determining the length and number of runs" on page 98) to determine how many additional simulation runs are required to obtain an accurate estimate of the model's performance. Some tools in ExtendSim for evaluating multiple simulation runs are:

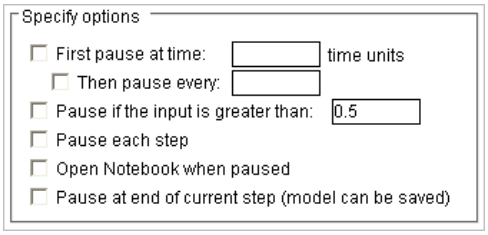
- Blocks in the Chart library
- Confidence intervals (reported by the Statistics block in the Report library) that show the probability that a certain range captures the true mean for a simulation model result

☞ See also "Length and number of runs" on page 96.

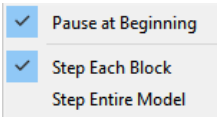
Stepping through a model

Sometimes you want to pause a simulation run to observe something of special interest, or step through the run to verify the action. There are several ways to accomplish this:

- Use the Pause Sim block (Utilities library) and clone one of its buttons (Run, Pause, or R/P/R) onto the worksheet to control the simulation run. This block causes the simulation to pause when certain conditions are met; the conditions are specified in the block's dialog (seen at right). These options are particularly useful when stepping through a model.



- Use the commands in the Run > Model Debugging menu to tell ExtendSim how far to go when the Step button or command is clicked.



  - The *Pause at Beginning* command automatically pauses the simulation after it starts so that you can step through the run from step zero. The first pause occurs before the first step but after the initial model processing (initialization, error checking, etc.) While the simulation is paused, the word **Paused** appears in the model's status bar.
  - The *Step Each Block* command controls the behavior of the Step command or button so that you can step through a simulation block by block. Only active when the Pause button or command has been activated.
  - The *Step Entire Model* command controls the behavior of the Step command or button so that you can step through an entire cycle of all the blocks in the model. Each Step command starts at the selected block and continues the simulation run until the execution order returns to the original block. Only active when the Pause button or command has been activated. This is a good way to examine what happens in the intervals between when a block is called.
- Slow down the simulation run as discussed on page 100.
- Give the command Run > Pause, or click the Pause/Resume button in the toolbar, as the simulation runs. Then give the command Run > Resume or click the Pause/Resume button to continue execution. This method works well for pausing a run one time to observe some-

thing of interest, but is not that useful for stepping through the entire model since it depends on your dexterity.

- Finally, if you use equation-based blocks you can set debugging points in the equation to step through model calculations.

Other points when running models

- When a model finishes running the status bar will state "Finished at TimeX" and display how long the simulation run took. If the device's sound is on and ExtendSim is set in the Options command to make a sound at the end of the run, there will also be a beep.

☞ If you run a model multiple times, the status bar reports the times for each run during the run, and the cumulative time at the end of the runs. Also consider using the RealTimer block (Utilities library) if you want more precise information.

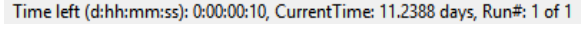
- You can run multiple models simultaneously. However, the Run > Run Simulation command only activates one model at a time. See "Working with multiple models" on page 100 for how to activate multiple model runs.
- Windows and dialogs, such as a notebook, a block's dialog, or a hierarchical block's worksheet, can be left open when the model runs. However, you cannot run a simulation with a block's structure open. In addition, leaving too many windows open (especially if they have parameters that update constantly) can slow simulation speed.
- The ExtendSim application communicates with blocks in a model and the blocks can communicate with each other by sending and receiving messages before, during, and after a simulation run. This unique communication method allows powerful modeling constructs and results in models that are visually logical and easily understood. See also "How ExtendSim passes messages in models" on page 101.

For more information about running models under different circumstances, see the following sections:

- "Animation" on page 110
- "Scenario analysis" on page 143
- "Sensitivity analysis" on page 138
- "Optimization" on page 158
- "Dotted lines for unconnected connections" on page 214
- "Model reporting" on page 216
- "Confidence intervals" on page 138

Status bar

ExtendSim shows information for the active model worksheet in a Status Bar at the bottom of the application window.



When you start the simulation run, ExtendSim displays some initial status information in the form of messages that appear momentarily in the status bar area. Depending on the speed of your computer, you may see the following messages: Wait, Checking Data, or Initializing Data. These messages inform you of status as ExtendSim checks and initializes the model prior to starting the run. On fast computers, the messages may appear too quickly for you to read.

While the simulation is running

The Status Bar changes as each running model is brought to the front. The following information is displayed while the simulation runs:

- An estimate of the actual time left in the simulation (expressed as hours:minutes:seconds) so you can determine how long it will run.
- The current time of the simulation in simulation time.
- The number of the simulation run (internally the numbering of simulation runs starts at 0 but the number is displayed as starting at 1 in the Status Bar).
- The word "Seed", if the model has been set to run with the same sequence of random numbers.

These values are determined by the entries in the Simulation Setup dialog described in "Setup tab" on page 84.

☞ The time remaining shown in the Status Bar is only an estimate. For continuous simulations, it is usually accurate. For discrete event and discrete rate simulations, however, it can be inaccurate, as discussed below.

After the run has finished

Once the run has completed, the Status Bar reports when the run finished and the total elapsed time it ran (for all runs, if there were multiple runs). The Status Bar also reports the settings for Runs and End time as specified in the Simulation Setup dialog.

Timer inconsistencies (event-based models only)

In discrete event and discrete rate models, the value for the actual time left sometimes varies from one moment to the next, especially at the beginning of a run when it is impossible to know when events are going to happen and thus how long it will take to run a model. The estimated time remaining may even increase if the simulation starts to run slower because items are being delayed more or because more events are being generated. The Status Bar will sometimes show the phrase *Initializing Data* during the initial steps of a discrete event simulation if there is a great deal of fast activity (for instance, if you use preprocessing).

Since it is only an estimate, do not be concerned about the value in the Status Bar; the model is running correctly regardless of the value shown there. However, if the timer continues to increase throughout the run and the simulation clock does not advance, it may indicate that there is an infinite loop in the model. Running the model with 2D animation on may help identify the cause of this.

Blocks that control or monitor simulation runs

In addition to the menu commands and toolbar buttons and the Status Bar, several blocks provide customized control over or information about the simulation run. And as is true for other blocks, you can clone dialog items from these blocks onto the model worksheet as discussed in "Cloning" on page 72.

Buttons (Utilities library)

Creates buttons to activate frequently-used commands such as running a simulation, opening a notebook, and saving the model. Since this block is most often

used to create a user interface for a model, it is discussed in “Creating a dashboard interface” on page 74.

Pause Sim (Utilities library)

Pauses the simulation when certain conditions are met. You can clone the block’s Pause, Resume, and R/P/R (Run/Pause/Resume) buttons to the worksheet. Furthermore, this block’s options give more flexibility than cloning the Buttons block’s Pause Simulation button (discussed on page 74). See “Stepping through a model” on page 89

RealTimer (Utilities library)

Reports information about simulation run times and profiles block usage. It should be placed at the far right on the model worksheet. Its Duration tab reports the duration of the current run as well as a specified number of previous runs. Its Profile tab reports the proportion of simulation time each block used. At the conclusion of the run, the dialog opens to display the run duration in hours, minutes, seconds, tenths of seconds, and ticks (sixtieth of a second). Block profiling information is stored in a database table.

Run Model (Utilities library)

Runs the simulation when the *Run Simulation Now* button is pressed. The options in this block give you more flexibility than cloning the Run Simulation button from the Buttons block (discussed at “Buttons” on page 74.)

Time Sync (Utilities library)

Synchronizes the model to run in real time. It does this by pausing on each simulation step until the amount of simulation time that has passed equals the amount of real time that has passed. This is only effective if the model is running faster than real time. If the model is running slower then real time, the block will have no effect. The timing starts at the beginning of simulation execution after all of the blocks have been initialized.

Notify (Value library)

This block has an input connector that receives a True or False value that is used to determine whether or not to play a sound, issue a prompt, or stop the simulation and alert the user. It is discussed more at “Notify block” on page 77.

Timing

Unless you stop them earlier, all simulations run for a specified period of time. ExtendSim determines the *duration* of a simulation run based on the values entered in the Run > Simulation Setup > Setup tab; the duration is the period from the start time to the end time.

Continuous simulation timing

In continuous simulations, the duration is divided into intervals or *steps* of equal length, where start time is the first step and end time is the last step. The length of time, in time units, for each step is known as *delta time* or *dt*. The delta time determines how frequently model data is recalculated. For more information, see the Continuous Process Modeling guide.

Discrete event simulation timing

In discrete event and discrete rate simulations, ExtendSim progresses from start time through end time through a series of events. Time progresses from one event to the next and the time between events is unlikely to be equal. The application calculates the time between events internally based on when an event occurs. The number of steps in a discrete event or discrete rate model is the number of events.

Simulation order *(continuous models)*

The Run > Simulation Setup > Continuous tab allows you to choose the order in which ExtendSim executes block data for continuous models. The choices are *Flow order* (the default), *Left to right*, and *Custom*.

It would be unusual to change the simulation order from the default choice, Flow order.

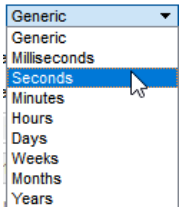
Since this information pertains to running continuous models, it is discussed in the Continuous Process Modeling Quick Start guide.

Time units

Time is the most common unit of measure in simulation. Each model has its own time unit that is managed in the Simulation Setup dialog.

Global time unit

The global time unit is the base unit for the entire model, defining the units for the start and end times of the simulation.



There are two ways to specify a global time unit:

- Use the *Generic* time unit as the default for the whole model.
- Select a specific global time unit, such as hours or seconds, to be the default for the whole model.

Global time units are set by going to the Run > Simulation Setup > Setup tab and selecting and defining the unit of time. Once you select a global time unit, all blocks with time-based parameters, including those subsequently added to the model, use it by default as their local time unit.

Using a generic global time unit

By default, the non-specified “Generic” time is a new model’s global time unit. Generic is a conceptual unit of time and has whatever meaning you give it. Generic time is the same for the whole model and for each block that has time-based parameters.

This option is used by most modelers to quickly create a model without having to select a specific time unit. It is also commonly used when the model’s blocks use the same time units as the model.

With the Generic global time unit, time-based blocks do not have a local time unit. Instead, all blocks that include a time-based parameter will specify generic “time units.”

Make sure to maintain all time-based parameters throughout the model in that same unit of conceptual time. For example, if you have a model that simulates a factory over the course of three hours and your conceptual time unit is minutes, set the end time of the simulation to be 180 in the Simulation Setup dialog and enter parameters based on minutes in block dialogs.

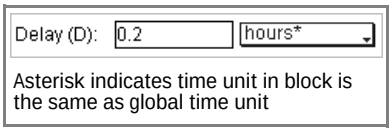
The Comments tab in the Simulation Setup dialog is a convenient place for noting what unit of time Generic represents in your model.

☞ Models cannot use Calendar dates or specify a local time unit if the global time unit is set to Generic.

Using a specific global time unit

In the Run > Simulation Setup > Setup tab you can change the default global time unit to be something specific, such as hours or seconds. Using a specific global time unit can add to the understanding of the model, since it allows you to specify parameters based on different time units throughout the model.

When a specific global time unit is selected, that setting becomes the default global time unit for the model as well as the default local time unit within time-based blocks. There will be an asterisk (*) next to the name of the time unit in the dialog of each block.



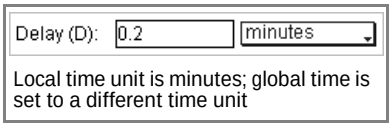
As discussed below, when the model uses a specific global time unit, you can change the local time unit to be anything you want.

☞ If you change the global time unit in the Simulation Setup dialog, be sure that the new time unit is appropriate for all blocks that use the default time unit. For example, if the global time unit was in hours and you change it to days, a block that used two hours will now use two days if it is set to use the default time unit.

Local time unit

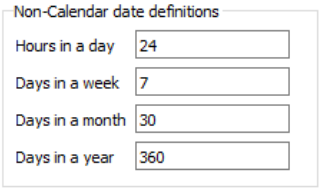
You can only change the local time unit in a block if the model is using a non-generic global time unit such as minutes, as discussed on page 94. By default, a block's local time unit will be the same as the global time unit for the model as a whole. For instance, if you change the global time unit in the Simulation Setup dialog, it will also change in every time-based block. However, as soon as you select a different time unit from the popup menu in a block, the local time unit changes from the default to that new time unit.

You can thus choose any unit of time for each time-based parameter in each block in the model. This means you can specify the parameter's time locally as milliseconds, seconds, minutes, hours, days, weeks, months, or years, regardless of the global time unit.



☞ In continuous and discrete event models, a local time unit applies only to its associated time-based parameter. For instance, an Item library block could have two local time units, each associated with a different parameter. In discrete rate models however, the local time unit applies to the entire block. There can only be one local time unit for a discrete rate block.

If a local time unit is specified, ExtendSim will automatically convert parameters based on the local time unit's relationship to the global time unit. If the model is using Calendar date definitions (see "Calendar dates", below) that conversion will be based on the calendar—the actual number of days in the specific month selected. If the model is not using Calendar dates, the conversion will use the non-calendar definitions entered in the Simulation Setup



dialog, as shown at right and discussed in “Time unit conversions (non-Calendar dates)” on page 96.

Selecting a specific global time unit provides consistency when adding new blocks to the model, since every new block will initially be using the same time unit. However, in some cases you will not want to keep all parameters in default mode. Explicitly selecting a time unit (even the same unit of time as the global time unit) for each local parameter in the model is a safer choice. Then, if you subsequently decide to change the global time unit, you will not accidentally affect the simulation results.

☞ If you do not want the local time unit to change based on a change in the global time unit, select a time unit from the popup that is not the default but which has the same name as the default global time unit. For instance, select *hours* from the popup, rather than *hours**.

Calendar dates

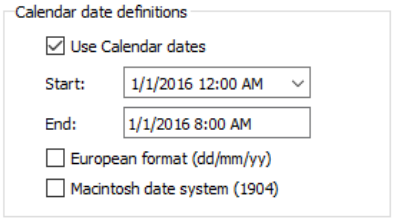
Sometimes it is helpful to have the duration of simulation expressed in terms of a calendar-based starting and ending time. Calendar dates is a time and date format that corresponds to the calendar and a 24-hour clock. It is an option that you can choose in the Simulation Setup dialog, along with the Start time, End time, and Global time unit entries.

⚠ Calendar dates are not available if the global time unit is Generic or Milliseconds for most models, or if anything other than Seconds, Hours, Days, or Weeks has been selected as the specific global time unit for a discrete rate model. (If Calendar dates has been selected, individual Rate library blocks will not be able to select Months or Years as their local time unit.)

To enable Calendar dates:

- Choose Run > Simulation Setup > Setup tab
- Select Use Calendar dates
- Click the Start field

A calendar appears from which to select a date and time that represents the beginning of the simulation. ExtendSim will use that information and the other entries you have made in the Setup tab to calculate the *End* date. (Of course, all dates are in simulation time, not in real time.)



There are two other options that affect Calendar date definitions. The *European format* option places the day before the month. The *Macintosh date* option, selected by default for Mac OS, is needed when Excel is set to use the “1904 date system” (see Excel > File > Preferences > Calculation.) If “1904 date system” is not checked in Excel on the Macintosh, uncheck this option in ExtendSim.

Selecting *Use calendar dates* for a model can also affect blocks that deal with time. For example, if Calendar dates is enabled, you can use the calendar format to create a schedule in the Create block (Item library). An example of this is the block named “Schedule” in the Scheduling Resources model located in the folder Documents\ExtendSim\Examples\Discrete Event\Resources and Shifts. (The model is discussed in “Scheduling resources” on page 370.)

The following allow you to select Calendar dates:

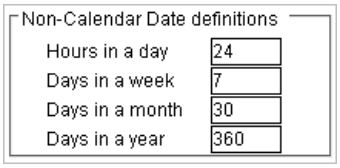
- Create (Item library) when it is set to *Create items by schedule* or *Create values by schedule*

- History and Shift blocks (Item library)
- Lookup Table (Value library) when it is set to *Lookup the: time*
- Some blocks in the Chart library
- Database tables

⚠ Do not use Calendar dates if block dialogs are set to use values smaller than 1 second (such as 0.0005 seconds), as it may affect numerical precision.

Time unit conversions (non-Calendar dates)

If a model has not been set to use Calendar date definitions when it runs, ExtendSim converts all time-based parameters from their local time units to the specific global time unit based upon constants entered in the Run > Simulation Setup > Setup tab. You can use the default settings or set your own definitions for all the time categories. Specifying that there are 8 hours in a day is an easy way to model a standard 8 hour working day. Note that, since each month is set to have 30 days, the default value for a year is 360, not 365.



Default constants in the Simulation Setup dialog

For example, assume you are using the default constants and the global time unit for the model is days, but you have selected a local time unit of hours for a block's time-based parameter. When the model runs, ExtendSim will cause that parameter to be divided by 24, converting it into the appropriate value for one day.

☞ Time unit conversions are only applicable if the model uses specific global time units and does not use Calendar dates.

Other Units

Flow units

Flow units (gallons of liquid, cartons of cereal, rolls of paper, tons of flour, and so forth) can be defined in the Rate library blocks. For more information, see "Units and unit groups" on page 441.

Length

The Transport and Convey Item blocks (Item library) utilize a unit of distance (feet or meters) to calculate the delay required to move an item from one point to another. Similarly, the Convey Flow block (Rate library) uses a user-defined unit of length to calculate a delay for moving units of flow.

Length and number of runs

An important consideration when building a model is to determine how long and how often the simulation should be run. Your entries for the *End time* and *Runs* in the Simulation Setup dialog will depend on four factors:

- Whether the system being modeled is *terminating* (has a natural end point) or *non-terminating* (has no obvious end time)
- The period of interest (what portion of time you are modeling)
- Your modeling objectives (estimating performance, exploring alternatives, or other)

- How the samples for statistical analysis are obtained (from running multiple short simulations or by analyzing portions of one very long simulation run)

A brief discussion of terminating and non-terminating systems follows. A comprehensive discussion on this matter is beyond the scope of this manual.

Terminating systems

Some systems obviously lend themselves to a natural determination of end time. For instance, most service operations have a point at which activities end. In these terminating systems there is an obvious time when no more useful information will be obtained by running the simulation longer. For example, when modeling one day at a walk-in customer service center that is only open 8 hours each day, you could safely set the simulation end time for 8 hours. Since customers would not wait overnight in line for service, the service queue would be empty or cleaned out at the end of the day and further simulation time would be unproductive.

Because terminating systems do not typically reach a continuing steady state, your purpose in modeling them is usually to look for changes and identify trends, rather than obtain system-wide statistical averages. For instance, in the customer service center mentioned above, it is more important to determine the peaks and valleys of activity than to calculate overall averages. Basing your decisions on average utilization in this case could obscure transient problems caused by multiple periods of understaffing.

Since the initial conditions in terminating systems will have an impact on results, it is important that they be both realistic and representative of the actual system. Terminating systems are simulated using multiple runs for short periods of time using different random seeds for each run. As discussed in “Confidence intervals” on page 138, the more frequently a simulation is run, the more confidence you can have in the results.

Non-terminating systems

A non-terminating system does not have a natural or obvious end time. Models of non-terminating systems are often called *steady-state systems* since, if they are run long enough, the results tend to a steady state. In these situations, simulation runs could go on indefinitely without materially affecting the outcome. Most manufacturing flow systems and some service situations (for example, 24-hour convenience stores, emergency rooms, and telephone service centers) are non-terminating systems.

Systems that have off-shift periods, for instance a manufacturing plant that operates only two shifts a day, may still be considered non-terminating. If the operation does not clear out at the end of the second shift, but instead the first shift starts up where the second shift ended, the system is considered non-terminating and the off shift period is just ignored for modeling purposes.

The important considerations when modeling non-terminating systems involve eliminating the initial bias caused by the warm-up period, deciding how to obtain samples for statistical analysis, and determining the length of the run.

- The *warm-up period* is the period from start-up to when processes operate at their normal or steady-state level. In simulation models, start-up conditions may be unrealistic or nonrepresentative of the actual system and may bias simulation results. To overcome this, you can either wait until after the warm-up period to gather statistics, reset statistics as discussed in “Clear Statistics” on page 136, or run the simulation for a long period of time to “swamp” the biasing effect of the initial conditions.

- To obtain multiple samples for statistical analysis, you could perform repeated runs after eliminating or compensating for the warm-up bias or you could do one extremely long run and calculate statistics on results occurring during various windows of time. As with terminating systems, the greater the number of samples, the higher the confidence in the results.
- The run length of a non-terminating system depends on various factors, including how you obtain samples, your period of interest, and your modeling objectives, as discussed below.

Determining the length and number of runs

When modeling terminating systems, the length of the simulation run is usually determined by the natural end point of the process being modeled. For instance, the 8 hours that a bank would be open is modeled for 8 simulation hours. For statistical analysis purposes, however, you may want to build a model that looks at a specific time period of a terminating system. For example, you could model the bank's busiest time period (say from 11 AM to 2 PM) to run multiple times to get a better statistical picture of how the bank operates during that time period.

For non-terminating systems, the length of the run depends on how you decide to obtain your samples (as discussed above) and on your period of interest. Theoretically, a model of a non-terminating system could be run indefinitely. In reality, it is usually simulated until the output reaches an adequate representation of steady-state. For example, you would run a model of a manufacturing operation for a long enough period of time that you feel confident that every type of event happens at least several times. In other situations you might want to limit the run to an artificially short period of time. For instance, you may want to only model the time it takes the manufacturing operation to go from start-up to operating in a normal manner.

The number of simulation runs is determined by statistical sample size calculations and your modeling goals. If your goal is to estimate performance, the number of runs is determined by the required range in a confidence interval. If your goal is to compare alternatives, the number of runs is based on acceptable levels of risk.

For more information on determining the length and number of simulation runs, please refer to a simulation or statistics textbook.

Speeding up a simulation

The more complex your model, the more important it is to have it run quickly. Models become complex as you add more detail to the workings of each part, or as you run it for longer or, for continuous models only, with smaller delta time increments. Although ExtendSim runs models at extremely high speeds relative to other programs, it can never hurt to think about speed considerations. The following topics discuss some common reasons why a simulation might run slower than it should and how to speed it up.

Select fastest run time

Unless you are running multiple different models at the same time, be sure the Run Mode is set to fasted. See "Run modes" on page 87 for more information.

Decrease the display of data or movement

The most important thing to remember about running speed is that anything that causes the screen to update will inherently slow down your model. Tips to keep in mind include:

- Only use animation when you need it, since animation will slow your model down more than any other activity.

- Close hierarchical block worksheets if animation is on or if the worksheet contains cloned dialog items.
- Do not keep graphs open when running your model if you are concerned about speed. To keep a graph closed, in the Display tab of the Chart block's Dialog tab select either *does not open automatically* or *opens at end of run*.
- Close dialogs and notebooks that have parameters updating while the model is running.
- Limit the number of cloned objects that update frequently during the simulation.

Look for inefficient settings or code

In some situations the simulation settings, or a block's code or the settings in its dialog, will cause a model to run slowly.

- *Delta time*. For continuous models, consider using larger delta time increments so that ExtendSim does fewer computations. For example, if each step is a day in your model, consider making each step a week. Of course, this will not work with every model because some calculations are based on the number of steps and will thus be not accurate if you reduce the number of steps.
- *Model profiling*. In some situations a block's code or the settings in its dialog will cause a model to run slowly. For example, you might have 80,000 items arriving in a Queue that is set to prioritize its inputs, or an inefficient section of code in a block you have created. ExtendSim provides two methods for profiling blocks:
    - Use the RealTimer block (Utilities library). Place the RealTimer block at the far right of the model. In its Block Profiles tab, enable block profiling and choose a location to store the report. Then run the simulation. A database table stores information about each block in the model: name, number, label, time spent executing, percentage of total time spent executing, the size (allocated memory) of the block at the beginning of the run, the difference in the allocated memory at the end of the tun, and information about hierarchical blocks. Since profiling significantly slows the simulation, the Block Profiles tab provides options for disabling profiling during subsequent runs.
    - Profile the model using a Model Debugging command. Select the command Run > Model Debugging > Profile Block Code, then run the model. A text file is generated with the following information for each block the model uses: name, number, time spent executing, percentage of total time spent executing.
- If a *block is linked to a data source* in a global array or an ExtendSim database, it will receive a message whenever the value in that data source changes. If a simulation is running slowly, and blocks have links to data sources, consider sending or receiving link alert messages only at the start or end of the simulation. For more information, see the caution at "Dynamic linking to internal data structures" on page 229.
- Change settings in the *Display Value block* (Value library. This block may cause the simulation to run more slowly because the block pauses while it shows you information. To speed it up, set its dialog's *Wait* value to *0* ticks or seconds. You can also unselect the *Dialog opens* option if you don't want to see the dialog during the entire model run.

- *Executive block*. The Executive block (Item library) stores information about each item in the model. As the simulation is run, the Executive block allocates additional items in groups of a fixed size when necessary. In the Control tab of the Executive block's dialog, you can specify how many items are allocated at the beginning of the run and the number of additional items allocated when required during the run. The procedure of allocating additional items during the run can slow the simulation down if performed numerous times. Therefore, set the number of items initially allocated to be the maximum number of items that are expected to be in the model at any given time, plus 10%. Note that unnecessarily allocating too many items will take up available memory and can slow down the simulation.

Other factors that affect simulation speed

- Increase the amount of physical memory (RAM) in your computer. If your computer runs out of RAM, it will use virtual memory, which can slow down your model significantly.
- Using a large number of Value library blocks connected together to perform a simple calculation can slow a model down. Where possible, replace large "webs" of blocks with equation-based blocks that perform the same calculation.
- For discrete event models, you may be able to scale the number of items you generate. For example, if your model is of a factory floor, instead of generating one item for each object manufactured, you might generate one item for each set of five objects. And if your factory is high-volume or high-speed, consider using the Rate library to model it instead of the Item library.
- When using equation-based blocks, limit making string comparisons if possible.

Slowing down simulations

You might want a model to run more slowly to debug it or critically visualize what the model is doing. If this is the case, the best way to slow down your model is to turn on animation and use the Slower Animation (turtle) button.

Another example of when you might want to slow down a model is if you want to synchronize the model to real-time events. This might be required if you are receiving real-world data. To accomplish this, refer to the TimeSync block (Utilities library).

If you set it to leave the dialog open while the simulation runs, the Display Value block (Value library) is also useful for slowing down the simulation.

See also "Stepping through a model" on page 89.

Working with multiple models

ExtendSim can have multiple model windows open and run multiple models simultaneously. This facilitates copying model sections or hierarchical blocks between models and running models that communicate with each other. You can also run one model in the background while constructing another.

The Run Simulation command only works on the active model. To run multiple models, start the front model running. Then bring each model forward and start their runs. While some models are running, other models can be built.

☞ Use the multi-threaded run mode when running multiple models at the same time. See "Run modes" on page 87 for more information.

How ExtendSim passes messages in models

ExtendSim uses a sophisticated messaging architecture to signal blocks into action. While messages can originate either from the ExtendSim application or from individual blocks, it is always a block that is on the receiving end of a message. Different types of messages result in the receiving block doing different types of things.

Since messages have the potential to affect the speed and behavior of simulations, understanding the ExtendSim messaging architecture will help you build more accurate and efficient models and will make debugging models easier.

Model messages can be divided into two categories:

3) Messages typically sent from the application to one or more blocks
4) Messages typically sent between blocks.

☞ All types of blocks can receive application messages. Continuous blocks neither receive nor send block messages.

Application messages

Application messages are usually sent to blocks by the ExtendSim application. There are several types of application messages: Simulation, Model Status, Block Status, Dialog, Connector, Dynamic Link, and OLE. They are discussed completely in the Technical Reference.

When sent

The application can send message before, during, and after the simulation run, including:

- At the start of the simulation
- At each step
- When a parameter is changed
- When a dialog is opened
- If a button is clicked
- At the end of the simulation

At the start of the model run, all blocks receive application messages to check their dialogs and connections for consistency and to initialize variables, import data, or allocate any arrays used during the simulation. At the end of the simulation run, blocks are triggered to collect statistics, dispose of any temporary arrays, and update the status of the model. Other application messages are sent only when a specific event, parameter change, or dialog click occurs. The Technical Reference has a complete list of application messages, including CheckData, InitSim, Simulate, EndSim and FinalCalc.

Major messages

The major application messages are:

- PreCheckData
- CheckData
- StepSize
- InitSim
- PostInitSim
- Simulate

- BlockReceive0-9
- FinalCalc
- EndSim

Link alerts

Link alerts are a special, and very powerful, type of application message that may cause changes in a block's data. For example, if a block is linked to a data source in a global array or an ExtendSim database, it has options to receive a message whenever the value in that data source changes.

The ExtendSim application automatically sends link alert messages in continuous, discrete event, and discrete rate models. In a discrete event or discrete rate model, the alerted block may, in turn, send messages out its connectors.

Managing this message is important because linking blocks to a data source that frequently changes may slow down the simulation. If a simulation model is running slowly, and blocks have links to data sources, consider sending or receiving link alert messages only at the start or end of the simulation. For the message options, see:

- "Link dialog" on page 230 for managing the alerts when block parameters and data tables are linked to an ExtendSim database or global array
- The check box *When receive database link alert msgs* in the Equation block (Value library)
- The *data source changes* check box in the Options tab of the Read block (Value library)

Continuous model messaging

In a continuous model, the most important application message is the *simulate* message, which causes code in blocks to execute as simulation time advances. At each step of the simulation, the application sends a simulate message to every block in the model in the sequence defined by the model's simulation order. (Simulation order is determined by the application at the start of the run based on settings in the Simulation Setup dialog, as discussed in "Continuous tab" on page 85.) In the simulate message handler, each block will perform its intended calculation. For example, if *Add* is selected in the Math block (Value library), the block will add all its inputs when it gets a simulate message. Likewise, a Decision block (Value library) will check its inputs and output a true or false value based on the options selected in its dialog.

Block to block messages

Blocks can send messages to each other either through connections or "through the air." This often happens when one block has calculated a new value and wishes to alert one or more blocks to this change. Continuous blocks, such as those in the Value library, do not typically send or receive block messages. Discrete event (Item library) and discrete rate (Rate library) blocks often send and receive block messages.

Discrete event block messaging

In addition to receiving application messages, discrete event blocks communicate with each other using block messages. A variety of messages are sent between blocks during the course of the simulation. These block messages fall into the following categories:

- *Event*. Communication between the Executive and various Item library blocks in the model.
- *Value connector*. Notifies connected blocks that the value of a output connector has changed or requests updated input connector information.

- *Item connector*. Propels items through the model.
- *Block-to-block*. Updates the status of other blocks in the model.

To avoid modeling errors and simulation speed issues in discrete event models, it helps to understand how the discrete event messaging architecture works. See “Messaging in discrete event models” on page 419 for additional information.

Discrete rate block messaging

In addition to receiving application messages, discrete rate blocks intensively use messages in order to initiate the calculation of the effective rates and to propagate information through the blocks. These block messages fall into the following categories:

- *Event*. Communication between the Executive block and various Rate library blocks.
- *Value connector*. Notifies connected blocks that the value of a output connector has changed or requests updated input connector information.
- *Flow connector*. Updates the effective rate through the connected blocks.
- *Rate blocks flow*. Defines the LP area – that part of the model that could be impacted by a change in the effective inflow and outflow rates.
- *Executive flow*. Updates all the blocks in an LP area with a new effective rate.

The discrete rate technology has a complex and unique messaging architecture. For optimal performance, it is important to understand discrete rate messaging when building rate-based models. For more information, see “Messaging in discrete rate models” on page 537.

How To

Presentation

Organize and enhance models
so they communicate your ideas

“In the modern world of business,
it is useless to be a creative original thinker
unless you can also sell what you create.
Management cannot be expected to recognize a good idea
unless it is presented to them by a good salesman.”
— David M. Ogilvy

Overview

Simple models are easy to follow. But as models become larger and more complex, worksheets can become crowded with thousands of blocks, making it harder to communicate what's happening. As shown in this chapter, ExtendSim provides several features that let you organize, explore, and enhance models.

For additional ideas to enhance a model's appearance, see the chapter "Customizing the User Interface" on page 71.

Text

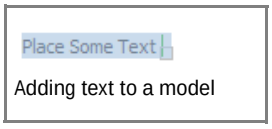
Text is used to:

- Document or enhance a model or a block
- Create the text labels used as named connections on model worksheets
- Create text connections so hierarchical blocks can communicate with the model outside

☞ All the objects in an ExtendSim model, including text, have properties so you can label them, search for them, move them programmatically, and so forth.

Entering text

To add text to your model, either:



Adding text to a model

- Double-click the location (worksheet or notebook) where you want the text to appear; that opens a text box.
- Or, select the Text Box button from the Shapes tool and click on the desired location to open a text box.

When you've finished entering text, click anywhere else in the window and the box surrounding the text will disappear.

Formatting text

The style, font, color, and size of text can be selected before the text is entered or changed after selecting the text box. Text can have color; individual letters can be styled using the Text tools.

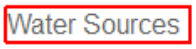
To type new text with a particular format, select the desired format (Text toolbar) and color (Fill Color button in the Shapes toolbar), *before* you start the text box. ExtendSim will remember that format every time you start new text in that model.

Text toolbar (top) and Fill Color button (bottom)

To change the style or size of *existing* text, access the text box and select the text, then choose a Text tool button, such as Bold. The Align Text button (located to the right of the style options, as shown above) is for toggling the text to be left, right, or center adjusted. By default text has a transparent background; to cause text to instead have a white (opaque) background, select the text then use the Transparent Background button that is to the right of the Align Text button.

To change the color of existing text, use the Shapes tool's *Fill Color* button—select the text in the box then click the tool and select a color. Use the handles in the corners of the text box to resize or reshape the box, for instance to have some of the text be on a second line.

☞ If you change the format or color of text while within an existing text box, ExtendSim won't use that format as the default for the model. To cause a format or color to be the default, change it before creating a text box.



By default, text does not have a border. To cause text to have a border that will move with it, use the Shapes tool's *Line/Border Thickness* button and select a size for the border. Then use the Shapes tool's *Border Color* button to select a color for the border. Putting a border around text is useful for distinguishing descriptive text from block labels or named connections. The text example above has a red border.

☞ The Border Color button in the Shapes toolbar, as well as the corresponding button in the text's Properties dialog, won't draw a visible border around the text unless the Line/Border Thickness has been set to have a width greater than 0.

Moving and copying text

To move the text, select it then drag the text box to the desired location.

When text is selected, you can use the Copy and Paste commands, then move the pasted text where you want it (or click on the window before pasting to position it in that spot). You can also use the Edit > Duplicate command or Ctrl+D to copy and paste model elements slightly offset from the original.

Drag and drop text

It is often easier to drag and drop a portion of some text than to copy and paste it. To do this, select a piece of text within the text box and drag it. When you drag the selection, an insertion point appears at the end of the cursor. Drag the insertion point to the desired location and release the cursor.

Drag and drop can be used within a text box on a worksheet, a text file, a block's structure window, and between any combination of the above with one exception: it will not work between two separate text boxes. For example, if you double-click on the worksheet to open a text box, enter text into the text box, then select a portion of the text, you will not be able to move that text to another text box using drag and drop. This is because you can have only one text box open for editing at a time.

Graphic shapes, tools, and commands

There are three sets of toolbars for creating and managing graphics, including text:

- Text tools (discussed in "Formatting text" on page 106)
- Shapes tools (discussed below)
- Alignment tools (see page 109)

Shapes tools

☞ All the objects in an ExtendSim model, including text and graphic objects, have properties. This means you can label them, search for them, move them programmatically, and so forth.

The buttons in the Shapes toolbar let you add graphic objects and text to the model, color them, and add borders.

Using the tools

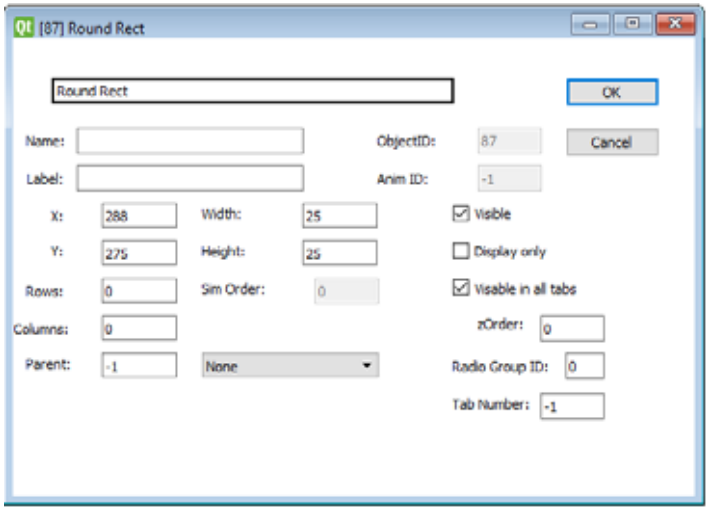
To use a Shape, select one of the buttons (shown above) then click on the worksheet where you want the graphic object to appear. Resize as necessary.

- Use the *Rectangle*, *Rounded Rectangle*, *Oval*, and *Polygon* buttons to add those shapes, as well as circles and squares, to your worksheet or a notebook.
    - To get a *square* or *circle*, place a Rectangle or Oval, respectively, on the worksheet or notebook. Then hold down the Shift key while resizing the shape. Or right-click a Rectangle or Oval shape to access its Properties, then set the object's width and height to be the same.
    - The *Polygon* button draws a shape with custom lines. Draw the desired shape by clicking multiple times on the worksheet or notebook to create hinge points, then double-click to finish the shape. Each hinge-point can be manipulated using the cursor or arrow keys

☞ Arrow keys move graphic objects 1 pixel at a time. To move objects 0.1 pixels per arrow key click, hold down the Alt key while clicking.

- *Lines:*
    - Have a default width of 1 pixel and can have other thicknesses depending on the selection in the Line/Border Thickness button, discussed below.
    - Can be colored using the Fill Color or Border Color buttons.
    - To get a perfectly straight *horizontal or vertical line*, select the Line button then hold down the Shift key while drawing the line on the worksheet or notebook.
- For *Text*, its font, style, and other properties are determined by buttons in the Text and Shapes tools, as described on page 106.
- *Fill Color* sets the color of text and shapes, including lines. Choosing this tool opens the Select Color window:
    - Choose a basic or custom color for the current object, or add a selected color as a custom color for future objects
    - The red, green, and blue (R, G, B) fields can be set from 0 (for no color) to 255 for full color
    - The *alpha* channel setting determines the amount of transparency, with 0 being fully transparent and 255 being fully opaque
- *Border Color* uses the Select Color window (discussed above) to sets the color of lines as well as the color for the borders around text and shapes. (Since text and shapes by default have no border, first select a line width using the Line/Border Thickness button.)
- By default, text and other shapes don't have a border and lines have a width of 1. The *Line/Border Thickness* tool sets the width of lines and the thickness of borders around text and shapes. The first option is for no thickness; the other options are from 1 to 5 pixels.

Resizing and controlling a shape's properties



Use the Graphics Cursor to move or edit shapes.

To resize or reshape a graphic object, select the shape using the Graphics Cursor, then click the small rectangle in the bottom right corner and drag to create the desired shape.

To resize an object proportionately, hold down the Shift key while resizing.

Every shape, including Text, has a Properties window that can be accessed by selecting the object and right-clicking or going to File > Properties. Properties let you customize and manipulate the shape. And if you program, you can cause a shape to appear or disappear, change shape or size, and so forth depending on model conditions or data.

☞ To get a square or circle, hold down the Shift key while resizing a Rectangle or Oval shape. Or, right-click the Rectangle or Oval shape to access its Properties, then set the object's width and height to be the same.

Alignment tools

The Alignment toolbar allows you to align selected text and graphics, cause an overlapping object to be sent to the back or brought to the front, flip a selection of multiple objects, and rotate a selected object either 90 degrees from where it is or (with Free Rotate) rotate it a specified number of degrees from zero.

☞ When using these tools on a mixed group of objects (for example, both text and graphics), the type of cursor you used to select the objects determines what happens. For example, to align all selected objects, use the All Objects cursor.

⚠ The Flip buttons only flip the relative positions of multiple objects that have been selected together. They do not result in the displaying of the reverse image of a graphic. Use an application such as Photoshop to create reverse images of graphics.

Working with pictures

☞ All the objects in an ExtendSim model, including pictures, have properties so you can label them, search for them, move them programmatically, and so forth.

To add a picture to ExtendSim, create or open the picture in a painting or drawing program and copy it; this places it in the operating system's internal clipboard. Then with an ExtendSim window open (model worksheet, notebook, or icon pane) choose Edit > Paste Picture to paste

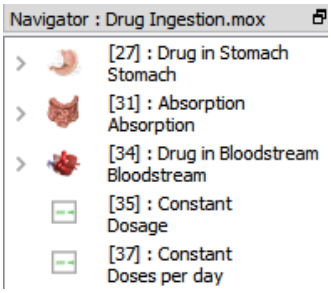
the picture into the window. For more information, see “Copy/Paste and Duplicate commands” on page 250.

ExtendSim supports multiple formats for pictures; see “Picture file formats” on page 115.

☞ It doesn’t matter what the original file format for the graphic was (JPEG, TIFF, GIF, etc.) After it has been copied into the system’s clipboard, ExtendSim treats it as a graphic object.

Navigator

The Navigator is an explorer-like window that is useful both for navigating through a model’s hierarchical structure and for getting quick access to block dialogs in large models.



The Navigator is especially useful when presenting a model to others. Use it to explore the model by drilling down through hierarchical layers to show subsystems or to quickly find blocks and access their dialogs, no matter where they are in the model.

To open the Navigator:

- Select the command Window > Navigator or click the Navigator button in the toolbar.

The name of the active model is listed at the top of the window. Each block’s icon is shown along with its information (object ID, name, and label). If no model is open, the Navigator opens as empty.

Animation

When presenting models to others you can show how graphs or cloned outputs change as the simulation progresses. However, the ExtendSim animation capabilities give a more visually descriptive representation of what is happening in the model.

Overview

As discussed in detail below, models can be animated using:

1) Blocks that have built-in animation
2) Specialized blocks for creating customized animation
3) Animation functions called in equation-based blocks or programmed into blocks you create

To see animation in a model:

- Choose Run > Show 2D Animation or click the Show 2D Animation button in the toolbar.
- Run the simulation.

Due to redrawing, animation tends to slow the simulation. You may want to leave animation on while you are testing, debugging, or presenting your model and then turn it off when you are running the model for analysis.

Use the Animation Slider, or click the Faster Animation (rabbit) and Slower Animation (turtle) buttons, to change the speed of the animation.

☞ If animation is running at its slowest speed, the Slower Animation button will be grayed out. Likewise, if the animation is already running at the highest speed, the Faster Animation button will not be available.

Blocks with built-in animation

Many of the ExtendSim blocks have animation built into them. A block's online Help says whether it is animated and, if so, which aspects are. The two types of built-in animation are:

- Animation on and around a block's icon, including displaying information
- Animation of items traveling from block to block (discrete event models only)

Animation on a block's icon

Many ExtendSim blocks have been programmed to show 2D animation on and around their icons. The amount and type of animation is specific to the block and is described in the block's Help. For example, the Holding Tank block (Value library) shows its contents relative to a minimum and maximum, the Select Item Out (Item library) shows which path items take, and the icon of the Convey Flow (Rate library) displays the distribution of its contents as the simulation runs.

Some blocks have an Animation tab for customizing some aspect of the 2D animation. For example, the Valve (Rate library) has a checkbox for displaying the rate above the block's icon and the Exit (Item library) has a checkbox to display the total number of items that have exited.

The icon of a hierarchical block can also be animated. See "Animating a hierarchical block's icon without programming" on page 113.

To see icons animate, select Show 2D animation from the Run menu or select Show 2D Animation in the Model tools toolbar.

Animating the movement of items between blocks (discrete event modeling only)

The blocks in the Item library can track the movement of items with *animation pictures* that travel along connections between blocks.

To animate the flow of items along connection lines in a discrete event model:

- Choose Run > Show 2D Animation or click the Show 2D Animation button in the toolbar.
- Choose Run > Animate Connection Lines.
- Run the simulation.

The Final Car Wash model, located in the folder ExtendSim\Examples\Tutorials\Discrete Event, shows cars moving between blocks and along connection lines.

To see animation pictures also move along the paths of named connections:

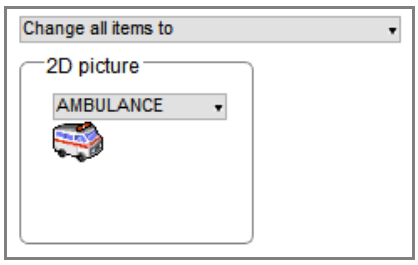
- With Show 2D Animation checked, choose Run > Animate Named Connections.

☞ Items can travel on curved paths if you right-click a connection line and choose Bézier to create a curved connection. See "Connection line appearance" on page 116 for more information.

Selecting an animation picture

When Show 2D Animation is enabled in a discrete event model, pictures representing items pass from block to block. Initially, all items are represented by the default animation picture (a green circle). You can cause a block to use any other animation picture included with ExtendSim, existing clip art, or pictures created in an external drawing package. The selection of which picture to use is made in each block's Item Animation tab.

☞ To find out how to add custom animation pictures, see "Pictures for animation" on page 115

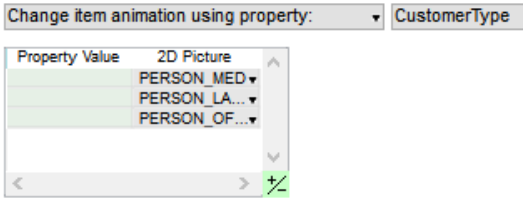


For example, the screenshot at right shows the Item Animation tab from an Activity block (Item library). There are three options for animation of items:

- Do not change item animation. In this mode, items that leave the block will have the same picture they had when they entered.
- Change all items to. This activates the 2D picture popup menu for selecting a picture to represent items as they leave the block. This is especially useful for showing the transformation of one type of item (e.g. cans) into another (e.g. cases of cans).
- Change item animation using property. With this choice, the item's animation will change depending on its property (attribute, quantity, or priority). For this option to work, the item's attribute, quantity, or priority must have been set in a preceding block. Then in this block you create a lookup table with properties corresponding to the desired animation pictures.

To cause the animation to change based on an attribute value:

- Choose ***Change item animation using property*** in the popup.
- From the popup menu to the right of that popup, select a value attribute.
- Click the green +/- resize button in the bottom right of the table. In the dialog that opens, enter the total number of attribute values that you want to assign animation pictures to. For example, if you enter "3" and click OK, 3 rows will appear in the table.



- Enter values in the *Property Value* column, then use the 2D popup menu to select which picture will animate the item when its attribute has that value.

For information about creating item attributes, see the Discrete Event Modeling section of this User Reference.

☞ As discussed on page 113, the Animate Item block (Animation library) is also useful for changing the animation picture of items that pass through it.

Blocks for customized animation

There are two ways to use blocks from the Animation library to add custom 2D animation to a model without programming:

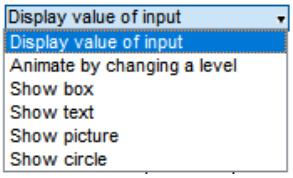
- Show animation when an item enters the block or when the block gets a value.
- Animate the icon of a hierarchical block.

Showing animation in response to model conditions

When connected to other blocks in a model, the Animate Value and Animate Item blocks (Animation library) show customized animation in response to model conditions.

Animate Value block

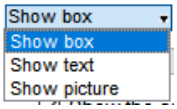
The Animate Value block can be connected to value connectors. There are several choices for how this block animates:



- Display the value that is input to the block. This is similar to using the Display Value block (Value library), except the value can be displayed in a selected color.
- Move a level up and down between limits. If you choose this, it can be displayed in a specific color that will move between the top and bottom of the block's icon. Be sure to specify maximum and minimum values that represent the entire range of possibilities.
- Flash a box, text, picture, or circle when the input goes above a specified value. To flash animation when the input is greater than or equal to a value, select a box, text, picture, or circle. Text must only be a few characters long so that it fits within the block's icon. To show an animation picture, select its name from the popup menu of pictures stored with ExtendSim; to find out how to add custom animation pictures, see "Pictures for animation" on page 115.
- For Mac OS only, play a movie when the input goes above a specified value. To play a movie in Mac OS, the movie must be stored in the Extensions folder and must be a QuickTime movie.

Animate Item block

The Animate Item block can only be connected to the item connectors of discrete event blocks. Depending on choices in its Block tab, when an item enters the block a colored box, piece of text, a selected picture, or a selected movie (Mac OS only) is shown on the block's icon. To show an animation picture, select its name from the popup menu of pictures stored with ExtendSim; to find out how to add custom animation pictures, see "Pictures for animation" on page 115. The Item tab is useful for changing the animation picture of the item traveling through the block.



Animating a hierarchical block's icon without programming

To animate a hierarchical block in 2D without programming, you create an animation object on the hierarchical block's icon, include the Animate Item or Animate Value block from the Animation library on the submodel worksheet, and select options in those blocks.

The animation block (either Animate Item or Animate Value) in the submodel reads the values from other submodel blocks and, using the animation object, controls the hierarchical block's icon animation based on those values.

Step 1: Create an animation object on the hierarchical block's icon

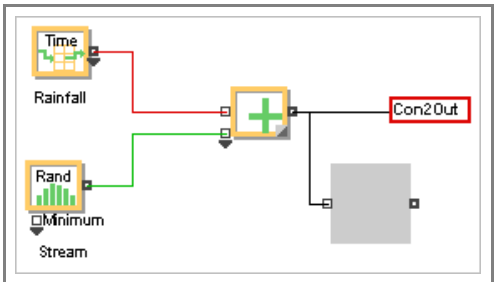
- Open the hierarchical block's structure window:
  - Select the hierarchical block in the model worksheet.
  - Choose Model > Open Hierarchical Block Structure or right-click and select Open Hierarchical Block Structure.
- Go to the block's Icon tab.
- Add an animation object to the icon:
  - If the Icon toolbar isn't already open, go to the menu command Tools > Icon

- Select the Animation Object button, the last button in the Icon toolbar.
- Click on the hierarchical block's icon where you want the animation object to appear
- Resize the animation object until it's the size you want

Animation objects display as rectangular shapes on the icon. You can add more than one animation object and you can resize them. Each animation object is assigned a number starting with 1. For example, a hierarchical block's icon with an animation object near the output might look like the image at right.

Step 2: Include an animation block in the model

- Go to the hierarchical block's Worksheet tab and determine which output you want animated.
- Select an Animate Value or Animate Item block (whichever is appropriate) from the Animation library.
- Attach the animation block to the output of the submodel's block that you want to animate. In the example to the right, The Animate Value block is used to animate the sum of the two water sources in the Water Sources hierarchical block. (If the output were an item output, you'd instead use the Animate Item block from the Animation library.)

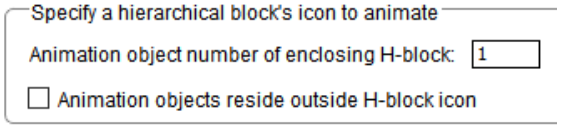


Animate Value block added to submodel

⚠ As shown here, the Animate Value block can be connected in parallel and its output does not need to be connected. For the Animate Item block items must pass through, so it cannot be connected in parallel. Instead, its item output must be attached to the item input of another block. Otherwise an item coming into the Animate Item block would have no way to exit.

Step 3: Select options in the animation block's dialog

- In the dialog of the animation block, choose the type of animation you want from the ***Specify icon animation*** popup menu,
- Enter the number of the animation object that the block is to control in the ***Animation object number of enclosing H-block*** field. For example, to animate an animation object that has number 1, enter "1" in the field.



- Close the hierarchical block's structure and save changes as appropriate.

☞ If the animation object number cannot be located on the hierarchical block's icon, ExtendSim will search the icon of the next highest enclosing hierarchical block. If it still fails to locate that animation object number, it will give an error message.

If you create custom blocks, including hierarchical blocks, use this same process of identifying the animation object number to add animation objects to your blocks.

☞ For an example of an animated hierarchical block, see the discussion of the "Markov Chain model" on page 17 or the "Animating Queue Contents" model discussed in the Discrete Event Modeling Quick Start guide (that model is not available with ExtendSim CP.)

Animation functions

If you build your own blocks, ExtendSim offers a suite of functions to design how the block will animate. For example, the Planet Dance model (located in the folder ExtendSim\Examples\Continuous\Custom Block Models) shows three planets orbiting in space. Their orbits are determined by numbers entered in the blocks' dialogs.

See the Technical Reference for a description of the animation functions. Also, see "Animating a hierarchical block's icon without programming" on page 113 for an example of how to add animation to a block's icon. You would use that same process to add animation to any blocks you create.

Pictures for animation

ExtendSim includes many animation pictures as bitmaps (Windows) or PICT resources (Mac OS). These pictures are used by discrete event blocks to show the flow of items, by the Animate Value and Animate Item blocks (Animation library) to show custom pictures, and in any custom block that calls the appropriate animation functions.

To add your own animation pictures to the available list, they must be of the correct file format and they must be stored in the ExtendSim/Extensions/Pictures folder. After placing the animation pictures in the Pictures folder, restart ExtendSim and the pictures will be available in animation picture popup menus. For information about these and other extensions, see the Technical Reference.

Picture file formats

ExtendSim supports the following image file types:

- BMP (Windows Bitmap)
- GIF (Graphical Interchange Format)
- JPG/JPEG (Joint Photographic Experts Group)
- PNG (Portable Network Graphics)
- PBM (Portable Bitmap)
- PGM (Portable Gray Map)
- PPM (Portable Pix Map)
- XBM (x11 Bitmap)
- XPM (x11 Pixmap)
- For backwards compatibility, Mac OS PICT (picture) resource files that have been converted to Windows format using the ExtendSim Mac/Win converter.

Pictures with names that start with the symbol "@" will not show up in the block's animation tab popup menu. This prevents the animation tab menus from being filled with other, larger types of pictures that are not suitable for animation between blocks.

☞ ExtendSim will scan all the pictures at the top level of the Pictures folder and add them to an internal list of images. They can then be accessed via Modl functions and through the animation picture popup menus in ExtendSim blocks.

Connections

Connections are the main method blocks use to send data, items, or flow between each other. ExtendSim has two types of physical connections, *line connections* and *named connection*s. In addition, Throw and Catch blocks allow data, items, and flow to be sent remotely between blocks. The following discussion shows how to draw and format connection lines and discusses the use of named connections.

☞ All the objects in an ExtendSim model, including connection lines, have Object ID's and properties so you can label them, search for them, move them programmatically, and so forth.

☞ The Value, Item, and Rate libraries each contain Throw and Catch blocks for sending data, items, or flow remotely without using connection lines or named connections. Use these blocks in situations where you cannot use a line or named connection, such as when sending between hierarchical levels.

Drawing a connection line

☞ The following shows how to draw a connection line so that data can be transferred between blocks. There are other ways to connect blocks so they transfer data, such as Named Connections (discussed on page 118), Throw and Catch blocks, and the Smart Block technique (discussed on page 60).

Connection lines are drawn using Point and Click. For example, to draw from one block's output connector to another block's input:

- Place the cursor over the block's output connector. Notice that the cursor changes color and shape and the output connector enlarges.
- Click once on the output connector, then move the cursor to the corresponding input connector on another block. (Connector types must be compatible!)
- When the connection lines thickens and turns blue, and the input connector enlarges, click once on the input connector. This draws a connection line between the two blocks so that information is transmitted during the simulation.
- Connection lines will turn solid dark gray when properly connected; they appear as a dashed red line if the connection has not been made correctly.

Note that it doesn't matter in which direction the connection line is drawn; you can start at one block's input and go to another block's output and vice versa.

Rather than needing to draw a connection line, blocks in the Item and Rate libraries automatically connect if they are added to the model using the Smart Blocks technique (where blocks are added from a right-click list) or the Auto Insert and Bump Connect techniques, which place and connect blocks. For more information about these techniques, see "Placing blocks on the model worksheet" on page 59.

☞ If the connection line is dashed red, it indicates that the connection has not been properly made.

Connection line appearance

ExtendSim provides several options for altering the appearance of connection lines

- A checkbox in the menu command Edit > Options > Model tab determines whether the default connection line for each type of connector is used in models. For instance, most discrete event models track flows of items as well as calculate and show values. The default

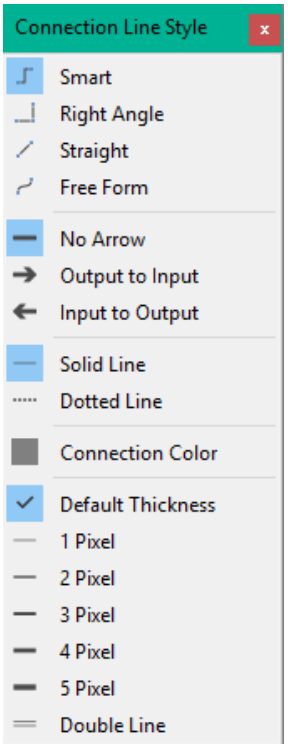
line makes it easier to visually differentiate the flow of items from the processing of values. The defaults are shown in this table:

Connector type	Default line
Value	
Item	
Flow	

- As discussed in *Styles* below, the Edit > Options > Model tab is also where the default style of connection lines for all new models—Smart, Right Angle, Straight, or Free Form—is set. The right-click menu can be used to override the default style for any selected line.
- The menu command Model > Connection Line Style, discussed below, has options for setting the style of connection lines for a particular model, as well as for choosing custom colors and other attributes for a model's connection lines.
- The right-click menu has options for adding or removing points along a connection line, making it easier to position the line in the model.
- If the connection line style is Free Form, the right-click menu has an option for changing it into a Bézier curve; adding points on the line can help here too. Right-click the line and toggle Bézier again, and the line returns to Free Form.

Model > Connection Line Style

This menu command is shown at right.



Styles

The default connection line style for all new models—Smart, Right Angle, Straight, or Free Form—is set in the Edit > Options > Model tab.

This can be changed for each model by right-clicking on the model worksheet or by selecting one of the first four options in the Model > Connection Line Style window before drawing the connection line. Furthermore, after a connection has been made, each line's style can be changed by selecting the connection line and right-clicking to choose a style.

Arrows

The second set of choices in the menu specifies whether or not you want arrows to appear on the connections.

Line attributes

The third set of choices gives the ability to make the line solid or dotted.

Colors

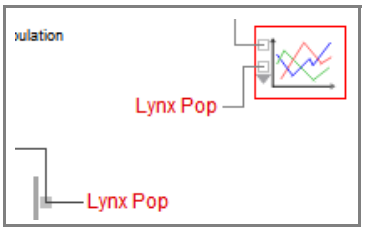
The fourth choice in the Connection Lines command deals with the color of connection lines. Like text or drawing objects, connection lines can be any color you choose. To have all connection lines be a specific color, choose the color before you draw the line. To change the color of an existing line, select the line and choose a color.

Line thickness

The final section is for setting the thickness of connection lines. The Default Thickness choice indicates that the thickness of the connection line will be based on the connector type, as shown on page 116. For example, the default connection between Item connectors is a double line. Choosing one of the other choices changes from the default for the connector type that is set in the Edit > Options > Model tab.

Named connections

Named connections are text labels used to represent one output at many locations in your model without using connection lines. If you have two text labels with the exact same text, you can use these to have the data, items, or flow jump from one part of the model to another. Named connections are often used when you do not want to clutter up your model with many lines. You can place the names near the blocks to which they connect and leave much of the area of your model free from connection lines.



By using named connections, you can eliminate a lot of connection lines in your model, making it easier to see which blocks are connected to each other and preventing lines from crossing over blocks or other model elements.

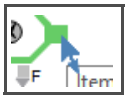
⚠ Named connections only work within one level of the model. This means that the data will not flow between hierarchical levels if you have a named connection on one level and a corresponding named connection in a block at a lower or higher level. To connect between hierarchical levels, use the Throw and Catch blocks in the Value, Item, or Rate libraries, or add connectors to hierarchical blocks. For more information on adding connectors to hierarchical blocks, see "Step 4: Add connectors to the icon in the Icon tab" on page 125.

How to create a named connection

- Create two text labels:
  - Use the Text tool or double-click the model worksheet to create a text box
  - Enter the relevant word; click elsewhere on the worksheet when finished
  - Copy the text box and paste onto the worksheet, creating a second text label with the same spelling

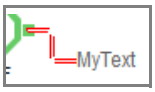
☞ Named connections are not case sensitive and spaces and returns are ignored, but you must use identical spelling in the text names.

- Draw a connection line to the first text label:
  - Without selecting the connector, hover the cursor over the first connector until the cursor becomes blue and double-ended, as shown here

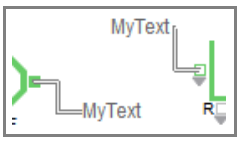


  - Click once in the connector, then drag the cursor to either side of the text label until the connection line turns thick and solid blue. (As Dan Cummings noted, there is a "magic, invisible spot" on the left and right of the text label.)
  - When the connection line turns thick and blue, click once on the text label to complete that segment of the connection

  - Release the cursor.
  - If drawn correctly, the connection line should be red and segmented, indicating that part of the connection has been made but the complete connection has not yet been made



- Draw a connection line to the second text label
  - Repeat the process, using the double-ended cursor to draw a connection line from the second connector to the second text label
  - Once the connection line turns thick and solid blue, click once on the text label, then release the cursor
  - If done correctly, the first connection's segmented red line will change to a normal connection line, indicating the full connection between text labels has been made



☞ It doesn't matter if you draw the connection line from the connector to the text label or from the text label to the connector. Also, you don't have to copy the original text label but it saves you from making a spelling error if you create a new text label from scratch.

Show Named Connections command

If you use many named connections in a model, you may accidentally connect a block to the wrong named connection. The Model > Show Named Connections command draws a connection line directly between all named connections. This helps you check that you made the connections that you wanted.

Showing and hiding model elements

Commands in the Model menu can change how model elements appear. Choose the Show Block Labels, Show Named Connections, Hide Connections and Hide Connectors commands to display or not display those elements of the model. These commands are especially useful when you have a lot of connections and blocks and you want to present the model to others without so much clutter in the workspace.

☞ These commands work at each level of hierarchy separately and take effect in whichever level is the active window. Thus if you want only the top level of the model to not show connectors, but the hierarchical levels to show connectors, choose the command when no hierarchical levels are open.

Hierarchy

The ExtendSim hierarchical capability lets you combine basic modeling constructs (such as a group of connected blocks that represent a process) into a single, higher-level construct, then combine several of those new constructs into an even higher-level construct, and so forth. This causes portions of a model, or even the entire model, to be displayed in layers, where you can drill down from the highest system level to the smallest detail. Note that this doesn't result in just a pictorial representation of a process; all of the submodel's behavior is encapsulated and incorporated into the new modeling construct.

The mechanism for creating hierarchical layers in ExtendSim is through the creation of a hierarchical block—a special type of block that usually has a group of blocks encapsulated onto its worksheet. Each grouping of blocks represents a portion of the model—a *subsystem* or *submodel*. Using hierarchical blocks you can create new modeling constructs without programming, by nesting subsystems in an unlimited number of layers for top-down or bottom-up modeling.

☞ Top-down modeling is starting with a new hierarchical block and building a model or a series of submodels in it. Bottom-up modeling involves selecting a group of blocks within a model and making them into a hierarchical block. You can create multiple hierarchical layers from the top down or from the bottom up.

Uses for hierarchy

Although you do not have to use them as you build models, hierarchical blocks can help organize models logically, make models appear more attractive, improve productivity, and enhance comprehension.

- Simplify a complex model by grouping areas of the model into hierarchical blocks containing submodels. These submodels can then be reused in other models without having to reproduce all of the connections.
- Instead of showing all the detail, present a model as a few simple but logical steps. To reveal the subsystems within a step, just double-click the hierarchical block.

- To increase productivity and comprehension, create a hierarchical block that represents a process element, then use it in several models that have that element in common. By changing the parameters, each instance of the hierarchical block can be customized for its model.
- When developing a new model, go from the simplest assumptions to more complex ones by creating more and more levels of hierarchy. This helps structure your thinking and makes models easier to follow as they become more complex.
- Although hierarchical blocks usually contain submodels of blocks, they don't have to. They can also be useful for enhancing and documenting the model, serving as popup windows that reveal pictures, text, and so forth when double-clicked. For an example, see "Help block" on page 80.

Characteristics of hierarchical blocks

Hierarchical blocks are unique; they have some characteristics of a block and some characteristics of a model worksheet. All hierarchical blocks have the following in common:

- A hierarchical block can contain no blocks, one block, a group of blocks, or even other hierarchical blocks. It can also contain text, graphics, cloned dialog items, and pictures.
- Hierarchical blocks can be copied to other areas of the model or to other models and used just like other blocks.
- The settings within a hierarchical block can be changed by double-clicking the hierarchical block, then double-clicking the desired interior block's icon, or by changing the desired block's cloned dialog items.
- Hierarchical blocks have a worksheet with notebooks and a structure.
    - Hierarchical blocks' worksheets are similar to model windows. On the hierarchical block's worksheet you can add blocks from a library, create a hierarchical block, add graphics, type labels and other text, clone dialog items, and so forth.
    - Hierarchical block notebooks are used in the same manner as for any model.
    - The components of a hierarchical block (icon, connectors, or Help text) can be changed by accessing the hierarchical block's structure window.
- Unlike other blocks, by default hierarchical blocks are saved directly in the model as copies. This characteristic allows them to be treated much like a copy of a portion of the model. You can copy a hierarchical block to another part of your model and make changes to its worksheet window without affecting the original hierarchical block. This is known as *physical hierarchy*.
- You can choose to save a hierarchical block in a library, in which case it can be treated much like a regular block. When you make changes to the hierarchical block's structure window and you also choose to update all instances of that block, it is known as *pure hierarchy*.

Important notes about hierarchical blocks

Named connections (discussed on page 118) only connect within one hierarchical level in a model. This means that the data will not flow between levels if you have a named connection on one level and a corresponding named connection in a block at a lower or higher level. To

connect between hierarchical levels, use the Throw and Catch blocks in the Value, Item, or Rate libraries, or add connectors to hierarchical blocks. For more information on adding connectors to hierarchical blocks, see “Step 4: Add connectors to the icon in the Icon tab” on page 125.

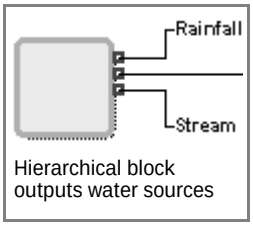
⚠ If an ordinary ExtendSim block is changed in its library, its manifestation in a hierarchical block is also updated. However, because hierarchical blocks contain both blocks and data, and because updating a hierarchical block might change its existing data, hierarchical blocks saved in a library will not automatically update their corresponding hierarchical blocks in a model.

Creating and accessing hierarchical blocks

Unlike blocks that are created using programming, hierarchical blocks are created using menu commands.

Creating

There are two ways to create a hierarchical block:



Hierarchical block outputs water sources

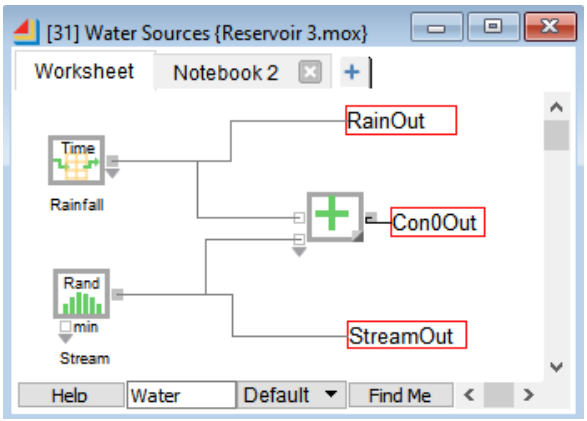
1) The easiest method is to select a group of blocks and either right-click or use a menu command to encapsulate them into a new hierarchical block. This is most often used to organize the complexity of an existing model and is discussed at “Making a selection into a hierarchical block” on page 123.
2) An alternative is to use a menu command to create a new hierarchical block from scratch; then build a submodel on its worksheet. This is used for top-down modeling as discussed in “Creating a new hierarchical block” on page 124.

Ways to access a hierarchical block

Once created, hierarchical blocks have two views:

- The *worksheet window*. This is what is displayed if you just double-click the icon of a hierarchical block. Use this window to label the hierarchical block, make changes to the submodel, and so forth as discussed below.
- The *structure window*. This is displayed if you create a new hierarchical block by giving the command Model > New Hierarchical Block or if you select an existing hierarchical block and either right-click or give the command Model > Open Hierarchical Block Structure. The structure window contains another view of the worksheet and additional panes for modifying the hierarchical block’s icon, adding Help, and so forth. It is shown on page 124.

Worksheet window



The most direct way to see a hierarchical block's submodel is by double-clicking the block's icon. Instead of a dialog inside, you see a worksheet and notebook, just as for a regular model window. The blocks that make up the submodel are on the worksheet and are laid out like a model, with connections, labels, and so on. Depending on how the hierarchical block was created, it might also contain text, pictures, or cloned dialog items that control the blocks within the hierarchical block.

The block's title bar shows the name of the top level model in braces. The submodel's connections with the rest of the model are shown as *connection text boxes* (named connections with red borders around them). In the example here, *RainOut* is one of the outputs of the hierarchical block named Water Sources.

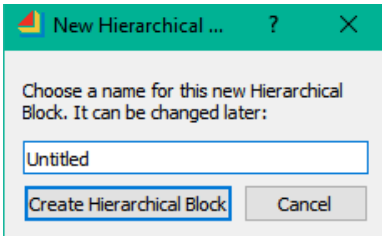
Making a selection into a hierarchical block

This is the easiest method for creating a hierarchical block. The steps are:

- Open an existing model or place blocks, text, and/or graphic objects on a new model worksheet.
- Since the selection tool determines what gets selected, choose the appropriate cursor (Block/Text, All Objects, etc.) for what you want to include in the hierarchical block.
- Use the cursor to select blocks and other objects by dragging over them or by holding down the Shift key and clicking each item.

⚠ Do not select the text labels of any named connections, otherwise communication between the submodel within the hierarchical block and the model outside of it will break.

- Right-click or choose the command Model > Make Selection Hierarchical.
- Enter a descriptive name for the hierarchical block and click Create Hierarchical Block.



When it creates the hierarchical block, ExtendSim makes all the connections and replaces the selected blocks in the model worksheet with the new hierarchical block and its default icon, a rounded rectangle with a border.

- Double-click the hierarchical block's icon to see the worksheet and its submodel.
- To alter aspects of the hierarchical block (such as changing from the default icon, adding blocks to the model, or moving or renaming connections), see "Modifying hierarchical blocks" on page 128.

Hierarchical blocks can be saved in libraries or just in the model, as discussed at "Saving hierarchical blocks" on page 127.

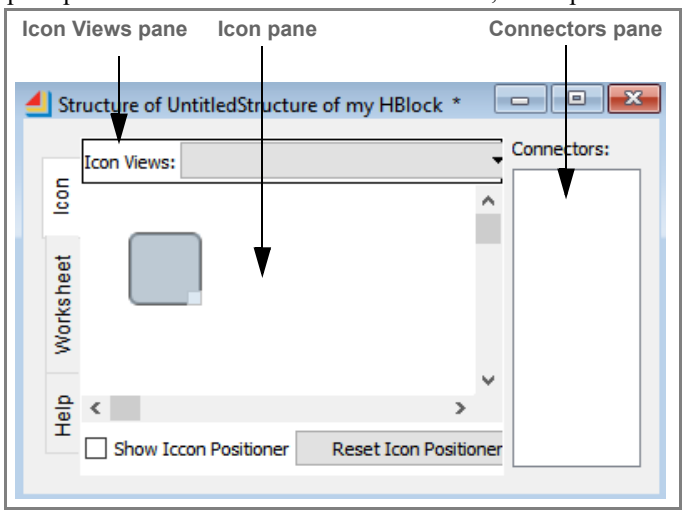
Creating a new hierarchical block

The following example shows how to create a new hierarchical block.

Step 1: Start a new hierarchical block

- Open an existing model or open a new model window.
- Give the command Model > New Hierarchical Block.

ExtendSim prompts for a name for the hierarchical block, then opens its structure window.



The hierarchical block's structure window has three tabs on the left side (Icon, Worksheet, and Help). The *Worksheet* tab is where the submodel for this hierarchical block will be built; this is what appears when you double-click a hierarchical block's icon. The *Help* tab is for entering any comments or assistance you want

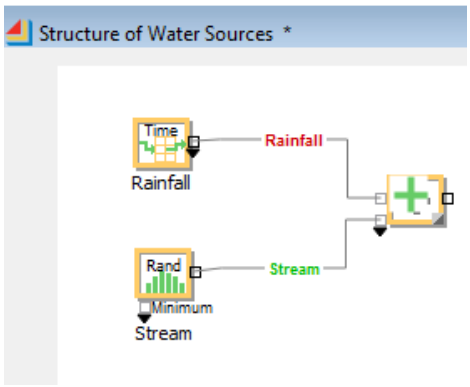
The *Icon* tab, shown above is divided into three work areas or panes for creating the block's icon and connectors:

- The area at the top left of the Icon tab is the *Icon Views* pane, for creating different views for the icon, such as a right-to-left view.
- The *Icon* pane in the middle is for designing the block's icon and adding connectors.
- The *Connectors* pane on the right displays the names of the input and output connectors for the block. It also appears on the Worksheet tab and can be used to rename the connectors.
- The Icon Positioner is only used if the position of the icon needs to be adjusted due to a change caused by a newer release of ExtendSim.

☞ You can also access a hierarchical block's structure by right-clicking on its icon.

Step 2: Build the submodel in the Worksheet tab

To build a submodel in a new hierarchical block, add blocks and/or graphic objects to the Worksheet tab using the same methods used as for a model worksheet.



For example, the Water Sources hierarchical block at right contains three connected blocks from the Value library: a Lookup Table block labeled Rainfall, a Random Number block labeled Stream, and a Math block set to add its inputs.

If you prefer, use the Copy and Paste commands from the Edit menu to copy a portion of an existing model onto the Worksheet.

⚠ Do not place an Executive block within a hierarchical block. There should only be one Executive block in the model and it needs to be at the top level of the model worksheet.

Step 3: Modify the icon in the Icon tab

In the Icon tab, a new hierarchical block starts with a default icon, a gray square. You can modify this icon, for example by changing its shape or color or by adding text or a picture. Or you can delete it and create a new icon using the graphic tools in the toolbar, or by pasting a picture from another program. For information on using the graphic tools, see “Graphic shapes, tools, and commands” on page 107.

Step 4: Add connectors to the icon in the Icon tab

Since the model encapsulated within a hierarchical block needs to be connected to the model outside of the hierarchical block, you must add the appropriate connectors to the hierarchical block’s icon.

Some hierarchical blocks have both input and output connectors, while others have just one kind. Also, as is true for other blocks, hierarchical blocks use specific types of connectors (value, item, flow, etc.) depending on what they are connected to.

Choosing a connector type

If the Icon tool isn’t already open, give the menu command Tools > Icon. This opens the Icon toolbar, a portion of which is shown on the right.

The first eight buttons add connectors: Value, Item, Flow, Reliability, Universal, Array, User defined, and Box. (These are discussed at “Connector types” on page 61.)

For this example, you would select the Value connector since the output comes from a value connector (ResultsOut) on the Math block.

☞ The connector types must match the input or output connectors of the blocks in the Worksheet tab that will be getting data from the main model worksheet or sending out data to the model.

Adding a connector

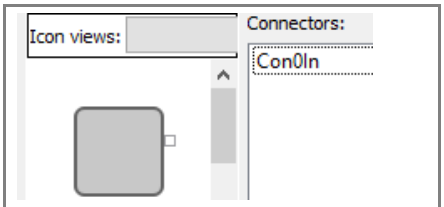
Connectors are added to icons the same way blocks are added to a worksheet—point and click.

- Go to the hierarchical block’s Icon tab.

- From the Icon toolbar, click the appropriate type of connector.
- Click at the desired position on the hierarchical block's icon. For this example, click on the right side of the icon.

☞ To add multiple connectors, hold down the Alt key and click multiple times.

This creates the default *Con0In* connector on the icon in the Icon pane, lists it in the Connector pane, and adds a red-bordered connector text label (similar to a named connection) in the Worksheet tab.



Value connector added to hierarchical block

☞ If you choose the wrong type of connector, just select the connector in the Icon pane, then select the correct type of connector from the Icon Tools.The connector will change to the new type.

Changing to an output connector

By default, when a connector is first added to an icon it is an input connector, as indicated by the word *In* at the end of its name. To cause it to be an output connector, the connector name needs to end in **Out**. And, since Con0 isn't very descriptive, you'll want to change the connector name to something more relevant such as "Water".

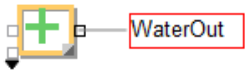
☞ You can name a connector anything you want as long as the name ends in *In* or *Out* and is 32 characters or less. Also, every connector must have a unique name.

- Double-click the connector name in the Connector pane and change its name to something that ends in Out, such as *WaterOut*.

Connect to the submodel in the Worksheet tab

When this connector is added to the icon in the Icon tab, a text label "WaterOut" is placed on the Worksheet tab. The connector label is how the submodel communicates with the model worksheet.

The next step is to connect the connector text label to the appropriate block in the model.



Connector changed to output connector and added to Math block

- In the Worksheet tab, move the WaterOut text label to the right of the Math block.
- Use point and click to connect from the output of the Math block to the WaterOut text. Once the line thickens and turns blue, release the mouse and the connection is made.

Step 5: Connect the hierarchical block in the model

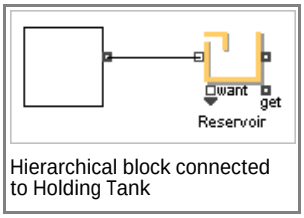
To connect the finished hierarchical block to the rest of the model:

- Close the hierarchical block's structure window.

When you close the structure window of a new hierarchical block, a dialog gives options for saving the block. These options are discussed in the following topic. For this example:

- Choose **Save Changes to This Block**.

▸ Connect the connector on the hierarchical block to other connectors in the model, just as you would any other block. In the example, the hierarchical block is connected to a Holding Tank block.



Hierarchical block connected to Holding Tank

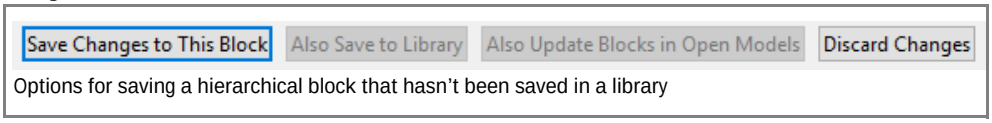
Saving hierarchical blocks

By default, hierarchical blocks are saved with the model. You can copy such a hierarchical block to other parts of the model or even to other models using Copy/Paste. Unless the block has been saved in a library, each instance of the block in a model can be then be made unique by modifying it as described in "Modifying hierarchical blocks" on page 128. You can also save a hierarchical block to a library, which makes it easier to add them to new models.

Saving hierarchical blocks in a model

How a hierarchical block is saved depends on how it was created and whether its structure has been modified:

- If you make a selection hierarchical, the hierarchical block is automatically saved with the model when the model is saved. Any subsequent changes made to the hierarchical block's worksheet also get saved when the model is saved.
- If you create a new hierarchical block, or make changes to an existing hierarchical block's structure window, closing the block's structure prompts with options to: *Save Changes to This Block*, *Discard Changes*, or *Cancel*. Note that the Save Changes option affects only this specific instance of the block in the model.



Options for saving a hierarchical block that hasn't been saved in a library

☞ The two additional options for saving (*Also Save to Library* and *Also Update Blocks in Open Models*) are not enabled unless the block already exists in a library, as discussed below.

Saving hierarchical blocks to a library

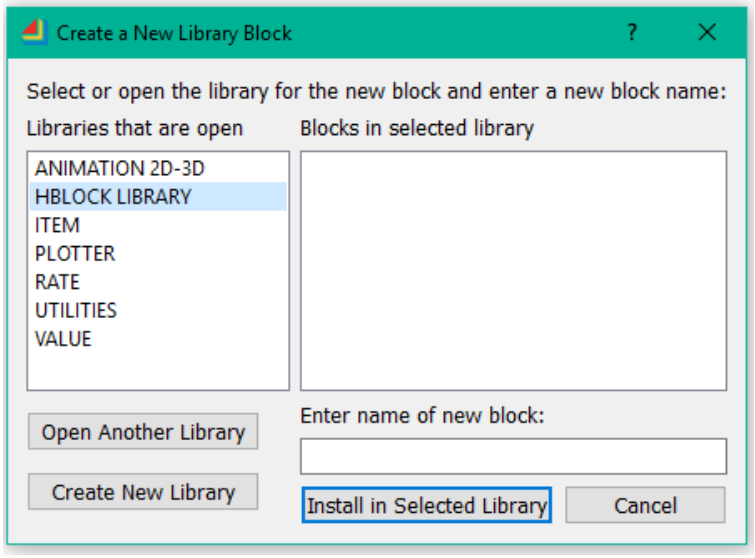
You can save a hierarchical block in a library, but what is saved in the library is only a "snapshot" of the hierarchical block at the time you saved it.

⚠ If you modify a hierarchical block's worksheet and the hierarchical block has been saved in a library, those changes will not be reflected in the master block in the library unless you specifically tell ExtendSim to also save the changes to the library. Furthermore, saving changes to that hierarchical block can only update copies of it that are in OPEN models. If a copy of that hierarchical block is in a model that is not open when the changes are saved, that copy will not get updated.

To save a hierarchical block in a library or to cause changes made to a hierarchical block to be reflected in its master block in a library:

▸ If it is not already open, open the structure of the hierarchical block by right-clicking on its icon or by selecting its icon and choosing Model > Open Hierarchical Block Structure.

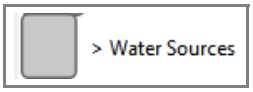
- Choose File > Save Hierarchical Block to Library As
- In that dialog, select a new or existing library for the hierarchical block and select *Install in Selected Library*.



⚠ Do not save hierarchical blocks in the libraries that come with ExtendSim!

- Close the hierarchical block's structure window and choose one of the save options:
  - *Save Changes to This Block* affects only this specific instance of the block in the model. It does not affect any other copies of the block in this or any other model.
  - *Also Save to Library* affects this instance of the block, the master block in the library, and blocks placed from the library into a model after the save.
  - *Also Update Blocks in Open Model**s*** also saves the changes to instances of this hierarchical block in any ***OPEN*** models.
  - *Discard Change**s***
  - *Cancel*

These options are discussed more in "Summary of results of modifying hierarchical blocks" on page 129.



A hierarchical block that is saved in a library has its name listed in the library and in the library window preceded by the symbol >. Like other blocks, hierarchical blocks that are saved in libraries list the name of the library in their title bar in angle brackets. If no library is listed, the hierarchical block is not saved in a library.

Modifying hierarchical blocks

How you modify a hierarchical block depends on whether the block is saved in a library and the type of modification you want to make:

- If the block hasn't been saved in a library, or you don't want to save the changes to the master block in the library, simply double-click the hierarchical block's icon and change the settings for individual blocks or modify the layout and appearance of the submodel (e.g. add blocks, clone items onto the worksheet, add text and drawings, etc.).

- To save the changes to the master hierarchical block in a library, you must open the hierarchical structure window. (You must also use the structure window to make any changes to the hierarchical block's icon, connectors, or Help, even if you don't want to save these changes to the library.)

The following indicates where to make changes to a hierarchical block:

Action	Worksheet Window	Structure Window
Add blocks and connections	X	X
Change parameters in submodel blocks	X	X
Add text, drawing objects, and pictures	X	X
Clone dialog items	X	X
Change icon, add alternate views		X
Rename the block		X
Add Help		X
Add animation		X

Changing the icon and adding alternate views

To change the icon to better suit the purpose of the hierarchical block:

- Open the hierarchical block's structure window.
- Click the icon in the icon pane.
- Change the size, color or pattern of the icon, or delete it and create another icon.

  To create a new icon, use any of the ExtendSim graphic objects or paste pictures from outside of ExtendSim. See "Graphic shapes, tools, and commands" on page 107, for information on how to use the drawing tools.

You need to delete the icon before creating a new one, but do not delete any connectors. If you do, ExtendSim will warn you. If you accidentally delete a connector, undo the delete or add another connector. Be sure to check the block's connections in the model.

- Optionally add alternate views using the Icon Views pane at the top of the Icon tab. An example of this is the Markov Chain Weather model discussed on page 17. See "Icon views" on page 66 for more information.

Renaming the block

With the structure window of a hierarchical block open, rename it by choosing Develop > Rename Block.

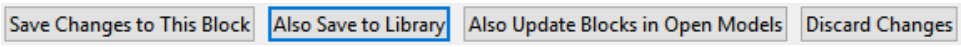
Adding animation

Hierarchical block icons can be animated, as discussed on page 113.

Summary of results of modifying hierarchical blocks

There are different results when a hierarchical block is modified, depending on whether its worksheet window or structure window was modified and whether the block is saved just in a model or is also saved in a library:

- If you modify a hierarchical block's submodel in its worksheet (accessed by double-clicking the icon), those changes only apply to that block. Changing a hierarchical block's submodel is similar to changing parameters in a regular block's dialog: the changes affect only that instance of the block on the worksheet and are saved with the model. This is true even for hierarchical blocks that were originally saved in libraries.
- If you modify the *structure* of a hierarchical block that is not saved in a library, those changes also only apply to that block. This has the same result as modifying a hierarchical block's submodel in the worksheet window. For example, you can make several copies of that hierarchical block in a model, but when you change one of the copies, the other blocks remain unchanged.
- If you modify the *structure* of a hierarchical block that has been saved in a library, you can choose how those changes should be reflected:
  - Only to this instance of the block on the model worksheet (choose *Save Changes to This Block*).
  - Also in the master block in the library, which only affects blocks placed in the model from the library *after* the change has been made (choose *Also Save to Library*).
  - Also in all instances of the block in *open* models (note that this does not affect models that are not open at the time); this is also called pure hierarchy (choose *Also Update Blocks in Open Models*).



⚠ Changing a hierarchical block that is saved in a library can have unintended consequences if the block is used in multiple places. Do not make changes to the master block in the library unless you understand the effect those changes could have on other models that use that block.

- To modify the structure of a hierarchical block saved in a library, copy a fresh hierarchical block from its library window onto the model worksheet, then work on the block directly in the model. Do not work on a hierarchical block that is already in the model or you might overwrite the master hierarchical block with a block that has been customized for a specific purpose in the model.
- If you change the hierarchical block's submodel, rename the block and save it to the library using the new name. This will prevent existing instances of that hierarchical block from being changed.
- Use caution if you change a parameter in a block in a submodel, then save the hierarchical block to the library with the "Also **update blocks in open models**" choice. In that case, each instance of the hierarchical block will have the changed parameter value.

⚠ Changes made to the structure of a hierarchical block that is saved in a library will not be reflected in models that were NOT open when the change was made. If that happens, open the model and replace its hierarchical block with a fresh copy from the library.

☞ The Constant block (Value library) has an option to *Retain constant if updated from hierarchy*. If that option is checked, the Constant will retain its value when the enclosing hierarchical block is updated from a library and existing instantiations of the block will not be changed

inside of existing hierarchical blocks. This is useful if each hierarchical block needs a unique identifier which does not reset if the hierarchical block is updated from a library.

Other presentation features

The How To chapter on "Customizing the User Interface", which starts on page 71, discusses many features that will help when presenting your models to others, such as the following.

ExtendSim databases

Databases are very useful when presenting models to others because a model's inputs and outputs can be stored in a centralized location and data can be organized in a logical manner. Each model can have one or more ExtendSim databases for storing, managing, and reporting information.

☞ For more information, see the separate document titled "ExtendSim Database Quick Reference".

Notebooks

Worksheets and hierarchical blocks have notebooks that you can customize to help organize and manage model data, facilitate presentations, and more. See "Notebooks" on page 79 for further information.

Cloning

A clone is an exact copy (similar to a shortcut or alias) of a dialog item, table, or graph that behaves identically to the original. Presentations can be enhanced by placing clones of dialog items in strategic locations on a model's or hierarchical block's worksheet or notebook. For more information, see "Cloning" on page 72.

Creating a custom user interface

ExtendSim lets you add customized buttons, popup menus, controls, and more to facilitate user interaction with a model. These are also helpful when presenting and explaining models to others. See "Creating a dashboard interface" on page 74.

How To

Analysis

Making sense of your models

“Prediction is very difficult, especially if its about the future.”
— Niels Bohr

Overview

Although simulation is a quantified subject, there is still room for analysis and judgmental procedures. This is especially true when your purpose in building a model is to spot trends or identify patterns of behavior, or when you must make an immediate "go, no-go" decision. How statistically precise your models should be depends on several factors, including:

- The nature of the problem
- The importance of the decision
- The level of risk you are willing to accept
- The sensitivity of the system to the input data

One of the main benefits of ExtendSim is that it provides several methods for analyzing the results of simulation runs. This chapter covers those techniques, including statistics, confidence intervals, scenario and sensitivity analysis, optimization, Stat::Fit, and graphs.

☞ Remember that when you use statistical methods to analyze simulation output, you are performing the analysis only on model results, not on the actual system. It is important to ensure that the numbers entered accurately represent the details of the actual system. The significance and relevance of your analysis will depend on how closely the model's inputs correspond with real-world data ("garbage in = garbage out").

Blocks that calculate statistics

Many ExtendSim blocks automatically calculate relevant statistics and display them on their Results tab. For example, the Activity block (Item library) reports utilization, average wait, maximum length, and so forth.

You can also use blocks in the Value library, such as the Equation and Math blocks, to perform calculations on model outputs.

As described below, other ExtendSim blocks are specially designed to report statistical information for particular types of blocks or for items that pass through the block, or to refine statistical data collection.

Statistics

The Statistics block (Report library) accumulates data and calculates statistics using a specified *statistical method*.

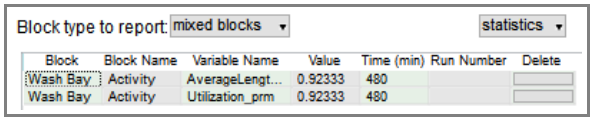
Types of blocks it reports on

Place a Statistics block anywhere in the model and, depending on selections made in its dialog, it will report statistics for the following blocks and types of blocks:

- Activities (Activity, Convey Item, and Transport blocks in the Item library)
- Mean & Variance block (Value library)
- Queues (Queue, Queue Equation, and Queue Matching blocks in the Item library)
- Resource Item block (Item library)
- Resource Pool block (Item library)
- Workstation block (Item library)
- Convey Flow block (Rate library)

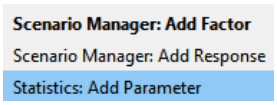
- Tanks (Tank and Interchange blocks in the Rate library)
- Rate library blocks
- Resource Manager block (Item library)
- Queue Matching block (Item library)
- Mixed blocks (Shift-click or clone drop, as discussed below

Shift-click or clone drop

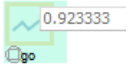


The *Mixed blocks* option is useful for adding a custom selection of output fields to the Statistics block's table. For example, in the screenshot here the length and utilization statistics from an Activity block have been added to the table. You can use either Shift-click or clone drop to add each targeted output field to the table.

☞ The output field must be in one of the supported types of blocks.

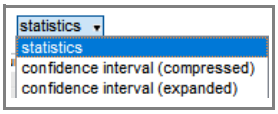


- To **Shift-click**, open the target block's dialog and hold down the Shift key while you click once on the targeted output field (utilization, length, etc.). Then click on the appropriate option that appears (*Statistics: Add Parameter*). This causes the output information to be displayed in the Statistic block's table.
- To **clone drop**, open the target block's dialog and use the Clone cursor to click once on the targeted output field (utilization, length, etc.). Then hover the Clone cursor over the icon of the Statistics block. When the icon of the Statistics block is highlighted, click once on that icon to release the cloned parameter. This causes the cloned output information to be displayed in the Statistics block's table.



Statistics/Confidence Interval popup

A popup menu on the Statistics tab allows you to choose whether the raw data (Statistics option) or calculated confidence interval information (compressed or expanded options) will be displayed in the table. Also see "Confidence intervals" on page 138.



Accumulating data

When a simulation is run, one row of information is recorded in the Statistics block's table for each block of the specified type each time an observation is made. Observations can be recorded continuously, which significantly slows simulation time, or at the end of the simulation. They are also made in response to an item or value arriving at the Go universal input connector. For instance, attach a Pulse block (Value library) to the Go connector to force a periodic or scheduled update of statistical information.

☞ The first column of the table in the Statistics block's dialog lists the block's label. For a block without a label, the first column gives the block's name.

To immediately see the current statistics, click the Update Now button; this is also useful if you want to see which blocks will have their statistical information collected during the simulation run.

Statistical methods

The Statistics block's Options tab allows you to select one of three statistical methods:

- Multirun analysis
- Batch means
- Custom

Choosing one of the first two methods causes specific settings to be selected in the block's Options tab. For the "custom" option, manually select which settings you want.

Export data

The block's Export tab is for exporting the results to Excel using a new or existing worksheet and for setting other export options. You can also create a database table and link the data to that.

Queue Statistics model

The Queue Statistics model uses the Statistics block to gather information about queues. This discrete event model is located in the folder ExtendSim/Examples/Discrete Event/Statistics.

Block	Block Name	Ave Length (M)	Ave Length (CI)
Queue 1	Queue	39.19	14.70
Queue 2	Queue	8.794	2.397
Queue 3	Queue	16.25	9.190
Queue 4	Queue	6.653	2.512
Queue 5	Queue	8.149	4.343

Clear Statistics

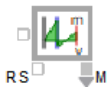
As discussed in "Non-terminating systems" on page 97, when you start a simulation run there is no data in the model. After a stochastic model has been running for a while, it gets to the point where it is functioning more like the real system. The interval from when the model starts to when it is functioning in a steady or normal state is called the *warm-up period.*

The Clear Statistics block (Value library) clears the statistics for selected blocks, eliminating the statistical bias of a warm-up period. The type of blocks include:

- Activities (Activity, Convey Item, Transport, Workstation)
- Exit block
- Mean & Variance block (Value library)
- Queues (Queue, Queue Equation, Queue Matching)
- Rate library blocks
- Resources (Resource Item, Resource Pool)
- Information block
- Max & Min block (Value library)

The block can reset statistical accumulators at periodic intervals or in response to a system event. It can be placed anywhere in the model. Choose in its dialog which types of blocks will have their statistics cleared. The Clearing Statistics example, a discrete event model located in the ExtendSim/Examples/Discrete Event/Statistics folder, uses the Clear Statistics block to restart model statistics after 40 seconds.

Mean & Variance

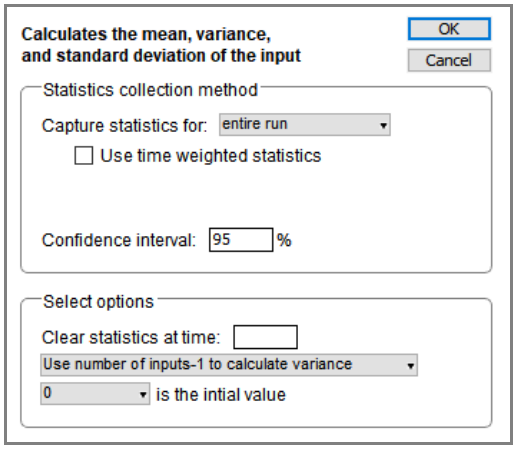


The Mean & Variance block (Value library) calculates the mean, variance, and standard deviation of the values that it receives during a simulation run, based on a specified confidence interval. The block reports these data points along with the number of observations, the confidence interval, and the relative CI error.

Methods for capturing statistics

The dialog provides four methods of capturing statistics:

- Entire run
- Moving average
- Interval
- Multiple runs



These settings allow you to use time weighted statistics, calculate a moving average for a selected interval, calculate the statistics over multiple runs, and more. See also “Confidence intervals” on page 138.

Options and use

There are also two options for how the variance is calculated:

- **Use number of inputs to calculate variance**. The variance is computed as the sum of (valid inputs - mean)^2 ÷ (number of inputs).
- **Use number of inputs-1 to calculate variance**. The variance is computed as the sum of (valid inputs - mean)^2 ÷ (number of inputs - 1).

A Quantiles option on the Results tab allows you to see what proportion of the observations fall within an interval.

The Mean & Variance block uses 1/(N-1) as an averaging factor. If an input is a NoValue, it is ignored and does not affect the statistics.

To use this block, connect a value output from the block of interest to the Mean & Variance block’s input.

The Monte Carlo model, located in the folder ExtendSim/Examples/Continuous/Standard Block Models and discussed on page 14, is an example of using the Mean & Variance block.

Information

The Information block (Item library) reports statistics about the items that pass through it. It is useful for counting the number of items and for determining cycle time statistics and the time between item arrivals (TBI). The block reports the average and the current statistics, as well as the minimum and maximum.

The Information block can only be used in models that use item-based blocks, and it must be connected to a block’s item output. This block is also used to calculate cycle time.

Cost Stats

The Cost Stats block (Report library) records in a table the input costs and total cost generated by each costing block in a model. This block reports cost information for item-based blocks, such as for a discrete event model. Information can be exported to a spreadsheet. The block can be placed anywhere in the model and can be set to different confidence intervals. See also “Confidence intervals” on page 138.

Confidence intervals

Confidence interval estimation tells you, with a given level of probability or confidence, how close the average simulation results are to the *model’s* true average or mean. The confidence *interval* is the range within which it is predicted the true mean is located. This is expressed as the probability that the true mean lies within interval x± y, where “x” is the average obtained by multiple observations and “y” defines the outer limits of the interval’s range. The confidence *level* is the probability that the true mean will be located within the range. Typical confidence levels are 90%, 95%, and 99%. Notice that, at higher levels of probability, the interval gets wider.

Used in blocks

The Statistics and Cost Stats blocks (Report library) and the Mean & Variance block (Value library) provide the option to specify a confidence level in their dialogs. For example, the Statistics block not only summarizes and reports statistical information, but also calculates confidence intervals for the information given various levels of confidence.

How to generate

To generate a confidence interval where each sample is the result from a single simulation run, in the Run > Simulation Setup > Random Numbers tab do one of the following:

- Randomize the seed by entering a blank or zero random seed.
- Or, select *Continue random number sequence* in the popup.
- Or, select *Use Database table __seed for values* in the popup.

To obtain sufficient sample data to determine the confidence interval, multiple observations of each statistic must be made and appended to the table of data. This is shown in the “Run for CI” model located in the ExtendSim/Examples/Tips/Modeling Tips folder. The model runs repeatedly based on the Mean & Variance’s Options tab settings *Calculate for multiple simulations* and *Replicate until relative error is <= 0.01*.

Sensitivity analysis

Sensitivity analysis allows you to conduct controlled experiments to explore how much of an impact a particular parameter has on model results. The ExtendSim sensitivity analysis features make it easy and convenient to specify a parameter to investigate and settings to use for the analysis.

Overview

You use sensitivity analysis to investigate the effect of changing one or more parameters upon an area of interest. Sensitivity analysis works with all numeric parameter fields. It also works with clones of those numeric items. You can add sensitivity to as many dialog values as you like. However, it is recommended that you only vary one or two dialog values at a time so as not to confuse the analysis. Once a parameter has been sensitized, specify a multiple number of runs and run the simulation.

The resulting values for the area of interest are usually graphed. Although you can use any of the standard graphs from the Chart library, it is most common to use the Line Chart and choose that each run be graphed on separate traces. To do this, use the Run and Run# columns in the Line Chart's Data Collection tab. The results of varying the parameter value over the selected settings will be displayed as the simulation is run multiple times.

Parameters are sensitized by right-clicking or by using the Model > Sensitize Parameter command. The Model > Open Sensitized Blocks command shows all the dialogs for blocks that have sensitivity settings, even if sensitivity analysis is not enabled. This is convenient if you have entered sensitivity settings for many parameters in a large model.

☞ To utilize sensitized parameters, the number of simulation runs must be greater than one.

Use sensitivity analysis

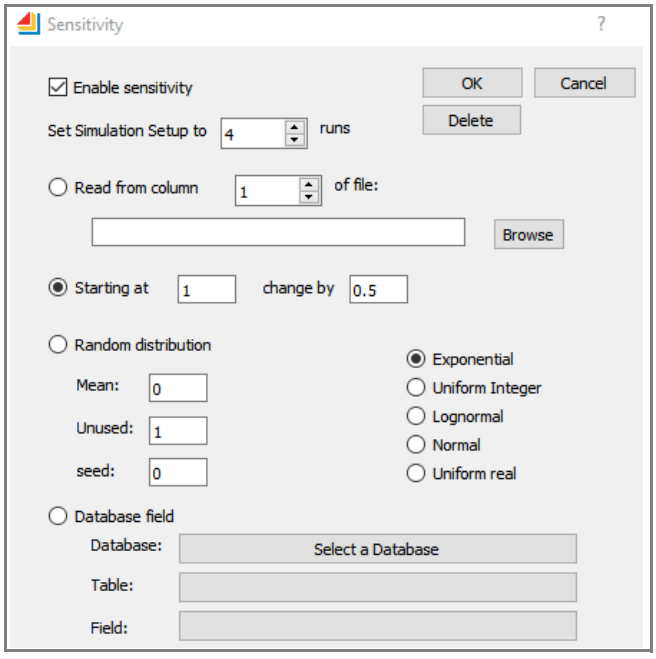
For this example, use the Reservoir 1 model discussed in the Tutorial module.

- Open the Reservoir 1 model; it is located in the Documents/ExtendSim/Examples/Tutorials/Continuous folder.
- So that you don't overwrite the original model, give the command File > Save Model As and save the model as "Sensitivity".

Sensitize the parameter

- Open the dialog for the Random Number block labeled "Stream".
- Open the Sensitivity dialog by doing *one* of the following to the *Maximum* entry field:
  - Click the entry field and choose the command Model > Sensitize Parameter.
  - Or right-click the entry field and choose Sensitize Parameter.

The Sensitivity dialog appears with the *Enable sensitivity* checkbox selected by default.



- Enter **Set Simulation Setup to: 4 runs**.

☞ The number of runs can be set in either the Sensitivity Setup dialog or the Simulation Setup dialog. Each controls the other so that the last value selected in either of them controls the number of runs.

- Enter **Starting at 1** and **change by 0.5**.

This will cause the stream's flow to increase by 0.5 for each subsequent run.

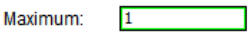
☞ Parameter values can be sensitized using values from a file, a specified range of values, a ran-

How To

dom distribution, or values from a database. The choices are described in “Specifying the sensitivity method” on page 140.

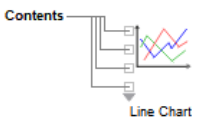
- Close the Sensitivity Setup dialog using the OK button, to save your changes.

In the Random Number block’s dialog, the parameter field now has a green border around it, indicating that it has a sensitivity setting and sensitivity analysis is active for the parameter.

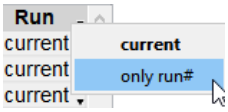


Output the results to a Line Chart block

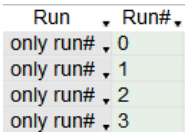
- Add a Line Chart block (Chart library) to the right side of the model and expose 4 inputs.
- Copy the text of the **Contents** named connection and create 4 named connections to the Line Chart’s inputs, as shown here.



- In the Dialog tab of the Line Chart, go to the Data Collection tab and click on the header of the **Run** column to see the popup.
- Change the popup from current to **only run#**, as shown here.

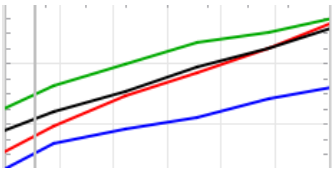


- In the new Run# column that appears
  - Leave row 0 at 0 for the Run#
  - Change row 1 to 1, row 2 to 2, and row 3 to 3



- Notice that the Run > Use Sensitivity Analysis command is checked by default.
- Run the simulation. ExtendSim runs the simulation four times, from run 0 to run 3.

If you zoom in on the chart you can see the variation between the runs in the four lines. As expected, increasing the stream’s flow increased the amount of water in the reservoir. In more complex models, the effect of making a change would not be so obvious.



Specifying the sensitivity method

The Sensitivity dialog lets you specify that a sensitized parameter will change from a list in a file, incrementally, randomly, or from based on the value of a database field. The choices are:

Option	Description
Read from column x of file	Assigns the values from a text file. This is the option you will most likely use when performing ad-hoc experiments. If the file has more than one column separated by tab characters, specify the desired column. Starting with the first row of the specified column for the first run, this option uses the value of each successive row in that column for subsequent runs.
Starting at... change by...	Specifies the starting value and the amount of change. By default, the starting value is the same as the parameter's value in the dialog. Increase the variable with a positive number or decrease it with a negative number.
Random distribution	Uses a random distribution to set the parameter. This is an easy way to make a single value in a model change randomly over many simulation runs while keeping the value constant within a single run. Choose one of the five distributions and enter the distribution parameters in the options to the left of the distribution. The *seed* is the number to use for the random number generator. As in the Simulation Setup dialog, BLANK or 0 for the seed is random.
Database field	Assigns values from the fields in an ExtendSim database. Starting with the first record of the specified field for the first run, this option uses the value of each successive record for subsequent runs.

☞ If you are a developer, you can use the CurrentSense variable to control the order of use for the values set for specific runs from within a block. For example, you can cause the first simulation run to use row 4 of a file, rather than the default use of row 1.

Turning sensitivity on and off

You can control sensitivity either globally for the model as a whole or locally at the dialog parameter level for a specific block:

- Use the Run > Use Sensitivity Analysis command to turn sensitivity analysis on and off for the model as a whole.
- Enable, disable, and delete sensitivity settings for a particular parameter using the Sensitivity dialog.

When you enter sensitivity settings for a value, sensitivity analysis is enabled as long as the *Enable sensitivity* box is checked in the Sensitivity Setup dialog. If you uncheck the box, a dialog value's sensitivity is temporarily disabled so that you don't have to re-enter the number for subsequent analysis.

To remove sensitivity settings from a parameter (as compared to simply temporarily disabling the settings by turning off the parameter's Enable sensitivity box), open the Sensitivity dialog using any of the methods discussed on page 139, then click Delete.

☞ Editing a sensitized parameter in a block's dialog disables the sensitivity settings for that block. When this happens, ExtendSim automatically unchecks the *Enable sensitivity* choice. ExtendSim assumes that if the value is edited, you want to use that new value, not the one that was entered in the Sensitivity dialog. To turn off sensitivity analysis for a parameter for the foreseeable future, open that item's Sensitivity dialog and click the Delete button. This will help prevent accidentally changing the value in a future run of the simulation.

A parameter that has sensitivity settings has a frame inside of it. If sensitivity analysis is active for the parameter (that is, if the *Enable sensitivity* choice is checked), the frame is green. If the

sensitivity analysis is inactive for the parameter or if it is turned off for the model as a whole, the frame is red.

The Model > Open Sensitized Blocks command shows the dialogs of all blocks with sensitized parameters, regardless of whether or not sensitivity is enabled.

Reporting the results

The ExtendSim reporting and tracing features are useful when analyzing output after using sensitivity analysis. The Reporting features show final values and the Trace feature shows values at each step or event. For more information, refer to “Model reporting” on page 184 and “Model tracing” on page 216.

Multi-dimensional scenarios

Sensitivity can be enabled on more than one item at a time. For instance, you may want to vary the values of two Constant blocks and see the interaction between the two items. If you set the sensitivity for the parameters with the *Starting at* option, both values will increment at the same rate. For instance, if you have one parameter start at 5 and increment by 1, and the second parameter start at 100 and increment by 50, and the simulation is run seven times the value pairings will be:

Run #	Variable 1	Variable 2
0	5	100
1	6	150
2	7	200
3	8	250
4	9	300
5	10	350
6	11	400

Often, however, you want to look at all the possible pairings of the two (or more) variables. In this example, you would want to run the model 36 times, with the following pairings:

Run #	Variable 1	Variable 2
0	5	100
1	5	150
2	5	200
3	5	250
...	...	...
7	6	100
8	6	150
...	...	...
35	11	400

In order to perform this kind of multi-dimensional analysis, you need to get the values from a file. The most convenient way to do this is to create a file that has two columns separated by a

tab character with all the desired pairings. For instance, the file for this example would start with:

5 100
5 150
5 200
...

In the Sensitivity Setup dialogs for the two parameters, choose **Read from file** and enter the file name. For the first variable, enter **1** for the column number; for the second variable, enter **2** for the column number.

When you run a multi-dimensional analysis, you usually use a Write block (Value library) to write out the values to examine. In the Write block, select a data destination on the Send Data tab and check **Row (or record) index is equal to run number** on the Options tab. If the data takes a long time to transmit, you should check **Only write to (data destination) at the end of the run** on the Send Data tab.

Scenario analysis

Scenario analysis is a method for systemically and strategically examining the outcome of different model configurations. The purpose is to support the exploration and analysis of alternatives, gaining insight into why your system behaves the way it does and how it can be improved and managed. ExtendSim facilitates scenario analysis through the Scenario Manager block (Value library) which can be added to any model to control all aspects of the analysis. The Scenario Manager essentially keeps track of multiple what-if models, all based on the same model. It offers a highly flexible framework for experimentation and analysis.

The ExtendSim Scenario Manager is a data storage and analysis system that supports analysts as they input, view, manage, and analyze various scenarios. It allows them to use a single model to explore and evaluate an unlimited number of options while providing precise control over model changes as well as easy access to the results. And it provides automated methods for quickly changing variable information and for generating scenario reports. It does this by assigning a named scenario to each combination of variables you choose to examine. For example, you can create best-, worst-, and typical-case assumptions for input values. Then recall each of these scenarios to see how the model behaves under different conditions. You can also recall scenarios in combination, such as best-case sales and worst-case costs.

The ExtendSim database is a central part of the Scenario Manager's data storage and organization. A reserved (hidden) database named *Scenario DB* is automatically created when a Scenario Manager block is added to the model. This database is used to store the inputs and results for the scenarios. The Scenario Manager also facilitates using a design of experiments (DOE) methodology for more efficient performance.

The Scenario Manager is one of three ExtendSim analysis tools; see “Comparison of analysis methods” on page 157 for differentiators. While the Scenario Manager will meet the needs of most ExtendSim modelers, like most ExtendSim blocks it is open source. Thus a programmer can modify and enhance it should the need arise.

How the Scenario Manager works

For scenario management you evaluate and compare a number of model configurations or *scenarios*, where each scenario is one set of model inputs or *factors*. These factors are typically specific dialog parameters (such as the maximum number of items in an Activity) but can also be complete databases, tables, fields, or records. A scenario can have any number of factors

and each factor can have any number, or *levels*, of different values that it can take on. (For example, setting the maximum number of items in the Activity to 2, 6, or 10 would be 3 levels.)

At the beginning of each set of simulation runs, or *replications*, the Scenario Manager copies the model factors for the current scenario into the appropriate locations in the model. Typically each scenario is run multiple times to capture any stochastic behavior. At the end of each of these replications, the results of interest (*responses*) are generated, recorded, and summarized. After all replications have completed, the scenario results are stored in tables in the Scenario DB database, which is a reserved database. You can also generate a report and export the data to a different ExtendSim database, an Excel workbook, JMP, Minitab, or a text file.

☞ It is best to set aside some time, or use the ExtendSim Analysis RunTime version, for all of the scenarios to run. Depending on the size and complexity it may take minutes to hours to complete the entire set of runs. You should also be careful not to create too many scenarios. This is particularly easy to do if you create a full factorial design.

⚠ If you use the Scenario Manager block to exchange data with JMP or Minitab, those applications and their drivers must be 64-bit compatible.

Tutorial I (dialog parameters)

☞ Most analysts would only use dialog parameters in their scenarios, so this tutorial focuses on those. For advanced analysis, you can also use databases, tables, and so forth, as discussed on page 150.

The following example uses a tutorial discrete event model to investigate how changes in a number of factors affect the average wait time and throughput. However, the same concepts apply for continuous process and discrete rate models.

The table below shows the model factors and how they will be varied:

Factor	Values
Car interarrival rate	3, 3.5, 4, 4.5, 5
Wash only time	5, 6, 7
Wash and wax time	6, 7, 8

Each scenario will be run 5 times to capture the variation in the results between one run and the next. To evaluate all of the possible combinations requires 225 replications (45 scenarios [5 x 3 x 3] at 5 runs each).

☞ Fortunately, this model runs very quickly so this number of replications is not a problem. However, if the model took significantly longer to run or if there were more factors, a design of experiments (DOE) should be used to reduce the number of scenarios.

Steps

The steps for performing scenario analysis are:

1) Add a Scenario Manager block (Value library) to a model
2) Identify and add the factors (model inputs) you want included in the analysis
3) Identify and add the responses (model results) you want to analyze

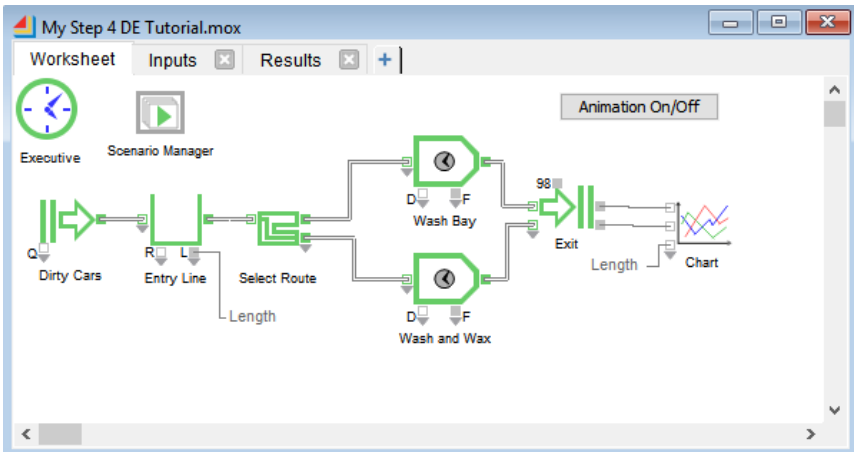
4) Determine what you want in the report
5) Determine the DOE, then generate and run the scenarios
6) Analyze the results
7) Optional. Export the scenarios to Excel, JMP, Minitab, a text file, or an ExtendSim database for further analysis

Add the Scenario Manager block

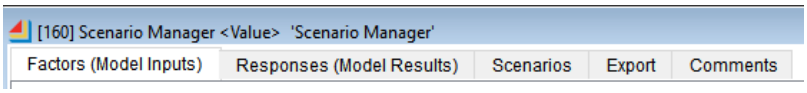
- Open the model named *Step 4 DE Tutorial* (Documents/ExtendSim/Examples/Tutorials/Discrete Event)
- Save the model as "My Step 4 DE Tutorial" so you don't overwrite the original model
- Place a Scenario Manager block (Value library) in the model. The block can be placed anywhere but the top of the model will provide easiest access.
- Since all the other blocks are labeled, label the block "Scenario Manager"

☞ Adding a Scenario Manager block to a model automatically creates a reserved (hidden) database named Scenario DB to the model. This is where the results of each individual scenario, as well as model factors, are stored. See "Analyze results" on page 149 for more information.

Your model should now look similar to the following screen shot:



- Open the dialog of the Scenario Manager block



As seen above, the Scenario Manager has tabs for Factors, Responses, Scenarios, and Export. These represent the steps in using the Scenario Manager.

Identify and add factors and values

The next step is to determine and add to the Dialog Factors table any dialog factors (model parameters that are inputs) that you want to include in the analysis.

⚠ Don't perform scenario analysis on input parameters that have been calculated by the system and are displayed in non-editable fields. For instance, don't use as a factor a delay time that is determined by an item's attribute value, such as the delay shown here in a display-only field.

Delay (D): 6.157127613

The information is in the following table.

Block Name (Label)	Parameter	Factor Name	Values
Create (Dirty Cars)	Mean	Cars Rate	3, 3.5, 4, 4.5, 5
Activity (Wash Only)	Delay (D)	Wash Time	5, 6, 7
Activity (Wash & Wax)	Delay (D)	Wash&Wax Time	6, 7, 8

Methods for referencing dialog variables as factors

There are 3 ways to add a dialog variable as a factor to the Dialog Factors table:

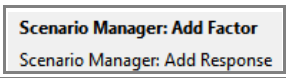
- Shift-click. Shift-click the target parameter in the target block's dialog and select where it should go.
- Clone drop. Click on a clone of the target parameter from the target block's dialog, then click on the Scenario Manager's icon.
- Manual. Manually enter the dialog variable name into the factors table.

Interarrival time factor

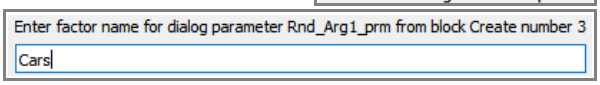
☞ This section uses Shift-click to associate the parameters with the Scenario Manager. The response section that follows uses clone-drop.

In the Queue (Dirty Cars) dialog:

- While holding down the Shift key, click the *Mean* variable.
- In the popup menu that appears, select **Scenario Manager: Add Factor**



- In the dialog that appears (shown on the right), name the factor **Cars Rate** and click OK to save the name.



- Close the Queue (Dirty Cars) dialog.

Notice that the Scenario Manager's Dialog Factors table now contains all the information about Dirty Cars – the block it came from, the number it is currently set at, and so forth.

Other factors

- Repeat the steps used for the Cars Rate factor, above, to place the remaining two factors (from the table on page 145) in the Scenario Manager's Dialog Factors table

The Dialog Factors table should now look like this:

	Factor Name	Block Name	Block Number	Block Label	H-Block Name	H-Block Label[num]	Dialog Var
1	Cars Rate	Create	1	Dirty Cars		Scenario Manager[-1]	Rnd_Arg1_prm
2	Wash Time	Activity	4	Wash Bay		[-1]	waitDelta_prm
3	Wash&Wax Time	Activity	13	Wash/Wax Bay		[-1]	waitDelta_prm

Entering values

Now that the factors have been entered, it is time to enter their proposed values. There are two methods for entering the values:

- If you choose a *DOE method*, enter the minimum, maximum, and step (the increment between the minimum and maximum values) for each dialog variable factor in the table in the *Factors* tab.
- If you choose to *manually enter the scenario configuration*, enter a value for each factor and every scenario directly in the Scenarios table of the *Scenarios* tab.

Since this example will use the Full Factorial DOE method:

- Go to the Factors tab of the Scenario Manager block
- Enter the minimum, maximum, and step values for each of the factors as shown below

Factor Name	Min Value	Max Value	Step
Cars Rate	3	5	0.5
Wash Time	5	7	1
Wash&Wax Time	7	9	1

The Wash Time has a Step of 1 with a Min Value of 5 and a Max Value of 7. That means that the Scenario Manager will explore times of 5, 6, and 7 minutes as requested in the table on page 145.

Identify and add targeted responses

The next step is to add responses to the Scenario Manager.

☞ While you could use any of the 3 options from page 146, this section uses the clone drop method to associate the parameters with the Scenario Manager.

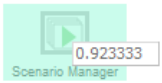
The responses (results of interest) are in the following table:

Block Name (Label)	Tab	Parameter	Response Name
Queue (Wash Bay)	Results	Length: Average	Ave Length 1
Queue (Wash & Wax)	Results	Length: Average	Ave Length 2

Wash Bay

- Close the dialog of the Scenario Manager block
- Open the Queue block and go to its Results tab
- Select the Clone cursor, either by right-clicking in the dialog or by going to the menu
- Use the Clone cursor to click in the variable field for *Length: Average*; this creates a clone of that variable

- Hover the cursor over the icon of the Scenario Manager block until the icon is highlighted and the cloned variable appears on the block's icon, as shown here



- Click once on the Scenario Manager's icon
- In the dialog that appears, select **Response**
- Name the response **Ave Length 1**
- Close the Wash Bay's dialog

This causes the selected variable to be placed in the Scenario Manager's Responses tab.

Wash & Wax

- Use either the Shift-click or the clone-drop method to add the field for *Length: Average* response of the Wash & Wax queue to the Scenario Manager
- Select the **Response** option and name the response **Ave Length 2**

The Dialog Responses table in the Responses tab should look like the following:

	Resp Name	Block Name	Block Number	Block Label	H-Block Name	H-Block Label[num]	Dialog Var
0	Ave Length 1	Queue	10			Scenario Manager[-1]	AveLength_prm
1	Ave Length 2	Queue	91			Scenario Manager[-1]	AveLength_prm
2							

Determine what will be in the report

The fourth step is to determine what type of report you want and what you want in the report.

The Dialog Responses table in the Responses tab is where you specify what level of statistics you want the report to calculate and which responses you want included in the report. In most cases, you would leave them set to the defaults. However:

Min/Max	Report Set	Include in Report
None	M	☑
None	M	☑

- If you are using JMP, the *Min/Max* column provides additional information for analyzing the results in JMP.
- The *Report Set* column determines which report is run for the specific response. The default is to only show the mean or average value of the response for each scenario, but more detailed statistics can be added by selecting a different report set as shown here.

- The *Include in Report* column is for choosing which responses will be included in the report.

For this example:

- Keep the default settings

Select the DOE and run scenarios

The Scenarios tab of the Scenario Manager block is for choosing a method for the DOE as well as for creating and running the scenarios. There are several DOE methods you can choose:

manual, full factorial, JMP custom, and Minitab optimal; they are described more on page 157. For this tutorial you will use the full factorial method.

- Go to the Scenarios tab of the Scenario Manager block
- In the Run Control section, choose **DOE method: Full factorial design**
- Click the **Create Scenarios** button

This generates all of the combinations of the factor values. There will be a total of 45 rows (5 x 3 x 3 levels) in the Scenarios table. By default all of the scenarios are selected; to not run a specific scenario, uncheck its *Select* check box.

- Since you want all of the scenarios to run, leave all of the **Select** check boxes selected
- In the Scenarios tab, enter **Runs per scenario: 5**
- Click the **Run Scenarios** button

As the scenarios run, the Status frame reports the run count and the scenario count. As displayed, 225 replications (5 runs x 45 factors) are required to complete the scenario analysis. When the scenarios have completed running, you should see the following in the Scenarios table:

Scenario Name	Cars	Wash Time	Wash&Wax Time	(M) Ave Length 1	(M) Ave Length 2
Scenario 01	3	5	7	0.89820995	0.49471272
Scenario 02	3	5	8	0.90764734	0.46442931
Scenario 03	3	5	9	0.84853145	0.55478145
Scenario 04	3	6	7	0.9437500	0.40583333
Scenario 05	3	6	8	0.96843652	0.35593652

☞ See page 156 for suggestions on how to cause scenarios to run faster.

Analyze results

The Scenario DB database, a reserved database, automatically stores all the responses of each scenario in a series of tables. The tables correspond to the rows of the Scenario Name column in the Scenarios table. The All Scenarios table of the Scenario DB contains information about all the model factors and responses. You can use the Scenario DB to perform analysis, or export selected information to a different ExtendSim database, to Excel, or to JMP or Minitab, as discussed on page 150.

Some additional things to notice for your analysis are:

- To not include a response in the report, uncheck *Include in Report* in the Responses tab
- Each response will have the type of report indicated in the Report Set column of the Responses tab. The default is to display the mean value for each response variable but you can also display the confidence interval, standard deviation, and so forth.

☞ Changing a Report Set after the scenarios have been run causes the Scenario Manager to recalculate the results with the new set. This is useful if you change your mind about the report you want.

- Depending on what has been selected as the Response tab's Report Set, each response will have one or more columns in the Scenarios table of the Scenarios tab. These columns summarize the results of each set of replications for each scenario. The type of report, for example *CI for confidence interval*, is indicated in parentheses in the header of the column.

- On the Scenarios tab, the Details column in the Scenarios table opens a database table containing the original data that was used to calculate all the statistics. The table is from the Scenario DB database that was created when the Scenario Manager block was first placed in the model.
- By default all of the scenarios are selected in the Scenarios table; to not run a specific scenario, uncheck its Select box

Export the results (optional)

The form of the Scenario DB database tables may not be exactly what you want. To do additional analysis using a different format, export a report on the Export tab. For example, you could export complete results, including the values from every scenario, to JMP. Then use JMP's analysis capabilities to compare the different scenarios. To export the results:

- Choose the type of report.
  - *Complete Results* exports the contents of the Scenario DB database – the factors and responses for every run in every scenario
  - *Displayed Statistics* exports the statistics for each scenario as shown in the Scenarios table. What gets displayed depends on what is selected in the Response tab's Report Set column.
  - *Complete Statistics* exports all of the available statistics for each scenario. This is every possible statistic allowed by the Report Set column of the Responses tab.
- Select the destination for the report: an ExtendSim database, Excel, Minitab, JMP, or a text file.


So that you don't accidentally overwrite the data, do not export to the Scenario DB database.

- Set the destination parameters for the exported data. This varies based on the selected export destination.
- Click Export Report

Scenario name	Cars Rate	Wash Time	Wash&Wax Time	(M) Ave Length 1	(M) Ave Length 2
Scenario 01	3	5	7	0.922650931	0.449166667
Scenario 02	3	5	8	0.8611874	0.552002468
Scenario 03	3	5	9	0.891692454	0.4575
Scenario 04	3	6	7	0.949616531	0.40875
Scenario 05	3	6	8	0.937171168	0.440504502
Scenario 06	3	6	9	0.898900914	0.472178514

Once you have exported the results, analyze the scenarios by sorting, plotting, or generating comparison statistics.

Tutorial II (database variables)


This section assumes you understand how to create ExtendSim databases, including how to link to them and how to enter parent/child relationships. Databases are described "ExtendSim databases for internal data storage" on page 232 and in the separate document titled ExtendSim Database.pdf.

So far you have controlled only dialog variables with the Scenario Manager. While most analysts would use only dialog parameters as factors and responses, a select few of you will want

to do more advanced analysis. With the Scenario Manager you can use entire databases or a subset of a database (tables, fields, or records), as factors and/or responses. These database variables can be used in addition to, or instead of, dialog variables.

☞ Using database variables is more complicated than using dialog parameters. However, it is very useful when you have sets of data, rather than single variables, that you want to swap in and out for each scenario. For example, several production schedules for a manufacturing model.

To use the Scenario Manager with a database variable, you must create an ExtendSim database and any necessary tables, fields, and records (this tutorial assumes you already know how to do that). You also need to specify targets and sources, as discussed below.

Targets and sources

The concept of targets and sources is specific to the Scenario Manager and allows more sophisticated analysis.

Definitions

- A *database variable* is a variable that references an entire database or a subset of a database, such as a database table or field.
- A *source* is a database variable that gets copied to a target
    - *Factor sources* are the set of values that a database variable can take for all the different scenarios. They contain all of the information for all of the scenarios.
    - A *response source* is the value of a database output variable for a particular replication. Each response source only has the information for the current set of replications.
- A *target* is a database variable whose value is defined by a source
    - *A factor target* is a database input variable whose value has been defined by a particular factor source. Each factor target only has the information for the current set of scenario replications.
    - *Response targets* are a collection of output database variables where each variable's value is defined by a particular response source. They contain all of the results for all of the scenarios.

Overview

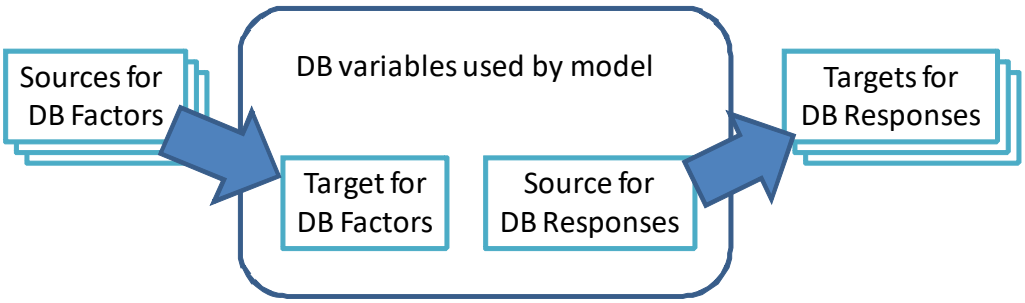
- The architectures of the targets and the sources must match. For example, if the source is a database table, the target must also be a database table with the same number of fields.
- A minimum number of sources and targets must be created for the model:
    - For database *factors*, at least two sources and at least one target is required. (Each set of data is a source so having one source does not make any sense when there are sets of data.)
    - For database *responses*, one source and a number of targets is required. The number of targets depends on whether ExtendSim creates the source and targets or you do it manually. If *Auto-Create Target* is selected in the Database Responses table, ExtendSim will create a new database target for every simulation run. If you do it manually, create one source and one target per scenario.

- If you run the model for testing purposes without using the Scenario Manager, the identified variables can be used in the model. However, these variables are just placeholders and their contents will change when you use the Scenario Manager.
  - *Factor target*. When testing the model, use the *target* database factor in your blocks. (Either link a parameter or data table to the target variable or use a Read or Write block to reference the target variable.)
  - *Response sour*ce. Similarly to the above, build the model as if you will be using a placeholder database response variable. However, use the *source* rather than the target.
- Sources are always copied to targets. When and how this happens depends on whether the database variable is a factor or a response.
  - *Factor.* At the start of each set of scenario replications, the Scenario Manager copies the contents of the source database factor variable into the target database factor variable. This is done automatically by the Scenario Manager.
  - *Response*. At the end of each simulation run the database variables used in the model are copied from the source to the target database variables for that scenario.

☞ The contents of the table, rather than the table itself, is copied. Thus the index of the target database variable does not change.

Diagram of sources and targets

The following diagram shows how sources and targets are related to factors and responses.



☞ Database factors are copied from the sources to the target at the start of each scenario run. Database responses are copied from the source to the targets at the end of each run.

Steps for scenario analysis

The steps for performing scenario analysis using database variables are:

1) Open a model
2) Create the required database, tables, fields, and records for the model
3) Optional. If you plan to perform DOE, create an additional table for each database variable, where each table has a list of the possible values for that variable.
4) Create links between the model and the database variables
5) Enter information in the Database Factors table
6) Enter information in the Database Responses table (this example does not do this step)
7) Generate and run the scenarios

8) Optional. Export the report.

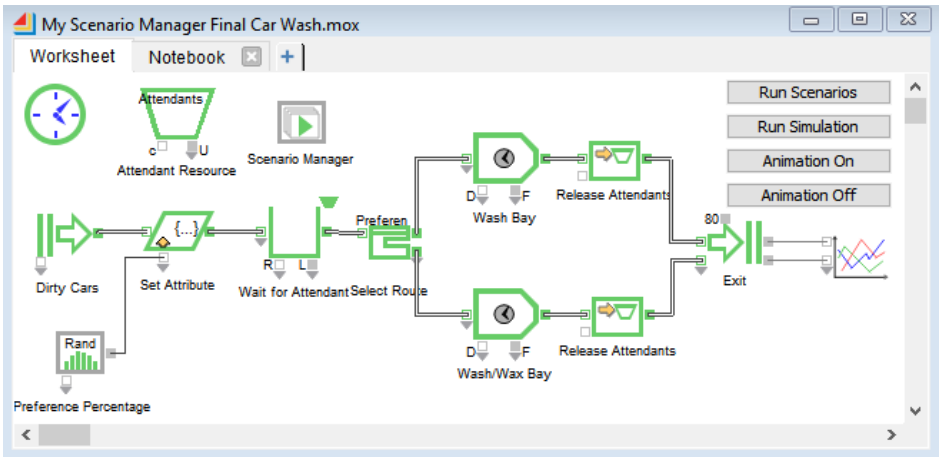
☞ The above steps assume database tables are used for factors and responses. You can also use fields or records, or even entire databases, as targets and/or responses.

Open the model

In the previous tutorial you used dialog variables as factors and responses. This tutorial uses database tables as factors and responses.

☞ This tutorial uses a similar model that already has the required database structure in it.

- Open the Scenario Manager Final Car Wash model (ExtendSim/Examples/How To/Scenario Manager)
- So that you don't overwrite the original, save the model as My Scenario Manager Final Car Wash.



This model has some differences from the previous model, but like that model this model has:

- Three factors in its Dialog Factors table: *Wash Time*, *Wash&Wax Time*, and *Cars Rate*
- Two responses in its Dialog Responses table: *Ave Length 1* and *Ave Length 2*

This tutorial will keep these dialog factors and responses but will add a fourth factor (the percentage of cars wanting *Wash* or *Wash & Wax*) and put it in the Database Factors table.

About the tutorial

In the model, the proportion of cars wanting either wash only or wash and wax is set in the empirical table of the Random Number block (labeled "Preference Percentage"). The table is shown at right.

	Preference	Probability
0	Wash only	0.75
1	Wash and Wax	0.25

For this tutorial the Wash or Wash & Wax factor will have three combinations of percentages, which represent 3 levels:

1) 75% wash and 25% wash & wax – the base case
2) 90% wash and 10% wash & wax – the more wash case
3) 50% wash and 50% wash & wax – the less wash case

As in the earlier tutorial, each scenario will be run 5 times to capture the variation in the results between one run and the next. To evaluate all of the possible combinations now requires 675 replications (135 scenarios [5 x 3 x 3 x 3 levels] at 5 runs each).

A database structure for the model

So that you can focus on the tutorial, the required database has already been created for this model. As shown at the bottom of the Database menu, the Scenario Manager Final Car Wash model has a *Model Data* database. This database stores the different options and probabilities for the wash or wash/wax factor in 6 tables:

- A Parent table, *Wash Type*, that lists the options for the two wash types: wash only and wash & wax. Its contents are shown on the right.

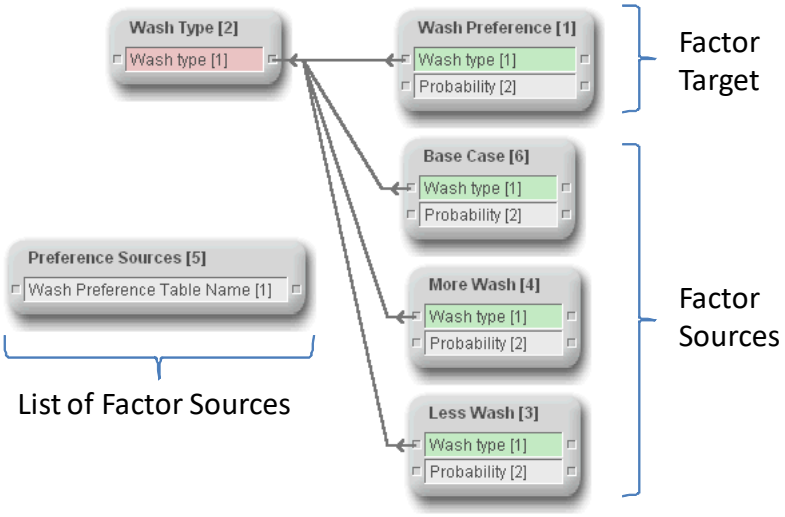
	Wash type[1]
1	Wash only
2	Wash and wax

- Three factor source tables, each of which has one of the probability combinations: *Base Case* (75% wash/25% wash&wax), *Less Wash* (50%/50%), and *More Wash* (90%/10%). These are Child tables that get their wash type information from the Wash Type table.
- A factor target table named *Wash Preference*. Each record in the factor target table has a wash type field and a probability field associated with that wash type. This table is used as a placeholder for model data. Thus before the scenario runs begin, the table is identical to the Base Case source table.
- A factor source list table, named *Preference Sources*. This table has one field named Wash Preference Table Name; its field type is List of Tables. (In other words, this is a table of tables.) The field has three records where each record is the name of one of the factor source tables. The field and its records are shown at right. This table is used when performing DOE, so that you will know which tables to select from when creating the design. See “Create a source list (optional)”, below, for more information.

	Wash Preference Table Name[1]
1	Base Case
2	More Wash
3	Less Wash

☞ Because they would commonly be used, the Model Data database has two tables that weren’t strictly required for the example: the Parent table (Wash Type) and the list of source tables (Preference Sources).

The structure of the Model Data database is shown below.



Create a source list (optional)

If you plan to perform DOE, you should consider creating an additional table for each database factor variable where each table has a list of the possible values for that variable. This is a table of tables, or a factor source list.

Using a factor source list allows you to choose from a subset of database variables rather than from all of the database variables in the model. This makes it easier to create the scenarios manually and to use DOE. Without this list of factor sources, any table in the database would be listed in the Database Factors table as a possible factor source table.

As discussed above, a factor source list named Preference Sources, has already been created for this example. As you will see later, checking *Use Source List* in the Database Factors table limits the possible entries to those in the source list (in this case, to the Preference Sources database table).

Link to the target table

To control the model's wash preference probabilities, the Random Number block's empirical probability table must be linked to the target table – Wash Preference:

- Click the **Link** button in the Random Number block's Preference/Probability table
- In the Link dialog popup menu, select the database named **Model Data**
- Choose the table named **Wash Preference**, then click the **Link** button
- Close the Random Number dialog

Enter database factors

Unlike with dialog variables, you cannot Shift-click or clone drop a database variable. However, the Database Factors table also has popups for selecting the appropriate information.

- In the Factors tab of the Scenario Manager's dialog, check the **Enter database factors** check box to enable the Database Factors table

- Use the +/- resize button to add one row to the table
- Use the down arrows to select, or if needed manually enter, the following information in the Database Factors table:

Column Header	Enter or Select This Information
Target Name	Wash or Wash/Wax
Target Type	Table
Target DB	Model Data
Target Table	Wash Preference
Target Field	(leave this cell blank)
Target Record	(leave this cell blank)
Use Source List	(Check the box. See page page 155 for more about source lists.)
Source Database	Model Data
Source Table	Preference Sources
Source Field	Wash Preference Table Name

Generate and run the scenarios

Go to the Scenarios tab and generate and run the scenarios as you did in "Select the DOE and run scenarios" on page 148. If you use the Full Factorial method and have 5 runs per scenario, 135 scenarios should be created and it will take 675 replications.

Other models

For other example scenario analysis models, see the *Scenario Manager DB Factors* model or the *Call Center Scenario Manager* model. They are located at Documents/ExtendSim/Examples/How To/Scenario Manager.

Suggestions for running scenarios faster

To have the scenarios run as fast as possible:

- Turn off animation
- Set all the charts and graphs to not open automatically
- Rather than use the Run Scenarios button in the Scenario Manager's dialog, close the dialog and run the scenarios using the *Run Optimizations or Scenarios* button in the Model toolbar. Or clone the Run Scenarios button from the dialog to the worksheet and use that to run the scenarios, after closing the block's dialog. (If you use the Run Scenarios button in the dialog, you won't be able to close the dialog until all the runs end.)
- Close any other open dialogs.
- Off-load the model and its runs to a different device using the Analysis RunTime version of ExtendSim.
- Uncheck *Play sound at end of run* in the Edit > Options > Model tab. It won't cause the scenarios to run any faster but you'll avoid the annoying beep at the end of each run.

☞ Running scenarios can take a long time. Once the runs have started, you can click the Stop button in the Scenario Manager's dialog to stop the remaining scenarios from running. That works

better than clicking the Stop button in the toolbar or pressing Ctrl/Command + period on the keyboard, which only stop the current scenario run.

DOE methods

As introduced on page 148, the Scenarios tab of the Scenario Manager block allows you to choose the DOE method. These are described more fully below.

DOE Method	Description
Manual	Enter factors for each scenario manually
Full factorial	*Create Scenarios* creates one scenario for each combination of factor values. Since every possible combination of factors is generated, this takes the longest to run.
JMP custom (Windows only)	*Create Scenarios* opens JMP and causes it to create a custom or D-optimal design, then displays the factors and responses in both ExtendSim and JMP.
JMP custom – create and run (Windows only)	*Create and Run Scenarios* opens JMP, causes it to create a Custom or D-optimal design, displays the factors and responses in both ExtendSim and JMP, then immediately runs the scenarios.
Minitab optimal (Windows only)	*Create Scenarios* opens Minitab and causes it to create an Optimal or D-optimal design, then displays the factors and responses in both ExtendSim and Minitab.
Minitab optimal - create and run (Windows only)	*Create and Run Scenarios* opens Minitab, causes it to create an Optimal or D-optimal design, displays the factors and responses in both ExtendSim and Minitab, and then immediately runs the scenarios.

The options for JMP and Minitab require either JMP from SAS Corporation or Minitab from Minitab Inc. to generate the design and analyze the results.

☞ The JMP custom and Minitab optimal methods result in far fewer scenarios than full factorial.

⚠ If you use the Scenario Manager block to exchange data with JMP or Minitab, those applications and their drivers must be 64-bit compatible.

Comparison of analysis methods

The Scenario Manager is just one of the ExtendSim analysis tools. As described in the table below, each tool has a different purpose.

Technique	Sensitivity Analysis	Scenario Manager	Optimization
Purpose	Study the behavior of the model in response to changes in a single input parameter	Study and compare the behavior of the model in response to changes in multiple input parameters. Learn how the system behaves under different conditions.	Determine the best configuration of model parameters, given a specific goal (usually to minimize cost or maximize profit)

Technique	Sensitivity Analysis	Scenario Manager	Optimization
Question	If a parameter had this value, how would it effect model outputs?	If several parameters had these values, how would that effect model outputs?	What configuration of parameters allows a model to behave optimally?
Advantages	Easiest to specify Included in all ExtendSim products	Gives the most complete understanding of how the model reacts to different factors Factors can be from databases as well as dialog variables Automatically records each response for every simulation run Design of experiments can reduce the number of scenarios required to study factor interactions	Determines the optimum model configuration Constraints can be added to filter out infeasible model configurations Included in all ExtendSim products
Disadvantages	Evaluating more than one factor is cumbersome Additional modeling is required to record model results	A set of scenarios must be created by the modeler May require many simulation runs to evaluate all of the scenarios	An objective function is required Does not record the results of each individual run There is a practical limit on the number of factors, and their possible values, that can be examined May require many simulation runs to determine the optimal configuration
Refer to	“Sensitivity analysis” on page 138.	“Scenario analysis” on page 143.	“Optimization” on page 158.

Optimization

Optimization is a powerful feature that can automatically determine ideal values for parameters in a model. It does this by running the model many times using different values for selected parameters, searching the solution space until it is satisfied that it has found an acceptable solution. It then populates the model with the optimized parameter values.

ExtendSim facilitates optimization by making the optimization algorithm available within a block that can be added to any model to control all aspects of the optimization. Furthermore, having a block do the optimization increases flexibility and opens up the method and source code to users who might want to modify or create their own customized optimization blocks. The Optimizer block (Value library) uses an evolutionary algorithm to reduce the number of times the model has to run before a solution is found.

How optimization works

Optimization, sometimes known as "goal seeking," is a useful technique to automatically find the best answer to a problem. The "problem" is stated as an *objective function* or *cost equation* that ExtendSim tries to minimize or maximize to save you going through the tedious process of manually trying different values with each model run.

Like most optimization algorithms, the ExtendSim Optimizer solves models using an initial population of possible solutions. Each solution is explored by running the model several times using different values for some selected parameters, averaging the samples (for stochastic, random models), and sorting the solutions. The best solution sets of parameters are then used to derive slightly different but possibly better solutions. Each new derived solution is called a generation. This process continues for enough generations until the Optimizer determines that there are probably no better solutions in sight. The Optimizer then terminates the simulation runs and populates the model with the best solutions it has found.

The downside to optimization is that the model needs to run repeatedly and this can take a long time with large models. Also, optimization algorithms have an inability to tell when the best solution has been found, or even if the best solution has been attempted. A good approach is to allow the optimization to run for a sufficient number of cases and to then see if the population of solutions has converged. Then try the optimization procedure several additional times to make sure that the answers agree (or are close) and that the first answer is not a false or sub-optimal one.

☞ There are no optimization algorithms that are guaranteed to converge to the best answer in a finite time. The more time that you can give optimization to run, the better chance that it will provide the optimum answer. Consider using the Analysis RunTime version of ExtendSim when running optimizations, since you can off-load the model and its runs to a different device.

Steps for using optimization

The steps needed to optimize a model are listed below. The tutorial that follows illustrates these steps.

1) Add an Optimizer block (Value library) to a model.
2) Define the form of the objective function.
3) Determine which variables the equation needs and "clone-drop" them onto the Optimizer.
4) Set the limits for those variables in the Optimizer's Variables table.
5) Derive the equations for the objective function.
6) If variables need to constrained to certain values, add constraint equations.
7) Set the Optimizer's Run Parameters for a random or non-random model, then run the optimization.

☞ The following tutorial assumes that you know how to clone parameters (see "Cloning" on page 72) and that you are comfortable deriving equations (if not, see "Equation-based blocks" on page 189).

Tutorial assumptions

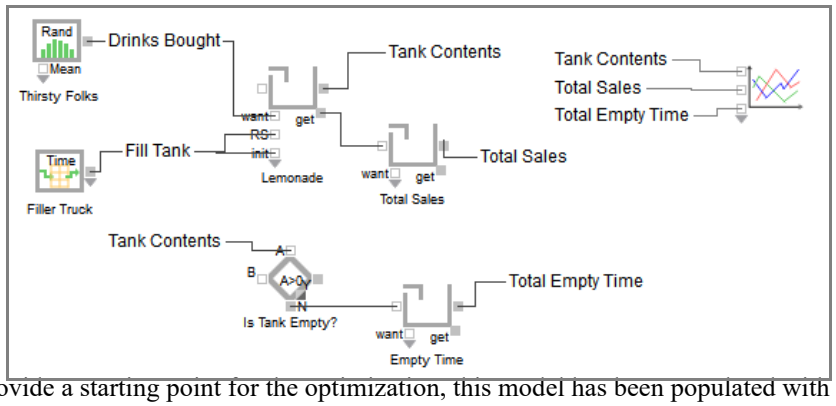
The model for this tutorial represents a drink stand at a county fair; it has these assumptions:

- Drinks are dispensed from beverage tanks that can range in size from 1000-8000 drinks. (In the example model, 1000 is used for the initial tank size setting.)
- A truck delivers the beverage tank at the start of the day and exchanges it periodically throughout the day. (The model is initially set to get a second tank after 240 minutes.)
- The truck exchanges the beverage tank for a new one of the same size. There is a $1 per drink beverage charge plus a delivery charge of $1,000 per tank.
- So that the beverage company will know when to deliver tanks and what size tank you will use for the day, arrangements regarding tank size and delivery frequency must be made at the beginning of the day.
- People purchase drinks according to a random distribution; the cost per drink is $2.50.
- If the beverage tank becomes empty, you lose an estimated $100 per minute in sales because customers already in line go to another stand and new customers are discouraged from getting in line.
- If you exchange the tank too often, you lose money because the beverage inside the old tank gets taken away along with the tank.
- This model ignores other expenses, such as labor.
- The model runs for a simulation time period of 480 minutes (8 hours).

The goal of this tutorial is to optimize both how big of a tank you should order and how often the tanks are exchanged. It is a simple continuous model, but a good example to show some of the optimization techniques you will use in any type of large model.

Open the model

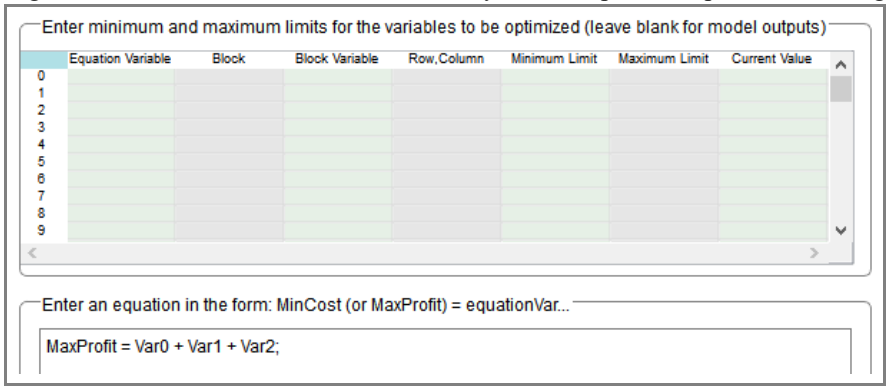
- Open the Optimize 1 model from the folder Documents/ExtendSim/Examples/How To/Optimization
- So that the example file isn't overwritten, save the model as "MyOptimizer".



To provide a starting point for the optimization, this model has been populated with initial assumptions for the decision variables: deliveries are repeated every 240 minutes and the tank size is 1,000 drinks.

Add an Optimizer block

- Place an Optimizer block (Value library) in a convenient location in the model.
- To get an idea of what it looks like without any entries, open the Optimizer's dialog.



As seen here, the Optimizer's Objectives tab has two sections:

- A *Variables table* for entering variables and specifying their limits.
- An *Equation pane* for entering an objective function. An objective function consists of one or more equations that use the variables.

- Close the Optimizer's dialog.

Determine the form of the function

The Optimizer block will try to reach a goal by changing the values of model variables based on an equation that measures how close the goal is.

The first step is to lay out the form of the objective function. This helps clarify what will be optimized and which factors affect that goal. In most cases, you want to minimize a cost or maximize a profit.

This model tries to maximize the beverage stand's profit. The factors that affect profit are:

- Beverage costs ($1 per drink plus $1,000 per tank)
- The revenue for each drink sold ($2.50 per drink)
- A penalty or reduction in profit if you run out of drinks to sell ($100 per minute)

Using this information, a (non-code) form for the objective function could be:

MaxProfit = $2.50#Sold - #Deliveries*($1000 + #Drinks*$1.00) - time empty * $100*

☞ You'll see the actual objective function below. For now, note that you cannot use the # sign in the actual equation since # indicates a preprocessor directive, such as #include.

How to add variables to the Optimizer

The form for the objective function is the basis for the cost equation that you will enter in the Optimizer's dialog. Before doing that, you need to obtain the necessary model variables (called *decision variables*) so the equation can reference them. In some cases the required variable will be a dialog parameter and in others it will need to be calculated based on a dialog parameter.

An Optimizer block's access to dialog variables is accomplished by using either:

- Clone dropping, where you use the Clone cursor to place clones of the desired dialog variables on the Optimizer's icon
- Or Shift-clicking the variable with the Block/Text cursor and selecting *Optimizer: Add Parameter*.

Either operation adds information about the variable to the Variables table. It also enables the Optimizer block to remotely read and change the value of that variable in the model, so that it can explore possible solutions.

Creating the variables

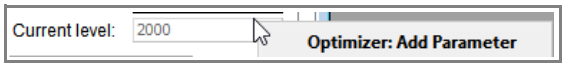
Using the variables nomenclature from the form for the equation (*MaxProfit = $2.50*#Sold - #Deliveries*($1000 + #Drinks*$1.00) - time empty * $100), the steps are:*

 # sold

The total number of sales for the day is an output variable calculated by the model; it is directly available in a block's dialog.

Use Shift-click to add the total sales variable:

- Open the dialog of the Holding Tank block labeled "Total sales".
- While holding down the Shift key, use the Block/Text cursor to click once on the value (not the label) for the **Current level** parameter (2000).
- In the dialog that appears, select **Optimizer: Add Parameter**.



- In the subsequent dialog, enter **numSold** as the name for the variable. This puts the variable into the first row of the Optimizer's Variables table.
- Close the Holding Tank's dialog.

☞ Starting with the first row, each dialog item is automatically placed into successive rows of the Optimizer's variables table.

 # deliveries

Unlike the other required variables for this example, the number of deliveries is not directly available as a dialog item. However, that value can be calculated using the frequency of deliveries, as you will see on page 164.

Use clone-drop to add the variable:

- To get the repeat delivery time, open the dialog of the Lookup Table block labeled "Filler Truck".
- Select the Clone cursor, either by right-clicking in the dialog or by going to the menu.
- Using the Clone cursor, click once in the variable field (but not the label or checkbox) for the **Repeat table every** parameter (240). This creates a clone of that parameter.
- Hover the Clone cursor over the icon of the Optimizer block until the icon is highlighted and the cloned variable appears on the block's icon.
- Click once on the Optimizer's icon to release the clone.

- In the dialog that appears, name the variable **deliveryTimes**.
- Keep this dialog open.

☞ From now on, use whichever method you felt most comfortable with: Shift-click or clone drop.

#drinks

The decision variable for the number of drinks per tank is also located in the Lookup Table block labeled “Filler Truck”.

- Use Shift-click or clone drop to add the data table in the Lookup Table’s dialog to the Optimizer block.
- Name the data table **delTankSize.**
- Close the Lookup Table’s dialog.

time empty

The model calculates the amount of time that the drink tank is empty; this is an output variable.

- Open the dialog of the Holding Tank block labeled “Empty Time”.
- Use Shift-click or clone drop to add the **Current level** parameter to the Optimizer block.
- Name the parameter **emptyTime.**
- Close the Holding Tank’s dialog.

The Optimizer’s variables table

The variables table in the dialog of the Optimizer block should appear as shown here.

	Equation Variabl	Block	Block Variable
0	numSold	Total Sales	Contents_prm
1	deliveryTimes	Filler Truck	Repeat_prm
2	delTankSize	Filler Truck	Data_tbl
3	emptyTime	Empty Time	Contents_prm

Setting limits for the variables

Now that the Optimizer has the necessary variables, you need to enter limits for some of the variables so the Optimizer will know that it should try to change them. For data tables, you also need to specify which cell is to be used in the equation.

☞ Variables without limits are considered outputs from the simulation and the Optimizer will not try to change them.

- Open the Optimizer’s dialog so that the Variables table in the Objectives tab is visible.
- On the first row of the Variables table (numSold) do not enter any limits. This value is not an input to be changed but rather an output value from the Holding Tank.
- On the second row (deliveryTimes) enter a Minimum Limit of **30** minutes (an estimated minimum) and a Maximum Limit of **480** minutes (the simulation end time). This sets the time between deliveries.
- On the third row (delTankSize)**,** you need to tell the Optimizer which cell of the data table to use and what its limits are:
  - In the Block column click once on the cell named “Filler Truck”, causing the dialog of the Lookup Table block to open. Since data table rows and columns start at 0, the cell that holds the number of drinks per tank is at row 0 and column 1 (# Drinks) of the data table.
  - Close the dialog of the Filler Truck.

- In the Optimizer's dialog, for the Row, Col value enter **0,1**.
- For this variable's limits, enter a Minimum Limit of **1000** and a Maximum Limit of **8000** drinks.

☞ Do not enter a decimal point. The absence of a decimal point tells the Optimizer that the value for the number of drinks has to be an integer.

- On the fourth row (emptyTime) do not enter any limits because this value is a model output.

The columns of the Objectives table should now look similar to the table on the right.

	Equation Variabl	Block	Block Variable	Row,Column	Minimum Limit	Maximum Limit	Current Value
0	numSold	Total Sales	Contents_prm				2000
1	deliveryTimes	Filler Truck	Repeat_prm		30	480	240
2	delTankSize	Filler Truck	Data_tbl	0,1	1000	8000	0
3	emptyTime	Empty Time	Contents_prm				378

How to determine the objective function

Now that the limits have been entered, substitute each variable's name for the row that holds the value of interest. The result is:

```
MaxProfit = 2.50*numSold - numDeliveries*(1000.0+delTankSize*1.00) -
  emptyTime*100;
```

The only variable from this equation that has not yet been specified is the **#deliveries** factor, which can be calculated using the **deliveryTimes** variable. The form for an equation to convert the deliveryTimes variable to the #deliveries factor is:

```
numDeliveries = int((endTime-1)/deliveryTimes + 1)
```

How this equation is structured is as follows:

EndTime is a global variable representing the end time of the run, in this case 480 minutes. DeliveryTimes is how frequently the deliveries are repeated, or every 240 minutes. If you were to divide the endTime value of 480 by the deliveryTimes value of 240, it would result in 2 deliveries a day, which appears correct. However, if you instead divide by 250, you get only 1.92 deliveries, even though you still have 2 deliveries, one at the beginning of the day and one at 250 minutes. There will always be a delivery at the beginning of the day, so you need to add 1 to deliveryTimes and truncate any fractional result with the int() function. This results in 2.92, which when truncated gives 2, the correct answer.

If you had a delivery every 480 minutes, there would be 2 deliveries, but the last one would happen at the same time as the stand closes. To remedy this, reduce the endTime by 1, preventing the 480 minute case (the maximum limit) from causing 2 deliveries.

For more details about objective functions, see page 169.

Enter the objective function

The entire objective function can now be entered in the Optimizer block's Objectives tab:

- Delete the default equation from the Equation pane.
- Define a new variable for the number of deliveries by entering it in the Equation pane

```
Integer numDeliveries;
```

☞ Don't forget the semicolons that are needed to end each statement!

- Below that variable definition, enter the equation that converts deliveryTimes into numDeliveries:

```
numDeliveries = int((endTime-1)/deliveryTimes + 1);
```

- Below the conversion equation, enter the cost equation:

```
MaxProfit = 2.50*numSold - numDeliveries*(1000.0+delTankSize*1.00)
  - emptyTime*100;
```

☞ The variable definition and the two equations are the objective function for the Optimizer.

The Optimizer block's equation pane should looks like:

```
Integer numDeliveries;
numDeliveries = int ((endTime-1)/deliveryTimes + 1);
MaxProfit = 2.50*numSold - #deliveries*(1000.0+delTankSize*1.00) - emptyTime*100;
```

Run the optimization

- Open the Optimizer block's dialog.
- Select the Run Parameters tab. Since this model has random elements, in the *Random model* section click the **Quicker Defaults** button. This quickly sets up all the parameters for a stochastic (random) model needing multiple samples, but limits the number of samples by default so you can get results more quickly.
- Run the optimization by clicking **New Run** in the Optimizer's dialog, clicking the Run Optimization or Scenarios tool on the toolbar, or by giving the command Run > Run Optimization or Scenarios.

☞ For comparison purposes to your model, see the model Optimize 2 located at Documents/ExtendSim/Examples/How To/Optimization

Results

While the optimization run is progressing, notice how the MaxProfit and convergence values are increasing on the graph.

☞ The graph in the Optimizer's dialog has many of the same features and customization opportunities as the blocks in the Chart library discussed starting on page 174.

At the end of the run, you should get a message stating the convergence value of approximately 95%. When the run is finished, the Optimizer opens and displays the Results tab.

BEST	numSold	deliveryTimes	delTankSize	emptyTime	MaxProfit	Samples	+/- Error
0		243	4992		12336.2688	5	252
1		242	4986		12235.2039	5	186.5
2		244	4968		12188.1189	5	337.9
3		243	4968		12075.3053	5	383.3
4		242	4986		12011.4884	5	369.6
5		250	5029		11995.8905	5	218
6		243	4992		11976.4121	5	427.6
7		244	4968		11922.9817	5	306.1
8		242	4968		11904.9338	5	460.3
9		244	4983		11566.1356	5	406.6

The best values for the selected parameters (delivery times and tank size) will have already been placed into the model, the drink tank is never empty, and the profit will be maximized. The results of the run as seen on the Objectives and Results tabs are:

- NumSold (total sales) is blank because it has not been changed; it is an output from the model.
- DeliveryTimes (Filler Truck repeat prm) = around 240 minutes between deliveries (2 deliveries).

- DelTankSize (Filler Truck data_tbl) = around 5200 drinks per tank delivered.
- EmptyTime (Empty Time contents) is blank because it is an output value from the model.
- MaxProfit (on the Results tab) = around $12,000.

Your results will vary a little from those shown here because of the randomness of the model and because you have set the convergence and samples for a *quick* result.

☞ As mentioned earlier, it is a good idea to run the model with optimization additional times to make sure that the answers agree or are at least close. This provides assurance that the first answer is not false or sub-optimal.

About constraints

The optimization results indicate that the profit would be maximized if you ordered tanks that hold approximately 4900 drinks and if the second tank were delivered about mid-day. But what would happen if the drink distributor only had specific size tanks and delivery times? By constraining a parameter, values that fall outside the constrained boundaries are not considered as part of the possible solution space.

You can constrain parameter values in almost an infinite number of ways by entering constraint equations in the Optimizer's Constraints tab. There are two types of constraints:

- *Individual constraints* are applied only to the specific variable and cause it to be changed in some manner. For example, an individual constraint could cause all tank sizes to be in increments of 1000. Individual constraints are expressed as *equationVariable = (calculate constrained value)*.
- *Global constraints* can cause an entire set of parameters to be rejected so that the Optimizer will try a different set. For instance, a global constraint could reject a solution set if the sum of all the variables was greater than a certain value. Global constraints are expressed as *if (conditions) Reject = True*.

These are discussed more in "Constraints" on page 171. For this model the constraints to consider are:

- Delivery tanks only come in sizes that hold 1000, 2000, 3000, 4000, 6000, or 8000 drinks.
- Tanks can be exchanged every 30 minutes except...
- Tanks with 6000 drinks or greater can't be exchanged more often than every 60 minutes.

Calculating individual constraints

There are two individual constraints: the size of the delivery tank and how often it can be exchanged if it holds fewer than 6000 drinks.

The constrained tank size

The individual constraint equation to granularize the drink tank sizes recalculates delTankSize as follows:

```
// round to delivery amounts (e.g. 1k, 2k, 3k, 4k, 6k, 8k)
if (delTankSize<=4000) delTankSize = int(delTankSize/1000.0 +
  0.5)*1000.0;
  else delTankSize = int(delTankSize/2000.0 + 0.5)*2000.0;
```

☞ Comments are preceded by // or ** and are omitted from the calculation.

This equation puts the tank sizes of 4000 or less into multiples of 1000, and tanks sizes greater than 4000 into multiples of 2000, using an individual constraint. For example, if the Optimizer suggests a size of 3300, the equation would convert it to a potential tank size of 3000. If the suggested size were 5200, the ELSE part of the IF statement would round it up to 6000 since 5200 is closer to 6000 than to 4000.

The constrained delivery time

The individual constraint equation to granularize the delivery times is:

```
// round fillup time to listed delivery times (e.g. 30 minutes)
DeliveryTimes = int(DeliveryTimes/30.0 + 0.5)*30.0;
```

The delivery time is converted to multiples of 30 minute intervals using an individual constraint, similar to how the delivery tank size was converted, above.

Calculating global constraints for 6000 and 8000 drink tanks

Although the preceding equation constrains the delivery times for most cases, you also need to reject cases where the drink tank is 6000 drinks or more and the delivery time is less than 60 minutes.

Rather than the individual constraints used in the previous two equations, for this equation you need a global constraint. Delivery times for tanks of 6000 and 8000 drinks are not limited to multiples of a specific number as was true for the tank size; they can be any value as long as it is greater than or equal to 60. So the solution space is somewhat unlimited. In addition, it would not be valid to just round up a delivery time that is less than 60. If the equation rounded up all the potential delivery times below 60 to exactly 60, as the preceding delivery time equation does, it would cause a severe bias toward getting deliveries every 60 minutes. Instead the equation needs to reject the entire solution set if the delivery time is below 60. This will cause the Optimizer to use a new random delivery time to generate a different set of solutions that will be less biased.

☞ If you use too many global constraints, or constraints that are too restrictive, the equation would unnecessarily reject almost all cases and the Optimizer could take too long to run or might fail to reach an acceptable solution.

The global constrained equation for deliveries of 6000 or more drinks is:

```
// can't deliver 6000 or more drinks sooner than 60 minutes apart
if (DelTankSize >= 6000 && DeliveryTimes < 60)
  Reject = TRUE;
```

The Optimizer pre-defines “Reject” as a special variable to be used only with global constraints. The Reject variable, if set to TRUE, will reject that case and cause the block to calculate another possible case that could be acceptable. If Reject is not set to TRUE, the current case will be used for the next series of runs.

Enter the constraint equations

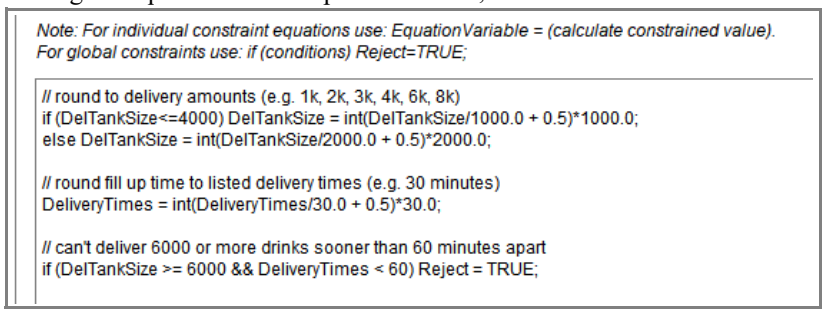
In summary, the equations to enter on the **Constraints** tab of the Optimizer block are:

```
// round to delivery amounts (e.g. 1k, 2k, 3k, 4k, 6k, 8k)
if (delTankSize<=4000) delTankSize = int(delTankSize/1000.0 +
  0.5)*1000.0;
  else delTankSize = int(delTankSize/2000.0 + 0.5)*2000.0;
```

```
// round fill-up time to listed delivery times (i.e. 30 minutes)
deliveryTimes = int(deliveryTimes/30.0 + 0.5)*30.0;

// can't deliver 6000 or more drinks sooner than 60 minutes apart
if (delTankSize >= 6000 && deliveryTimes < 60) Reject = TRUE;
```

After entering the equations in the Optimizer block, the Constraints tab should look like:



Running the optimization

Click the New Run button in the Optimizer's Run Parameters tab or give the command Run > Run Optimization or Scenarios. It is a good idea to run the optimization with the new constraints multiple times to make sure that the first answer is not a false or sub-optimal.

Because there are random elements in the model, the results will differ from, but should be close to, the following:

- numSold (total sales) is blank because it has not been changed; it is an output value from the model.
- deliveryTimes (Filler Truck repeat time) = 270 minutes between deliveries (2 deliveries a day).
- delTankSize (Filler Truck data_tbl) = 6000 drinks per tank delivered.
- emptyTime (empty time) is blank because it is an output value from the model.
- MaxProfit = around $10,500.

Note that the profit may have decreased slightly from the previous optimum results. This is due to the constraints put on the model parameters.

☞ For comparison purposes, the model with constraints titled Optimize 3 is located in the Documents/ExtendSim/Examples/How To/Optimization folder.

Using the Optimizer block

This section examines, in detail, how to get the most out of the Optimizer block.

Variables table

When a dialog's variable is added to an Optimizer block, the Optimizer places that variable into its Variables table. This table holds the variables needed in the Optimizer's cost or profit equation. These variables can be of two types:

- Decision variables to be optimized. These need their lower and upper limits entered.
- Output variables from the model. These should have no limits entered.

Both lower and upper limits for a decision variable need to be entered directly in the Variables table. Variables that are outputs from the model, such as *Exited* from an Exit block (Item library), should not have any limits entered or the Optimizer will attempt to incorrectly change them.

☞ If the variable is an integer type, such as a number of machines, do not enter a decimal point in the limits. Conversely, variables that need to have real values, such as a delay time that could vary from 1.0 to 2.2, need both limits entered with decimal points.

Specifying data table cells in the Variables table

If a data table is added as a variable to the Optimizer block, the *Row,Col* indexes need to be entered, separated by a comma. Since data tables are zero-based and start at row 0 and column 0, the first cell would start at 0,0.

For example, to use the first row and second column cell of a data table, enter the Row,Col value **0,1** where 0 is the first row and 1 is the second column.

☞ The Variables table's Row,Col column is only used to access a cell within a data table.

Objective functions

Optimizations can maximize profit, minimize a cost, or approach a constant, depending on the form of the equation. Just like the equation field of the Equation block (Value library), optimization equations can be multi-lined, define variables, have control constructs (i.e. IF-ELSE, FOR...), and call ModL functions. To see more information on equations, see "Equation-based blocks" on page 189.

Maximizing profit or minimizing cost

For example, you may want to optimize the number of machines used to maximize the profit. In that case, you can set up an equation:

```
MaxProfit = thruput*dollarsPerUnit - numMachines*dollarsPerMa-
  chine;
```

Or, you might want to write the equation as a cost, and minimize that cost:

```
MinCost = numMachines*dollarsPerMachine - thruput*dollarsPerUnit;
```

The variable names used in the MaxProfit or MinCost equations come from the **Equation Variable** column of the Optimizer's Variables table. Depending on how you write the equation, the Optimizer will either maximize or minimize the equation value by changing the values of the decision variables until the correct condition converges.

Approaching a constant

A special case for an objective function would be that the result needs to approach a constant K as close as possible. In this case, you can write a profit equation.

For the case where the equation should approach a nonzero constant K, the form would be:

```
// maximizes when equation equals constant
  MaxProfit = 1.0 - RealAbs(1.0 - equation/K);
Note that the RealAbs(x) function returns the positive absolute
  value of x.
```

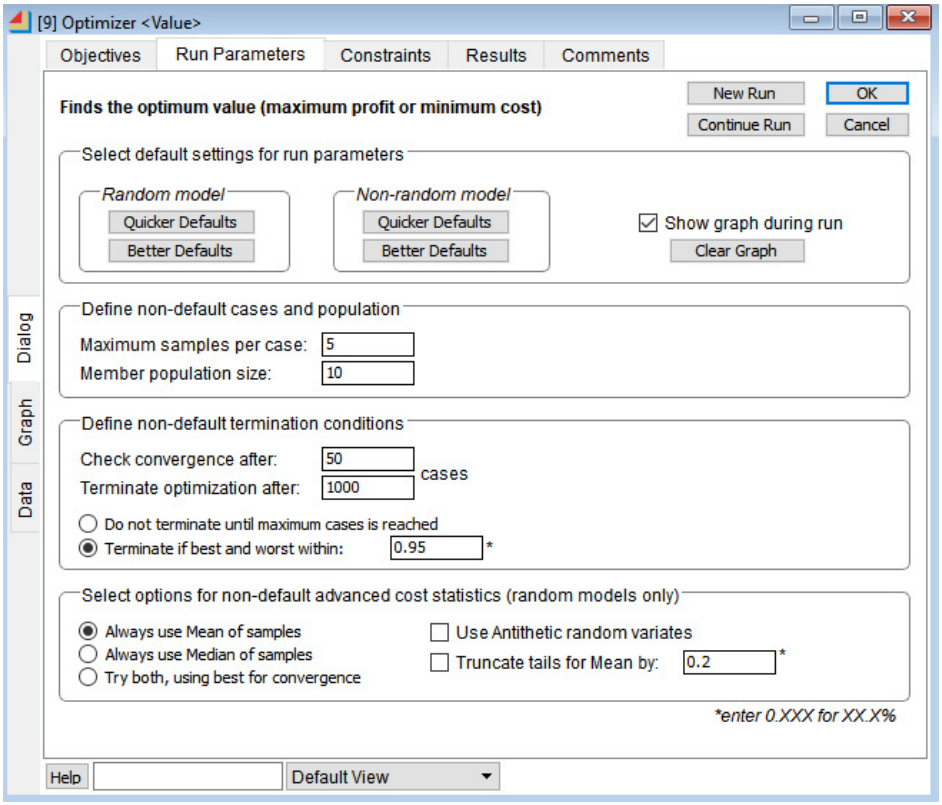
For the case where the equation should approach zero, the form would be:

```
// this becomes maximum when equation equals zero
  MaxProfit = ConstantValue - RealAbs(equation);
```

In this case, ConstantValue should be small, but large enough so that most of the population MaxProfit results are *initially* positive. If ConstantValue is too small, the convergence calculation will fail because there will be both negative and positive values in the final population results. If ConstantValue is correct, all of the values will tend to become positive as the system converges, and the convergence calculation will then be valid. If ConstantValue is too large, the convergence calculation will tend to be insensitive and high all of the time, causing a premature end of the optimization run.

These techniques can be adapted to solve any "approaching a constant" type of problem.

Run Parameters tab



In most cases, clicking either of the *default* buttons for the type of model you are optimizing (random or non-random) will quickly set all of the parameters in this tab to useful values.

Two of the parameters are especially important to convergence of the optimization:

- *Maximum samples per case:* The maximum number of runs averaged to get a member result. For non-random models, this should be 1. For random models, this needs to be high enough to get a useful mean value at the expense of run time. The Optimizer block starts the number

of samples at 1 and increases them with each generation until it reaches the maximum. Sometimes it is useful to reduce the maximum number of samples (possibly to 5), to get a rough idea of the best solution without spending too much time running samples. Most of the time, a useful result will occur and, even if it is not the optimum one, it will be close.

- *Terminate if best and worst within (percent):* This value is used to examine the current population of results to see if they are within a specified percentage of each other. The default of 0.99 might not be high enough for a precise answer in a noisy model. Increasing this value (i.e. 0.9999) will cause the optimization to continue until the population converges more closely, increasing the likelihood of a more optimum answer at the expense of run time.

Constraints

When building a model, there are almost always some parameter constraints that have to be satisfied. The two types of constraints are *individual constraints* and *global constraints*. The Optimizer block makes it easy to apply virtually any kind of constraint to a model's parameters.

Individual constraints

Individual constraints are used to change a decision variable's value if the value has to be limited or if it depends on the values of other decision variables. These constraints are entered as equations, usually with IF or IF-ELSE statements. For example:

```
if (NumQueueSlots > 7)
  NumSlots = NumActivities+3; // change it
```

Sometimes you might need the IF-ELSE form:

```
if (Var2 >= 3)
  Var2 = Var3-Var4;
else// var2 was less than 3
  Var2 = Var5/2;
```

In some cases you just need to modify the value of a variable. For example, if you need to constrain its values to multiples of 0.5 (i.e. 1.0, 1.5, 2.0). You do this by multiplying the variable by 2, adding 0.5 before the Int() function truncates it so that it rounds it to the nearest integer, and then dividing by 2.0, forcing the result to floating point values that are granular to 0.5:

```
Var2 = Int(var2*2.0+0.5)/2.0; // 0.5 Granularity
```

In any case the newly calculated Var2 value will replace the old Var2 value.

Global constraints

Global constraints are useful to reject an entire case if any or all of the decision variables don't meet a specific criteria. Global constraints are entered as equations, usually with IF statements, to assign the value TRUE to the variable REJECT if the variables are not within the constraint. They are entered like this example:

```
if (Var4+Var5 > 7)
  Reject = TRUE; // only reject if the sum is too large
```

☞ Reject is a special optimization variable for use with global constraints. If set to TRUE, it will reject that case and cause the block to calculate another possible case that could be acceptable. If Reject is not set to TRUE, the current case will be used for the next series of runs.

Sometimes you might need a more complex form:

```
if (Var4+Var5 > 7 || Var4 < 2) // the || means OR, && means AND
  Reject = TRUE;
```

In any case, any global constraint will abort that particular case and the Optimizer block will keep attempting to create cases until the global constraint doesn't set REJECT to TRUE. It will try to create 500 new cases before it gives up and prompts the user with an error message. If this occurs, the global constraint is probably faulty.

Interpreting results

The Results tab shows the entire population of solutions, sorted with the best one on top (row 0). As the optimization progresses, new and better solutions will replace inferior solutions in the population table.

If the optimization terminates for any reason, either via normal convergence or running for the maximum number of generations, the best solution set found so far is automatically placed in the model.

☞ For comparison purposes, see the Optimize 2 and Optimize 3 models located in the Documents/ExtendSim/Examples/How To/Optimization folder. For additional examples, see the Inventory Lab model (Examples/Discrete Event) and the Call Center Optimizer model (Examples/How To/Optimization).

Stat::Fit (Windows only)

In simulation models, it is often useful to characterize a random input (for example, interarrival times and demand rates) using a probability distribution. Typically, this involves obtaining historical data that documents the system's behavior, then analyzing the data to determine an appropriate distribution to represent it. There are two advantages to using statistical distributions rather than raw historical data as inputs to a model:

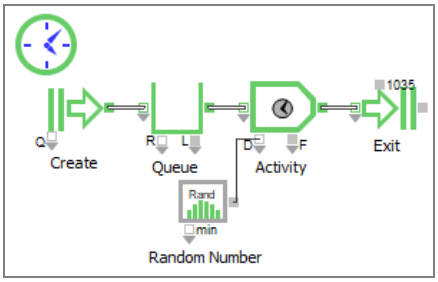
- Values for input random variables are not limited to what has happened historically.
- For continuous distributions, an infinite pool of data exists. With historical data, there is seldom enough collected data to support multiple simulation runs.

Stat::Fit is a software package from Geer Mountain Software (www.geerms.com) that helps the analyst determine which distributions, if any, offer a good fit for the underlying data. ExtendSim has an interface so you can easily access the power of Stat::Fit.

☞ Stat::Fit is a Windows application included with the ExtendSim DE and ExtendSim Pro products. It can be purchased separately for use with ExtendSim CP.

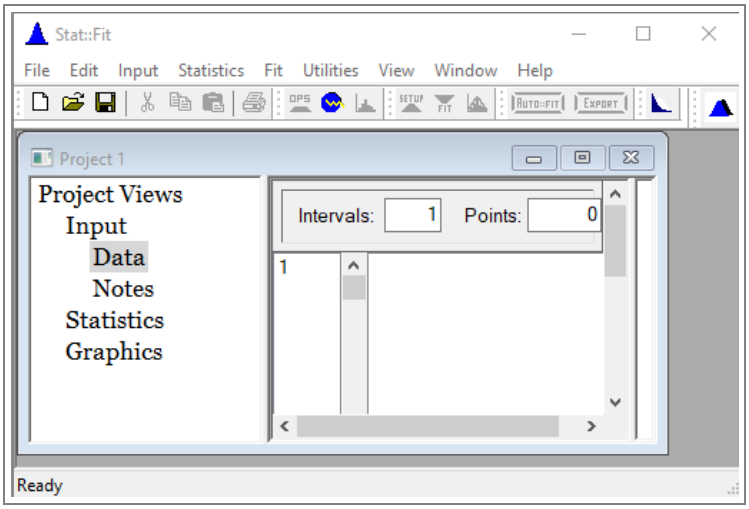
Tutorial

The discrete event model shown on the right has a Random Number block (Value library) set to the Uniform Real distribution with a Minimum of 0 and a Maximum of 1.

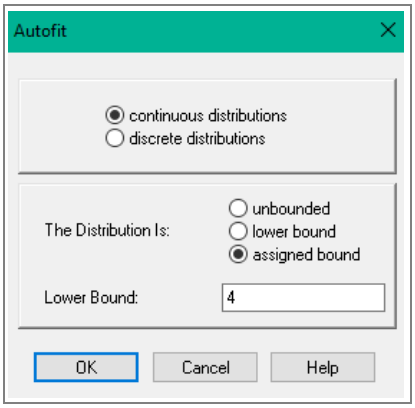


This tutorial uses a Stat::Fit project file—a pre-built text file with 32 historical data points—to define which distribution and associated parameters might be more appropriate than what is set in the Random Number block.

- Open the *StatFit Example* model located in the folder Documents/ExtendSim/Examples/How To.
- In the *Distribution Fitting* tab of the Random Number block:
  - Select **Stat::Fit** as the distribution fitting software
  - If it isn't already there, enter the path (statfit/statfit3) in the block's path field
  - Click **Open Stat::Fit**



- The Stat::Fit application launches and opens a new blank document.
- In Stat::Fit, select File > **Open** to open the project file.
- Find and select the file **StatFtEx.sfpx**. It should be in the same folder (ExtendSim/Examples/How To) as the StatFit Example model.
- In Stat::Fit, choose the menu command Fit > Autofit or click the Autofit button in the toolbar. This opens the dialog shown here.



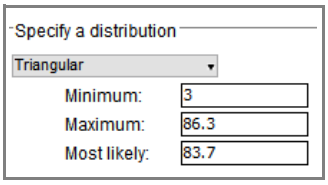
- In the Autofit window, click **OK** to use the default settings.

After Stat::Fit does some computation, a window appears displaying a list of parameterized distributions that have been ranked according to goodness of fit.

- In Stat::Fit, choose File > Export > Export Fit or click the Export button in the Stat::Fit toolbar.
- In the Export Fit window:
  - Select "ExtendSim" as the Application (it should be the default).
  - Then choose the distribution you want based on the goodness of fit results. (The Triangular distribution, which had the highest ranking, is selected by default.)
  - For the Output, select Clipboard.
  - Click OK
- Return to ExtendSim and the Distribution Fitting tab of the Random Number block.
- Click the **Import Distribution from Clipboard (Stat::Fit)** button.

- You may exit Stat::Fit if you wish

If you selected the highest ranking distribution, the Distributions tab in the Random Number should reflect the Triangular distribution and parameters Stat::Fit selected (Triangular,3,86.3,83.7).

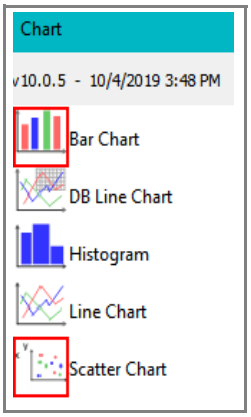


Additional Stat::Fit documentation is available in Stat::Fit's Help menu and in the *Statfit3* documentation located within the ExtendSim/Documentation folder.

Chart library

⚠ As of ExtendSim 10 the Plotter library is in legacy status and has been replaced by the Chart library, discussed here. The Plotter library is only supplied with the current release so that you can open older models and convert them to using the new Chart library. **Do not use the Plotter library to create new models as it will be discontinued in a future release**.

While the Chart library blocks also store and display data, their main purpose is to provide a graphical representation of simulation data. The Quick Start Guides show some basics of how to use the Chart blocks. This section discusses the blocks in more detail and describes how to customize their functioning and appearance.



Using charts

All the stand-alone ExtendSim charts and graphs are located in the Chart library. Certain other blocks, such as the Optimizer, use ModL functions to show charts or graphs in their dialogs. Likewise you can create your own charts, graphs, and plotters or add those features to your custom blocks, using ModL functions. However, as shown below, chart appearance and function can easily be customized without programming.

You can have multiple and different types of charts in a model, and you can place them at any location in the model. Chart blocks can have up to 20 traces on their graphs.

By default chart blocks open automatically after the simulation is run. If a chart block is set to not open (as discussed on page 176), you can open it by double-clicking its icon on the model worksheet. For constant visibility, the graph portion of a Chart block can be cloned onto the model worksheet or notebook.

The blocks in the Chart library adapt to the type of model they are placed in and change their options accordingly. For example, if the model contains an Executive block from the Item library, the chart block will accommodate discrete event or discrete rate messaging. Also note that some of the chart features discussed here will be absent or slightly different when used in continuous models.

☞ Once a chart block is in a model, it adapts to the type of model. Thus if you remove the Executive block from a model that has one or more chart blocks, you must replace all the chart blocks with new chart blocks. If you don't, you will get an error message when the simulation is run.

Types of charts

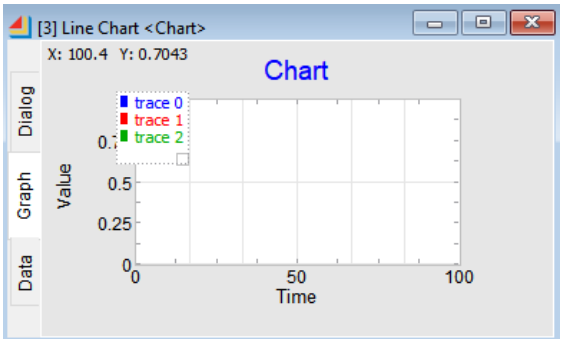
As indicated in the following table, each Chart block has a distinct purpose.

☞ These blocks can be used in any type of model—continuous, discrete event, etc. If a model has an Executive block, which marks it as being discrete event or discrete rate, the chart block will automatically adapt to that environment. Otherwise, the chart block assumes the model is continuous.

Chart	Description
Bar Chart 	Displays comparisons between Categories of data for one or more Series. Data is categorized into discrete groups (categories) such as months in the year, age group, types of machines, etc. Can have multiple Series and be used as a grouped bar chart; for example, rainfall data from 4 locations (4 series) for each month of the year (12 categories). The bars for a grouped bar chart can be stacked or side by side.
DB Line Chart 	Displays ExtendSim database data as a graph and reports that data in a table. Either traces the history of values received in a database cell over time or traces the content of database tables. Can be used for multiple runs. See the ExtendSim Database Tutorial and Reference for how to use this block.
Histogram 	Displays the frequency distribution of data. Groups data into bins, where bins are successive numeric intervals of equal size.
Line Chart 	Charts values over time. Gives graphs and tables of data for up to 20 time-associated inputs. Can be used for multiple runs and also as a worm chart.
Scatter Chart 	Shows two sets of data plotted as X,Y value pairs on up to 20 traces. You must connect both the X and Y inputs of at least one pair in order to plot data. Can also be used for multiple runs and also as a worm chart.

Chart structure

Whether they open automatically during or after a run, or are opened manually by double-clicking their icons, chart blocks open to the Graph tab as shown here.



Most of the ExtendSim chart blocks have these features in common:

- Three tabs on the left side as seen here and described in the table below.
- Legend and Cursor Values boxes that appear by default on the graph, as seen here.
- Trace Editor and Graph Properties dialogs that appear if the graph is right-clicked or if the Legend or Cursor Values boxes are double-clicked. These dialogs are used for customizing the appearance and format of the traces and graph.

- Underlying structure with a user-interface dialog and ModL code that determines how the block appears and behaves.

The three tabs on the left side of chart blocks

Left-Side tabs	Description	See Page
Dialog	Determines how the block works. At the top of this tab are three additional tabs: 1) The Data Collection tab controls which points will be plotted and if/when the data will be recorded. 2) The Display tab determines if/when graph the appears during the simulation run and if/when it will autoscale. 3) The Comments tab is for making notations. (The Histogram has two additional tabs: Bins and Results)	 177 176
Graph	Displays the data visually and provides access to dialogs that control how the traces and chart appear.	
	Legend and Cursor Values boxes are displayed on the graph by default.	182
	Trace Editor is opened by right-clicking on the graph or double-clicking the Legend.	182
	Graph Properties is opened by right-clicking on the graph or double-clicking the Cursor Values.	183
Data	Reports the data in a table format.	183

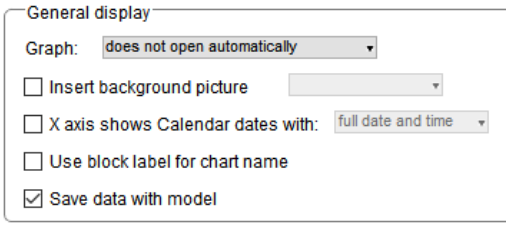
☞ Click the chart block's Help button for specific information about each option in its tabs. The Help button is active whenever the Dialog tab is the active window.

Display tab (top of the Dialog tab)

The Display tab is one of the three tabs that appear at the top of the window when the Dialog tab is the active window.

General display section

The *General display* section of the Display tab determines if and when the graph automatically opens:



- Never (the default)
- At the beginning of the simulation
- At the end of the simulation

Depending on the block, there could be additional options in this section, as seen here for the Line Chart block.

Background picture option

For example, to insert a background picture:

1) Put an @ sign at the beginning of the name of the file
2) Place the file in the ExtendSim/Extensions/Pictures folder
3) Restart ExtendSim
4) In the Display tab's General display section, check the *Insert background picture* checkbox
5) Select the file from the popup menu

Stacked option (Bar Chart and Histogram)
If multiple Series are specified in its Data Collection tab, the Bar Chart becomes a grouped Bar Chart. In this situation, the bars for each Series can be displayed either side by side (the default) or stacked on top of each other. See "Configure the bar chart options (Bar Chart and Histogram)" on page 179 for more information about Series.

Scaling section

The *Scaling* section of the Display tab determines if and when the graph will be auto-scaled:

- Never
- During the simulation
- At the end of the simulation (the default).

There are also additional options for defining what happens during autoscaling and for setting fixed minimum and maximum axis limits rather than the limits deduced from the simulation.

Data Collection tab (top of the Dialog tab)

The Data Collection tab is one of the tabs that appear at the top of the Dialog tab when it is the active window. As discussed below, the Data Collection tab controls which points will be plotted on each trace (or how bars will appear) and provides options for how much data is recorded and when.

☞ The options shown below will be displayed in most chart blocks in discrete event and discrete rate models; there are fewer options in continuous models. Also, the options differ depending on the chart block.

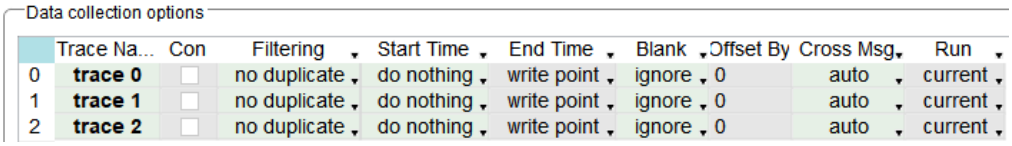
Data Collection Tab->	Data Collection Options Frame	Recording Options Frame	Other
Bar Chart	page 177		"Configure the bar chart options (Bar Chart and Histogram)" on page 179
DB Line Chart	page 181	page 179	"Data collection type (DB Line Chart)" on page 180
Histogram	page 177	page 179	
Line Chart	page 177	page 179	
Scatter Chart	page 177	page 179	

Data collection options (Bar Chart, Histogram, Line Chart, Scatter Chart)

☞ The following information applies to the Bar Chart, Histogram, Line Chart, and Scatter Chart blocks. See page 180 for the corresponding information for the DB Line Chart block.

Each block has options for setting how the block will collect information from its inputs.

- The Line Chart and Scatter Chart have a table with one line per trace.
- The Bar Chart and Histogram offer different options for setting data collection and specifying how the data is organized between bars.



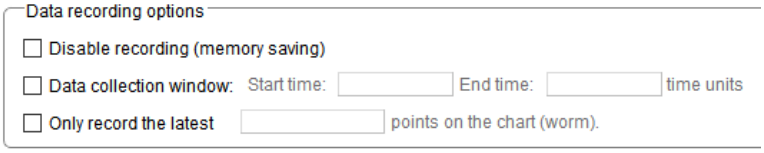
This frame contains a table with multiple columns. Each row in the table represents one trace on the graph; the columns are customizable properties. Since they aren't needed in a continuous model, most of these options only appear in discrete event and discrete rate models.

- *Trace Name*. Name each trace in this table or in the Trace Editor discussed on page 182.
- *Series*. Name each series in this table or in the Series Editor that appears if you right-click on the Graph. Applies to the Bar Chart and the Histogram.
- *Con* checkbox (Histogram, Line Chart, and Bar Chart) and *X* and *Y* checkboxes (Scatter Chart). These checkboxes indicate if there is a connection to the input connector represented by the column and row.
- *Filtering*. Fine tunes which points are written on the graph. An additional Duration column appears if the Scheduled filtering option is selected.
- *Start Time* and *End Time*. Controls whether or not a point is graphed at the beginning or end, respectively, of the simulation run. Use this for discrete event models, where it is more accurate for the graph to stop drawing the trace at the last event rather than at the End Time of the model run.
- *Blank*. If a Blank value is received, the block can ignore it or interrupt the line. Applies to the Line Chart and Scatter Chart.
- *Offset By*. Offsets the display of values on the Y or Y2 axis (Line Chart) or the X, Y, or Y2 axis (Scatter Chart) so that traces don't get written on top of each other.
- *Cross Msg* (Histogram, Line Chart, and Bar Chart). Auto is the default option. If you don't see points you think should be appearing on the graph, consider choosing one of the other options. Those options give more control over messaging in case a block does not actively notify the chart that the value it is sending has changed. This setting is only enabled if filtering is not scheduled and the block is used in a discrete event or discrete rate model.
  - *Auto* means that the connector is dependent by default but becomes independent if it gets a message.
  - *Dependent* causes a point to be written if the value at the corresponding input has changed and that connector doesn't get a message but one of the other connectors does get a message.
  - *Independent* means that a point is only written if the corresponding input connector gets a direct message.

- *X Msg* (Scatter Chart). Scatter Charts need values for both the X axis and the Y axis. When the input for the X axis gets a message, the block has three options for how to get a value from the opposite (Y) axis:
  - *Pull from the opposite connector*, which causes the Y input to actively refresh then report its value to the block.
  - *Read from the opposite connector*, which causes the Y input to passively report to the block whichever value it currently has.
  - *Ignore message*
- *Y Msg* (Scatter Chart). Reverse the explanation for the X Msg, above, so it is about the input for the Y axis getting a message and what the option is for the opposite (X) connector.
- *Run* (Bar Chart, Histogram, Line Chart, and Scatter Chart). Use this when performing multiple simulation runs. The Run options are "Current" or "only run#". By default, "Current" is selected and each simulation run will overwrite the traces. If "only run#" is selected and traces are specified in the Run# column that appears, each run in a multiple run will be written to a different trace. This is equivalent to having a multi-sim plotter. To see how this is used, go to "Output the results to a Line Chart block" on page 140.

☞ The Scatter Chart and Histogram also have an "all runs" option that accumulates data over all the runs.

Data recording options (Histogram, Line Chart, Scatter Chart, DB Line Chart)

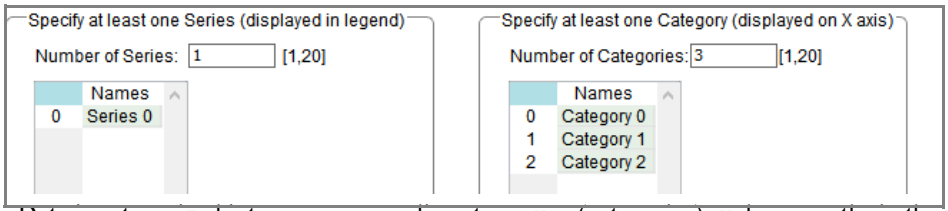


For the Line Chart, Scatter Chart, and DB Line Chart blocks, the bottom frame in the Data Collection tab allows you to either:

- Completely turn off data recording (saves recalculating time and memory)
- Only collect data during a specific window of time
- Record only the latest specified number of points ("worm chart"). This applies only to the Scatter Chart, Line Chart, and DB Line Chart.

Configure the bar chart options (Bar Chart and Histogram)

At the top of the Bar Chart's Data Collection tab is a frame for configuring the chart's categories and series.



- Data is categorized into one or more discrete groups (categories) such as months in the year, age group, types of machines, etc. The categories being compared are listed along the

How To

graph's horizontal axis; the height of a bar is proportional to the bars' represented values. By default, there is one series (and therefore one bar) for each category.

- Specifying two or more series allows the Bar Chart to be used as a grouped bar chart. In this case each categorical group would have two or more bars, each representing a series. Each series can have its own color and is listed in the legend. For example, a grouped bar chart could have rainfall data from 4 locations (4 series) for each month of the year (12 categories). By default, the bars are side by side; an option in the Display tab allows them to be stacked.
- The Histogram is close to the Bar Chart except the categories are called Bins, which are set in the Bins tab.

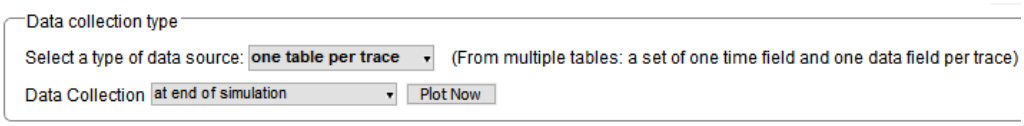
⚠ The Bar Chart can have up to 250 bars, 20 Categories, and 20 Series. However, those numbers are constrained by the fact that the number of bars equals the number of Categories multiplied by the number of Series. So if you have 20 Categories, you are limited to 12 Series.

Data collection type (DB Line Chart)

The DB Line Chart block graphs and displays ExtendSim database information. Instead of reading values from input connectors, this block displays values from the database. Its Data Collection tab is very different from the one discussed above.

⚠ To use this block, the model must contain an ExtendSim database. To learn more about how this chart is used, see the document titled *ExtendSim Database Tutorial & Reference*.

The *Data collection type* frame at the top of the Data Collection tab is for deciding what should be shown on the graph and when data is collected.



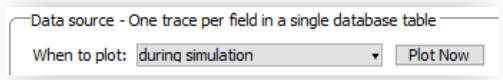
What should be shown on the graph

- *One cell per trace*. This is the easiest way to view the history of one or more database cells. The values for each selected cell are displayed on a separate trace in the block's Graph tab and reported in a separate column in the Data tab. The source cells can be from the same or different databases. The block reports and displays the values of the cells during the simulation run such that each point on the graph's trace represents the value of the cell at that specific time.
- *One field per trace*. For graphing data from different fields of the same database table. This choice gets the time information from one (and only one) of a table's fields and gets the data information from one or more other fields in that database table. All the fields must come from a single specified database table and that database table must have a time field. The block will report and display the value of each field's records at the associated times in the time field. Thus each record is a point on the graph and a value in the block's data table. The display can occur during or after a simulation run or whenever the graph is manually plotted.
- *One table per trace*. Used to chart data coming from fields in different database tables. For this option, each table must have its own time field as well as at least one data field and each trace will represent one time field and one data field from one database table. If all the fields

come from the same database table, it is similar to what happens if you choose “one field per trace”.

When the source data is collected
The options for when the data is collected are:

- For the “One cell per trace” mode, the data is always collected during the simulation run.
- For the “One field per trace” and “One table per trace” modes, the data collection options are:



  - During the simulation. Note: this option can slow the run because data will be constantly collected.
  - After the simulation.
  - Only when the Plot Now button is clicked. Use this option if you just want to view what is currently in the database. For example, when you don’t want to run a simulation or you have paused the simulation and want to view intermediate results.

Data collection options (DB Line Chart)

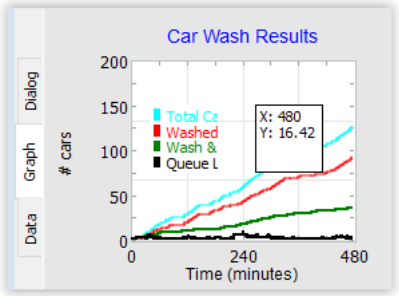
The *Data collection options per trace* frame in the middle of the DB Line Chart’s Data Collection tab is for selecting the location that holds the data to be graphed, one row per trace. Most of the option headers are self-evident; the rest are explained here.

☞ Depending on what data is collected, the following options may not all appear.

- *Open*. After the complete data source location (address of database, table, field, and record, as appropriate) has been entered, click the button in the Open column to open the Database Viewer. Until then, the button will have an x on it and won’t be able to open the Database Viewer.
- *Selection*. If present, use the Selection column to quickly specify the exact database address, rather than selecting each component of the address from the other columns.
- For the *Blank*, *Offset By*, and *Run* columns, see the data collection options for the Line Chart, on page 177.

Graph tab

The Graph tab is one of the three tabs that appear on the left side of a chart block; the graph visually displays data collected during the simulation run.



- The values that actually get displayed depend on settings in the Dialog tab’s Data Collection and Display tabs, discussed above.
- You can directly name the chart, change the beginning and ending axis numbers, and format the axis labels by selecting and changing the text on the Graph tab. To control more properties, right-click on the graph or double-click the Cursor Values box to open the Graph Properties dialog, discussed on page 183.

By default the *Legend* and *Cursor Values* boxes appear on the graph, as discussed below.

Legend

Each chart block has a key or Legend that appears by default on the Graph tab and shows the name and color of each trace.

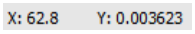
- The appearance of the Legend (its border, color, and whether it gets displayed or not) is controlled in the Graph Properties dialog, discussed onpage 183.
- The name, color, thickness, style, symbol, and Y1 or Y2 display of each trace shown in the Legend is controlled by the Trace Editor, discussed below.

To resize the Legend, select it and use the resizing square that appears on its lower right side. To reposition the Legend, select it and move it with the cursor.

☞ Double-clicking the Legend opens the Trace Editor.

Cursor Values

Each chart block has a Cursor Values box that appears by default on the Graph tab. It reports the X and Y coordinates of whichever point the cursor is at.



To resize the Cursor Values box, select it and use the resizing square that appears on its lower right side. To reposition the box, select it and move it with the cursor.

☞ Double-clicking the Cursor Values box opens the Graph Properties dialog.

Trace Editor

The Trace Editor allows you to set the properties for each trace; the selected name and color are displayed in the Legend that appears by default on the Graph tab.

To open the Trace Editor, right-click on the Graph tab or double-click the Legend on the Graph tab.

The *Trace Name* text at the left of the dialog describes the traces and is used to label the columns in the data pane.

The *Color* and *Thickness* choices affect how the trace looks. Clicking the color box opens the standard color window. To change the width of the line, repeatedly click in the Thickness box to select one of the five line widths.

The *Style* choice affects how the trace is drawn. Repeatedly click in the Style box to select one of the three styles:

- The top style shown here indicates that the data points are connected by diagonal lines (interpolated)
- The middle style indicates that points are connected by horizontal and vertical lines (stepped).
- The bottom style indicates that you want the points plotted with no lines between them.

The Trace Editor also has many *Symbol* types, such as dots, squares, and circles, as well as a trace numbering choice that displays a different number for each trace depending on the row. Repeatedly click in the Symbol box to select one of the symbols. If you choose a non-numeric symbol, it will be drawn at each point on the trace. Numbers will instead be spaced evenly along the trace. Each symbol is drawn using the trace color and width settings.

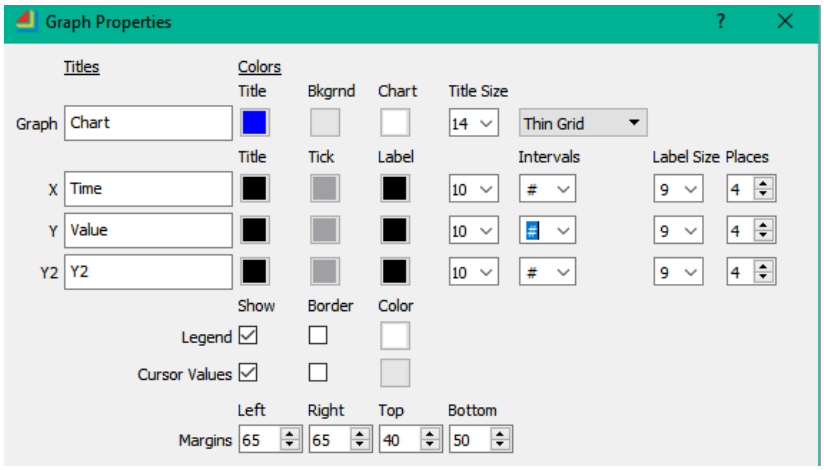
The *Y1/Y2* choice tells which axis to plot the numbers against. The default choice, Y1, plots values on the left axis. Click once on the Y1/Y2 box to change it to the axis on the right side.

The *Show/Hide* choice tells whether to display the trace at all. If you click on this choice, the eye is crossed out and the trace is not drawn. This is useful if you have many traces and you temporarily want to hide one or more of them or if one trace is on top of another and you want to make the chart clearer. (See also the Offset By option discussed in "Data collection options (Bar Chart, Histogram, Line Chart, Scatter Chart)" on page 177.

Graph Properties

To see the Graph Properties dialog, shown below, right-click on the Graph tab or double-click the Legend.

The Graph Properties dialog is for formatting the titles, colors, and other properties of the graph. (The Trace Editor is for formatting the traces that appear in the graph.)



As seen above, the Graph Properties dialog allows for a lot of customization. Most options are self-evident. The rest are explained below:

- For the Graph row, the *Bkgrnd* (background) choice colors the area around the actual charting area while the *Chart* choice colors the charting area.
- For the X, Y, and Y2 rows:
  - Ticks are the short lines that appear around the perimeter of the charting area.
  - Intervals indicate how many spaces are used to separate the numbers on the axis between the beginning and ending numbers—each interval represents one space between two numbers on the axis. If the default (#) is used, the number of intervals will change with the size of the window. Using any other option fixes the number of intervals regardless of window size.
  - Places determines how many significant digits are displayed on the corresponding axis.

Data tab

The Data tab is one of the three tabs that appear on the left side of a chart block.

If you change the data in the Data tab, those changes are reflected in the Graph tab. You can change numbers, paste in rows, and so on. Use this capability to view how various data would be graphed, or to plot a reference line.

The color of the column header text is the same as for its corresponding trace; those colors are set in the Trace Editor, described on page 182.

Model reporting

The blocks in the Report library track simulation results and provide statistical analysis of data.

☞ The ExtendSim database is also useful for storing results for output to external applications.

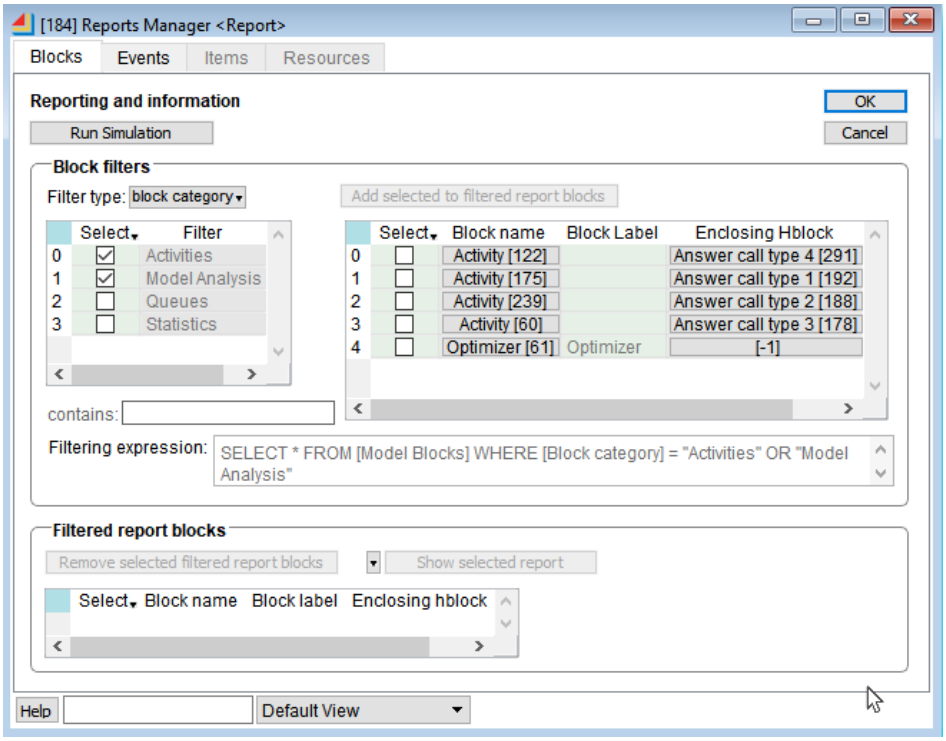
Report Library	Description
Cost Stats 	Place this block anywhere in the model and it will report the following statistics, with confidence intervals, for each block that has costing enabled: Block Number (or label, if a label is entered in the block), Block Name, Cost Per Item, Cost Per Time Unit, and Total Cost. If costing is enabled, statistics are reported for the following categories of blocks: Activities, Generators, Queues, and Resources.
DB Statistics 	Calculates the mean, variance, standard deviation, minimum, maximum, median, and mode for a field in an ExtendSim database table. Select a field and the results will be calculated automatically. Choose *calculate statistics* at the end of each run to only calculate the values when each run completes or at the end of all runs to calculate when all of the simulation runs have completed.
Item Log Manager 	Collects customized information about how item states change over time, then packages this information into a customizable database report. The block allows you to customize which blocks will contribute to the report, what information will be included in your log, and in what order the information will appear. For example, you may only want blocks with the label "Process 52" to collect data for attributes "SKU" and "waitTime". You can place as many different Item Log Manager (ILM) blocks in your model as you want. See also "Using the Item Log Manager to get item information" on page 399.
Reports Manager 	The primary interface for creating reports from simulation runs. See page 185 for more information.
Statistics 	Accumulates data on certain types of blocks and reports statistics in a single table using a specified statistical method. You can select which blocks or types of blocks will have their information collected: activities, queues, Mean & Variance, Resource Item, Resource Pool, Resource Manager, Workstation, Convey Flow, Queue Matching, Rate blocks, and tanks. To choose a selection of fields from supported blocks, select the "mixed blocks" option. Then either Shift-click or clone drop output fields as discussed on page 135. This causes the output information to be displayed in the table on the Statistics tab.

Reports Manager block

The Reports Manager block is the primary interface for creating reports from simulation runs for blocks, events, items, and resources. The dialog of the Reports Manager block provides filtering interfaces to specify which blocks, events, items, and resources to generate reports for.

- Block reports collect and centralize information from the *Results* tabs of user-specified blocks organized by block category.
- Event reports collect information for events that have properties matching user-specified filtering conditions.

At the end of the simulation run the information collected for these reports is stored in tables in an internal ExtendSim database. These tables can be accessed from the dialog of the Reports Manager block as well as from the database menu of the ExtendSim application.



Block filters

In the Blocks tab, blocks can be filtered by the following:

- Category that was defined when the block was created, such as Inputs or Math
- Name of the block, such as Activity or Lookup Table
- Label on the block, as entered in the block's dialog
- Enclosing hierarchical block

Selections can be made across more than one block filter type, such as for category and for label.

Event names

In the Events tab, you can select one or more of the following general categories of events:

- Creation
- Arrival
- Downstream blocking
- Resources unavailable
- Departure
- Exit
- Route
- Batch
- Unbatch
- Spawn
- Preempt
- Shutdown
- Item allocation

How To

Math and Statistical Distributions

Working with equations and distributions

"I know that two and two make four - & should be glad to prove it too if I could –
though I must say if by any sort of process I could convert 2 and 2 into 5
it would give me much greater pleasure."
— George Gordon Noel Byron

Overview

ExtendSim provides an extensive palette of tools for integrating mathematical equations and randomness in a model. You can even control the exact moment that calculations will take place. This chapter discusses equation-based blocks, randomness, and probability distributions.

☞ See also the How To chapter titled “Analysis” on page 133.

Blocks that represent functions

ExtendSim libraries are toolkits of blocks for quickly building a graphical representation of model logic. As shown below, some blocks have specific mathematical functionality and perform calculations automatically based on settings in their dialogs. Other blocks provide even more flexibility and perform calculations based on equations you enter in their dialogs; they are discussed on “Equation-based blocks” on page 189.

☞ For information about blocks that perform statistical analysis, see “Blocks that calculate statistics” on page 134.

The following blocks may be used in any type of model to provide mathematical functionality based on dialog selections:

Decision (Value library)

Compares the value at one input to the value at another input and reports a result. For example, use this block to determine if one model value is greater than, less than, or equal to another value during the simulation run.

Integrate (Value library)

Provides different integration methods to integrate the input value over time. You can also set an initial value in the dialog.

Math (Value library)

Calculates a mathematical, financial, logical or trigonometry function depending on the option selected in its dialog. Set the block to add a number to its input value and output the result. Or have it calculate the exponent of the input value. Provides over 30 functions from a popup list.

Mean & Variance (Value library)

Calculates the mean, variance, and standard deviation of the input. You can set an initial value in the dialog and select options to calculate a moving average, use a specified confidence interval, collect interval statistics for a specified time period, and use time weighted statistics.

Random Number (Value library)

Generates random numbers for the distribution selected in the dialog. Select from over 30 distributions or use an empirical table to create a custom distribution. This block is discussed more in "Random numbers" on page 197.

Data Fitter (Utilities library)

Uses matrix techniques to obtain a least mean square curve fit to a set of data. Enter or import the data into the dialog's data table and select a fitting function to solve for.

☞ In addition to the blocks listed above, most blocks automatically calculate and report statistical information during or at the end of a simulation run. For example, the Math block (Value library) reports the results of the selected mathematical calculation in its Options tab and the Queue block (Item library) displays statistics about the queue length, average wait time, utilization and so forth in its Results tab.

Other options

There may not be an ExtendSim block that provides the specific function or equation that you want. Or, you may want to combine the functionality of several blocks into one. Some possible solutions are:

- Select several blocks and make them into a hierarchical block, as discussed in "Hierarchy" on page 120.
- Add features to an ExtendSim block by modifying the structure of a block (its dialog and code) as discussed in the ExtendSim Technical Reference.
- Use the Equation block (Value library) or the Equation(I) or Queue Equation blocks (Item library) to directly combine functions or to obtain behavior not available in other blocks. These blocks are discussed in the next topic.

Equation-based blocks

The equation-based blocks calculate values for, and perform operations in, models based on formulas and ModL code entered in their dialogs. There are five mathematical equation-based blocks:

Block	Library	Purpose
Equation	Value	Computes user-defined equations and outputs the results
Equation(I)	Item	Computes user-defined equations when an item arrives, then outputs the results
Queue Equation	Item	Stores items and releases them based on the results of user-entered equations.
Query Equation	Value	A user-defined equation selects one record from an ExtendSim database table
Query Equation(I)	Item	A user-defined equation selects one record from an ExtendSim database table when an item arrives

☞ The Buttons block (Utilities library) is also equation-based but is used for more specific purposes. It is discussed on page 74.

Advantages

- Access to over 1,200 internal functions plus you can use operators to enter logical statements, write compound conditions, and specify loops.
- The equation can be as simple as performing a mathematical operation on the value from an input connector or it could be as complex as a full programming segment.
- You can use include files (See *Include Files* in the Technical Reference) within your equation to reuse your own functions among many equation blocks
- You can use the source code debugger (See "Debugging equations" on page 211) to check the equation if you have a problem. The equation is automatically compiled when you click OK in the block's dialog.

⚠ Equation-based blocks support most but not all of the programming features available when creating blocks. However, message handlers cannot be used. And in order to be used in equation blocks, user-defined functions and procedures and constants must be defined in an include file. See "Include files for equations" on page 196 for more information.

The Equation(I), Queue Equation, and Query Equation(I) blocks can only be used in non-continuous (discrete event and discrete rate) models. They typically perform calculations when items arrive or depart. Although they are continuous blocks, you can use the Equation or Query Equation block in non-continuous models. This is common when you want the equation to calculate independent of item status.

Overview

An equation-based block takes input variables, uses those values in an equation, and outputs the results of the calculation. Equation-based blocks are similar to the formula bar of a spreadsheet. Most of the usual components (operators, values, functions, and so on) are the same. There are two differences - instead of a cell reference, these blocks have input and output variables that are identified by name in the equation, and the results of the equation can be output to different destinations.

Equation components

The components of an equation are the input variables, the equation, and the output variables.

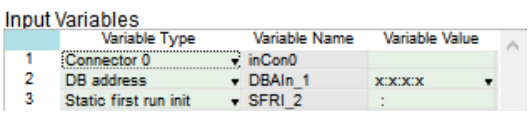
All of the equation-based blocks have a scrollable area for entering the equation. Most equation-based blocks also have a separate Equation Editor for viewing and editing the equation; it is discussed on page 195.

You can define local (temporary) input and output variables in the equation area or the Equation Editor as well as static (permanent) input and output variables using include files.

As described below, most equation-based blocks also have two tables, Input Variables and Output Variables, with predefined static variables you can use in equations.

Input variables

Input variables are the model values used in an equation. With the exception of the Buttons and Optimizer blocks, the equation-based blocks have a table for creating



the input variables. Each row in the Input Variables table (shown on right) has a popup menu for selecting a pre-defined type of input variable and a field for its name and value. Variables created here are static (permanent).

To modify the number of rows used in the variable tables:

- Change the number of rows in the table by clicking the green +/- resize button in the table's bottom right corner, selecting Resize, and entering the number of rows desired.
- Delete rows by first selecting the rows you wish to delete, clicking the green +/- resize button, and then selecting the option to "delete selected rows".
- Duplicate any rows by first selecting the row you wish to duplicate, clicking the green +/- resize button, selecting the option to "copy selected row", and choosing how many copies you want to create.

The Variable Type popup provides many options for input variables, as shown below. For more information about each option, see the blocks' Help.

☞ The Queue Equation, Query Equation, and Query Equation(I) blocks have additional specific input variable types not listed in this table. See "Query Equation and Query Equation(I) blocks" on page 246 for more information.

Input Variable Type	Equation	Equation(I)	Queue Equation	Query Equation	Query Equation(I)
AR Order ID – reads an item's advanced resource order ID. If an item has advanced resources attached, the value of this property is the record index into the Resource Orders table.		X	X		X
Connector – value input connector	X	X	X	X	X
DB read value – reads a value from a fixed location in a database table. Specify location in the Variable Value column.	X	X	X	X	X
DB read value using attrib – reads a value from a variable database location specified by a DB address attribute		X	X		X
DB read PRI – reads the Parent Record Index (PRI) from a cell in a Child field. Read location is fixed and is specified in the Variable Value column. See note below.	X	X	X	X	X
DB read PRI using attrib – reads the PRI from a variable database location specified by a DB address attribute. See note below.		X	X		X
DB address – a database address chosen in the Variable Value column	X	X	X	X	X
DB database index – a database index chosen in Variable Value column	X	X	X	X	X

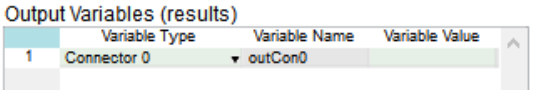
Input Variable Type	Equation	Equation(I)	Queue Equation	Query Equation	Query Equation(I)
DB table index – a database table chosen in the Variable Value column	X	X	X	X	X
DB field index – a database field chosen in the Variable Value column	X	X	X	X	X
DB record index – a database record chosen in Variable Value column	X	X	X	X	X
Static first run init – a static variable that gets initialized to its starting value only at the beginning of the first run of a multirun simulation. Specify starting value in the Variable Value column.	X	X	X	X	X
Static mult run init– a static variable that gets initialized to its starting value at the beginning of each run for a multirun simulation. Specify starting value in the Variable Value column.	X	X	X	X	X
Static open model init – a static variable that gets initialized to its starting value when the model is opened. Specify starting value in the Variable Value column.	X			X	
Next calc event time –Used to get this block's next event time. (Only enabled if the "Use 'calc event time' output variable" option is chosen.)	X			X	
Attribute – the value of the attribute named in Variable Name		X	X		X
Item quantity – the item's quantity value		X	X		X
Item priority – the item's priority value		X	X		X
Item index – the item's index value from the Executive's item array		X	X		X

For the Variable Name column, use the default names, assign names that have more relevance to the model, or select a name from a popup, depending on the type of variable selected. The type of input variable selected also determines the options for the Variable Value field: enter a value directly, select the value from a database location, and so forth.

☞ The notation PRI (Parent Record Index) is used in blocks that interface with the ExtendSim database. It describes the type of value that is being read or written when dealing with a Child field.

Output variables

Output variables are where the results are recorded when the equation is calculated. With the exception of the Buttons and Optimizer blocks, the equation-based blocks have a table for creating the



output variables. Each row in the Output Variables table (shown at right) has a popup menu for selecting a pre-defined type of output variable and a field for its name and value. Variables created here are static (permanent).

To modify the number of rows used in the variable tables:

- Change the number of rows in the table by clicking the green +/- resize button in the table's bottom right corner and entering the number of rows desired.
- Delete rows by first selecting the rows you wish to delete, clicking the green +/- resize button, and then selecting the option to "delete selected rows"
- Duplicate any rows by first selecting the row you wish to duplicate, clicking the green +/- resize button, and then selecting the option to "copy selected row."

The Variable Type popup provides many options for output variables, as shown below. For more information about each option, see the blocks' Help.

☞ The Queue Equation, Query Equation, and Query Equation(I) blocks have additional specific output variable types. See "Query Equation and Query Equation(I) blocks" on page 246 for more information.

Output Variable Type	Equation	Equation(I)	Queue Equation	Query Equation	Query Equation(I)
Connector – value output connector	X	X	X	X	X
DB write value – writes a value to a fixed location in a database table. Specify location in the Variable Value column.	X	X	X	X	X
DB write value using attrib – writes a value to a non-fixed database location specified by a DB address attribute		X	X		X
DB write PRI – writes a Parent Record Index (PRI) to a fixed location in a database table. Specify location in the Variable Value column. PRI is the index of the parent cell that is being written to the child.	X	X	X	X	X
DB write PRI using attrib – writes the PRI (see above) to a non-fixed database location specified by a DB address attribute. See note below.		X	X		X
Calc event time – Used to set this block's next event. (Only enabled if the "Use 'calc event time' output variable" option is chosen.)	X			X	
Attribute – the value of the attribute named in Variable Name		X	X		X
Item quantity – the item's quantity value		X	X		X
Item priority – the item's priority value		X	X		X

For the Variable Name column, use the default name, assign a name that has more relevance to the model, or select a name from a popup, depending on the type of variable selected. The Variable Value field reports the results for the selected output variable. In the Equation(I) block there is also a column to specify what should happen to the result if an output is needed but there is no item to trigger the equation's recalculation.

☞ The notation PRI (Parent Record Index) is used in blocks that interface with the ExtendSim database. It describes the type of value that is being read or written when dealing with a Child field.

Equation

An equation is a list of commands to be executed on one or more input variables, resulting in one or more output variables. In the equation pane, enter an equation that uses the names of the input and output variables and any of ExtendSim's built-in functions and operators. (Note that user-defined functions and procedures must be defined in include files; they cannot be called directly in the equation block.) The equation must be of the general form output = equation; (the semicolon at the end is required).

For example, an equation that uses the values from input connectors named Input1 and Input2 and outputs the result to an output connector named OutputA would be:

```
if(Input1 > Input2)
  OutputA = 2;
else
  OutputA = 5;
```

☞ To find out how to debug your equations, see "Debugging equations" on page 211.

The equation does not have to use all the names assigned to the input and output variables. However, if an input connector is connected, ExtendSim assumes that you will want to use it in the equation. If you don't use it, ExtendSim will give a warning when the simulation runs. ExtendSim will also warn if the equation uses a connector that is not named or is not connected.

☞ Equation-based blocks can calculate separate results and output them using any number of output variables.

The timing and control of equation calculations

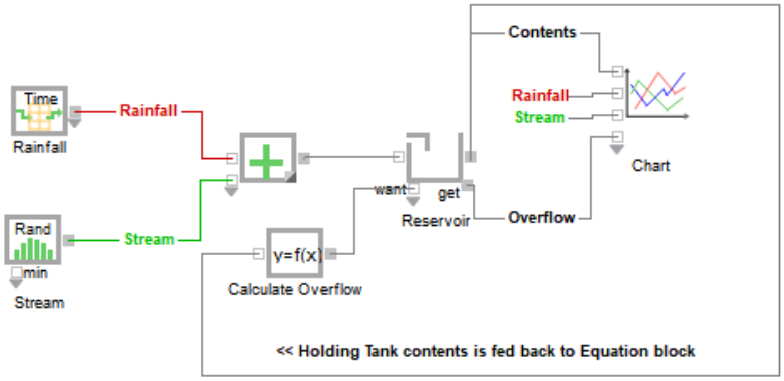
The equation-based blocks evaluate their equations:

- *Equation, Equation Query*. In a continuous model, at every step. In a discrete event or discrete rate model, when a message is sent to its value input or output connectors. The frequency of calculations can also be customized in the block's Options tab.
- *Equation(I), Equation Query(I)*. Each time the block gets an item at its item input connector.
- *Queue Equation*. Each time an item arrives or leaves and when a message is sent to one of the value input connectors.

The timing of equation calculations can have significant consequence in a simulation, affecting model behavior. To control when the equation is evaluated so that extraneous messages aren't generated, see the Step Message block (Utilities library) or the Pulse block (Value library).

Equation block example

As seen in the Reservoir 3 model (Examples/Tutorials/Continuous) one Equation block replaced four blocks in the Reservoir 2 model that calculate and remove overflow from a Holding Tank block.



Equation Editor

All of the equation-based blocks have a scrollable area for entering equations. Other than the Optimizer (Value library) and Buttons (Utilities library) blocks, the equation-based blocks also have a separate Equation Editor window for viewing and editing the equation.

Although the equation could be edited within the block's dialog, there are advantages to editing the equation in the Equation Editor window:

- The Equation Editor window can be resized to fit larger equations
- Tabs can be used for indenting
- Code colorization is allowed, so code is more understandable
- Other editing commands, such as matching braces, are supported

☞ Code completion is enabled in the Equation Editor window. That makes it easier to recall function names and their arguments, since you just start typing the first part of a function's name and choose it from the list that appears. Once the function has been placed in the script, enter an open parenthesis "(" immediately following it. This causes the parenthesis to turn red and causes call tips to display the function's arguments. The first argument will be bolded. When you enter it, the parenthesis will turn black. As you enter each argument, subsequent arguments get bolded until all are entered.

To see the Equation Editor:

- Open the Reservoir 3 model located at Documents/ExtendSim/Examples/Tutorials/Continuous
- In the model, open the dialog of the Equation block; it is labeled Calculate Overflow

- In the block's dialog, click *Open/Close Equation Editor*. A portion of the window that opens is shown in the screenshot to the right.

```
14  // The reservoir depth is 50 inches.
15  // represent the reservoir height, mal
16
17  real reservoirDepth;          // define
18  reservoirDepth = 50.0;      // set its
```

As you can see, the contents of the window automatically uses the indenting and code colorization features. To use code completion, type the first part of the function name in the code pane, then scroll through the list of functions that appears and choose the one you want.

Include files for equations

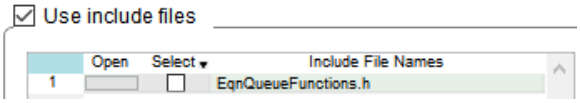
It is common that several blocks would use the same or similar variable definitions and functions. Include files simplify programming tasks that are repeated in multiple blocks. They are especially helpful when you define your own functions for one block and want other blocks to have access to those functions. In addition, include files allow you to use constants and user-defined functions and procedures in equation-based blocks, which don't directly support them.

For those who are building complex equations, most of the equation-based blocks have an interface for adding include files. This feature is implemented in the Equation and Query Equation blocks (Value library), Equation(I), and the Queue Equation, and Query Equation(I) blocks (Item library).

Include files in equations are similar to the standard ModL include header files used when creating a block and discussed in the Technical Reference. They can contain ModL symbol definitions such as *#define*, conditional compilation statements such as *#ifdef*, and function definitions.

The main differences between the two methods of using include files are:

- Include files used with equations are normally saved in the same location as the model using them; this makes it easy to move both the model and the includes it uses to a different location. Include files used in block code should be saved in the Extensions/Includes folder.
- Include files are added to block code via the *#include* syntax in the code, such as *#include "filename.h"*. However, include files in equations are added via an interface (the checkbox labeled *Use include files*) in the equation block's dialog. You cannot use the #include syntax in an equation block.



To create a new include file for an equation-based block:

- Check **Use include files** in the block's Equation tab
- Click the **Create New** button
- Give the new include file a name
- In the window that appears, enter the procedure or function. For instance:

```
//*********************************
//START: Equation Include Prototypes
```

```
procedure myproc();//this will be displayed in the prototype list
//END: Equation Include Prototypes
//*********************************
procedure myproc()
{
userError("Hello World");
}
```

- Use the procedure in the equation. In this case, when the equation is called the block will display a user error that says Hello World.

☞ If you have ExtendSim DE or Pro, see also the Equation Include File model; it is located at Documents/ExtendSim/Examples/Discrete Event/Tips.

Random numbers

The ability to include randomness and show dynamic aspects through time is one of the most valuable characteristics of a simulation experiment. Introducing randomness into a model mimics the patterns and unpredictability of the real world, increasing model accuracy. Since most models have randomness, it is important to understand random numbers.

ExtendSim has blocks, features, and functions that provide randomness in models. For example, the Random Number block (Value library) and the Create and Shutdown blocks (Item library) calculate random numbers using random distributions accessed through ModL functions. You can also specify random settings when using Sensitivity Analysis and for values in the ExtendSim database, and developers can directly access several random number functions.

Random number generators

The random functions produce random numbers based on a repeatable algorithm known as a *pseudo random number generator*. This generates the uniform random numbers used in the distribution functions. These 32-bit functions are seed-based and update their seed after being called. ExtendSim supports two types of random number generators:

- The recommended, default generator known as the minimal standard random number generator initially by Lewis, Goodman, and Miller, using new coefficients by Gerald P. Dwyer, Jr. (See Numerical Recipes in C, 2nd edition, pp.279 “A portable and reasonably fast minimum standard random number generator that uses Schrage's algorithm” and L'Ecuyer - Comm. of the ACM, Oct. 1990, vol. 33 and L'Ecuyer and Cote, ACM Transactions on Mathematical Software, March 1991.)

- An optional generator based on Schrage, “A More Portable Fortran Random Number Generator,” ACM Transactions on Mathematical Software, Vol 5, No. 2, June 1979, pages 132-138.

The first type of generator is specified by default in the Run > Simulation Setup >Random Numbers tab and is highly recommended for building models. The second generator is used mainly for backwards compatibility, so that models developed before ExtendSim 4.0 will retain results that are consistent with those older versions of ExtendSim.

Random seeds

A *random number stream* is a sequence of random numbers; the numbers in the stream are derived based on *seed* values. The pseudo random number generator is the internal mechanism in ExtendSim which calculates the numbers in the stream.

ExtendSim provides independent random number streams with the ability to specify a seed so that sequences of random numbers can be repeated. A random number is generated based on a seed.

You can specify a seed for the model as a whole in the Run > Simulation Setup > Random Numbers tab. A seed value of 0 or blank uses a random seed. Any other value causes repeatable sequences of pseudo-random numbers and the word "Seed" will appear in the model's status bar during the run. This gives repeatable results, allowing you to determine exactly how changes affect the model.

ExtendSim automatically assigns a separate seed to each block in the model that generates random numbers. To specify your own seed, enter a value in the *Use block seed* field in the dialog of those blocks. Any number entered as a seed in a block dialog will result in an independent random number stream that will not change across runs. Since each stream of random numbers is based on a seed, and you can have a separate seed for each block that generates random numbers, random number generation in ExtendSim is independent. The blocks that generate random arrivals or numbers are the Create and Shutdown blocks (Item library) and the Random Number block (Value library).

Each cell in an ExtendSim database can have an independent seed for the selected distribution. The database seed is dependent on what is set in the Run > Simulation Setup > Random Numbers tab. The seed field in the Sensitivity Setup dialog is independent of the seed that is set in the Random Numbers tab.

Resetting random numbers for consecutive runs

The Run > Simulation Setup > Random Numbers tab has three options for when a model is run repeatedly:

- Reset random numbers for every run
- Continue random number sequence
- Use Database table __Seed for values

These options are described in "Random Numbers tab" on page 86.

Probability distributions

In real life you cannot know exactly when an event is going to occur until it happens. For example, you do not know when the next customer will enter your store. However, by using the correct *statistical distribution* you can approximate what happens in the real world.

A distribution (also known as a *probability distribution* or a *random distribution*) is a set of random values that specifies the relative frequency with which an event occurs or is likely to occur. ExtendSim's random number distributions express both a probability that something will occur and a range of values that specify the maximum and minimum value of occurrence.

Distributions represent the data observed in real-world situations. When you gather data for a simulation model, it is seldom in a useful form. By "filling in the gaps," distributions help to compensate for information which was overlooked during data collection. For example, distributions account for extreme or outlying values which may have been missed during typically

short data-gathering intervals. Stochastic models use distributions as a handy method for converting data into useful form and inputting it into models.

Characteristics of distributions

The functions that produce a distribution have one or more parameter arguments which define and control its characteristics. The most important characteristics are a distribution's *shape*, its *spread*, and its *location* or *central tendency*. Shape is often used to identify distributions; for example, the bell-shaped curve of a normal distribution is widely recognized. Shape can be characterized according to *skewness* (leaning to one side or another) and *kurtosis* (whether it is peaked or flat). You specify the characteristics for the selected distribution by the values you enter for these arguments.

Choosing a distribution

Using random numbers means either choosing the theoretical distribution that best describes the variability of the raw data, describing the data using a user-defined or *empirical* distribution (such as the empirical distribution in the Create block), or fitting known data to a distribution. As seen below, there are many distributions in ExtendSim and it also has the ability to interface with external distribution-fitting software, as discussed in the next topic.

The choice of one distribution over another is not an exact science, but rather is dependent on the type and extent of the data which is gathered, the detail required for the process being modeled, and (in the case where little data is available), informed guesswork. If the data does not fit any of the distributions described below as "typical" for your process, but fits a distribution which is not typical, go with what your data tells you. It is usually better to use an approximate distribution than it is to keep a value constant.

Distribution fitting

There are also software applications which fit data to distributions. Use these tools in situations where there is empirical data you want to model using random distributions, but the ExtendSim distributions do not exactly fit. These products can help find the statistical distribution that best emulates the real-world data.

Some ExtendSim packages come with the Stat::Fit distribution fitting application (see "Stat::Fit (Windows only)" on page 172.) The Random Number block has a Distribution Fitting tab from which to launch a distribution fitting package, analyze empirical data, and determine the appropriate statistical distribution for a given data set.

ExtendSim distributions

When there is sparse or no data, this list of common uses may help you select a plausible distribution in the Create, Shutdown, or Random Number blocks, within the Sensitivity Setup dialog, or when formatting a cell in the ExtendSim database:

Distribution	Definition
Beta	Distribution of random proportion, such as the proportion of defective items in a shipment, or time to complete a task.
Binomial	The number of outcomes in a given number of trials. Most often used to express success/failure rates or the results of experiments, such as the number of defective items in a batch or the number of customers who will arrive who are of a particular type.

Distribution	Definition
Cauchy	Used to represent the ratio of two equally distributed parameters in certain cases or wildly divergent data as long as the data has a central tendency. It has a sharp central peak but broad tails that are much heavier than the tails of the Normal distribution.
Chi Squared	Used in statistical tests but, since it does not have a scaling parameter, its utilization is somewhat limited. It is a subset of the Gamma distribution with beta = 2 and alpha = nu/2.
Constant	This does not produce a random number, but a constant value which does not change. Used when there is exactly the same amount of time between arrivals or as a method to reduce the effects of randomness in the early stages of model building.
Empirical	Used to generate a customized or user-defined distribution with a special shape when the probability of occurrence is known. The options are: *discrete* (the block will output the exact values given in the table); *stepped* (values in the table will be used as probabilities of ranges of data); and *interpolated* (the probability distribution will be interpolated between the data points).
Erlang	Frequently used for queueing theory to represent service times for various activities or when modeling telephone traffic.
Exponential	Primarily used to define intervals between occurrences such as the time between arrivals of customers or orders and the time between failures (TBF) or time to repair (TTR) for electrical equipment. Also used for activity times such as repair times or the duration of telephone conversations.
Extreme Value Type 1A	describes the limiting distribution of the greatest values of many types of samples. Used to represent parameters in growth models, astronomy, human lifetimes, radioactive emissions, strength of materials, flood analysis, seismic analysis, and rainfall analysis. Its peaked shape is always the same but it may be shifted or scaled.
Extreme Value Type 1B	Describes the limiting distribution of the least values of many types of samples. Represents parameters in growth models, astronomy, human lifetimes, radioactive emissions, strength of materials, flood analysis, seismic analysis, and rainfall analysis.
Gamma	Typically used to represent the time required to complete some task. The distribution is shaped like a decaying exponential for shape (2) values between 0 and 1. For shape values greater than 1, the distribution is shaped like a bell curve skewed towards the low end.
Geometric	Outputs the number of failures before the first success in a sequence of independent Bernoulli trials with the probability of success on each trial. Typically used for the number of items inspected before encountering the first defective item, the number of items in a batch of random size, or the number of items demanded from an inventory.
HyperExponential	Usually used in telephone traffic and queueing theory.

Distribution	Definition
Hypergeometric	Describes the number of defects, x, in a sample of size s from a population of size N which has m total defects. It is used to describe sampling from a population where an estimate of the total number of defects is desired. It has also been used to estimate the total population of species from a tagged subset.
Inverse Gaussian	Originally used to model Brownian motion and diffusion processes with boundary conditions. It has also been used to model the distribution of particle size in aggregates, reliability and lifetimes, and repair time.
Inverse Weibull	Describes several failure processes as a distribution of lifetime. It can also be used to fit data with abnormal large outliers on the positive side of the peak.
Johnson SB	Used in quality control to describe non-normal processes, which can then be transformed to the Normal distribution for use with standard tests. It is a continuous distribution that has both upper and lower finite bounds.
Johnson SU	Used in quality control to describe non-normal processes, which can then be transformed to the Normal distribution for use with standard tests. It is an unbounded continuous distribution.
Laplace	Used in error analysis and to describe the difference of two independent, and equally distributed, exponentials.
Logarithmic	Describes the diversity of a sample, that is, how many of a given type of thing are contained in a sample of things. For instance, this distribution has been used to describe the number of individuals of a given species in a sampling of mosquitoes, or the number of parts of a given type in a sampling of inventory.
Logistic	Most often used a growth model: for populations, for weight gain, for business failure, etc. Can also be used to test for the suitability of such a model, with transformation to get back to the minimum and maximum values for the Logistic function. Occasionally used in place of the Normal function where exceptional cases play a larger role.
Log-Logistic	For Shape = 1, it resembles the Exponential distribution. For Shape < 1, it tends to infinity at Location, and decreases with increasing X. For Shape > 1, it is zero at Location, and then peaks and decreases.
LogNormal	Often used to represent the time to perform an activity (especially when there are multiple sub-activities), the time between failures, or the duration of manual activities. This distribution is widely used in business for security or property valuation, such as the rate of return on stock or real estate returns.
Negative Binomial	Number of failures before Sth success. P specifies the probability of success.
Normal	The well-known Gaussian or bell curve. Most often used when events are due to natural rather than man-made causes, to represent quantities that are the sum of a large number of other quantities, or to represent the distribution of errors.

Distribution	Definition
Pareto	Represents the income distribution of a society. It is also used to model many empirical phenomena with very long right tails, such as city population sizes, occurrence of natural resources, stock price fluctuations, size of firms, brightness of comets, and error clustering in communication circuits.
Pearson Type V	A distribution typically used to represent the time required to complete some task. The density takes on shapes similar to lognormal, but can have a larger "spike" close to x = 0.
Pearson Type VI	A distribution typically used to represent the time required to complete some task. A continuous distribution bounded by zero on the left and unbounded on the right.
Poisson	Models the rate of occurrence, such as the number of telephone calls per minute, the number of errors per page, or the number of arrivals to the system within a given time period. Note that in queueing theory, arrival rates are often specified as poisson arrivals per time unit. This corresponds to an exponential interarrival time.
Power Function	A continuous distribution with both upper and lower finite bounds. It is a special case of the Beta distribution with q = 1. The Uniform distribution is a special case of the Power Function distribution with p = 1.
Rayleigh	Frequently used to represent lifetimes because its hazard rate increases linearly with time, e.g. the lifetime of vacuum tubes. This distribution also finds application in noise problems in communications.
Triangular	Usually more appropriate for business processes than the uniform distribution since it provides a good first approximation of the true values. Used for activity times where only three pieces of information (the minimum, the maximum, and the most likely values) are known.
Uniform Integer	Describes a integer value that is likely to fall anywhere within a specified range. Used to represent the duration of an activity if there is minimal information known about the task.
Uniform Real	Describes a real value that is likely to fall anywhere within a specified range. Used to represent the duration of an activity if there is minimal information known about the task.
Weibull	Commonly used to represent product life cycles and reliability issues for items that wear out, such as the time between failures (TBF) or time to repair (TTR) for mechanical equipment.

These distributions and their arguments are described more fully in the Help of blocks that use them.

Integration vs. summation in the Holding Tank block

he Holding Tank block (Value library) has integration capabilities. The block's dialog gives the option to either sum or integrate its input. In general, a good rule of thumb is:

- Integrate when the value going into the Holding Tank is based on simulation time units, such as a rate. For example, you would integrate when the Holding Tank block's input represents dollars per year or gallons per hour.

- Sum when the value is to be added to the block at each step or dt calculation. For example, you would sum when the Holding Tank block's input is orders or people.

Summing adds the given input to the total at each step, regardless of the time units. Integration considers the input to be spread evenly over each time unit; at each step integration adds a portion of the input to the total. For example, the following table shows the effect of inputting $2000 to the Holding Tank block when the time units are in years and dt (delta time) is set to 0.25 (for 1/4 of the year).

As seen in the table, if the Holding Tank is set to sum its inputs, it would be the equivalent of adding $2000 to the account every quarter. If the Holding Tank is set to integrate, it would be the equivalent of adding $500 per quarter.

Time	Step	Summed	Integrated (delay)	Integrated (no delay)
0	0	2000	0	500
0.25	1	4000	500	1000
0.50	2	6000	1000	1500
0.75	3	8000	1500	2000
1	4	10000	2000	2500

- *Summation* occurs at each step so there is an amount calculated at time 0. Since summation treats its input as an amount, the entire 2000 is added at each step.
- The *integrated (delay)* choice treats its input as a rate and calculates a new result at the next step. Because this backward Euler integration occurs during the interval between steps, there is no amount at time 0.
- The *integrated (no delay)* choice also treats its input as a rate. However, it calculates a new result at the current step. Because its integration occurs at each step, there is an amount at time 0.

It is also important to note that if you subsequently change the delta time to something other than 0.25, the total amount in the summed Holding Tank would be different from the amounts shown above, but the total amount in the integrated Holding Tank would remain the same.

Your choice of integration methods depends on the model:

- You would generally use the *integrated (delay)* choice when there is only one integrating block in the model, when the integrating blocks are not interdependent or cross-coupled, or when there is no feedback.

- In models with more than one integrating block, where the integrating blocks are interdependent or cross-coupled, the feedback between the blocks is usually a correction factor. If this feedback is delayed, the system may correct too late or overcorrect, causing the model results to become unstable. This is often observed as a graph where the traces oscillate with increasing magnitude as time progresses. The *integrated (no delay)* choice compensates for the feedback delays by outputting results one step earlier. For example, the Predator_Prey model is an example of interdependent Holding Tank blocks; the model is located in the folder Examples/Continuous/Standard Block Models.

How To

Debugging Tools

Learn how ExtendSim can
help find errors in your models

*"If debugging is the process of removing bugs,
then programming must be the process of putting them in."
— Unknown author*

Creating a model is not so fool-proof that your work is finished once the model has been built. Two important steps in any simulation project are *verification* of simulation results (compare the results to what was intended or expected) and *validation* of results (compare the model to the real system).

ExtendSim provides several methods for detecting problems in models and for debugging models that are not working as expected, including:

- Hints for debugging models
- Debugging item flow using item *Contents* tab of queues and activities
- Debugging equations in equation based blocks
- Verifying results at each step of the model-building process
- Specific blocks used for debugging
- Getting the information you need to debug a model
- Using the Find command or the Find and Replace block (Utilities library) to easily locate blocks or dialogs that require attention
- Dotted lines to show unconnected connection lines
- Running with animation to see if a model is behaving as expected
- Using the Notebook to keep track of critical parameters in one location
- Stepping through the model as it progresses
- Show Simulation Order command
- Model tracing for comparison with the real system
- An automated test environment to compare model results between one release of ExtendSim and another

☞ This chapter is concerned with debugging a model, not with debugging custom blocks. The ExtendSim Technical Reference discusses in detail how use the ExtendSim Debugger to find bugs in blocks you create.

Debugging hints

Efficient debugging of a simulation requires an organized, logical approach. Following the following steps will shorten the process:

- Duplicate the bug. Fix the random number stream so that the problem occurs the same way and at the same time whenever the model is run.
- Describe the bug. Defining the difference between the correct behavior and the observed behavior can lead to insight into the source of the problem. This also helps in formulating a strategy for locating the source of the bug.
- Assume the bug is yours. The vast majority of modeling errors are caused by the modelers themselves. It makes sense to start with the most likely source of the bug.
- Divide and conquer. Determine the source of the bug. And, determine where the bug is not. Build the simplest model that duplicates the error. This will make the model run faster and there will be fewer variables to consider in the debugging process.

- Think creatively. Bugs don't always come from the expected locations. If the source of the problem is not immediately evident, you may be looking at only a symptom. Look at other places in the model that could be the actual source of the error.
- Leverage tools. ExtendSim comes with a variety of tools for debugging a simulation model. For discrete event models, adding a History (Value library), Item Log Manager (Report library), or Record Message (Utilities library) block can provide insight into the operation of the model. In continuous models, writing a sequence of values to the ExtendSim Database or to a global array is an easy way to record the values at a specific point in the model. Trace files are also useful at this point.
- Start heavy debugging. Focus on the problem at hand and on how it can be fixed
- Learn and share. This may be a problem that could occur in someone else's models. Share your experiences with other modelers through the ExtendSim E-Xchange or the ExtendSim LinkedIn site.

Verifying results as you build a model

One of the most efficient methods for debugging models is to verify that the model is working correctly at each step during the model building process. It is a lot easier to find problems as you create each section of the model than to try to debug a finished model. You can use almost any of the debugging features discussed in this chapter, but the two most common methods to debug at each step are by examining connector information and cloning dialog items.

Connector information

Connectors provide helpful information when you are debugging models. Run the model at any point in the model building process. As the model runs, or at the end of a run, hover the cursor over a connector to see its name and current value. You may also see additional information depending on how the block is programmed.

Cloning dialog items

To focus on a particular parameter, clone it to the model worksheet and watch it change as the simulation runs. You can also turn animation on to compare the cloned parameter to the behavior of the surrounding blocks. Cloning dialog items is discussed at "Cloning" on page 72.

Blocks for debugging

ExtendSim libraries have many blocks that are useful for debugging models either during or after a simulation run, including:

Block	Library	Category	Use
Display Value	Value	Inputs/ Outputs	Displays the value of its input connector at each simulation step on the block's icon and in the dialog. In the dialog, set the time between displays.
Notify	Value	Inputs/ Outputs	Plays a sound or stops the simulation when its input is ≥ 0.5. Optionally displays a message set in its dialog.
Statistics	Report	Statistics	Displays information about all blocks of a certain type, such as all queues or all activity blocks.

Block	Library	Category	Use
History	Item	Information	Displays statistics and history (arrival time, priority, and attributes) of items that pass through it. See “Using the History block to get item information” on page 398.
Information	Item	Information	Counts items that pass through it and reports the time between item arrivals and cycle time statistics. See “Information” on page 137.
Item Log Manager	Report	Information	Collects specified data in selected History blocks and blocks with Contents tabs. Stores the data in an ExtendSim database for analysis and output as a report. See “Using the Item Log Manager to get item information” on page 399.
Reports Manager	Report	Information	The primary interface for creating reports from simulation runs for blocks, events, items, and resources. See “Reports Manager block” on page 185.
Event Monitor	Utilities	Discrete Event Tools	Reports the contents of the ExtendSim discrete event calendar.
Find and Replace	Utilities	Information	Finds specified dialog items and replaces their values. Drag a clone of a dialog item onto this block’s icon to search for similar dialog items.
Item Messages	Utilities	Discrete Event Tools	Records the messages sent over item connectors.
Link Alert	Utilities	Discrete Event Tools	Sends a message when the data in the linked database location changes. Use with the Record Message block. See the *ExtendSim Database Tutorial and Reference*.
Memory Usage	Utilities	Information	Reports the size of each block, array, or database table in the model in bytes.
Model Compare	Utilities	Information	An automated testing environment for model verification.
Pause Sim	Utilities	Model Control	Causes the simulation to pause when certain conditions are met. Click Resume to continue execution
Record Message	Utilities	Discrete Event Tools	Records messages sent over value connectors in a discrete event or discrete rate model.
any graph	Chart		Add a chart block any place in a model and connect it to the values you want to track. See “Chart library” on page 174.

Measuring performance to debug models

ExtendSim provides several methods for obtaining model information when debugging models:

- Dialog boxes display data pertinent to the specific block and in some cases automatically perform statistical calculations. For instance, the dialog of the Queue block (Item library)

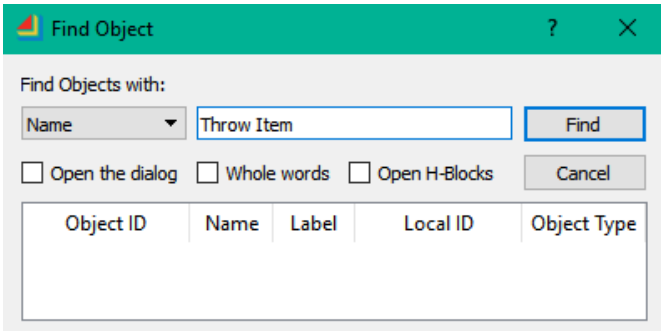
reports utilization and maximum queue length as well as the number of arrivals and departures.

- You can clone dialog parameters to the model window or to the Notebook to create customized reports and control panels, as shown in “How to clone a dialog item” on page 72.
- Many of the blocks in the libraries have value output connectors that give direct access to specific information. For example, the **U** output connector on the Activity block outputs utilization values. You can attach any value output to a plotter to display information about model performance. You can also attach value outputs to value inputs on diagnostic-type blocks, such as to the Display Value block (Value library) to display information about that output.
- Charts, from the Chart library, conveniently display graphs and tables of data over time. These blocks are useful not only for showing results but for identifying trends and anomalies. You can choose what you want plotted and how you want it displayed, and you can use as many plotters in a model as you want. See “Chart library” on page 174 for more information.
- Animation shows the flow of items in a model, levels of values, etc. ExtendSim blocks have built-in customizable animation; you can also add custom animation using the Animate tabs in block dialogs or by using blocks from the Animation library. Animation is especially useful for verifying a model since it can show if portions of the model are operating as expected. Since animation can slow model performance considerably, it is common that you would use animation in the early stages of model-building or for presentations. See “Animation features for debugging” on page 215 for more information.
- There are numerous blocks that can be used for debugging models and verifying results. For instance, the Notify block (Value library) can stop the simulation and notify you when its input goes above or below a specified level. The Information block (Item library) provides information about the output of the block it is connected to (the interval between arrival times, how many items are currently present at the output, and so forth).
- Sensitivity analysis allows you to vary a parameter incrementally, randomly, or in an ad hoc manner to determine how sensitive model results are to changes in one variable. See “Sensitivity analysis” on page 138 for more information.
- Running simulations multiple times, such as for Monte Carlo simulations, gives ranges of values indicating the possible outcomes for the model. See “Running a model multiple times” on page 88 for more information.
- The Statistics block (Report library) reports and statistically evaluates results. For example, it can display information about every queue-type block in the model and calculates the confidence intervals based on the results. As discussed in “Clear Statistics” on page 136, the Clear Statistics block (Value library) resets statistical accumulators at random intervals or in response to a system event; this is used to eliminate statistical bias during the warm-up period.

- The Utilities library contains two blocks that are useful for debugging discrete event models. The Record Message block, when connected between two value connectors, shows all of the messages, the values transferred, and whether the message came in the input or output connector. The Item Messages block records the message communication between two item connectors. See the Discrete Event Quick Start guide for a detailed discussion of the item-based messaging system.

Find command

If a model is large, it might be difficult to find all the blocks by sight. For example, you may see in the Trace file that a particular block didn't get the expected input. The Edit > Find command lets you locate a block or other object by its name, ObjectID, or label.

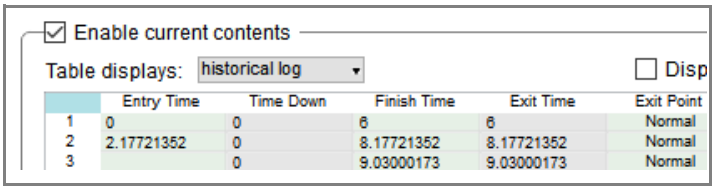


This command opens a different window depending on where the search is initiated: the worksheet, the block's structure, or a database window. See "Find..." on page 543 for more information about this command.

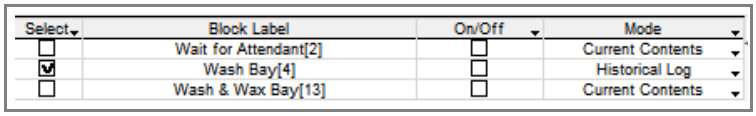
☞ You can also use the Find and Replace block, listed in the table of debugging blocks above, to find blocks. It is even more useful for finding specific dialog items in the located blocks, so you can replace their parameter values.

Item Contents of queues and activities

The Contents tab found in queues, activities, and the Resource Item blocks allows you to record and view either which items currently reside in the block or which items have historically traveled through the block. The table is customizable—expand the table and use the popups to select additional properties, such as cost or item quantity, that you want tracked.



While this feature can be quite useful, turning it on can slow model execution speeds and unnecessarily grow the size of your models. So you wouldn't want to leave it on all the time.



However, since this feature will most often be used for debugging purposes, the Executive block has an "Item Contents" tab where you can remotely control the Contents tabs in all the blocks in your model from one location. This will make it easier for you to selectively turn contents tracking on during the debugging phase of your model development and then turn it off once debugging has been completed.

We also expect customers to find uses for contents tracking beyond debugging. For example, linking a block's contents table to a database table would make that block's current item content easily accessible to other blocks in the model.

☞ See the Help for each block, including the Executive, to see what the dialog settings mean.

The Source Code Debugger

The ExtendSim Source Code Debugger is the ultimate model debugging tool. With the debugger you can step through every statement, inspect the value of every variable, and view the sequences of messages sent from one block to another. Using the debugger requires that you understand the ModL language and how blocks interact with each other. For more information, see the Technical Reference.

Debugging equations

The equation-based blocks allow you to enter equations that are simple logic statements (if x, then y) or complex calculations that control the model using ModL functions. While in many cases you won't need to debug the equation, ExtendSim provides an equation debugger so you can determine if equations are performing as expected. This works like the Source Code Debugger that ExtendSim developers use, but it is specific to the equation-based blocks used by many modelers.

The equation debugger allows you to:

- Set breakpoints for equations. A breakpoint is where the debugger initially stops execution of the equation so you can step through the code.
- Set breakpoint conditions for equations. Setting a breakpoint condition causes the breakpoint to stop execution only under certain circumstances.
- Step through lines of code and examine the variables. Step commands allow you to trace the execution of the equation and examine the effects on the variables defined in the code.

Tutorial for the equation debugger

- Open the Reservoir 3 model located at Documents/ExtendSim/Examples/Tutorials/Continuous
- So that you don't overwrite the original model, save the model as **Equation Debugging**
- In the model, open the dialog of the Equation block; it is labeled Calculate Overflow

☞ Although the following tutorial uses a continuous model, the equation debugger can be used with any of the equation-based blocks. For a list of equation-based blocks, see "Equation-based blocks" on page 189.

Setting breakpoints

To step through each line of code, it is first necessary to stop the code execution. In this example, it requires placing a breakpoint before the "if" statement at the beginning of the equation:

- In the dialog of the Equation block, check the *Enable Debugger* check box
- Click the **Set Breakpoints** button

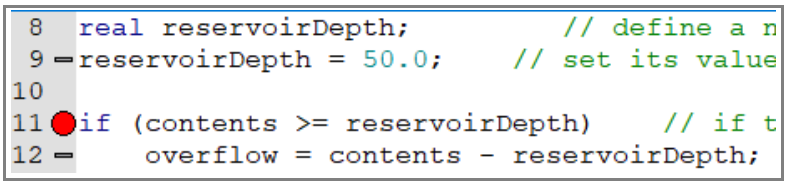
This recompiles the equation in debugging mode and opens two windows:

- The *Set Breakpoints* window is where you set the breakpoints

- The *Breakpoints* window lists the breakpoints that have been set. It is used later in this tutorial, for "Setting a conditional breakpoint" on page 213.

☞ Code in the Set Breakpoints window cannot be edited.

The gray breakpoint lines in the left margin of the Set Breakpoints window indicate the only places where breakpoints can be set.



▸ In the left margin of the Set Breakpoints window, click the breakpoint line that is at the left of the *if* statement. This turns the gray dash into a red circle. If you click at the wrong line of code, just click it again to remove the breakpoint.

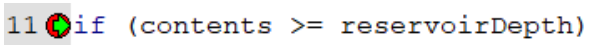
▸ Close the Set Breakpoints window.

▸ Run the simulation. The simulation will run until the breakpoint is reached.

Stepping through the code

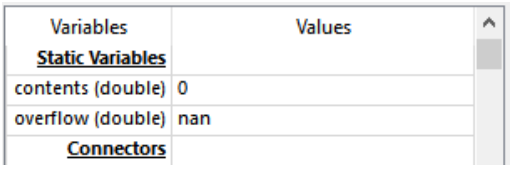
When the breakpoint is reached, the Debugger window opens.

- In the Source pane a green arrow is placed at the left of the breakpoint. This indicates that the code before the green arrow has been executed.



☞ The green arrow indicates which line of code will be executed next, once the run is continued.

- The Variables pane at the top right shows the variables used in the equation and the value of each variable at the point before the line of code at the breakpoint is executed.



- Along the top of the window is a toolbar with six buttons. These are used to continue the run, to step over, into, or out of functions, and to stop the run with or without code editing. (The buttons are explained fully in the Debugging chapter of the Technical Reference.)
- The Call Chain pane at the upper left tells where control is in the chain of command. It is not used for debugging equations unless there are blocks in the model that have debugging code.

The "if" statement has not yet executed. To watch the statement execute:

▸ Repeatedly click the **Step Over** button at the top of the Debugger window. Alternatively, use the right arrow to step over the function.

The green arrow in the Source pane indicates the flow of the code; the Variables pane shows the change in value for the variables.

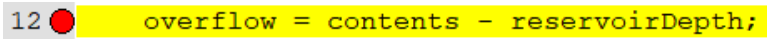
☞ Hovering the cursor over a variable in the Source pane will also show its current value.

Getting to the overflow calculation

The next step is to set a breakpoint to find out when the overflow value first gets calculated.

In the Debugger window:

- Remove the first breakpoint by clicking its red circle
- Add a new breakpoint at the line below it, the "overflow = contents..." line



- Do one of the following to continue the run:
  - Click the **Continue** button at the top of the Debugger window
  - Or, press *F5*
  - Or, use Shift+right arrow

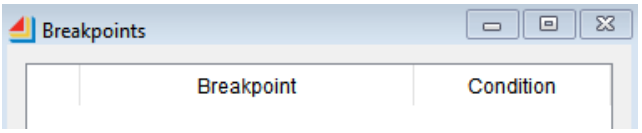
Although the breakpoint is set at the overflow calculation line, the green arrow indicates that the calculation has not yet been performed. Thus the overflow value will not be current.

In order to see the current overflow value:

- Click the **Step Over** button to execute that line of code. The calculated overflow value is shown in the Variables pane.

Setting a conditional breakpoint

Conditional breakpoints cause a calculation to break only when the overflow value is calculated and within a specified range or specific value.



The *Breakpoints window* is for setting breakpoint conditions. It allows you to:

- See which breakpoints have been set and where they have been set
- Add conditions to a breakpoint
- Click a breakpoint's red circle to disable or enable any breakpoint, without deleting it. (In place of the red circle, disabled breakpoints have an open circle.)
- Delete a breakpoint by selecting it's name and clicking the Delete or Backspace key

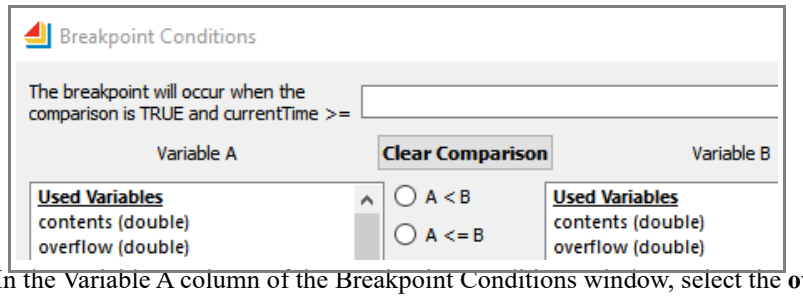
To set a condition on the breakpoint so it will break only if the overflow is greater than 0.0:

- In the Debugger window:
  - Remove the existing breakpoint by clicking its red circle in the Debugger window.
  - Add a new breakpoint to the Debugger window after the entire "if" and "else" statement; in this case at the ending brace that is below the "overflow = 0.0" statement.

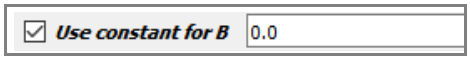
The next step is to write a comparison statement that will compare the overflow value to zero.

- In the Breakpoints window:

- Double-click in the Condition column that is to the right of the Breakpoint column. This opens the *Breakpoint Conditions* window, shown below.



- In the Variable A column of the Breakpoint Conditions window, select the **overflow (double)** variable; this will become A in the equation.
- Since you want the condition to determine when the overflow is greater than zero, choose **A>B** as the comparison in the middle column.
- Below the Variable B column, check the **Use constant for B** check box and enter **0.0** as the constant.



- Click OK

Notice that the breakpoint's red circle in the Debugger window has changed to a blue circle, indicating there is a condition on the breakpoint.

Since you have set conditions, and you want to find the first time the condition is met, the calculation must start again from the beginning. To start the run over:

- In the Debugger window, click the red Stop Debugging button This closes the window and stops the model run.
- Run the simulation. The first overflow value greater than 0 will be displayed in the Variables pane.

As an additional exercise, try using different values for the overflow condition, such as 2.0, to see when that value first appears.

☞ Advanced modelers might benefit from reading about the Source Code Debugger in the Technical Reference.

Dotted lines for unconnected connections

If a block is not getting an input or is not generating an output when you think it should, it may not be connected properly in the model. This can happen when connections run underneath blocks when you thought that they were connected to the blocks.

ExtendSim shows incomplete connections as a *red dotted line*. To fix these, delete the incomplete connections by double-clicking them and reattach them to where they are supposed to be. Clicking one segment selects that segment; double-clicking a segment selects the entire connection line.

Animation features for debugging

Running the model with 2D animation on is useful for debugging. If the animation goes by too quickly for debugging, slow down the process with the Slower Animation (turtle) button in the toolbar. To resume speed, use the Faster Animation (rabbit) button.

To learn more about how models, blocks, and connection lines are animated see "Animation" on page 110.

Animating the model

If you know how a block is animated (see the block's Help), you can watch its icon to determine if something is not acting as expected. For example, if a block indicates the status of its contents, you can use that information to debug the model.

Discrete event models have additional animation capabilities and can show the flow of items along connection lines and between named connections. This gives a visual representation of what is happening in the model. Many times these additional animations are specified in the block's dialog.

Animating item properties (discrete event models only)

Changing an item's animation depending on a property (attribute, quantity, priority) is helpful when debugging models. For example, in a block's *Item Animation* tab you can visually differentiate between types of items by setting separate animation objects for each attribute value. Then observe the individual items as they flow through the model.

Notebook

Notebooks are a convenient way to see the final values for several dialog items in a model or to compare inputs to outputs. This is useful in debugging because you can group the dialog items by what their final values are expected to be.

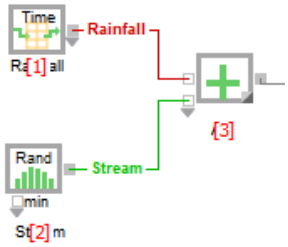
For more information, see "Notebooks" on page 79.

Stepping through the simulation

The Model toolbar has buttons that help if you are debugging a model. There are also Stop, Pause, Resume, and Step commands in the Run menu and Pause at Beginning, Step Each Block, and Step Entire Model commands in the Run > Model Debugging menu.

See "Stepping through a model" on page 89 for methods to use.

Show Simulation Order command (continuous blocks only)



ExtendSim normally determines the order that blocks in a continuous model are executed by following the path of connections. If a continuous model is behaving in an unexpected manner, you may want to see explicitly the order ExtendSim is using to execute calculations in a model.

Selecting the Model > Show Simulation Order command puts a small number on each block's label indicating its order of execution.

☞ Although also accurate for groupings of continuous blocks in non-continuous models, this display will be inaccurate for discrete event (Item library) and dis-

crete rate (Rate library) blocks in those models. This is because discrete blocks can generate block-to-block messages and override the system's simulation order.

Slow simulation speed

There are many reasons why a simulation would not run as quickly as you might expect. Some common causes, and the methods to detect and avoid them, are discussed in "Speeding up a simulation" on page 98.

Model reporting

The blocks in the Report library are useful for generating customized reports of data to help debug models. However, model tracing (discussed in the next topic) can offer more detailed information. So you will probably use that option more for debugging and the reports option for analysis.

To learn more about how to generate reports in ExtendSim, see "Model reporting" on page 184.

Model tracing

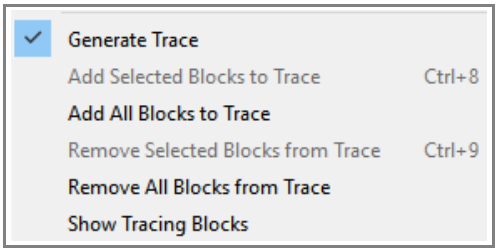
Model tracing is useful for finding anomalies that occur as the simulation runs, generating a trace text file that shows the details of block values at every step or event in the simulation.

Because of the large amount of information generated by the model tracing commands, most people don't use model tracing often. However, tracing is a highly effective method for following a single block or a few blocks to watch for values that do not match expectations.

The trace is saved as a text file and opens automatically at the end of the run. If the *Runs* field in the Simulation Setup dialog is greater than 1, each consecutive trace is added to the end of the file so that you can compare traces from earlier runs.

Generating traces

The tracing commands are located in the Run > Model Debugging menu. To generate a trace:



- Select the blocks you want included in the trace:
  - To specify individual blocks for tracing, select them and choose the **Add Selected Blocks To Trace** command.
  - To get a trace on every block, choose **Add All To Trace**.
- Choose the Run > Model Debugging > Generate Trace command to create a trace the next time you run the model.
- Run the simulation to see the trace results. ExtendSim prompts you for a name and location for the trace file. Trace reports are opened, closed, and edited just like any other text file in ExtendSim.
- To trace fewer blocks in the next simulation run, select the blocks you want out of the trace report and select Remove Selected Blocks from Trace. To start over on the selection of blocks, choose Remove All Blocks from Trace.

You can see which blocks are included in the trace by choosing Show Tracing Blocks; each traced block in the model displays the word *Trace* on its icon.

Tracing example

For the Math block from the Reservoir 1 model, the top of the tracing report is:

```
ExtendSim Trace - 9/16/2017 3:48:57 PM
Run #0
### calculation at Math number 2. CurrentTime:0.
ValuesIn = 2.6
ResultOut = 3.1550072961999
----------------------  Step #1  ----------------------
### calculation at Math number 2.  CurrentTime:1.
ValuesIn = 4.4
ResultOut = 5.3917648062033
----------------------  Step #2  ----------------------
### calculation at Math number 2.  CurrentTime:2.
ValuesIn = 6.7
ResultOut = 7.1088551154443
----------------------  Step #3  ----------------------
### calculation at Math number 2.  CurrentTime:3.
ValuesIn = 3.4
ResultOut = 3.8298759283776
```

☞ If you build your own blocks and want to use the Trace features for debugging, add special code to the blocks, as described in the Technical Reference.

Automated test environment (Windows only)

To assist you in your modeling efforts and to assist your company in the version accreditation process, ExtendSim includes an automated test environment for model verification. This helps identify any differences in two situations:

- Verifying that models will generate the same results when there is a new release of ExtendSim. The test environment simplifies the process of comparing simulation results between two ExtendSim releases – looking for any differences between models that were saved in a previous release and the same models run in the new release. See page 218 for how to do this.
- Determining if changes made to a model affect the results. In the same manner as verifying between releases, you can compare copies of the model saved before and after changes as discussed in “To compare results for one model” on page 219.

⚠ The test environment only runs on Windows. In addition, the first time you launch the Compare Results application, you will need to run ExtendSim as Administrator. See Windows documentation for how to run as Administrator.

Components

There are two components to the test environment:

1) The *Compare Results* executable

2) The *Model Compare* block, located in the Utilities library

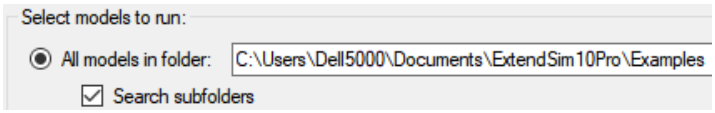
As discussed below, you use these differently depending on whether you want to test several models at a time or only one model.

To perform verification for several models:

- Launch the mini-app named Compare Results:
  - Go to the Start Menu
  - In the list of Programs, select ExtendSim/Documents
  - Locate the Compare Results.exe application
  - Click "Compare Results" to launch that application. This also launches ExtendSim, if it is not already open.

⚠ It is important to note which ExtendSim release will be used for the test. If ExtendSim is not open, the Compare Results tool will launch the default version, which may not be the most current release. The safest method is to first open the ExtendSim release you want to evaluate.

- In the Compare Results window that appears (shown at right), use the Browse buttons to select all the models located in a particular folder or to load a text file of models to run.



☞ Note that when you select one folder, any subfolders will also be selected by default. To avoid this, uncheck the *Search subfolders* checkbox or create a text file with the specific models you want to run, as discussed below.

- Click the Run Models button. This does the following:
  - If no model is open, creates a new model with some text in it. (Do not close the new model until the Compare Results process is finished.)
  - Opens the selected models
  - Temporarily turns off animation in those models
  - Temporarily places the Model Compare block (Utilities library) in each model as it is run
  - Runs the models one at a time. Note that you may get a "Run as Administrator" message when the first model is run. As noted in the alert above, you need to run ExtendSim as Administrator to use the Compare Results application.
  - Reports the project's progress in a green status bar below the list of models; the status and results for each model are reported in the columns.
- Any models without differences are automatically closed without saving, discarding the Model Compare block.
- Models with differences are left open in ExtendSim. (You can also choose that the Compare Results window only display those models.)
- If there are any model differences, go to ExtendSim and open the Model Compare block that is in the upper left corner of the model. Each parameter that has a difference will be indicated by an entry in the column labeled *Different*.

- When you close these models, the Model Compare block will be discarded and the model will revert to its prior saved condition.

Creating a text file

You can use the application to create a text file or create one in Word or some other application.

To use the Compare Results application to generate the list of models to run:

- Browse to the folder that contains the models you want
- Click the Save Model List button to save the generated list as a text file
- Edit the text file so that it only contains the models you want to verify
- In the Compare Results application, browse to the edited list of models

Alternately, you can directly create a text file using any appropriate application. Create the list by entering the full path to each model in each row (for example: C:\Users\John\Documents\ExtendSim9.2\Examples\Continuous\Standard Block Models\Monte Carlo.mox)

To compare results for one model

While designed to assist in model verification between releases, this test environment can also be used to run different versions of a model and compare the results. For example, you could make a change to a saved model and run it before saving again to see if the results changed.

To do this:

- Run and save the model
- Then make your changes
- Place the Model Compare block in the model, open its dialog, and click the *Run and Compare* button

The results will be displayed in the Model Compare block's dialog.

	Block Name	Block Number	Parameter Name	Before Value	After Value	Different
0	Queue	17	AveLength_prm	0.29283793521	3.01856715612	X
1	Queue	17	AveWait_prm	1.20138640087	12.3151675342	X
2	Queue	17	MaxLength_prm	3	7	X


To have your changes saved, you must first delete the Model Compare block, then Save the model. Simply closing the model will discard your changes along with the Model Compare block.

How To

Data Management and Exchange

Managing and transferring data within ExtendSim
and between ExtendSim and other applications

"It is a capital mistake to theorize before one has data."
— Sir Arthur Conan Doyle

Overview

ExtendSim provides a variety of standards-based options for managing and sharing data internally and with other applications. This chapter covers:

- User interfaces for exchanging data within ExtendSim and between ExtendSim and external data structures
- Internal structures, such as ExtendSim databases and global arrays, for storing and managing data
- Blocks for accessing and managing data
- How data source types are indexed and organized
- Transferring data between ExtendSim and devices
- Standard communication technologies, such as ADO and COM Automation


This chapter is concerned with the communication of data. For information about working with graphic objects or pictures, see “Graphic shapes, tools, and commands” on page 107 or “Copy/Paste and Duplicate commands” on page 250.

User interfaces for data exchange

The table below summarizes the methods for exchanging data within ExtendSim and between ExtendSim and external applications and devices. Some of these methods (such as Dynamic Data Linking) are specific to ExtendSim and some (such as ADO) are industry standards for inter-application communication.

User Interface	Method	Examples	Data Direction	Dynamic Live Link	ExtendSim or External Application	See Page
Copy/Paste	Menu commands	Share data through the clipboard	One way	No	Both	250
Import/Export between ExtendSim and external files	Data Import Export block (Value library)	Excel, ADO compliant databases, XML and text files, FTP	One way	No	Both	224
Read/Write data, usually within ExtendSim	Read, Write, and Query Equation blocks (Value and Item libraries)	ExtendSim databases and global arrays, block data tables, text files, Excel, etc.	One way	Possible	Both	226
Directly link dialog components to ExtendSim database	Dynamic Data Link (DDL) menu commands and dialog settings	ExtendSim databases, global arrays, and dynamic arrays	Two way	Yes	ExtendSim only	229

As indicated in the table above, each method has its own specifications and properties:

- **Data Direction**. The flow of data can be one-directional or two-directional. With two-directional data flow, a change in data at the source affects data at the target and a change at the

target affects data at the source. One-directional data changes at the target do not affect the source.

- **Dynamic Live Links**. A communication method that has live links sends a data update message along with the data. These *link alerts* mean that as soon as the data at the source changes, the target is dynamically notified and can take the appropriate actions to react to the new value. See "Link alerts" on page 247.
- **ExtendSim or External Applications.** Dynamic data linking is only used to access data in internal ExtendSim data structures such as ExtendSim databases and global arrays. The other user interface methods can be used either internally or externally (for instance, copy/paste from one ExtendSim dialog to another or copy/paste from ExtendSim to Excel).

☞ Most of the above methods are supported by an underlying industry-standard technology, as discussed in "IPC protocols and technologies for communication" on page 237.

Copy/Paste

See "Copy/Paste and Duplicate commands" on page 250.

Importing and exporting data

Simulation models require input data and generate output data. This data is usually derived from one or more sources that reside in applications such as Excel, relational databases, SAP, and from files residing on remote computers. The management of this data is an integral component of the simulation modeling process.

As the complexity and fidelity levels of simulation models increase, the amount of data that has to be managed increases as does the time devoted to model configuration and data management. As the volume and complexity of data being managed increases, it becomes more desirable to automate as much of the data management process as possible.

Automation reduces data input errors, scenario setup time, and output data management time. Consequently, it is highly desirable to automate the direct exchange of data between simulation models and data management applications.

User interfaces

In addition to options that are available with programming, ExtendSim provides the following UI methods to support the import or export of data:

1) The Data Import Export block (Value library) is the most common method for interfacing between ExtendSim and other applications and files. It enables ExtendSim databases and global arrays to directly exchange data with external files, typically at the beginning or end of a simulation run. See "Data Import Export block" on page 224.
2) Read and Write blocks (Item and Value libraries) are most commonly used to exchange data within ExtendSim and during a simulation run. The Read and Write blocks in the Value library can be set to transfer data from text files, a local table, ExtendSim database tables, global arrays, or Excel. See "Read, Write, and Query Equation blocks" on page 226.
3) Database menu commands that, among other things, allow you to export an entire ExtendSim database as a text file. And the ExtendSim DB Add-In for Excel, which can be used to transfer data between an ExtendSim database and Microsoft Excel. The ExtendSim database and the Add-In are discussed in the separate document *ExtendSim Database Tutorial and Reference*.

In some cases the data is imported/exported directly within ExtendSim or directly between ExtendSim and the other application. In other cases it is imported/exported in the form of text or ASCII files, a standard communication technology discussed in “Text files” on page 240.

Data Import Export block

The Data Import Export block (Value library) provides modelers with a mechanism for controlling the automated exchange of data between ExtendSim models and external data sources and targets. It performs two primary functions:

1) Transfers data from specified data sources to specified data targets where either the source or the target can be an external application or file
2) Controls the timing of when data exchanges occur—at the beginning, during, or after the simulation run

☞ The main advantage of using the Data Import Export block is that information is exchanged as one piece rather than step by step during the simulation run. Thus it is typically used to load the model with data before the simulation run and export results at the end of the run.

Compared to copy/paste and read/write

Importing and exporting provides additional capabilities compared to copy/paste or read/write:

- Some types of importing/exporting can be automated; copy/paste always requires user interaction.
- As opposed to copy/paste, you can import/export while the simulation is running.
- Import/export might be faster than reading and writing data (discussed on page 226) because the data gets copied as one piece (a local copy). Whether the exchange happens before, during, or after a simulation run, all the data is made available to the target at the same time, rather than being transferred piece by piece during the simulation run.
- The information can be accessed even if it originated on a different computer or an external device.

For example, simulations often run faster because the data is exchanged as one piece—all the data is made available to the target at the same time rather than transferring piece by piece during the simulation run. This is true whether the data is imported/exported before, during, or after a simulation run.

Modes and interfaces

The Data Import Export block can be configured to operate in either *import* or *export* mode and interfaces between several internal and external data points. Importing/exporting is one-directional—changing data at the target does not affect data at the source and vice versa. The data can reside locally, be remotely accessed over a network, or accessed via the Internet using FTP protocols.

Internal sources and targets

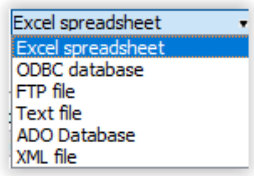
Internally, you can use the Data Import Export block to share data from:

- ExtendSim database tables
- ExtendSim global arrays

External sources and targets

The supported external interface types are:

- A Microsoft Excel workbook (specified by rows and columns, named ranges, or by workbook table)
- External databases that are ODBC or ADO compliant (Microsoft Access, Oracle, MySQL, SQLServer; Windows only).



☞ The Oracle and SQLServer databases are also useful as automated bridges between ExtendSim databases and ERP programs such as SAP.

⚠ If you use the Data Import Export block to exchange data with an ODBC or ADO database such as Access, the database and its drivers must be 64-bit compatible.

- An FTP (file transfer protocol) data source type, where the file resides on a remote machine and access to it requires a user name and password.
- Text files such as those generated by Notepad or Excel or any other applications where the modeler has read access to a file that resides on the local computer or a local area network (LAN).
- An XML (extensible markup language) file, a flexible way to share structured data with external databases, the World Wide Web, intranets, and elsewhere.

☞ The Data Import Export block allows you to export an ExtendSim database table or even the entire database to an ADO-compliant database, such as Access.

Left: import to an ExtendSim database. Right: export from an ExtendSim database

Configuring the block for external applications and files

Each of the allowed applications and file types listed above have their own set of requirements for a successful import or export and the Data Import Export block changes settings and fields accordingly.

For example, when exporting from an ExtendSim database to an Access database file, the block presents a field for the name of the Access file. However, depending on the external application or file type selected, the fields for specifying it as a source or target will vary. In addition, for server-based applications such as SQL Server and Oracle, you will need access to the server and/or a user name and password.

☞ For ADO compliant databases, the path to the external source or target file varies depending on which database is selected. For Access it is specified by a file name. For MySQL and SQL Server the source/target is specified using a server name and a database name. For Oracle, a user name and database name must be specified.

If information other than a file name and location is required, the block's Options tab will be enabled. The Options tab is helpful for configuring tables, entering a data source name (DSN) data structure, and so forth. As is true when files are selected and configured on the Import Export tab, the fields on the Options tab will vary depending on the external application or file. If there is nothing additional to configure, the Options tab will be disabled.

☞ If the block will exchange data with an ExtendSim database's child field. see the information about string data types, PRI, and PRV in the *ExtendSim Database Tutorial and Reference.*

Import and export timing

There are three options for when the external source or target exchanges data with the ExtendSim database or global array:

- Manually whenever the Import Now or Export Now button is clicked, even if the simulation is not running.
- During the simulation run, when the block's input connector gets a value greater than 0 (zero).
- During a specified phase of the simulation run:
  - If the *Import at beginning of simulation* checkbox is checked, data is automatically imported at pre-CheckData. The Options tab offers alternative import points.
  - If the *Export at end of simulation* checkbox is selected, data is automatically exported at EndSim.

For more information, see

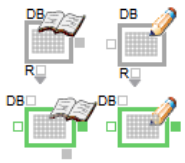
- The *ExtendSim Database Tutorial and Reference*, which has a tutorial for the Data Import Export block
- The Dynamic Resource Qualification model located at ExtendSim/Examples/Discrete Event/Resources and Shifts/Advanced Resources

Reading and writing data

In contrast to the importing/exporting described on page 224, reading and writing is commonly used to exchange data between ExtendSim model components and piece-by-piece as required during the run. For instance, models may need to write and read data while the simulation is running since it is common that the data that is written is in turn read and used in another part of the model.

⚠ A potential disadvantage is that reading and writing the data each time the block is calculated can impact simulation speed. If the data to be read or written will not change over the course of the simulation, you should import/export the data as a local copy in the block before or after the run. If the data is not going to change at all, consider using another method, like a one-time copy/paste or import/export.

Read, Write, and Query Equation blocks



The Read and Write blocks (Value library) shown at the top on the right, and the Read(I) and Write(I) blocks (Item library) shown on the bottom, make it easy to exchange data between ExtendSim model components. The four blocks are discussed below.

The Query Equation and Query Equation(I) blocks can use an equation to search an ExtendSim database table for the highest ranking record available during the run. Since they are only used with the ExtendSim database, and are only available in certain ExtendSim products, they are discussed in the *ExtendSim Database Tutorial and Reference*.

Read and write interfaces

The blocks that read and write are typically set to transfer data during a simulation run.

☞ The Read and Write blocks in the Value library have multiple data sources and targets; the Item library blocks only exchange data between the model and ExtendSim databases

Reading

The two Read blocks look up information found at a specified data source:

- For the Read block (Value library) the data source could be an ExtendSim database or global array, an Excel workbook or local table, or a text file
- For the Read(I) block (Item library) the source is an ExtendSim database.

Once they've read in the data, the blocks report that information through their value output connectors. The Read(I) block can also store the information as an attribute on items passing through, so the information can be used by other blocks in the model.

Writing

The two Write blocks send information to a specified data target or destination:

- For the Write block (Value library) the data target could be an ExtendSim database or global array, an Excel workbook or local table, or a text file
- For the Write(I) block (Item library) the data target is an ExtendSim database.

The information to be sent can come from their value input connectors. The Write(I) block can also send the attribute information that has been found on items passing through.

Addressing the data structure

By definition Read and Write blocks (Value library) are required to be interfaced to some type of data structure as their source or target, respectively. The pieces of information that are required to fully and properly specify where the information is to be read from or written to (the "address") depends on the type of data structure chosen, as shown in the following table.

	ExtendSim Database	Global Array	Microsoft Excel	Text File
File name			X	X
Database name	X			
Table, array, or sheet	X	X	X	
Field or column	X	X	X	X
Record or row	X	X	X	X

For instance, the fully specified address for accessing an Excel file would include the file name, sheet, column, and row.

Interface methods

Once you have specified a data structure type to communicate with and determined what the fully qualified address is, the next step is to determine where the data structure address is entered:

- In the block's dialog or through value input connector for the Read and Write blocks (Value library)
- In the block's dialog, through value input connector, or by using attributes on items that pass through the Read(I) and Write(I) blocks (Item library)

Triggers

The following table indicates the mechanism that will trigger when the data is sent or received.

	Read	Write	Read(I)	Write(I)
Connector message	X	X		
Beginning of run only	X	X		
End of run only	X	X		
Item arrival at block			X	X

Link alerts

When read/write blocks are dynamically linked to an ExtendSim database or global array, ExtendSim can send live update messages, called *Link Alerts*, between them. For example, if you select the option, and write to a location in an ExtendSim database, ExtendSim will send update messages to all the Read blocks in the model linked to that location. When the Read block receives the alert message, it will act on the fact that the data value changed. For more information, go to page 247.

Compared to dynamic data linking (DDL)

Dynamic data linking (DDL) is discussed in "Dynamic linking to internal data structures" on page 229, and a tutorial is provided in the separate document titled *ExtendSim Database Tutorial and Reference*. DDL consists of dynamically linking data from a block's dialog parameter or data table to a database cell or table.

The read/write blocks provide a lot more flexibility compared to DDL. For example:

- The read/write blocks allow more control over which database values get used, and where and when they are used.
- Database usage is clarified visually. Instead of a blue outline around a parameter field inside a block's dialog, it is more obvious when a database value is coming from a Read block.
- Dynamic linking is limiting because it is fixed point-to-point. While you can change the value at the data source, you can't dynamically change which database cell is accessed during a simulation run or in a series of runs.
- Rather than setting up a dynamic link for each parameter or data table, data can be more easily accessed at several places in the model.
- The Read and Write blocks support *name tracking* and model components are alerted if there is a change to an ExtendSim database component. DDL does not support this feature.
- In some blocks there is no dialog parameter or data table you can directly link to.
- It is often easier to perform mathematical calculations on the database values.
- It is more convenient for hierarchical blocks, allowing access to a different record for each instance of the hierarchical block in a model.

Where to get more information

- See the *ExtendSim Database Tutorial and Reference*
- See the Monte Carlo model located at ExtendSim/Examples/Continuous/Standard Block Models

- See the DB Job Shop model located at ExtendSim/Examples/Discrete Event/Routing. Note: this model uses blocks from the Item library and cannot be opened using the ExtendSim CP product.

Dynamic linking to internal data structures

ExtendSim supports a comprehensive, proprietary internal data linking method called *dynamic data linking (DDL)*. This application-level protocol tracks which dialog items (parameters and data tables, including their clones) are linked to which internal data structures (ExtendSim database tables, global arrays, and dynamic arrays).

⚠ For a tutorial on how to use DDL, see the *ExtendSim Database Tutorial and Reference*.

Link From	To ExtendSim Database (see page 232)	To Global Array (see page 233)	To Dynamic Array (see Technical Reference)
Parameter or its clone	Use the Model > Create/Edit Dynamic Link command	Use the Model > Create/Edit Dynamic Link command	Requires programming
Data table or its clone	Click the table's Link button	Click the table's Link button	Requires programming

Dynamic data links are extremely powerful because they are live and bidirectional. This means that the value of a linked dialog item can change immediately when the value of the data source changes, and vice versa.

⚠ There is overhead associated with the process of accurately updating a linked dialog item with a data structure. If you overload a model with links that frequently change, the model's performance can suffer. Consider the following suggestions: 1) Just link the data that is important to a model and not link every piece of data to every possible source. 2) Import values into a Read block (Value library) for use in the model, so they don't have to be continuously updated. 3) Send or receive link alert messages only at the start or end of the simulation as discussed in "Link dialog checkboxes" on page 231.

Interfaces

DDL links parameters and data tables to an ExtendSim database table, global array, or dynamic array.

- Parameters. Unless it is already linked for sensitivity analysis (see "Sensitivity analysis" on page 138), a dialog parameter can be dynamically linked to a specific cell of an internal data structure—ExtendSim database, global array, or dynamic array. For the database or global array, the linking is done through the menu command Model > Create/Edit Dynamic Link or through the right-click menu. For dynamic arrays, the linking is done through programming.

☞ Parameter fields that are dynamically linked to an internal data structure are outlined in light blue. (Parameter fields are outlined in green for active sensitivity analysis and red for inactive sensitivity analysis.)

- Data tables. Many ExtendSim data tables have a Link button in their lower left corner. If the Link button is not present for a table, the table does not support dynamic linking.

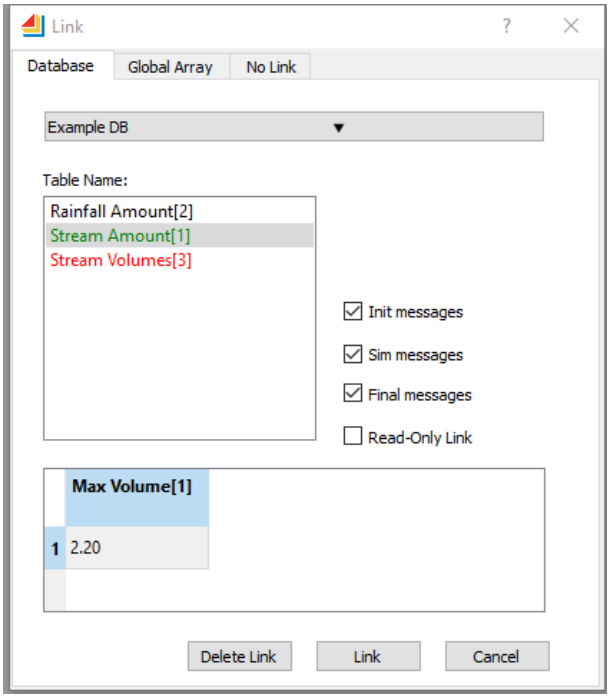
- If a data table is already linked, its upper left corner will be blue. If the linked source is a database (*DB*) or global array (*GA*), those initials will be displayed in that corner. If the link is to a dynamic array there will be no initials.
- If a data table is linked to a database, double-clicking the DB initials in the upper left corner will open the linked database table's viewer. (Double-clicking the GA initials does not open the global array, because global arrays don't have a user interface.)
- Since DDL can be a two-way live link, changing the value of one or more cells in the block's data table would immediately change the values in the corresponding global array or database table's cell, and vice versa. If you do not want that behavior, select the Read-Only option in the Link dialog.

☞ A data table linked to an internal data structure will have its upper left corner in light blue rather than the gray of the table header.

⚠ Because ExtendSim data tables and global arrays are zero-based, but databases are one-based, a data table's cell at row 0, column 0 (the top, leftmost cell) is linked to the database table cell at field 1, record 1 (the top, leftmost cell) or the global array cell at row 0, column 0 (also the top leftmost cell). For detailed information, see "Data source indexing and organization" on page 247.

Link dialog

This dialog opens if you click a data table's Link button or (by using the menu command or right-click) select Create/Edit Dynamic Link for a dialog parameter.



Use the Link dialog to create a dynamic link between a block's parameter or data table and a database or global array. It can also be used to change link settings and even change between data structures.

⚠ A parameter or data table can only link to one data structure. If you change any of the components, such as linking to a different database table, the previous linkage will be automatically deleted.

As shown here, the Link dialog opens with two tabs: Database and Global Array.

- If the data table or parameter is already linked to a database or global array, the dialog will display information about the existing link. For example: *Link: DB Link - Reservoir DB:Stream Amount:Max Volume*. You can change or delete the link.

- If the data table or parameter is already linked to a dynamic array, the Link dialog will display *Link: No user created link*. (As discussed on page 236, a dynamic array is an internal data structure only accessible through programming.) You can create a link to a database table or global array.
- If the data table or dialog parameter is not linked to anything, the dialog will display *Link: No link.* You can create a link to a database table or global array.

Link dialog checkboxes

Outside of a simulation run, blocks receive link alert messages whenever the dynamically linked data changes at the source. To maximize simulation speed you may not want these update messages sent during certain phases of a run.

When the Database Table or Global Array option is selected in the Link dialog, it displays four checkboxes that allow you to control the link's update behavior:

- Init messages
- Sim messages
- Final messages
- Read-Only link

The first three link options (Init messages, Sim messages, and Final messages) control whether or not blocks receive link alert update messages when linked values change during specific points in a simulation run. For example, the "Init messages" option determines whether a block receives a message if a linked value changes during the InitSim message.

- These three options default to on. Turning an option off prevents ExtendSim from sending data update messages to the linked blocks if the data source is updated during that phase of the simulation. For example, if you know the dynamic link only updates during the FinalCalc message, it will reduce the messaging and speed up the simulation to uncheck the Init messages and Sim messages options.
- When the Init messages and Final messages options are checked, linked blocks will receive an additional message during InitSim and/or FinalCalc, causing an update of the linked dialog items during those phases.

The fourth checkbox is the Read-Only link option, which is off by default. If checked, the linked data can only be changed from the internal data structure (database or global array). In that case, you will not be allowed to edit the parameter or the data table in the block's dialog and the live link is only one-way (from the database or global array to the data table or parameter).


Use these options carefully, since they can dramatically change linking behavior. You would probably only want to uncheck a message option (Init messages, Sim messages, or Final messages) if you are sure that the source data won't change during that message.

Delete Link button

The Link dialog is also used to delete a link.

Deleting a linked parameter

To delete a link between a dialog parameter and a database:

- Right-click in the linked parameter field and select **Create/Edit Dynamic Link**. (Note: don't select or left-click the parameter field first.)

- In the Link dialog that appears, click the **Delete Link** button.

Deleting a linked data table
To delete the link to a model's data table:

- Click the Link button at the bottom left of the data table
- In the Link dialog that appears, click the **Delete Link** button.

☞ ExtendSim knows which link is being deleted, so you don't need to select the table the parameter or data table is linked to before deleting the link.

Finding linked dialog items

The Model > Open Dynamic Linked Blocks command opens the Find Links dialog. This is useful for examining and locating linked dialog items and the options in the dialog allow you to selectively choose which types of links you want to open. The command is discussed fully at "Open Dynamic Linked Blocks..." on page 555.

Internal data storage and management methods

ExtendSim provides several internal structures for storing data for use in a model.

Structure	Definition	Discussed on	Access By
ExtendSim databases	Internal relational repositories for model data. Uses to store, manage, and report model data. Allows you to separate the data from the model for better project and experiment management.	See the "ExtendSim Database Tutorial and Reference"	Dynamic Data Linking (DDL)—see page 229 Data accessing blocks such as Data Import Export or a read/write block—see page 245
Global arrays	A two-dimensional (row and column) array of data. Used to store information that can be accessed by a row and column index, or to exchange data with external sources. Less flexible than an ExtendSim database.	Page 233	Dynamic Data Linking (DDL)—see page 229 Data accessing blocks such as Data Import Export or a read/write block—see page 245
Dynamic arrays	A multi-dimensional internal data structure that is only accessible through programming.	Page 236	Programming in ModL—see the Technical Reference
Linked lists	Queue-like structures of multiple types that maintain internal pointers between different elements. This allows the construction, manipulation, and rapid sorting of complex lists of data.	Page 236	Programming in ModL—see the Technical Reference

ExtendSim databases for internal data storage

⚠ For complete information, see the *ExtendSim Database Tutorial and Reference*, which is a separate document.

Overview

A database is a centralized repository for data. Relational databases organize data into one or more tables that have a logical connection, or relationship, to each other. Database tables hold data in cells, each of which is the intersection of a field and a record.

The ExtendSim graphical simulation database (GSDB) feature allows you to create internal relational databases for storing, managing, and reporting model data. ExtendSim databases also provide a convenient interface between models and external applications, such as spreadsheets and external databases.

Link alerts

Live update messages, called *Link Alerts*, can be automatically sent between ExtendSim blocks and the ExtendSim databases or global arrays they are dynamically linked to. For example, if you select the option, and write to a location in an ExtendSim database, ExtendSim will send update messages to all the Read blocks in the model linked to that location. When the Read block receives the alert message, it will act on the fact that the data value changed. For more information about link alerts, go to page 247.

Excel Add-In for ExtendSim databases

The DB Add-In for Excel is discussed fully in the *ExtendSim Database Tutorial and Reference*, which is a separate document.

The ExtendSim DB Add-In is a tool for externalizing the modification and construction of ExtendSim databases. Use the Add-In to:

- Import an ExtendSim database text file into Excel for editing, then export it back to ExtendSim.
- Create new, fully structured, ExtendSim database files within Excel, then export those files for use within ExtendSim.
- Let Excel be the master for documenting the data and database structure. For instance, charts and data tables can be added in Excel to help explain or analyze inputs to the database. (Any tables that do not start on row 20 are not considered ExtendSim database tables and will be discarded when the file is exported to ExtendSim.)

The data and structure of an ExtendSim database can thus be edited or created in Excel, separate from, and even in the absence of, the ExtendSim application. For example, analysts can structure or edit database files for use in models without knowing anything about ExtendSim.

Global arrays

A global array is a two-dimensional (row and column) array of data that is accessible by any block in a model. Like ExtendSim databases, global arrays are user-accessible structures for internally storing and managing data. Global arrays are used to share information between blocks when a direct connection is either inconvenient or impossible or to store information that can be accessed by a row and column index.

In most situations it would be best to use an ExtendSim database rather than a global array.

Compared to ExtendSim databases

Global arrays provide some of the same modeling advantages that ExtendSim databases do, but they differ from databases in some important aspects. Compared to databases, global arrays have a simpler data structure and take less time to access data. However, global arrays aren't as flexible as ExtendSim databases, which allow mixed data types, randomness, and parent/child

relationships. It is also easier to add or remove fields in a database than columns in a global array, and creating a global array requires adding a block to the model.

While it is more common to use a database for managing and exchanging data, global arrays are handy in situations where:

- You just want a simple array structure of a single data type. For instance, to create an integer array with 10 columns and 20 rows of data for use in a model.
- You program blocks that can use a simpler data structure than a database requires, and you don't need or want the data storage to be visible to the block user.

☞ Unless you are using an equation-based block or custom blocks that create global arrays, a Data Source Create block must be in the model to initially create the global array. Since the global array is saved with the model, the block can be removed after the array has been created.

Creating a global array

There are three ways to create a global array:

1) Use the Data Source Create block (Value library) as illustrated later in this chapter.

2) Through global array functions in an equation-based block from the Value or Item library. These blocks are discussed in "Equation-based blocks" on page 189.

3) Programing with ModL code. For more information, see the Technical Reference.

You can name a global array anything you want as long as the name is fewer than 32 characters and it does not begin with the underscore character (_). Global array names are not case sensitive; spaces between characters are allowed.

A global array can only be of one data type - either integer or real (if the data type is selected through the user interface), or integer, real, string, or pointertype (if the data type is selected through ModL code.) Each model can have one or more global arrays associated with it and each global array can have multiple rows and columns. Global arrays are stored and saved with the model, even if the Data Source Create block that created the array is deleted.


Following the standard conventions, ExtendSim databases are organized by field and record (similar to being organized by column and row.) However, arrays are organized by row and column. This is important to consider when transferring data between databases and global arrays. See "Communicating with external devices" on page 248 for more information.

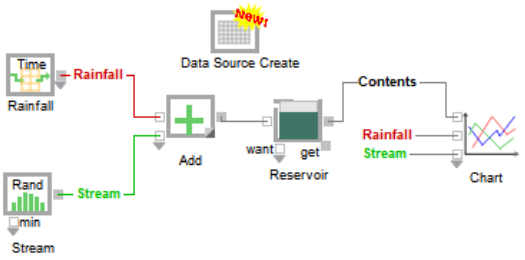
How to create and use a global array

The example that follows uses the Data Source Create block to create a global array for the Reservoir 1 model.

Using an example model

- Open the model Reservoir 1 from the ExtendSim/Examples/Tutorials folder.
- So that you don't overwrite the original file, give the command File > Save Model As and save the model as **ReservoirGA**.

- Place a Data Source Create block (Value library) at any convenient location on the model worksheet.



Creating the global array

- In the dialog of the Data Source Create block:
  - Choose **Type: Global array**.
  - In the "Create a new array" frame, click the **Integer Array** button, indicating that you want the values to be integer.
  - In the three dialogs that appear, name the array **My Array**, give it **12** rows, and give it **2** columns. Click OK after each dialog entry.

Notice that Data Source Viewer for the selected array now displays 12 rows and 2 columns. The table's upper left corner is light blue, indicating that it is a data source, and it has the initials "GA", indicating that the data source is a global array.

Data Source Viewer

GA	0	1
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0

Populating the array with data

Global arrays can be used as inputs to a model or to store model results.

- A common method of populating an array with data for use in a model is to enter values or copy data directly into the Data Source Create block's Data Source Viewer table when the array is created. You can also use the Data Import Export block (Value library) to import data to the array from a spreadsheet or external database, the Internet, or a text file.
- To store model results, it is most common to use a Write block (Value library) to send data to a global array.

☞ The Data Init block (Value library) can be used to initialize a global array table, column, row, or cell; it is described on page 246. The Data Specs block (Value library) gathers information about the global array and puts it into a report; it is discussed on page 246.

Exchanging data with a global array

ExtendSim provides the following methods for models to interact with global arrays:

1) Through the user interface, using dynamic linking to establish live links between dialog items and global arrays. See "Interfaces" on page 229.
2) With data access blocks, such as the Read and Write blocks (Value library), to exchange data between a model and a global array. Data access blocks are discussed on page 245.
3) Using global array functions in an equation-based block from the Value or Item library to manage data. These blocks are discussed in "Equation-based blocks" on page 189.
4) Programming using ModL code. See the Technical Reference for more information.

Dynamic arrays

A dynamic array is multi-dimensional internal data structure that is only accessible through programming. A model user can resize a data table that is linked to a dynamic array and can link the data in the table to an ExtendSim database or global array. However, the dynamic array itself cannot be accessed outside of ModL code.

A data table linked to an internal data source will have its upper left corner in light blue rather than the gray of the table header. If the source is a database (*DB*) or global array (*GA*), those initials will be displayed in the corner. If the source is a dynamic array, the upper left corner will not have any initials.

Since dynamic arrays are created and managed through ModL code, see the Technical Reference for more information.

Linked lists

A linked list is an internal data structure that allows the construction and manipulation of complex lists of data. It can be accessed only through programming. Linked lists are queue-like structures of multiple types that maintain internal pointers between different elements. This enhances the sorting for complex structures by speeding the movement of the elements within the list.

ExtendSim implements linked lists:

- In the queue blocks (Item library)
- Extensively in the Rate library
- With the Linked List functions described in the Technical Reference.

Since this involves programming, linked lists are described in the Technical Reference.

Exchanging information with external applications (IPC)

Interprocess Communication (IPC) is the mechanism that facilitates communication and data sharing between applications. It enables one application to control another and multiple applications to share the same data without interfering with one another. IPC utilizes a set of programming interfaces, called *communication protocols* (see “IPC protocols and technologies for communication”, below), that allows programmers to coordinate activities among different applications.

The IPC communication methods allow ExtendSim to directly communicate with other applications and with ExtendSim models running on other computers, *while the simulation is running*. This allows ExtendSim to work on a wide variety of tasks jointly with external applications and files, even when they are located on a server or on the Web.

Client and Server

An application that communicates with another application can be categorized as either the Client or the Server. A Client application requests a service from some other application, connecting to and requesting data and services. The Server application responds to the Client’s request, perhaps even allowing the Client to take control.

Like many applications, ExtendSim can act as a Client and a Server depending on the circumstances. When it acts as a Client, ExtendSim connects to and requests data and services from a Server application. ExtendSim can also act as a Server application that is controlled by any Windows application that can be configured as an Automation controller.

IPC protocols and technologies for communication

Many technologies have been developed as industry standards to allow applications to share and access data both internally and with each other independent of programming language, operating system, and file type. ExtendSim supports most of the standard types of communication technologies including:

Protocol	Description	Use
COM (Component Object Model)	A set of object-oriented programming technologies and tools for IPC.	Communicate with, exchange data with, or control another application. Link data that, when chosen, automatically starts another application for data editing.
ExtendSim COM Object Model	The objects and methods that ExtendSim exports for Automation, including Execute, Poke, and Request.	Primary method for ExtendSim to be controlled and communicated with as a COM Automation Server.
DLLs (Dynamic-Link Libraries) on Windows	Libraries of routines or functions written in an external language, such as C++. These are the standard methods for linking between the ExtendSim internal programming language (ModL) and other languages and applications.	Call an external code segment from a block's code and perform operations. A block can pass data to the DLL, cause the DLL to calculate using that data, and get the results back from the DLL. For example, in ExtendSim, the Rate library blocks communicate with the LPSolve DLL.
COM DLL	A library developed in an external language, such as C++, to create or define Active X objects.	ExtendSim can load up and communicate with a COM DLL or be controlled by a COM DLL. For example, see how ADO is implemented in ExtendSim.
Automation	A subset of COM (previously OLE Automation). Enables applications to expose their unique features (properties, methods, and events) to scripting tools and other applications. The exposed objects are called ActiveX objects.	Access and manipulate (i.e. set properties of or call methods on) shared *automation objects* that are exported by other applications.
ADO (ActiveX Data Objects)	An Automation protocol for exchanging information with ADO-compliant databases.	Send an entire table of information at one time; high-speed data interchange. In ExtendSim ADO is implemented as a COM DLL for communication with ADO-compliant databases such as Microsoft Access.
FTP (File Transfer Protocol)	Protocol for sending files or data to and from Internet web pages.	Import from, or export to, files located on a server or on the web.

Protocol	Description	Use
Mail Slots	Functions for inter-computer communication.	An easy way for ExtendSim to send and receive short (<400 byte) messages. Also provides the ability to broadcast messages across all computers in a network domain.
ODBC (Open Database Connectivity)	A standard programming language middleware API for accessing database management systems (DBMS).	This legacy technology has been superseded by ADO.

In addition to the uses listed above, these technologies are supported for block development. This is described in the Technical Reference.

Communicating with external applications and files

The IPC technologies listed above allow applications to share and access data independent of programming language, operating system, and file type. You can incorporate IPC into ExtendSim models by:

1) Using ExtendSim blocks, as discussed on page 245
2) Using the ExtendSim I/O functions—ExtendSim as IPC Client, as discussed in the Technical Reference
3) Using an external language to access ExtendSim Automation protocols—ExtendSim as IPC Server, as discussed in the Technical Reference

Table of external communications

The table below shows how ExtendSim can communicate with other applications and with ExtendSim models running on other computers, even while the simulation is running. This allows ExtendSim to work on a wide variety of tasks jointly with external applications such as spreadsheets, external databases, word processors, statistics packages, and so forth.

ExtendSim supports IPC through:	Excel	ADO-Compliant Databases	Text Files	Other	Notes	See
Read and Write blocks (Value and Item libraries)	X		X		Interfaces with the ExtendSim database and global arrays	page 226
Data Import Export block (Value library)	X	X	X	XML files, FTP files	Interfaces with the ExtendSim database and global arrays	page 224
Command block (Value library)	X				Choose a command and when to send it	Value library

ExtendSim supports IPC through:	Excel	ADO-Compliant Databases	Text Files	Other	Notes	See
ExtendSim Database	X		X		Integrated within ExtendSim	"ExtendSim Database Tutorial and Reference".
ExtendSim DB Add-In for Excel	X					"ExtendSim Database Tutorial and Reference".
OLE Automation—ExtendSim as Server				External languages control ExtendSim		page 242
IPC functions	X			Other applications		Technical Reference
OLE/COM functions				Other applications, COM DLLs		page 242
DLL and Shared Libraries functions	X	X		Other applications	DLLs on Windows; Shared Libraries on Mac	page 245
ADO (user-defined functions)		X				page 243
Mail Slot functions				Other computers		page 245
Internet Access				Web, Internet	XML and text files, FTP	Technical Reference
Serial I/O functions				Equipment		page 248
ODBC functions		X	X			page 244

Spreadsheets

You may want to use spreadsheet data as the input for an ExtendSim model. Likewise, you may want to send output data to a spreadsheet for further analysis or presentation. For instance, if you have a spreadsheet that performs calculations on large amounts of data, you may want that spreadsheet to respond dynamically as you model real world conditions in ExtendSim.

ExtendSim facilitates communication with spreadsheets by:

- The Command, Data Import Export, Read, and Write blocks (Value library). These blocks can exchange data with spreadsheet applications, send one value to a specified cell in a spreadsheet and request the recalculated value from another cell, or send commands or a macro to a spreadsheet. See "Data Import Export block" on page 224, "Read, Write, and

Query Equation blocks" on page 226, and the tutorials in the separate document *ExtendSim Database Tutorial and Reference*.

- The ExtendSim DB Add-in, which allows an exported ExtendSim database text file to be imported into Excel. The data can then be stored in Excel, edited and exported to an external database, or edited and exported back to ExtendSim as a database text file. See the separate document *ExtendSim Database Tutorial and Reference*.
- OLE and IPC functions to facilitate ActiveX/COM/OLE communication with spreadsheets. For more information, see the Technical Reference.

External databases

In many situations, the historical data for a a model is stored in an external database. ExtendSim models can share information with external database applications:

- Use the Data Import Export block (Value library) to send data to or from ADO-compliant database applications (Oracle, MySQL, SQLServer, and Access). This block can import data to, or export data from, an ExtendSim database or global array. See "Data Import Export block" on page 224.
- By using Excel as an intermediary application between an ExtendSim database and an external database. In this case, the data is imported from the external database to Excel, where it is manipulated. The data is then exported from Excel to ExtendSim as a database text file using the ExtendSim DB Add-in. See the separate document *ExtendSim Database Tutorial and Reference*.
- Using ADO functions to initiate an SQL query or perform other functions. For more information, see the "ADO (Windows only)" on page 243 and the Technical Reference.

Text files

A text file (also known as an *ASCII* file) is a file of unformatted information. Text files contain written text with the styles removed and/or numerical data separated by some delimiter or separator (such as tabs or spaces).

You can create text files in another application such as a spreadsheet, database, or word processing program, then read those text files into ExtendSim. Or create text files in ExtendSim, then read them into the other programs.

Text files contain text, data, or both data and text. There are many uses for text files. For example:

- To supply internal data as the basis for analysis when using Sensitivity Analysis.
- To share data with an external database or spreadsheet program such as Microsoft Access or Excel.
- To output model results to external applications, such as spreadsheets or word processing programs, for presentation or analysis.
- Text files of information are generated for model Reports or Traces.
- Most files transmitted from minicomputers and mainframes are text files.

Text files can reside locally, be remotely accessed over a network, or accessed via the internet using FTP.

Creating and opening text files

Some ExtendSim blocks automatically create text files. For example, selected blocks automatically output information in the form of a text file when the Generate Trace command is chosen for a run. If you program custom blocks, you can include ModL functions to have the blocks create or read text files. In addition, there are two methods to directly open or create a text file in ExtendSim:

1) Use the File menu commands to create or open text files. This allows you to look at report files, modify data input files, and so on, without having to open another application.
2) Use the Create or Open button in the dialogs of the Read and Write blocks (Value library) when Text File is selected as the data type. This is useful if you want to create a file of data for use in a model. The Read and Write blocks are discussed on page 246

Working with text files

- The Read and Write blocks (Value library) provide an easy method for working with text files in a model. With these blocks text files can reside locally in the model (a "local copy") or be remotely accessed over a network. These blocks also give added capability with text files in that they allow you to select which column and row of data to access. They are discussed in "Read and Write blocks" on page 246.
- When building custom blocks, use the file I/O functions described in the Technical Reference to read and write text files.

How to create a text file

This example shows how to use menu commands to create a file with text and data:

- Give the command File > New Text File.
- In the Text File window, enter the information shown on the right into two tab-delimited columns. For example,
  - Type **January**
  - Click the tab key
  - Type **2.6**
  - Click the enter or return key to start a new row
  - Continue entering the information, clicking the tab key after each month to separate the information into two columns and clicking the enter key after each value to start a new row of information.

January	2.6
February	4.4
March	6.7
April	3.4
May	1.9
June	1.1
July	0.7
August	0.5
Septembe	0.4
October	0.7
November	2.6
December	3.4

- Give the command File > Save Text Document As and name the text file **My Text**.

☞ Because it is unformatted, the information in the text file will not be lined up as shown in the screen shot above. It is important that you do not try to line up the data by adding extra tabs as that will affect how the data is interpreted. There must be one, and only one, tab character separating the columns.

Delimiting text file data

Columns in text files can be separated by a tab, a space, or a comma. Carriage return characters are always used to separate rows. When creating text files for use as inputs to ExtendSim blocks or other applications, consider how the blocks or programs want the data presented. For example, some ExtendSim blocks can take data columns separated a tab character, a space, or with another character such as a comma. On the other hand, Sensitivity Analysis and many external applications will only read files where the columns are delimited by tab characters. In general, it is best to use tab characters.

Also, remember that programs expect only one delimiter (one tab, one space, or one comma) between columns. For example, don't put in extra tab characters in order to make columns line up visually; if you do, the program is sure to get confused.

Changing text file font and size

To change the font or font size used when viewing text files, give the command Edit > Options > Model tab and use the **Text file font** option.

ActiveX/COM/OLE (Windows only)

ActiveX is Microsoft's set of object-oriented programming technologies and tools. There are two main uses of the ActiveX technology in ExtendSim:

- ActiveX/OLE automation
- ActiveX/controls (COM)

☞ See examples at ExtendSim/Examples/Developer Tips/OLE Automation

ActiveX/OLE automation

ActiveX automation is the process of using the ExtendSim OLE functions, or the scripting environment of another application, to communicate with, exchange data with, or control another application. This is, for example, the technology used by the Read and Write blocks to communicate with Excel.

When you use the OLE functions inside ExtendSim to communicate with another application, ExtendSim is the Client in the automation communication and the other application is the Server. When the other application is using its scripting environment to communicate with ExtendSim, ExtendSim is the Server.

When you develop custom blocks, automation is probably the most powerful tool that you can use for interapplication communication. It does, however, come with a cost. Developing code to use automation to control another application requires expertise with OLE/COM as well as knowledge of the object model of the target application.

An object model is essentially a list of the methods and objects that an application supports with regard to ActiveX automation. ExtendSim has a simple yet powerful object model that is described in some detail in the Technical Reference. Other applications have more or less complex object models. Excel, one of the more common target applications, has a quite complex object model; it is quite complex to deal with. The Object Mapper block (Custom Blocks library) is a tool that can be useful in learning more about the object model of an application or ActiveX control.

ActiveX controls

The main ActiveX technology is *COM* (Component Object Model). COM is the framework for developing and supporting *ActiveX controls* and *component objects*.

☞ While the terms “component objects” and “ActiveX controls” are sometimes used interchangeably, an object is commonly considered to have a source application that it derives from and a control is usually defined as a stand-alone object that doesn’t necessarily have an application behind it. For purposes of this manual, the word “object” applies to either a component object or an ActiveX control.

A component object is an identifiable part of a larger program that provides a particular function or group of related functions; it is roughly equivalent to a Java applet. ActiveX controls can be created using one of several languages or development tools, or with scripting tools. In implementation, an ActiveX control is a dynamic link library (DLL) module. It runs in what is known as a container, an application program that uses COM program interfaces.

OLE (Object Linking and Embedding) is Microsoft’s framework for a compound document technology for combining text, sound, animations, controls and so forth into a document. Each object is an independent program component that can interact with the user and communicate with other objects.

Whereas OLE provides services for the compound document that users see on their display; COM provides the underlying services of interface negotiation, life cycle management (determining when an object can be removed from the system), licensing, and event services (putting an object into service as the result of an event that has happened to another object.)

The Read and Write blocks (Value library) use OLE/COM to communicate with other applications. ExtendSim also has many OLE/COM functions to facilitate ActiveX controls and COM objects.

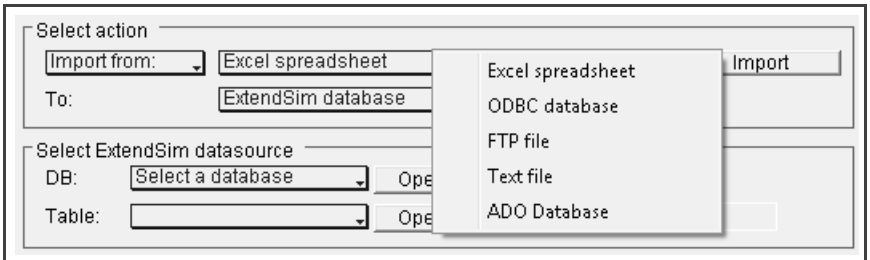
☞ ExtendSim has ModL functions to support its use as either an OLE automation client or server.

ADO (Windows only)

ActiveX Data Objects (ADO) is a method for exchanging information with various databases on Windows-based computers. ADO sends an entire table of information at one time, providing high-speed data interchange. The ExtendSim ADO interface allows you to communicate with Access, SQL Server, Oracle, and MySQL.

The interface is implemented as a Component Object Mode (COM) DLL. A set of ModL functions for accessing the DLL can be found in the ADO DBFunctions include file. These functions bundle a set of COM calls that perform specific ADO actions such as creating a table or transferring data. The Structured Query Language (SQL) is used to manipulate and select the data in the ADO database. SQL commands can be sent to the external database through the ADO interface.

The easiest way to communicate using ADO is to use the Data Import Export block (Value library) shown here. Otherwise, use the ADO functions from the ADO_DBFunctions include file, as discussed in the Technical Reference.



Options in Data Import Export block

☞ When using ADO, make sure that the ExtendSim table and external database table have compatible column formats and the same number of columns.

ODBC/SQL

Open Database Connectivity (ODBC) provides a standard application programming interface (API) method for accessing database data independent of programming language, operating system, and database system. The goal is to enable access to data from any application, regardless of which database management system (DBMS) is handling the data. It does this by inserting a middle layer between an application and the DBMS that translates data queries into commands understandable by the DBMS.

The ODBC API allows applications to access data in DBMS using SAG SQL (SAG = SQL Access Group; SQL - Structured Query Language) as the standard for requesting information from a database. SQL allows a single application to access different database management systems.

ODBC offers connectivity to a wide variety of data sources, including relational databases and non-relational data sources such as spreadsheets and text files.

⚠ Since Microsoft constantly updates and replaces these protocols, it is best practice to use the ExtendSim blocks (page 245) for ODBC communication. **Or better yet use ADO.**

FTP

File Transfer Protocol (FTP) is used to connect two computers over the Internet or an intranet for the purpose of transferring data and commands. In an FTP environment, one computer acts as a server and the other acts as a client.

As a client, ExtendSim initiates a connection to the server. Once connected, ExtendSim accesses a specified file on the server and either imports data from it or exports data to it.

ExtendSim implements FTP by:

- The Data Import Export block (Value library) can open a file on a remote computer and access the data for use in an ExtendSim model.
- Internet Access functions provide more flexibility than using the Data Import Export block. See the Technical Reference.

DLLs and Shared Libraries

Dynamic-Link Libraries (DLLs on Windows) and Shared Libraries (Mac OS) are segments of code written in a language other than ModL, the ExtendSim language. This standardized interface provides a method for linking between ModL and other languages. When and if a DLL or Shared Library file is needed, it is loaded into memory and run by ExtendSim.

DLLs and Shared Libraries can contain code, data, and resources. They are used to provide access to functions that are already written in another language or to solve problems that might be difficult or impossible to solve in ModL. Use a DLL or Shared Library to calculate some function, perform a task, access application programming interface (API) calls, or even control external hardware devices.

ExtendSim implements DLLs and Shared Libraries by:

- The Rate library uses the LPSolve DLL or Shared Library.
- ExtendSim functions allow you to call the external code segments from within a block's ModL code and perform operations. For example, a block can pass data to the DLL or Shared Library, cause the DLL or Shared Library to calculate using that data, and get the results back from the DLL or Shared Library. For more information, see the Technical Reference.

☞ For examples, see the folder ExtendSim/Examples/Developer Tips/DLLs.

Mailslots (Windows only)

The mailslots feature allows communication between two ExtendSim applications running on different computers on the same local area network (LAN). Since mail slots involve programming, they are described in the Technical Reference.

☞ For Mailslot send and receive examples, see the folder ExtendSim/Examples/Developer Tips.

Blocks for data management and exchange

Several ExtendSim blocks facilitate data storage and management. Some are used to establish and control dynamic links between ExtendSim and internal or external sources. Others are useful for importing or exporting data between ExtendSim and external applications.

In general there are several uses for data management and access blocks:

- Sharing information between blocks when a direct connection between them is inconvenient.
- Storing information which can be accessed by a row and column index.
- Accessing existing spreadsheets imported as a file.
- Accessing existing databases using ADO or ODBC/SQL.
- Accessing Internet-based data.

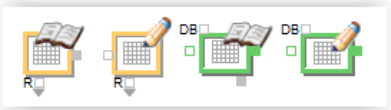
Most of the following blocks allow block-level access to data structures without having to program with the ModL functions. An additional group of blocks are designed to help block developers, serving as templates for custom-built data management blocks.

Data Import Export

The Data Import Export block (Value library) is used for the direct exchange of data between internal ExtendSim files (ExtendSim databases or global arrays) and external files (Excel, XML and text files, ADO and ODBC compatible databases, and files from the Internet via FTP.) See “Data Import Export block” on page 224 for more information about this block.

Read and Write blocks

The read/write blocks in the Value and Item libraries make it easy to get data into or send data out of model components.



See “Read, Write, and Query Equation blocks” on page 226 for more information about these blocks.

Query Equation and Query Equation(I) blocks

The Query Equation blocks are used to rank the records in an ExtendSim database table and intelligently select one record, based on a ranking rule. A user-defined equation in the block’s dialog is calculated once for each record in the table; the results are used to assign a ranking for each record. The record with the best ranking is the one that gets selected.

Query Equation blocks

☞ Since the Query Equation blocks are only used with the ExtendSim database, they are discussed in the *ExtendSim Database Tutorial and Reference.*

Data Source Create

The Data Source Create block (Value library) provides an easy method to create or modify global arrays or text files. You can also use this block to resize or delete a global array. For an example of how this block is used, see “How to create and use a global array” on page 234.

Data Init

The Data Init block (Value library) uses a data source to initialize a target with values. A popup menu in the block’s dialog allows you to choose as a target a database’s table, field, record, or cell or a global array’s column, row, or cell. The data source can be a value, a database’s table, field, record, or cell, or a global array’s column, row, or cell. Each row of the block’ table defines one initialization record for one target. Each initialization record can be set up to initialize the data it refers to at every run, the first run only, or to not initialize. Each cell in a given row of the table contains interface items that allow customization of that particular initialization record.

Data Specs

The Data Specs block (Value library) reports information about a selected ExtendSim database or global array. You can assign a name to the specification report, select its components through popup menus, initialize the outputs, and display the specification’s name and/or values on output connectors.

Command

When triggered by a value at its “send” input, the Command block (Value library) sends a command, such as an Excel macro, to a spreadsheet. You can select a command to send and choose when the command is sent.

Blocks for developers

The Utilities and ModL Tips libraries contain several blocks to assist block developers, such as the Object Mapper block in the Utilities library.

Link alerts

Live update messages, called *Link Alerts*, are sent between ExtendSim blocks and the ExtendSim databases or global arrays they are dynamically linked to. This means that as soon as the data at the source changes, an alert will be dynamically sent to all the blocks linked to that location, and those target blocks can take actions to react to the new value.

For example, if you write to a location in an ExtendSim database, ExtendSim will send update messages to all the Read blocks in the model linked to that location. When the Read block receives the alert message, it will act on the fact that the data value changed.

⚠ Writing to a cell in an Excel spreadsheet or to a text file will not have the same live link effect.

Link alerts can significantly slow simulation speed if enabled during the simulation run. To control when they are sent, see:

- “Link dialog” on page 230 for managing the alerts when block parameters and data tables are linked to an ExtendSim database or global array
- The check box *When receive database link alert msgs* in the Equation block (Value library)
- The *data source changes* check box in the Options tab of the Read block (Value library)

For the most part, link alerts will just make the blocks that have implemented them work as expected—when data changes, the blocks will respond appropriately. However, due to all the opportunities for response, link alerts are an especially powerful tool for those who use equation-based blocks or who create custom blocks. If you are developing a block that needs to respond to changes in data (ExtendSim database or global or dynamic arrays) or you are creating sophisticated equations that need this, or you need to use the Link Alert block, go to the Technical Reference to understand more about how Link Alerts work.

Data source indexing and organization

Communicating between various types of data sources has been greatly assisted by standardized technologies. However, each communication has its own conventions. If you program, it is important to keep in mind the following information:

- ExtendSim data tables have zero-based indexes and are organized by row and column. This is true whether the first row in the data table is labeled with a 0 or a 1.
- Spreadsheets are also organized by row and column, but they are one-based.
- Databases are one-based like spreadsheets, but are organized by fields and records (equivalent to columns and rows) rather than being organized by the spreadsheet convention of rows and columns.

For modelers, ExtendSim handles these differences internally. However, if you program you should be aware of the differences when transferring data from one type of source to another.

☞ See the Technical Reference for more information about indexing.

Communicating with external devices

In some situations, you may want to obtain data for a model directly from an external piece of equipment. For instance, when modeling a chemical process you might want to read the temperature of the actual process and compare it to simulated results.

If you create your own blocks with ModL code, there are two methods you can use to communicate with external devices such as scientific equipment and other hardware:

- Dynamic Link Libraries (DLLs) on Windows or Shared Libraries for Mac OS. DLLs and Shared Libraries are segments of code written in a language other than the ExtendSim ModL language, such as Visual Basic or C++. Their standardized interface provides a method for linking between other languages and ModL. They can also be used to perform complex calculations utilizing specialized hardware. For more information, see “DLLs and Shared Libraries” on page 245 and the Technical Reference.
- Serial port functions on Windows. To pass data through serial devices, use the serial port functions. These functions can read and write any data (including real-time data) to and from the computer’s serial ports. This is useful for transmitting and receiving data on a modem, for example. For more information, see the Technical Reference.

How To

Miscellaneous

Other important features
that didn't fit somewhere else

*"All truths are easy to understand once they are discovered;
the point is to discover them."*
— Galileo Galilei

This chapter covers some ExtendSim features and topics not covered in other chapters, including printing, scripting, the automated testing environment, and model sharing.

Printing

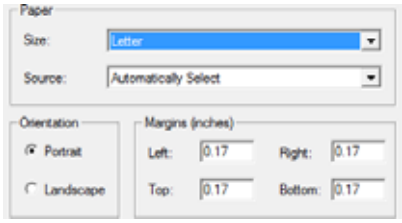
The File menu has three commands related to printing: Show Page Breaks, Page Setup, and Print.

Show Page Breaks

Causes ExtendSim to draw a set of page boundaries on the active window (worksheet, notebook, etc.) and place page numbers in the upper left hand corner of each page. The page boundaries show where page breaks will occur if you print the window. The command applies to each active window separately and the location of the page breaks is dependent on choices in the Page Setup command.

Page Setup

This command is for selecting the size and source of paper, the orientation for printing, and the margins.



The selected parameters are saved with the model.

Print command

The Print command is for printing to paper, a file (including PDF format), or a fax. The availability of options depends on which drivers are installed on your computer.

Clicking the Apply button in the Print dialog saves the settings you've selected without printing.

What gets printed and why

In general, whichever window or dialog is active gets printed when the File > Print command is given. However, that command doesn't apply to library windows. To print a library window, right-click the library window and choose Print Library Window.

The Model > Show Object IDs command causes unique identifying numbers to appear on all the objects in the model. For blocks, these numbers appear on their icons and also in the title bar of the block's dialog. Choosing this command before you print the model worksheet and dialog boxes will help to match dialogs of blocks with their icons in the model.

Copy/Paste and Duplicate commands

ExtendSim uses the Cut, Copy, Paste, and Duplicate commands in the Edit menu and toolbar just like other programs. An internal clipboard is used to pass information (text or graphics) within ExtendSim or between ExtendSim and other applications.

☞ Before you choose the Copy or Duplicate command, it is important to select the cursor that will allow you to copy the desired items. For example, choosing the All Objects cursor and frame-selecting a section of a model causes all items in the frame (blocks, drawing objects, clones, etc.) to be selected and available for copying. However, if instead you use the Block/Text cursor, only blocks and text will be selected.

Copying within ExtendSim

Blocks

If you copy and paste or duplicate blocks within ExtendSim, the clipboard also holds block parameters and connections. This allows you to duplicate portions of models, including the variables in the dialogs of the blocks, to another section of a model or to other ExtendSim models.

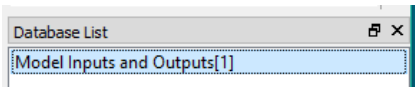
Drawing objects and text

Copy or duplicate drawing objects and stylized text from one section of a model to another section or from one model to another. To copy text, frame-select and copy the entire text box, rather than selecting the text within the box.

Data

While the Edit menu's Copy, Paste, and Duplicate commands work the same as in most applications, there are a few items you should note:

- When copying/pasting the *data* from data tables or database tables:
  - To copy all the data in a specific column of a data table or database table, click in the column's *header* so that the entire column is selected. Then give the command Edit > Copy.
  - To copy all the data from all the columns in a data table or database table, click in the *upper left corner* of the table so that the entire table is selected, then give the command Edit > Copy.
  - To paste data into a data table or database table, click in the upper left corner, in the upper left cell, or in a desired starting cell and give the command Edit > Paste. The table cells will be populated downward to the extent of the data in the clipboard or the number of rows in the data table.
- When copying/pasting an entire database table, select the table in the Schema View and give the Copy command.
- When copying/pasting an entire database, open the Database List from the Window menu; the list will attach to the library windows on the right side of the worksheet. Then select the database in the list, give the Copy command, and paste it into that Database List or into a Database List for another model.



☞ When copying data tables within ExtendSim, do not select *Edit > Copy With Headings*. If it is selected, the titles will be added to the data and will cause the original data to be displaced.

Copying from ExtendSim to other applications

Data

Data from parameter fields and data tables can be copied from ExtendSim and pasted into other applications such as word processors and spreadsheets. When you copy data, ExtendSim puts the data into the clipboard as unformatted text. See above for some additional information when copying data from data tables or database tables.

Pictures

Graphs, notebooks, parts of models, and dialogs are available as pictures to be copied from ExtendSim into other programs:

- Copy an ExtendSim graph to other programs for use in reports or presentations. To do this, simply click in the graph then choose Edit > Copy Graph Image.
- Notebooks and models contain various types of items such as drawing objects, cloned dialog items, and so forth. Before choosing the Copy command, it is important to select the tool from the toolbar that will allow you to copy the desired items. For instance, to copy an entire notebook, choose the All Objects tool from the toolbar. Then frame-select the entire Notebook (or choose Edit > Select All) and choose Copy.
- To copy all or parts of a block's dialog, choose the Clone Selection tool from the toolbar and select the dialog items you want to copy using any conventional method (frame-select, shift-select, etc.). Then choose Edit > Copy to Picture.

☞ Copying an ExtendSim picture into the clipboard changes it into a drawing object. To paste the contents of the clipboard into the receiving document, it is common to use that document's Edit > Paste Special command.

When it is not possible to copy parts of a model directly using the Copy command (such as for the entire plotter window), do the following:

- Windows: Use the Ctrl + Prnt Scrn key combination to save the entire screen as a bitmap. Or use a capture program to capture specific parts of the screen.
- Mac OS: Use the Command + Shift + 3 key combination to save the selection as a picture. Or use a capture program to capture specific parts of the screen.

Copying from other applications to ExtendSim

Data

You can copy data from other applications into an ExtendSim parameter field, data table, text file or database. However, if there is a lot of data it is usually better to import the data into a data table or into an ExtendSim database. For more information, see "Exchanging information with external applications (IPC)" on page 236.

Pictures and text

Pictures and text can be copied into ExtendSim in color or black-and-white to use on model worksheets, notebooks, hierarchical windows, or in the icon pane of a block's structure window. Create a picture with a painting or drawing program and copy the picture to the clipboard. Then paste the copied picture into an ExtendSim window with the Edit > Paste command.

☞ It doesn't matter what the original file format for the text or graphic was (JPEG, TIFF, GIF, etc.) Once it has been copied into the clipboard, ExtendSim treats it as a drawing object.

In the model worksheet and notebook windows, ExtendSim will paste the text or graphic object wherever you last clicked on the window. Pictures and text pasted from another program into ExtendSim become a drawing object which can be resized and repositioned using the Graphics cursor.

Pictures are treated like other drawing elements. For example, use the Graphics cursor to drag a picture around the window and resize it however you want. Or use the Alignment tools to

rotate or flip the picture, To delete a picture, select it with the Graphics cursor and choose Edit > Clear or press the Backspace or Delete key. Pictures that are copied, like objects created with ExtendSim drawing tools, always go behind ExtendSim blocks and text.

☞ Pictures can be proportionately resized. To do this, hold down the Shift key as you reshape the picture. It will resize with all dimensions proportional to the original.

Tooltips

For modelers, tooltips for block icons and connectors are turned on or off in the Edit > Options > Model tab. They help to quickly identify block information without having to open the block's dialog.

- Resting the cursor above a block causes a small window to appear containing the block's name, its local and global numbers, and (if *Include additional block information* is selected) the first sentence of the block's online help.
- Placing the cursor over an input or output connector gives information about it, such as its name and the number of items that have exited or the block's constraining rate.

For block developers, tooltips also exist to identify the variable name for block dialog items. These are turned on or off in the Edit > Options > Misc tab.

Scripting functions

ExtendSim scripting functions allow you to build, run, and control models indirectly, create custom wizards to automate modeling tasks, and develop self-modifying models. For example, the Smart Blocks, Auto Insert, and Bump Connect techniques, discussed in "Placing blocks on the model worksheet" on page 59, were developed using scripting functions. For more information, see the Technical Reference.

Referencing dialog variables

Sometimes a modeler must use a source block's dialog variable as a parameter in a target block. For instance, this is how to set model factors and get model responses with the Scenario Manager block, discussed in "How the Scenario Manager works" on page 143.

Referencing a parameter from an Activity to the Optimizer

Blocks that use the interface

The following blocks reference dialog variables from other blocks:

- Find and Replace (Utilities)
- Optimizer (Value)
- Scenario Manager (Value library)
- Statistics (Report library)
- Custom built blocks, if they use specific functions

Supporting the interface

All of the Value, Item, and Rate blocks have code that supports these interfaces. The feature is also available to block developers, as discussed in "Remote access to dialog variables" in the Technical Reference.

Methods for referencing dialog variables as factors

There are 3 ways to add a dialog variable to target blocks:

- Shift-click. Shift-click a parameter in the source block's dialog. In the window that appears, select which target block should get the parameter.
- Clone drop. Drag a clone of the source parameter from the source block's dialog and drop it onto the target block's icon. This method is especially helpful when one or more of the blocks is within a hierarchical block.
- Manual. Manually enter the dialog variable name into a field or table in the target block's dialog.

☞ These methods only work if one of the target blocks is in the model.

Changing parameters dynamically

Another name for the variable or constant number you input in a model's blocks is "parameter." Parameters that do not change during the entire simulation run are *static*. It is more likely that you would want to have a model parameter change *dynamically*.

One of the most powerful features in ExtendSim is the ease with which you can dynamically change the parameters of a block during the course of the simulation. This is usually done to represent the value of the parameter as a function of something else in the simulation model. You may also want to change a parameter to allow someone else to "game" (modify a value manually) to see how a change impacts the simulation.

Methods

There are a number of means for changing parameters dynamically:

- The most common method is to use a block's connector that corresponds to its dialog parameter field. Connect this to a block that calculates the desired value, such as a Random Number or Lookup Table block (Value library). This approach visually shows the connection between the parameter value and where the value comes from. In general, this is also computationally efficient since only the blocks connected to the parameter are recalculated.
- A second approach is to link the parameter field with a record in an ExtendSim database. To do this, right-click the parameter and select *create/edit dynamic link*. (You can also create this link programmatically by creating a custom block that links the parameters of other blocks with an ExtendSim database or global array.) Whenever the linked value in the database changes, the parameter will change as well. This method hides the relationship between the parameter and where it is calculated, however it is very useful for centralizing the location of certain parameters in the database or for broadcasting the same value at the same time to a number of blocks throughout the model.
- In a discrete event model, some blocks (such as the Activity or Equation (I) block) allow you to specify a property value for a parameter. Using an item's property is an easy way to make the parameter value a function of the item in the block.
- Sensitivity analysis allows you to change a parameter from one simulation run to the next. Sensitized parameters change values at the start of each new run. They can change randomly, based on a value in a file, or by a fixed increment. Changing parameters is useful for sensitivity analysis and Monte Carlo simulation. To utilize sensitized parameters, the number of

simulation runs must be greater than one. For a complete discussion, see "Sensitivity analysis" on page 138.

- Some blocks allow you to specify that a parameter is random. While this is not directly changing the parameter dynamically, this feature allows you to simulate a parameter that changes randomly each time the block is recalculated.
- There are other methods, such as calling the SetDialogVariable function in an equation block or custom block, but these are generally just used for specific situations.

Sharing model files

You may want to provide access to models you have built, without giving users the ability to change the model structure. ExtendSim provides two methods for preventing changes:

- Locking a model to prevent anything other than parameter changes when the model is run in the full version of ExtendSim.
- Using a RunTime version of ExtendSim to open and run models.

Protecting the model

Protecting a model is useful anytime you want to prevent changes to a model's layout, such as when giving the model to others who may accidentally move or delete blocks. The Model > Protect Model command prevents any modification of a model other than changing dialog values. Since this command hides many of the tools in the toolbar, the user also cannot add or change connection lines, graphics, and so forth.

This command does allow the user to save the model and any dialog value changes. However, even if they give the command File > Save Model As, the new model will also be password protected.

When this command is selected, a dialog opens for entering an optional password of 5 to 15 characters. To lock a model without using a password, leave the password fields blank and click OK. To unlock a model, simply choose the Protect Model command again and enter the password, if any.

☞ If a password is not used, any user can unlock the model by giving the Protect Model command.

The Protect Model dialog also has a *Protect hierarchical blocks* checkbox. Select this to prevent a user from opening a hierarchical block to see the underlying submodel worksheet.

Once a model has been locked with a password, the ONLY way to unlock it is to use the password. So remember the password and only lock a copy of a model, not the original.

Discrete Event Modeling

Items, Properties, and Values

Generating and removing items, and using item properties

As discussed in the Discrete Event Quick Start guide, items are what flow through a discrete event model, properties contain information about items, and values provide information about model conditions. This chapter discusses items and their properties and how information about them is reported as values. It will cover:

- Generating items randomly and by schedule
- The Create block's Start connector
- Attributes, priorities, quantities, and other item properties

☞ Most of the models illustrated in this chapter are located in the folder \Examples\Discrete Event\Items and Properties. For other models, location information is provided at the beginning of their respective discussions.

Blocks of interest

The following blocks will be the main focus of this chapter. The block's library and category appear in parentheses after the block name.

Item generating and removing

Create (Item > Routing)
Creates items randomly, by schedule, or infinitely. Can also be used to create values randomly or by schedule. Can initialize newly created items with properties, such as attributes or priorities.

Exit (Item > Routing)
Passes items out of the simulation. Reports the total number exited and the number that were taken from each input.

Item properties

Get (Item > Properties)
Displays the value of user-assigned and system level item properties: attributes, priorities, quantity, and item index.

Set (Item > Properties)
Attaches user-assigned properties (attribute, priority, and quantity) to items passing through.

Equation(I) (Item > Properties)
Can be used to set, modify, or check attributes on existing items. Calculates the equation when the item arrives.

Executive
Its Item Attributes tab is used for attribute management, such as renaming or deleting attributes or locating where they are used in a model. It is also where string/value equivalents are declared for string attributes.

Property-aware blocks

Item properties include attributes, priorities, and quantities. In addition to the blocks listed above, the following blocks in the Item library provide an interface for viewing, selecting, or modifying existing item properties or for adding new ones:

Activity	Queue Matching
Batch	Read(I)
Cost By Item	Resource Item
Equation(I)	Select Item Out
History	Shutdown
Information	Throw
Query Equation(I)	Unbatch
Queue	Workstation
Queue Equation	Write(I)

Item generation

Items for a model are usually generated using the Create block. While it can also generate values, the Create block can create items:

- Randomly. A random distribution causes items to be generated with a random or constant *interarrival time*. The distribution determines the time *between* item arrivals; a smaller interarrival time indicates that items will arrive more frequently. See the examples below.
- By schedule. Creating items by schedule causes an item to be generated at a specific *arrival time*. The schedule defines *when* the item will arrive and the time between arrivals is fixed. See the examples in "Generating items according to a schedule" on page 262.
- Infinitely. This provides an infinite supply of items that are available *on demand*. For instance, connecting a Create block with this behavior to a Gate block would provide an item to the Gate block each time it opens.

⚠ A Create block is set to *Create items infinitely* should *never* be connected to an infinite capacity queue, since generating an infinite supply of items would overwhelm the system.

☞ In a model, each item can represent an individual entity or a collection of individual entities. For instance, 50 items coming into a model could represent 50 people or it could represent 50 bus loads of people. How you characterize items is completely up to you.

Generating items at random intervals

The Create block can generate items that arrive to the model at random times. When set to "Create items randomly", the Create block outputs items at random intervals; the arguments of the distribution define the interarrival time.

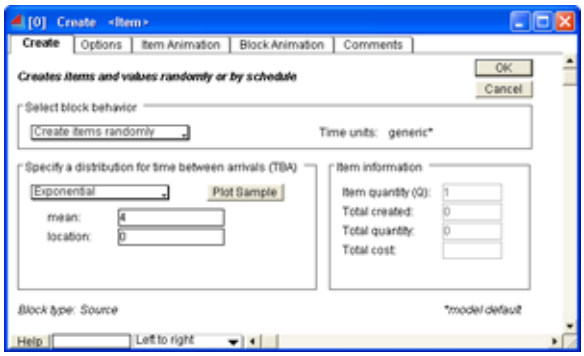
Example model

As you saw in the tutorial in the Discrete Event Quick Start guide, the Car Wash model is an example of using the Create block to generate items at random intervals.

☞ The Car Wash model is located in the \Examples\Tutorial\Discrete Event folder.

Choosing a distribution in the Create block

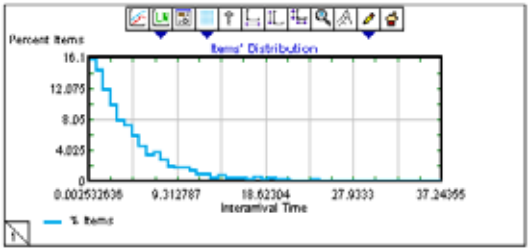
The dialog of the Create block contains several distribution choices in its "Specify a distribution" popup menu, as well as a table for entering data when an empirical distribution is selected. Each distribution is described in the Create block's Help; they are also discussed briefly in "Probability distributions" on page 198.



Exponential distribution selected; mean is 4

Choosing a distribution in the dialog of the Create block defines both the interval between item arrivals (the interarrival time) and the characteristics of the rate of arrival.

In the Car Wash model, for example, selecting an exponential distribution with a mean of 4 will cause one car to arrive approximately every 4 minutes for the duration of the simulation. This results in an interarrival time of 4. However, the *shape* of the exponential distribution dictates that it is more likely that the time between arrivals will be between 0 and 4 than between 4 and 8.



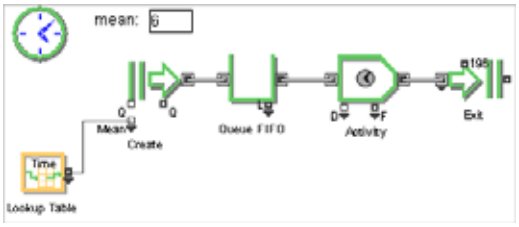
Distribution of outputs when mean is 4

Random intervals with dynamic parameters

You may want the parameters in a random distribution to change as a function of time or model status during the simulation run. The arguments for a given distribution can be controlled dynamically through the Create block's value input connectors.

Random Intervals model

In the Random Intervals model, time dependent arrival rates are modeled by connecting the Lookup Table block (Value library) to the Create block's value input connectors. A table in the Lookup Table block provides the mean values for an exponential distribution that has been set in the Create block. This causes the timing of item arrivals to be based on the time of day.



Random Intervals model

Specifying the dynamic parameters

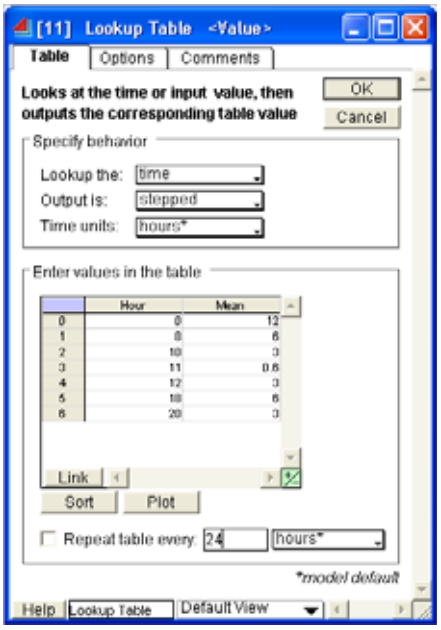
As seen in the dialog of the Create block, items arrive exponentially. Notice that the exponential distribution has a “Mean” parameter. Connecting the output of the Lookup Table block to the Mean input connector of the Create block causes the mean of the distribution to come from the Lookup Table during the simulation run, overriding any entry in the dialog. This dynamically changes the average interarrival time.

☞ The distribution determines the interarrival time. A smaller mean value indicates that there is less time between arrivals and items arrive more frequently.

In the Lookup Table’s dialog, the mean is smallest from hour 10 until hour 12, causing items to arrive more frequently during that period.

Choosing time units for the columns

When the block is set to “Lookup the: time”, the Lookup Table block looks at the current simulation time and outputs a corresponding value. Its table is used to determine the value that is output (by default the Output column; in this model, the Mean column) at a given simulation time (by default the Time column; in this model the Hour column). Its time units popup menu (in this model “hours”) represents the unit of time for the values in the Time/Hour column. However, that does not control what the output of the Output/Mean column represents.



Scheduling interarrival time

☞ The time unit for the Lookup Table block’s output column is determined by the block that receives its output values.

When a value passed to a block’s input connector is used to set a time parameter (as in this case), the value sent must be defined in the time unit that the receiving block expects. In this model, the values from the Mean column are sent to the Create block and are used to calculate the average time between arrivals. Since the Create block is using minutes as its local time unit, the values in the Mean column of the Lookup Table block also represent minutes. For instance, the value of 6 in the Mean column represents an average of 6 minutes between arrivals, even though the event time in the Hour column is in hours.

☞ Do not set the mean of a distribution to 0. The Create block will warn you if you make this modeling error.

Making sure the arrival occurs when expected

To avoid unexpected results, it is important to understand what happens in the Create block when you vary the mean of the arrival intervals over time. The block’s default behavior is to generate an arrival time, called “nextTime”, for the next item based on the current input parameters. When simulation time reaches nextTime, the Create block releases an item and generates a new nextTime based on the current values of the input parameters. For the period of time between releasing items, the Create block will not react to changes in the input parameters. If

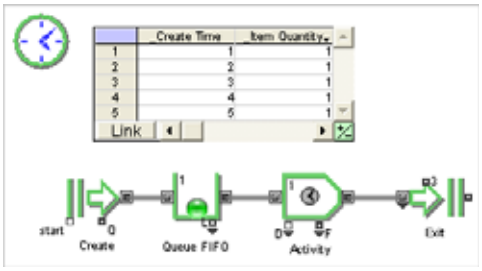
the inputs change drastically, this can cause unexpected results as discussed in “Cycle timing” on page 412.

Generating items according to a schedule

Scheduling item arrivals can provide more flexibility and precision than having them generated randomly. Setting the behavior of the Create block to “Create items by schedule” allows item arrivals to occur at fixed intervals that are specified in a table.

Scheduled Intervals model

Assume you want five items generated, one per minute, but that the simulation takes ten minutes to run. Setting the Create block to “Create items randomly” and selecting a Constant distribution would generate one item per minute, but there will be ten items. Generating items by schedule allows you to customize when items will be created and how many items the model will have.



Scheduled Intervals model

In the example model, the dialog of the Create block is set to “Create items by schedule.” The schedule of arrival times, which is cloned onto the model worksheet, indicates that one item will be generated at Create Time 1, another item at Create Time 2, and so forth up to time 5. With this schedule, the model has five arrival events.

The items proceed to a Queue to wait for processing by the Activity, which takes three minutes to process each item. Since the simulation runs for ten minutes, only three items exit; one item is left in the Queue and one item is still being processed in the Activity.

This method is especially useful when the time between item arrivals is known but not regular. For instance, the first item could be generated at Create Time 1, the second item at Create Time 3, and the third item at Create Time 3.5.

Notice that the Item Quantity column has a default value of 1 for each item generated. This means that each item generated represents 1 item. Item quantities are described fully in “Quantities” on page 272.

☞ An alternative method would be to use a Create block set to *Create items randomly*, and select a *Constant* distribution with a value of *1* in its dialog. Then in the block’s Options tab, select *Maximum items generated: 5*. This method is less flexible than the earlier method, since each item would have to have the same interval between arrivals.

The Start connector

When the Create block is set to *Create items by schedule* it has a *start* value input connector that can be used to control when the schedule is executed. The timing in the Create block depends on whether or not the start input is connected:

- If the start connector *is not* connected, the schedule's item creation times are synchronized with the simulation run's absolute time. For example, if the schedule's first create time is at time 2 and the simulation's start time begins at time 0, an item would be created when simulation time reaches 2. However, if the starting time entered in the Simulation Setup dialog is 4, the item scheduled at time 2 is never created.
- If the start connector *is* connected, such as to a Decision block (Value library), the schedule's item creation times are relative to when the connector is activated. For instance, assume simulation starting time is 0 and the first item is scheduled for creation at time 11. If the start connector gets activated at simulation time 5, then the first item will be created at time 16 (5 plus 11).

☞ Starts are activated whenever the connector receives a message with a True value (defined as greater than or equal to 0.5).

The Create block's Options tab provides choices for how the start connector should behave, as discussed below:

- *Follows schedule*. This is the default option and should be used for most situations. Once the start connector is activated, the entire schedule will be executed. Any new activation signals arriving before the current schedule has completed are ignored.
- *Generates one item per message*. This is an advanced choice for special situations. It is most often used when a custom-created block is connected to the start connector and you want an item to be instantaneously generated for every message. With this choice, the schedule is restricted to one row. Each time the start connector is activated, the row is executed.
- *Generates one item per event*. This is an advanced option for special situations. With this choice, the schedule is restricted to one row. Each time the start connector is activated, a zero-time event is scheduled. Once the Create block gets the zero-time event message, it will execute the schedule.

Item properties

A property is a quality or characteristic that stays with an item as it moves through the model. Some properties can be assigned to items by the model builder; others are automatically assigned by the system.

An item's properties include:

- User-assigned attributes. These are discussed in the next section.
- Priority. See page 271.
- Quantity. See page 272.
- System-assigned attributes. See page 275.

Item attributes

Because they allow items to be distinguished from each other, item attributes play a very important role in a discrete event simulation. They are especially useful for telling an activity-type block how long the item should be processed, or for determining where the item should be routed before or after processing. The following sections describe how to create, use, and manage attributes.


For examples of how to use attributes, see the Documents\ExtendSim\Examples\Discrete Event\Items and Properties folder. A good starting point is the Attributes Final Car Wash model.

☞ Flow attributes provide a similar functionality for Rate library blocks, which are used for discrete rate simulations. For more information, see "Flow attributes" on page 431.

Attribute names and values

Each item attribute is composed of a name and a numeric value:

- An attribute's *name* identifies some general characteristic of the item such as "size", "route", "CarType" or "tank capacity". Attribute names are limited to 15 characters.
- An attribute's *value* indicates one dimension of the named characteristic. For instance, an item' s "size" attribute could have a value of "8" or a value of "12", while an attribute named "CarType" could have a value of "1" (for Ford), "2" (for Toyota), or "3" (for Volvo). An attribute value is not just a number; it can also be the address of data in a database.

Attributes are meant to be unique; if you attempt to add a new attribute with exactly the same name as an existing one, ExtendSim warns you that the name already exists. While attribute names are not case sensitive ("Type" is equal to "type"), spaces are significant and should be avoided.

Attribute names and values are stored in a pair of dynamic, global arrays, described in "Attribute arrays" on page 270.

Number of item attributes in a model

In a model, each item can contain up to 500 attributes that uniquely describe the item. Every item contains the full set of attributes that have been defined in the model. The Executive block's Item Attributes tab displays all of the model's attributes.

Each attribute contains a value that can represent either:

- A number that can be used for routing, timing, and so forth.
- The address of data in a database or global array. The data pointed to can contain a single number or an unlimited amount of additional data that describes the item, its route, its properties, and so forth.

☞ If you use attributes efficiently, there is almost no limit to what can be represented. If you do approach the 500 attribute limit, consider using DB address attributes (discussed below) to reference information in the ExtendSim database.

Item attribute types

ExtendSim supports three types of attributes for items:

- A *value attribute* holds a real number as its attribute value.
- The value of a *string attribute* is still a number, but it is represented in the model by a string. With string attributes you enter a descriptive text label (string) for each potential attribute value in a lookup table in the Executive block's Item Attributes tab. The string can then be used in the model in place of the corresponding number. For example, a string attribute named "CarType" might have three possible values: 1, 2, and 3. Once the lookup table for this attribute has been properly configured, the blocks referencing the CarType attribute will display the strings "Ford", "Toyota", or "Volvo" instead of the numbers 1, 2, and 3.

- The value of a *DB address attribute* contains a single value that represents a location or address in a database. This address is composed of four separate numbers, where each number is an index for an ExtendSim database, table, field, and record. Taken together, the numbers target a specific location in the database. (Incomplete DB addresses are allowed. For example, an item may have a DB address attribute with only the database and table indexes defined.)

In addition to item attributes, the Rate library blocks of ExtendSim Pro have two types of flow attributes. These are discussed starting on page 431.

☞ The value of a DB address attribute cannot be used directly. It must be "decoded" using a Get. Read(I), or Write(I) block or by accessing DB attribute functions in one of the equation-based blocks.

Using item attributes

The following table lists some common attribute-based modeling activities and the blocks that are usually used to facilitate them. All blocks are from the Item library.

To Do This:	Use Block(s)
Set new attributes on items or modify values for existing attributes	Set, Equation(I), Queue Equation
Initialize newly created items with attributes	Create (when "Create items by schedule" is the selected behavior)
Define default attributes for item resources	Resource Item
Check attributes on existing items	Any property-aware block; see the table on page 259.
Route items based on attributes	Select Item Out (when "Select output based on attribute" is chosen)
Sort and release items from queues based on attributes	Queue (when it sorts by attribute value), Queue Matching
Sort items based on attribute values and conditionally release them based on an equation	Queue Equation
Pull in items and batch them based on attribute values	Batch (when "Match items into a single item" is the selected behavior)
Use attribute values to specify a delay or processing time	Activity (when "Delay is: an item's attribute value)
Define the value/string correspondence for item string attributes	Executive (Item Attributes tab in "Declare string attribute values" mode)
Find which block uses an attribute	Executive (Item Attributes tab in "Manage all attributes" mode)
Managing attributes and their names, such as renaming or deleting an attribute	Executive (Item Attributes tab in "Manage all attributes" mode)

To Do This:	Use Block(s)
Calculate an item's cycle time	Set an attribute to the current time in a Set block or use the *Timing attribute* feature in the Create block's Options tab. Then use the *Timing attribute* feature in the Information block so that it calculates the difference from start to end time. See "Cycle timing" on page 412.

Adding item attributes to a model

The Item library blocks that deal with attributes are listed in "Property-aware blocks" on page 259. These blocks provide a popup menu interface for selecting existing item properties or for adding new ones.

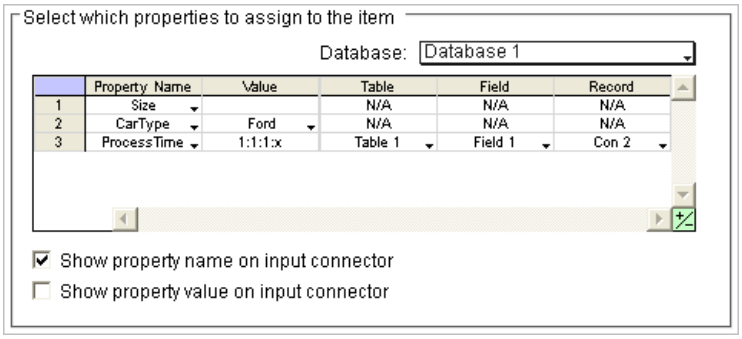
☞ Depending on the block, the popup menu may offer different choices of attribute types to set. For example, the Set block allows creating value, string, or DB address attributes.

Creating a new attribute causes it to appear in the list of properties in the block's dialog and makes the attribute accessible by every property-aware block in the model.

The following information describes how to create different types of attributes in the Property Name column of the table in the Set block's dialog:

- To create a new value item attribute, click the Property Name popup menu, choose New Value Attribute, type the name of the new attribute in the dialog that appears, and click OK. For example, a value attribute might be named "size".
- To define a new string item attribute, select New String Attribute from the popup menu in the Property Name column, enter a name, and click OK. This automatically opens the Executive block's Item Attributes tab. The table in this tab is where the attribute values (and the corresponding strings) for string attributes are declared. An example of this is shown in the tutorial in the Discrete Event Quick Start guide. An example of a string attribute could be "CarType" and the corresponding string/value combinations might be Ford/1, Toyota/2, and Volvo/3.
- Creating a new DB address attribute requires an existing ExtendSim database. In the Set dialog, select an ExtendSim database from the popup list that appears. To create the DB address attribute, click the popup menu in the Property Name column, choose New DB Address Attribute, type the name of the new attribute in the dialog that appears, and click OK. For example, a DB address attribute could be named "ProcessTime".

With the three different types of attributes, the Set dialog could look like:



After attributes have been created, they must be attached to items in the model and they must have unique values assigned to them.

Selecting attributes and attaching them to items

To allow an attribute to be used, define the attribute and assign a value to it. This is done using one of the attribute-handling blocks, such as Set or Create.

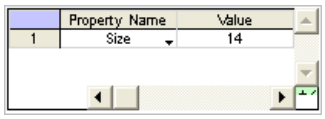
The most common method for assigning attributes to an item is to select an attribute in the dialog of a Set block, then pass the items through the block. The value of the attribute can be defined in the Set's dialog or through its value input connectors.

Another commonly used block is the Create block when it is in "Create items by schedule" mode. This is a convenient way to initialize new items with a set of attributes as they are introduced into the model. In the Create's schedule table, you can select an attribute from a popup menu in one of the columns, then enter a value for that attribute for each Create Time row in the table. An item generated at the specified times will have the attribute name and value indicated in the table. Other blocks, like the Resource Item block, can also be used to attach attributes to items.

☞ The information that follows assumes that you are using the Set block to assign an attribute to an item and that you have already created the attribute using a method described on page 266.

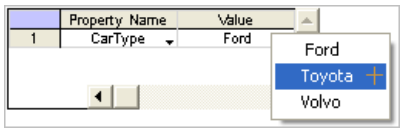
Value attribute

To set a value item attribute, select an attribute in the Property Name column's popup menu. (Existing value attributes are listed below the New Value Attribute divider.) Then enter a number in the Value column. In the screenshot at right, the Size attribute has been selected and 14 has been entered as the value.



String attribute

To set a string item attribute, select an existing attribute (listed below the New String Attribute divider) from the popup menu in the Property Name column. Then click the cell in the Value column to bring up a popup menu containing all the string values that have been defined for the attribute. In the example at right, the selected string attribute is CarType and the Value popup menu contains the strings Ford, Toy-



ota, and Volvo. If Toyota is selected, the corresponding value that gets stored on the item for the CarType attribute will be the number 2.

Connecting a Random Number block (Value library) that uses an Empirical table to a Set block that accesses a string attribute will cause the strings for that attribute to appear as a popup list in the empirical table's Value column. This is shown in the tutorial in the Discrete Event Quick Start guide.

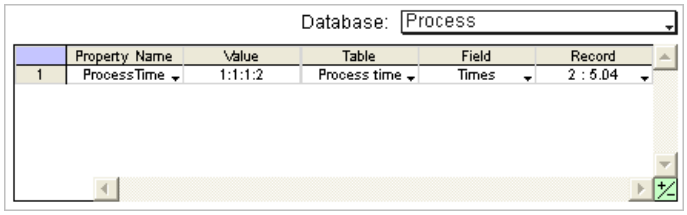
DB address attribute

Each Set block only points to one ExtendSim database, which becomes one element of the DB address. To set a DB address attribute in a Set dialog:

- Choose a database from the popup menu.
- In the Property Name column of the Set block's table, select a DB address attribute from the popup menu or create a new one; existing attributes are listed below the New DB Address Attribute divider.

Once the DB address attribute has been selected, the Set dialog's table enlarges to display the other elements of the address (Table, Field, and Record). The value for the DB address attribute is defined by clicking the appropriate popup menus in the table, selecting whether that element's information should be selected from a list, entered as an index, or accessed from a connector. The screenshot below is an example of the ProcessTime DB address attribute, which gets its value from a record in the Times field of the Processing Time table in the Process database.

☞ You don't need to select every element for a DB address attribute. For example, you may only want to specify the database, table and field indexes and ignore the record index.



For a DB address attribute, the Value column displays the database address, as determined by the indexes of the settings in the Table, Field, and Record columns. In the screenshot above, the Value notation is 2:1:1:2, where 2 is the index of the Process database, 1 is the table index for Processing Time, 1 is the field index for Times, and 2 is the index for the selected record, which has a value of 5.04.

Once the attribute has been set, the attribute information indicated in the Set block's dialog will be assigned to each item as it arrives to the block. Attribute values may also be defined dynamically using the Set block's value input connectors to override values set in the dialog.

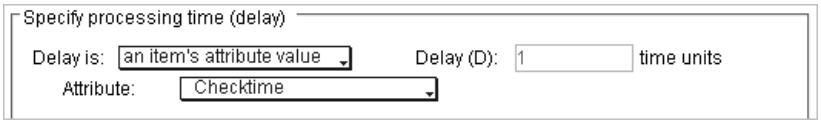
Every model has an internal list of all the attribute names that have been created for use by items in that model. However, not all items in the model will make use of every attribute name. For an item to use an attribute name, the value of the attribute must be explicitly set using an attribute modifying block (such as a Set block).

Getting attribute values and reporting changes

In order to manipulate an item based on the attribute, usually to route it or process it, you need to get the item's attribute value. The most common method for getting attributes is to select the

attribute by name from the list in the attribute popup menu in an attribute-reading block, such as the Activity or Get block.

Activity or Workstation blocks
In the dialog of an Activity or Workstation block, you can specify that an item's attribute value be used as its processing time, as shown below.



Get block
When items pass through the Get block, it accesses information about the attributes that have been specified in the table in its dialog. It then reports the information in the table and on its value output connectors. What the Get block reports and where, depends on the type of attribute:

- *Value attributes*. The value for the attribute is posted in the Value column of the attribute table and on the value output connector that corresponds to the attribute.
- *String attributes*. The string text is displayed in the Value column of the attribute table and the number that corresponds to the string is posted on the appropriate value output connector.

☞ Connecting a Lookup Table block (Value library) that is set to *Lookup the: input value* to a Get block that accesses a string attribute will cause the strings for that attribute to appear as a popup list in the Lookup Table block's left most column.

- *DB Address attributes*. You can get either an individual element of a database address or its entire address. To do this, from the popup menu in the table's "DB attrib reports" column, select which of the 5 components will be retrieved (db index, table index, field index, record index, or db address). The first 4 choices provide individual elements of the address; the "db address" choice provides the entire address. The information will be reported in the Value column and on the value output connector for that attribute.

☞ To access all five elements of a DB address attribute, add five rows to the table. Each row should have the same DB address attribute listed in the Property Name column, but different selections for the "DB attrib reports" column. This comes in handy when the Get block is working in conjunction with the Read or Write blocks (Value library). It allows the read or write location to vary based on what information is traveling on the item.

In addition to value outputs for reporting an attribute's value, the Get block has a Δ (delta) connector for reporting when an attribute's value changes. The Δ connector outputs a 1 when an item's attribute value (for the first attribute specified in the dialog) differs from the previous item's attribute value. Otherwise it outputs 0. This is useful for determining when there is a new type of item or when an attribute value used for processing time has changed. For example, you can have an attribute called "Type" with values that specify the type of item. When the value of Type changes, indicating a new type of item, the Δ connector outputs 1. This is shown in "Adding setup time" on page 324.

Modifying attribute values

The most flexible way to modify the value of an item's attribute or other property is with the Equation(I) block (Item library). This block can look up property information and modify it by applying some mathematical formula, then use the result as the new attribute value for the item. For example, if an item arrives with a value of 5 for the attribute "nextRecord", you could add a 1 to the 5 and create a new attribute value of 6 for that item's nextRecord attribute. The Air Freight model discussed on page 368 is an example of this.

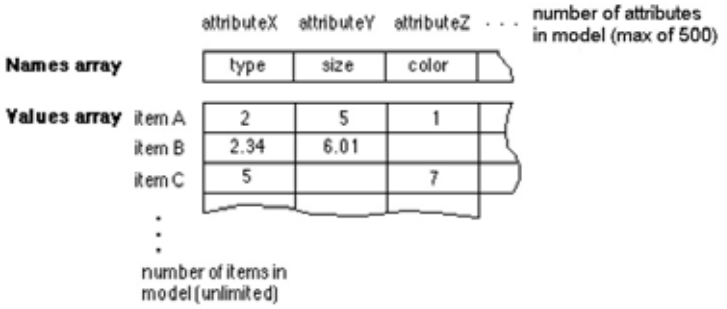
Another way to modify properties is to connect from a Get block's value output to a Math or Equation block (both from the Value library). Then have that block apply some mathematical formula and output the results to a value input on a Set block. The property must be selected in the dialogs of the Get and Set blocks, and the value connectors must be for that property.

Attribute arrays

Attribute names and values are stored in a pair of dynamic global arrays:

- The one-dimensional *Names array* stores the name of each attribute currently used in the model. Attribute names can be up to 15 characters long. You will receive an error message if you attempt to give an attribute a name greater than 15 characters. Attribute names are not case-sensitive.
- The two-dimensional *Values array* stores the value of each attribute for each item in the form of real numbers.

The following picture represents the attribute arrays:

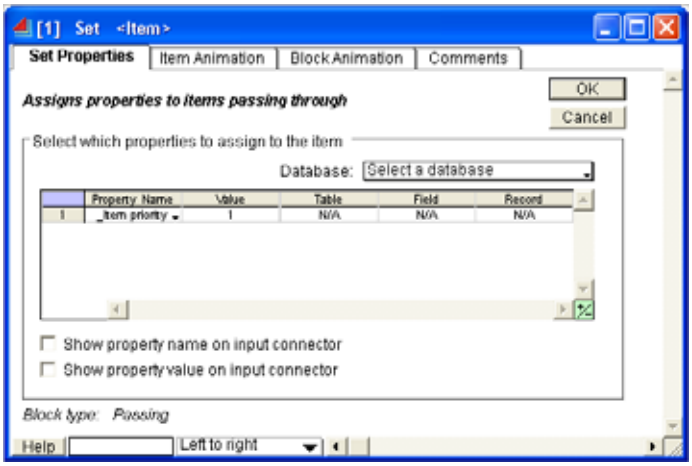


As new attribute names are added to the model, new cells (array elements) are appended to the Names array and new columns are appended to the Values arrays, up to a maximum of 500.

As new items are created during the simulation run, new rows are added to the Values array. The number of rows in the Values array is unlimited and will be the same as the number of items in the model. As shown in the above picture, item A has an attribute named "type" that has an attribute value of 2 and item B has an attribute named "size" with a value of 6.01.

Note that each attribute named in the model causes a cell to be reserved in the Values array for every item. However, not every item uses every attribute. To allow an item to use an attribute, you must assign a value to the attribute using one of the attribute-handling blocks (such as the Set block). If there is no value assigned, the attribute is not used by that item. This is shown in the figure above, where item B has no assigned value for the attribute name "color" and item C does not have a value for the attribute "size".

Priority



Like attributes, a priority is a type of item property that can be assigned to an item. Priorities signify the importance of items. Using the _Item priority property, you can assign priorities to items and manipulate them based on their priorities.

Priorities are particularly useful when you want to examine a population of waiting items and determine their processing order. For example, you might have a step in a manufacturing process where a worker examines the pending job orders and chooses the one that is the most urgent.

☞ Items can only have one priority. If you need multiple levels of priorities, use attribute values instead.

When a new priority is added to an item that already has a priority, the new priority prevails. When items are batched, the highest priority of the items prevails in the resulting batched item.

☞ The lowest value (including negative values) represents the top priority.

Setting, getting, and using priorities

The following table lists some common priority-based modeling activities and the blocks that are usually used to facilitate them. All blocks are from the Item library.

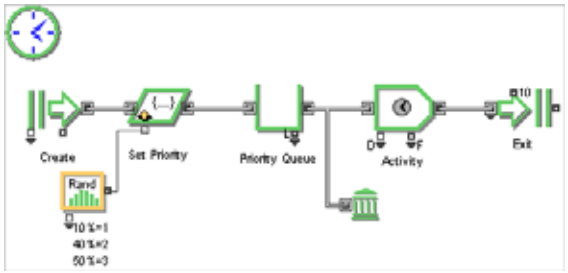
To Do This:	Use Block(s)
Initialize newly created items with priorities	Create (when “Create items by schedule” is the selected behavior)
Define default priorities for resource items	Resource Item
Set, modify, or check priorities on existing items	Set, Get, Equation(I)
Select incoming items based on priorities	Select Item In (when “Select input based on item priority” is chosen)
Sort and release items based on priorities	Queue (when it sorts by priority)
Sort items based on priority and conditionally release them based on an equation	Queue Equation
View an item’s priority	Get, History
Allocate resource pool units to the highest ranked item first	Resource Pool

☞ The Select Item Out block does not assign or use items' priorities. Instead its output connectors can be prioritized so that an item will be routed to the first available connector that has the highest priority. As shown in "Explicit ordering" on page 305, the Select Item Out block prioritizes the *path* an item will take rather than the item itself.

Priorities model

In the example, a Random Number block (Value library) outputs values to a Set block, as follows:

- 10% of the time it outputs a 1
- 40% it outputs a 2
- For the remaining 50% it outputs a 3.



Priorities model

The table in the dialog of the Set block indicates it will assign priorities to incoming items. Connecting from the Random Number block's output to the Set block's ItemPriority value input connector causes the priorities to be set according to the values from the Random Number block. Since the lowest number is the highest priority, 10% of the time items will be assigned the highest priority.

The Queue block is set to sort by priority. This means that the highest priority items held in the block will be made available to the Activity block before other items. A History block, added to the model by right-clicking on the Queue's output connector, shows that only top priority items are processed; the Activity cannot keep up with the demand.

The section "Interrupting processing" on page 329 shows how priority values are used to determine if one item should preempt another.

☞ For an item to be ranked by priority, there must be other items in the group at the same time. For example, items will only be sorted by priority in a Queue block if they have to wait there with other items.

Quantities

Quantity is another type of property that can be assigned to items. Each item can be a single entity or a group of duplicates. As is true for priority, an item can only have one quantity assigned to it at a time; the default quantity is 1. If the quantity property for an item is 1, it represents one item. If the quantity is other than 1, it represents a group. Item quantities are typically set in the Create and Set blocks.

☞ An item's quantity can be any number, including a negative number. An item with a quantity of 0 or less disappears when it reaches a queue.

For most purposes you would not want to change the quantity of an item from its default value of 1. However, to model a change of shift consisting of five workers going off duty at the same time, to simulate the delivery of a box of 300 pieces of mail to a mail room, or for similar situations, set the quantity of the item to be other than 1.

How blocks treat items with quantities other than 1

Items with quantities other then 1 are treated differently depending on the nature of the block processing them. They will travel together as a unit, being processed essentially as one item, until they reach an Exit, a Queue, a Batch, or a Resource Item block, or are sent into a universal connector (such as Change, Demand, Select, or Start.)

- When an item with a quantity other then 1 reaches an Exit, Queue, Batch, or Resource Item block, it is decomposed into separate identical items. For example, when it enters a Queue, an item with a quantity of 10 will become 10 distinct items, each with a quantity of 1 and each with the same properties (attributes, priority, and so on) of the original item. An item with a quantity of 0 will disappear when it reaches a Queue.

- When items with quantities other than 1 are sent into a universal connector (such as *demand* or *select*), they are treated as one item, but the quantity of the item may be used by the block as control information. See below for more information on how universal connectors deal with the incoming items that have quantities other than 1.

All the other blocks deal with items that have quantities other than 1 as a single item, ignoring the quantity associated with it.

☞ The Activity block accepts an item with a quantity greater than 1 as a single item and process it as one unit. To have the items be processed separately, precede the Activity block with a queue, since queues decompose items with quantities greater than 1.

Setting an item's quantity

A quantity can be assigned to an item in the Create, Set, or Equation(I) blocks.

Set block

In the table in the Set block's dialog, select _Item quantity in the Property Name column and enter the quantity in the Property Value column or input a value to the Block's _Item Quantity value input connector. Each item that passes through the Set block will be assigned that quantity.

Create block

The default setting in the Create block is that one item is input to the model at each arrival event; this is the most common case when building models. How you specify that a multiple number of items be released at each event depends on which behavior is selected for the Create block:

- Create block is set to "Create items randomly". Change Item quantity (Q) in the Options to an integer number other than 1. Or input a value to the Create block's ItemQuantity (Q) value input connector.
- Create block is set to "Create items by schedule". Enter values in the table's Item Quantity column for each Create Time field that has arrival times.

For example, assume you want to show that one item arrives randomly approximately every 4 minutes. To do this, use the same settings as in the Car Wash model from the discrete event

tutorial: in the Create block select the Exponential distribution and enter Mean: 4; in its Options tab leave Item quantity (Q): 1. To show that 2 car/items arrive every 4 minutes, keep the settings at Exponential with a mean of 4, but enter Item quantity (Q): 2. The block will now output one item with a quantity of 2 approximately every 4 minutes.

As discussed in "How blocks treat items with quantities other than 1" on page 273, the blocks that follow the Create block determine how an item with a quantity greater than 1 is treated. For instance, if an item with a quantity of 2 goes directly into a Queue or Resource Item block, it will be split into two items each with a quantity of 1. However, if the item goes directly into an Activity block, it will be treated as a single item with a quantity of 2. In most cases, you will want to follow the Create block with a Queue, which will decompose the item into two separate items.

☞ In most cases, you probably will not want to generate more than one item at each event. For example, rather than inputting 2 items every 4 minutes as discussed above, you would probably want to generate 1 item every 2 minutes. This is because, unless they are inside a container of some sort, it is not common to see two items arrive at exactly the same time; items are more likely to arrive at slightly different times.

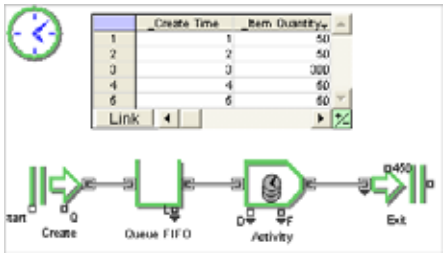
Quantities model

For example, assume you will receive 500 items a week, but that almost all of them are received on Wednesday. In this case, there are five arrival events (one event each on days 1 through 5), each with an item quantity of either 50 or 300.

This Quantities model is similar to the Scheduled Intervals model from page 262, except each item the Create block generates has a quantity greater than 1 and the Activity processes 5 items at a time.

The dialog of the Create block is set to Create items by schedule. The arrival times (Create Time) and the number of items arriving at the scheduled time (Item Quantity) are entered in the table, which has been cloned onto the model worksheet.

The table indicates that on the third day (Wednesday) 300 items arrive but that 50 items arrive on each of the other days.



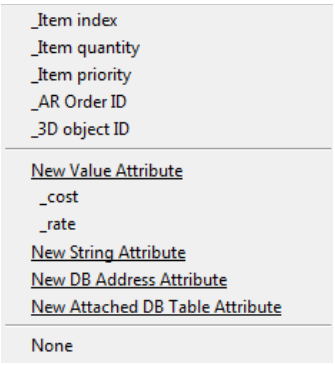
Quantities model

The Create block outputs an item with a quantity greater than 1 as if it were a group of items all arriving at the same time. When the item goes to a Queue block, it becomes multiple copies of itself. In this example, as each item is sent from the Create block to the Queue block, it will become either 50 or 300 units, depending on its quantity.

Running the model shows that the Create block creates 5 items, but that 500 items have arrived to the Queue.

Other item properties

In addition to the item properties discussed above, such as the item quantity or user-defined attributes, ExtendSim can assign properties to items. As shown in the property popup menu below, these system properties are preceded by the "_" character and include:



History block Properties menu

- _Item index. This property is available in the Get, Equation(I), and History blocks and points to where the item is located in the item arrays stored in the Executive block. It is used by block developers for debugging.
- _AR Order ID. This property is only available if the model contains a Resource Manager block (Item library). It refers to the resource order an item is associated with. The value of this property is the record index in the Resource Orders table of the item's resource order. For more information, see the separate document about Advanced Resource Management.
- _3D object ID. When you select a 3D animation object to represent an item, this property stores the index of the object. Note: 3D is not available in ExtendSim 10.
- _Cost or _Rate. If there is an entry for cost somewhere in the model, ExtendSim will add the _Cost and _Rate attributes to property popup menus. For more information, see "Working with cost data" on page 387 and "Combining multiple cost accumulators" on page 392.
- _Resource Order ID. This item property is only available in the Queue and Queue Equation blocks (Item library) if *Store Resource Order ID in attribute* is checked and the blocks are part of the Advanced Resource Management (ARM) system. The property is the value of a record index in the Resource Orders table of the Advanced Resources database. The record represents the last requested requirement that was assigned to an advanced resource item. For more information, see the separate document about Advanced Resource Management.

Discrete Event Modeling

Queueing

Storing items in buffers or waiting lines

A queue provides a buffer or waiting line to store items awaiting further processing. Queues can have simple behavior, such as holding items in first in. first out (FIFO) order, or more complex behavior, such that items are held and released in groups based on their attributes. You can also set an option in the Queue block's dialog to specify how long an item will wait until it reneges, or prematurely leaves.

This chapter covers:

- Queueing disciplines: LIFO, FIFO, Priority, Attribute, and User-Defined
- Queue/server systems
- Blocking, balking, and reneging
- Sorting items using the Queue Equation block
- Least dynamic slack, minimizing setup, and maximizing service levels
- Using the Queue Matching block to match items into groups based on their attributes
- Viewing and initializing queues with the Queue Tools block
- Displaying queue contents through animating

☞ This chapter's examples are located in the folder Examples/Discrete Event/Queueing.

Blocks of interest

The following blocks are the main focus of this chapter. Each block's library and category appears in parentheses after its name.

Queue (Item > Queues)
Stores items until there is downstream capacity. As a sorted queue, holds items in FIFO or LIFO order, or sorted by their priority or attribute value. As a resource pool queue, holds items in FIFO order.

Queue Equation (Item > Queues)
Stores items. Calculates an equation when it receives an item or when it is triggered by a value connection. When there is downstream capacity, releases items based on the results of the equation.

Queue Matching (Item > Queues)
Has a specified number of internal queues for holding items in separate groups. Releases a group when there is downstream capacity and the group requirements have been met. This block is useful for matching one type of item with another.

Queue Tools (Utilities > Discrete Event Tools)
When connected to the L (length) output of a queue, views and initializes the queue's contents. Displays information about item properties in a table. Can add an initial number of items, with specified properties, to a queue.

☞ In this chapter the focus is on using a Queue block to represent a sorted queue. For information about using the Queue block as a resource pool queue see "Resource pool blocks" on page 360.

Queueing disciplines

ExtendSim supports several scheduling algorithms, also known as *queueing disciplines*, through the queue blocks.

- *FIFO*. When set to be a sorted queue, the Queue block can represent a first in, first out (FIFO) queue, also known as a first come, first served queue. When set as a resource pool queue, the Queue block represents a FIFO queue for resource pool units. The “MM1 model” on page 280 is an example of a FIFO queue and most of the models in the Discrete Event module use a Queue block in FIFO mode. For more information about resource pools and how the Queue is used as a resource pool queue, see “Resource pool blocks” on page 360.
- *LIFO*. When set to be a sorted queue, the Queue block can represent a last in, first out queue. As is true when the Queue is set to FIFO mode, the Queue block automatically takes care of LIFO sorting.
- *Priority*. As a sorted queue, the Queue block can read priorities and pass items with the highest priority (lowest number) out first. For this to happen, the arriving items must have a priority. Items that have not been assigned a priority in the model have a default priority with a Blank value; they get relegated to the end of the waiting line. To see a Queue sorting items based on priorities, see “Priority queues” on page 280 or “Animating queue contents” on page 291.
- *Attribute*. As a sorted queue, the Queue block can use attribute values to sort items in the queue. In addition, the Queue Matching block allows you to define custom scheduling algorithms based on item attributes. It groups items based on certain attributes and releases them as a group once requirements are met. For this sorting rule, items must have attributes assigned to them before entering the Queue. Items that have not been assigned an attribute in the model have a default attribute with a Blank value; they get routed to the end of the waiting line. The process for having a Queue sort items based on attributes is similar to the process for sorting using priorities.
- *User-Defined*. The Queue Equation block allows a user-defined equation to decide the sorting order for items it holds. This can be used to specify any user-defined criteria for sorting, including Least Dynamic Slack, Minimize Setup, Maximize Service Level, and any other combination of sorting rules. A discussion of these ranking rules and example models start on page 283.

☞ It is important to remember that, except for a FIFO queue, there must be other items in a queue at the same time to allow the queueing disciplines to work appropriately and affect the order of the items. For example, if you set a Queue block to sort by priority, and there is never more than one item in the block at a time, the effect of queueing based on priority is negated.

Queue/server systems

Queue/server systems involve the creation of items which then wait in a queue until they can be processed by one or more servers. The following blocks in the Item library are used to represent queue/server systems:

- The *Create* block is used to provide items at exponential interarrival times (and many other interarrival times as well).
- A *Queue* block, set to sort in FIFO, LIFO, or some other order, holds the items and releases them in the designated order. It can have a maximum queue length specified in its dialog.

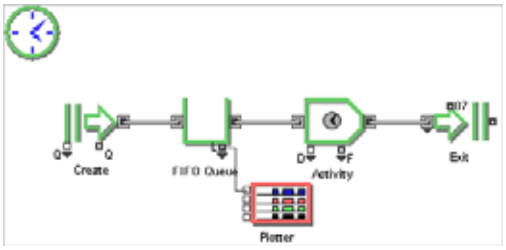
- The *Activity* block represents servers: you can specify an exponential or other distributional service time within its dialog or by connecting a Random Number block (Value library) to its D (delay) connector.

M/M/1 queues

A standard notation often seen in queueing theory is M/M/1. This is a basic construct, representing a single server queue. The notation translates to: *exponential interarrival times/ exponential service times/ single server.* It is also common to see the designation M/M/1/×, where the × translates to *unlimited queue length*, or the designation M/M/1: ×/×/FIFO, which translates to *exponential interarrival times/ exponential service times/ single server*: *unlimited queue length/ infinite population/ first in, first out service*.

MM1 model

A typical M/M/1 system expressed using ExtendSim blocks, with the addition of a plotter and an Exit block, would look like the screenshot below.



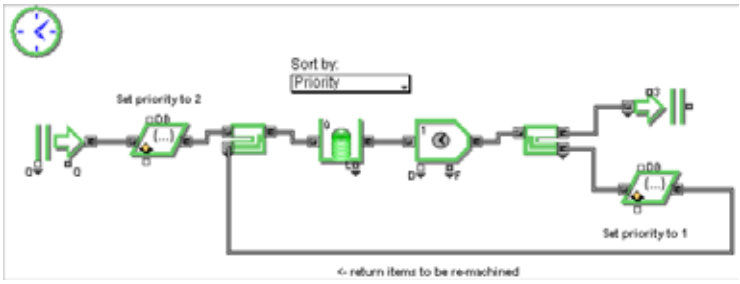
MM1 model

Priority queues

As is true when any other sorting rule is used, a Queue block that sorts by priority will hold items until there is downstream capacity. Once the downstream block can accept an item, the Queue searches through the contents of the queue and releases the item with the highest priority. For the Queue to work properly in this mode, items that enter should already have their priority set; items without a priority are assigned a default Blank priority and get sent to the end of the waiting line.

Priority model

In the Priority example, items enter the model and immediately have their priority set to 2. They then enter a Queue block set to Sort by: priority. After the machining processes, each item is inspected for flaws. If the item does not pass inspection, its priority is re-set to 1 and it is sent back to the Queue block where it waits to be re-machined. When the machine can accept a new item, the Queue block will release the item with the highest priority. In this case, any item waiting to be re-machined will be released first.



Priority model

Run the model with animation turned on to watch the items with a priority of 1 (red circles) bypass items with a priority of 2 (green circles) while waiting in the Queue.

Queueing considerations

Once items are generated for the model, it is common that they will be held in a queue, typically a Queue block. In addition to the queueing disciplines discussed above, queues and the items in them can exhibit other behaviors.

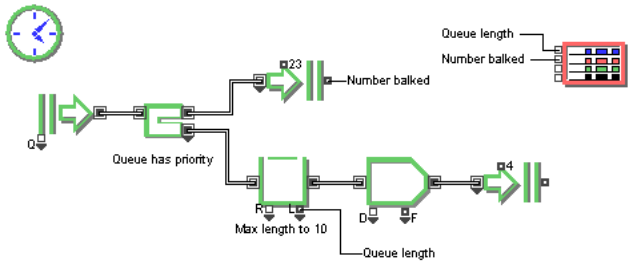
Blocking

Blocking occurs when an item is prevented from leaving a block because there is a downstream capacity constraint. Blocking is common in serial operations where there are several activities in a row without queues in between; each activity has the potential for blocking arriving items. It also occurs when activities are preceded by queues with finite capacity, causing backups in the preceding activity. Blocking increases the waiting time for items in queues and is added to the calculation of their utilization.

The examples in the sections “Processing in series” on page 317, “Sequential ordering” on page 304, and “Machines that can only process certain types of items” on page 312 illustrate potential blocking situations.

Balking

Sometimes customers enter a facility, look at the long line, and immediately leave. This is an example of *balking*. In the Balking model. balking is easily represented by setting a maximum queue length and then using a Select Item Out block to prioritize sending items to that queue. If the queue is full, the Select will send the items to another part of the model; in this case, an Exit block..



Balking model using Select Item Out to prioritize the Queue

Balking can also be represented by having a Decision block (Value library) look at a queue’s length or wait time. If the line meets certain conditions (is too long, takes too long to move, etc.), a Select Item Out block routes the item out of the model before it enters the queue.

Reneging

Reneging occurs when an item, having entered a queue, leaves before it reaches the output. An example of this is telephone callers who, after being put on hold, will hang up without getting help if they feel they have waited too long for assistance.

To simulate reneging, select an option in a Queue block’s Options tab. The choices are:

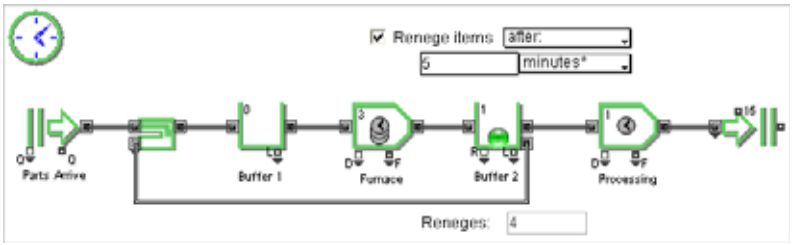
- Renege items after a specified number of time units. The number of time units can be set in the block’s dialog or through its R (renege) connector.

- Renege items immediately when the R (renege) connector gets a true value (0.5 or greater).

When either of these choices is checked, an alternate item output appears on the right of the Queue block. Items that renege leave through that Renege output. They can be routed back to the original line (as in the example below), routed elsewhere in the model, or they can exit the model.

Reneging model

In the Reneging model, parts wait in the first buffering queue until they can be heated by a furnace, then wait for processing in a second buffer. If too much time passes before a part is processed (such that it cools down), the part is sent back to the first buffer to wait for reheating.



Reneging model

The Options tab of the Queue that represents Buffer 2 specifies that a part will wait 5 minutes before it must be returned for reheating. The relevant information has been cloned onto the model worksheet. As seen in its Results tab, the Queue block automatically counts and reports how many items have reneged.

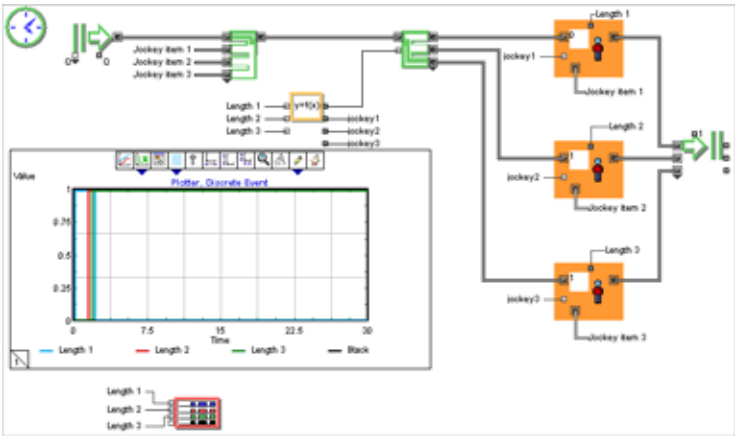
Jockeying

Jockeying is when items move from one waiting line to another in an attempt to gain some advantage. To see a good example of jockeying, go to any supermarket and watch as people leave the end of a slow moving cashier's line to try and get onto a faster line.

The reneging feature on the Options tab of a Queue block is useful for building a model of this type of behavior. Normally, items renege if they have spent too much time in a queue. But the Queue block has a connector that can force reneging of the last item in the line. The Jockey.mox model is a good example of this.

Jockey model

In this model customers arrive from the Create block and are routed through the Select Item Out block to the shortest of three possible queues.



Jockey model

As customers wait in the queues it is possible for the lines to move at different speeds. The last customer in each queue has the option to move to another queue if a shorter line opens up.

Sorting items using the Queue Equation block

The Queue Equation block (Item library) is used to release items in an order that is based on a ranking rule. Items enter the queue and are stored in the order of their arrival (FIFO). However, instead of the items leaving in FIFO order, an equation in the block's dialog determines which item (if any) should leave the queue next. The equation is automatically calculated once for each item in the queue and the results are used to assign a ranking for each item. The item with the best ranking is the one that is allowed to leave next.

How the block works

Importantly, only the equation results from the winning item are used for that particular calculation cycle. For example, if the queue currently holds 10 items, the equation will be calculated 10 times and 10 individual sets of equation results will be collected. But only the equation results from the item with the best ranking will be used; results from the other 9 items are discarded.

So that the item leaving next will be properly chosen, by default a new calculation cycle is initiated every time an item enters or leaves the queue or when a value input connector receives a message. To override the default behavior, the factors that initiate calculation cycles can be controlled through check boxes in the block's Option tab.

The tables below describe a number of equation input and output variables that are unique to the Queue Equation block. Along with the common input and output variables described in "Equation components" on page 190, these variables are designed to help users write equations that will properly select the next leaving item. In particular, the *QEQ item rank* variable is used to assign a ranking to each item in the queue.

☞ At least one QEQ item rank output variable must be defined for this block to function properly.

Once each item has been assigned a ranking, the items are released from the queue according to the ranking rule selected in the Options tab's Release popup; the ranking rules are listed on page 285. The block's internal data structure keeps track of the ranking assigned to each item.

☞ Giving an item a Blank ranking makes it an ineligible candidate for leaving the queue for that particular query cycle

Variables

The Queue Equation block has several types of input and output variables. The tables below (input variables and output variables) describe the common set of variables shared by all equation-based blocks. The following variables are unique to the Queue Equation block.

To modify the number of rows used in the variable tables:

- Change the number of rows in the table by clicking the green +/- resize button in the table's bottom right corner and entering the number of rows desired.
- Delete rows by first selecting the rows you wish to delete, clicking the green +/- resize button, and then selecting the option to "delete selected rows."
- Duplicate any rows by first selecting the row you wish to duplicate, clicking the green +/- resize button, and then selecting the option to "copy selected row."

Input variables

Input Variable	Uses
QEQ arrival time	The time the item arrived to the queue
QEQ FIFO position	The item's first-in, first-out (FIFO) position in the queue
QEQ current best item rank	The best (highest or lowest) item rank result for the current calculation cycle
QEQ attrib last item to exit	Provides the chosen attribute value of the last item to exit the block
QEQ num items in queue	The number of items currently in the queue
QEQ static calc cycle init	A static variable that gets initialized to its starting value at the beginning of every calculation cycle. Changes to this variable remain fixed (static) across equation calculations.
QEQ static item value	A static variable used to temporarily store information on items during their stay in the queue.
AR requirement is avail	This boolean variable is True if an advanced resource (AR) requirement is available for the current item. See note below.
AR num requirements avail	Provides the number of advanced resource (AR) requirements available for the current item. See note below.

☞ The AR variables are only shown in the Queue Equation block if Enable Advanced Resources (AR) is checked in the block's Options tab and the model contains a Resource Manager block (Item library included with ExtendSim DE and ExtendSim Pro).

Output variables

Output Variable	Uses
QEQ item rank	Defines each item's rank (position) in the queue. *At least one of the output variables must be of this type.*
QEQ halt calculation cycle	If set to True, this variable will prematurely halt the current calculation cycle.
QEQ next item	Controls which item in the calculation cycle is evaluated next.
AR allocate requirement	If set to True, the current item's advanced resource (AR) requirement will be allocated to it. See note above.

As mentioned earlier, while you can specify other equation results, one of the output variables has to be of the type *QEQ item rank*. Furthermore:

- Only the results for the item ranked best will be output.
- You can use more than one *QEQ item rank* variable; the secondary ranking variable will be used to arbitrate in the case of tied ranking.

☞ These variables are only available for the item that is currently being released.

Ranking rules

The Release popup on the Options tab provides the following selections to determine which item should be released first:

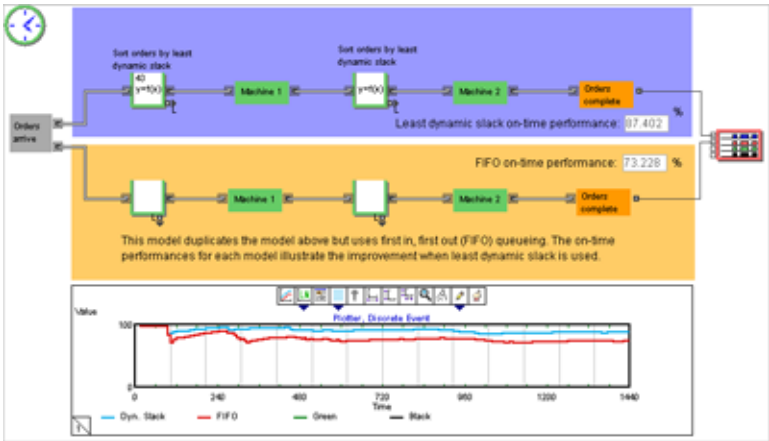
- Highest ranked item. Each item's rank value is calculated by the equation and the item with the highest ranking is allowed to exit.
- Lowest ranked item. Each item's rank value is calculated by the equation and the item with the lowest ranking is allowed to exit.
- The first True ranked item. The equation will be calculated once for each item until either an item receives a True ranking or there are no items left to evaluate.
- All True ranked items. All items receiving a True ranking are moved to an internal departures list where they are immediately and permanently available to be moved downstream.
    - The results are saved for each item in the departures list. These results are used (or broadcast) at the time the item exits the queue.
    - Items in the departures list are no longer evaluated by the equation. This can reduce the number of times an item is evaluated, reducing run times.
    - Items are released in FIFO order.

Least dynamic slack

Least dynamic slack is a ranking rule for queues that tends to reduce the "lateness" of a sequence of orders. Slack is defined as the due date minus the remaining processing time. In essence, it is the "float" that is available before the item is due to be completed. When using slack to sort items, priority is given to those items that are closest to being late. In a model that represents orders for services or goods, choosing the order with the least dynamic slack tends to minimize the number of late orders.

Least Dynamic Slack model

The example model Least Dynamic Slack illustrates the improvement in on-time performance that can be achieved by sequencing orders by least dynamic slack instead of first in, first out ordering. The two models are identical, except the top model uses Queue Equation blocks with



Least Dynamic Slack model

least dynamic slack calculations and the bottom model uses Queue blocks and typical FIFO ordering. In the model, the equations in the Queue Equation blocks calculate the dynamic slack for each item. The item with the smallest dynamic slack (least amount of time before being late) will be selected first. As seen on the plot which has been cloned onto the model worksheet, on-time performance is higher using least dynamic slack (top line) compared to FIFO (bottom line).

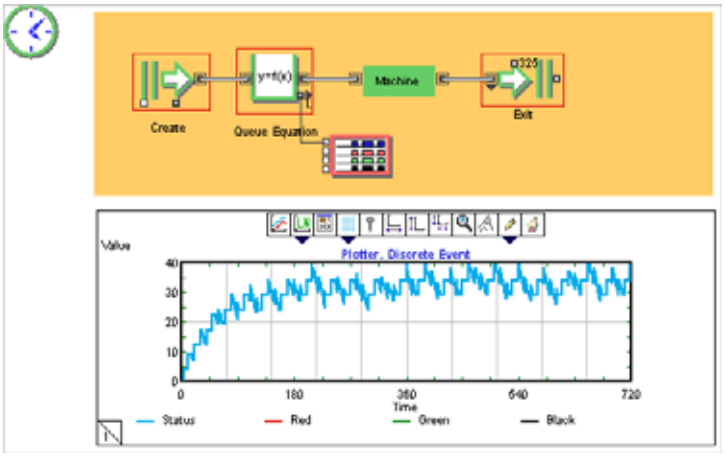
Minimizing setup

In some systems, setup time (the changeover from one product to another) can add significant delay to the processing of items. If this is the case, it may be useful to process the same item type until there is no longer any of that item type in the queue. Only when a particular type of item has been exhausted will another type of item be processed. Giving priority ranking in a queue to the same type of product that has just exited the queue reduces the number of setups or changeovers between products. Like least dynamic slack, minimizing setup time is another type of queue ranking rule.

Minimize Setup model

The model Minimize Setup compares the Product attribute for each item in the Queue Equation block to the Product attribute on the last item to leave the queue. The first item with its Product attribute value equal to the item that has just exited is released first. If no item in the queue can be found with an attribute value that matches the last exited item, the first item in the queue is selected. The plot shows the effects of this rule: the queue builds up initially until it can combine enough batches together to gain an efficiency from minimizing the setup time.

(The example includes a second model, with a FIFO queue instead of a Queue Equation block, for comparison purposes.)



Minimize Setup model

Maximizing service levels

In a service system, the service level can be defined as the number of customers served within a certain time period. For instance, technicians might be rated on the percentage of customer requests fulfilled within a certain time period. To maximize this, a queue that applies the maximize service level rule gives priority to those customers who waited less than the service level time, leading to dramatic improvement in the service level. However, this type of system might not be popular with real-life customers since some of them may have to wait a very long time while other customers who arrived later would wait a much shorter time.

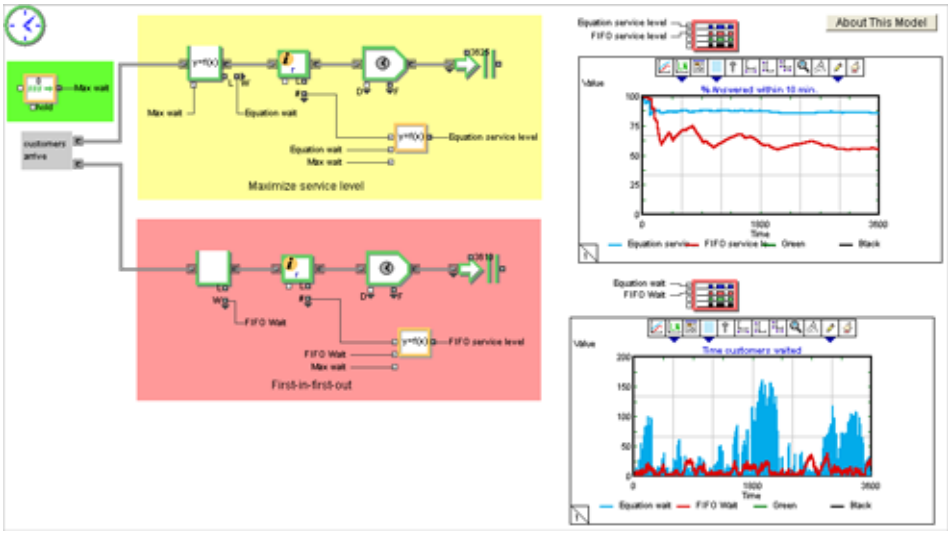
Maximize Service Level model

The top section of the Maximize Service Level model uses a Queue Equation block to sort the queue into two priority levels:

- The service priority is the customer who has spent the longest time (but less than 10 minutes) in the queue
- Customers who have waited longer than 10 minutes are placed at the back of the line

The lower section of the model is exactly the same as the top, except it uses a Queue block in FIFO sorted queue mode. As the model is run, two plotters show the effect on service level and wait time. Comparing the service level of the upper model to the bottom model, it is obvious that there is a dramatic improvement if the sorting rule is used. However, the second plotter

makes it equally as obvious that some customers have a much longer wait when the service level is maximized.



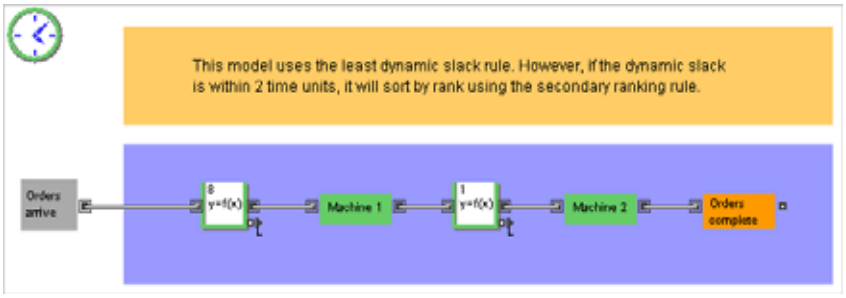
Maximize Service Level model

Combined rules

Because of its tie-breaking capabilities, the Queue Equation block can be used to model situations where two items are considered equal using the primary sorting rule but a secondary rule is used to determine the item with the higher priority.

Combined Rule model

The Combined Rule model uses the least dynamic slack as the primary rule. However, if the least dynamic slack is within 2 time units, the rank (order of the items in the queue) is used.



Combined Rule model

Matching items using the Queue Matching block

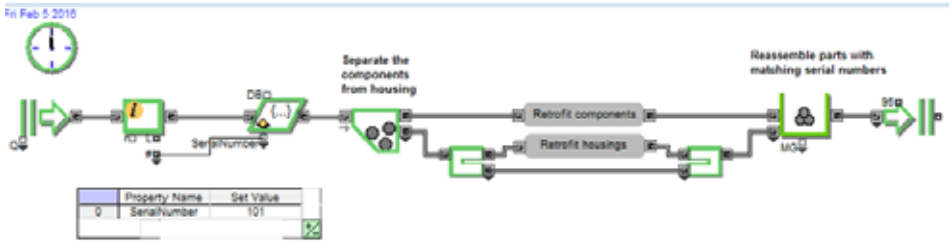
The Queue Matching block has a variable number of item connectors where each connector represents a separate internal queue. Within each queue, the block sorts items into different groups based on each item's match attribute value. Items are released from a group only when the required number of items are present in each group in each queue.

This block is especially useful for making sure that items have a particular characteristic at a specific time. For instance, you would use this queue to reassemble parts in the correct order or to insure that subassemblies are correctly matched with each other.

☞ For more detail, please see the Queue Matching online help by clicking the "Help" button in the lower left of the block's dialog.

Queue Matching model

In the following example, electronic systems arrive from the field, are separated into their individual components for refurbishing, and are reassembled. In this operation, all of the components but only 40% of the housings need to be reworked. In addition, it is important that the housings for each system be reassembled with their original (refurbished) component set.



Queue Matching model

The Information block counts each electronic system as it arrives and outputs the total number that have passed through the block. The Set block uses this value to set a Serial Number attribute for each electronic system. After refurbishment, the system is reassembled using its original parts.

Other models that use the Queue Matching block

The folder Examples\Discrete Event\Queue Matching contains additional models that use the Queue Matching block. Those models explore more advanced topics such as modeling fixed and variable requirements for specific items.

Advanced queue topics

This section discusses viewing, initializing, and animating the contents of a queue. These techniques are useful in creating a more exact model as well as for debugging and/or validating a model.

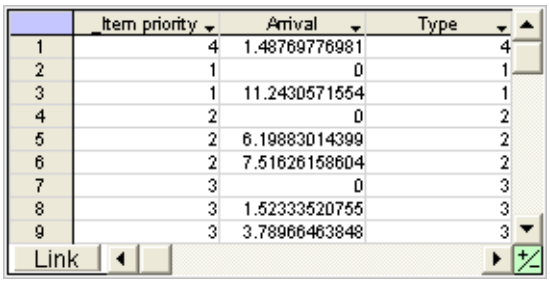
Viewing and manipulating queue contents

Although the Queue itself has a *Contents* tab that can show and report the items within the Queue at any time (See "Item Contents of queues and activities" on page 210), a powerful ExtendSim feature is the ability to also manipulate the contents of a queue. The Queue Tools block (Utilities library) allows you to view items in a queue, manually manipulate the ordering of those items, and initialize the queue's contents. To use this block, connect from the L (length) output of a Queue or Queue Equation block to the value input connector of a Queue Tools block. The block has two tabs, View and Options, as discussed in the following sections.

View tab of Queue Tools block

The Queue Tools block's View tab is used to manipulate items in a Queue or Queue Equation block and display information about the items. When the model is run, every item in the attached queue will have an entry in the table.

Popup menus at the top of the columns are for selecting which of the item's properties (attribute, quantity, priority, and so forth) to view. There are also buttons (Up, Down, Destroy) on the View tab that can change the rank of an item in the queue or delete an item from the queue. For instance, in the screenshot to the right an item with a priority of 4 has been moved in front of other items with priorities of 1.



Manipulating a queue

To manipulate the items in a queue, run the model, pause the simulation at the desired point, select an item, and use the buttons in the dialog to move or destroy it. (To pause a simulation run, click on any cell in the Queue Tools table or use the Pause button in the ExtendSim toolbar or the Pause menu command.) Leaving this block open (or having a clone on the model worksheet) while the simulation is running will slow the model down; it is best to close it when not needed.

The View tab of a Queue Tools block is cloned onto the worksheet of the Initializing and Viewing model, discussed in the next topic.

Initializing a queue

Sometimes it is useful to introduce items into the model at the start of the simulation run. The Options tab in a Queue Tools block (Utilities library) can be used to preload a queue with items at start time. Situations where queues might be initialized with items include:

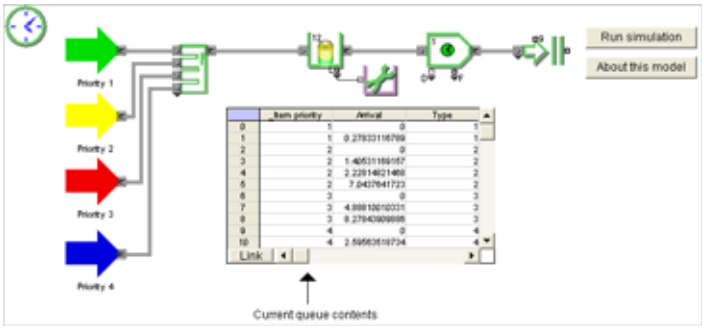
- Reducing start-up bias. By placing items in queues at the start of a simulation, the model begins in a state that is closer to steady-state.
- Importing current system status in a scheduling model. When using simulation to model a detailed schedule, it is necessary to start the simulated system with the same work-in-progress as the real system.

To use this block, connect from the L (length) output on a queue to the value input connector on a Queue Tools block. The block's Options tab has three choices:

- No queue initialization: Items are not added to the queue at the start of the simulation.
- Initialize queue: The queue is initialized with the number of items entered in the number field and property values as specified in the Properties table.
- Initialize from global array: The number of items and property values are read in from a specified global array. Because global arrays can themselves be initialized from a number of sources including Excel, a database, or the Internet, this is a very useful way to import the contents of a queue from an external source.

Initializing and Viewing model

The Initializing and Viewing example uses the Options tab of a Queue Tools block to introduce items into a Queue block at the start of the simulation. A clone of the block's View tab has been placed on the model worksheet.



Initializing and Viewing model

In this model, ten items are added to the queue at time 0. As seen in the Properties table in the Queue Tools's Options tab, these initial items have their item priority set to 1, their item quantity set to 1, their Type attribute set to 1, and their Arrival attribute set to 0. The tab also indicates that those items are represented by a cyan circle for animation. After the initial 10 items, items from the four processes are animated as circles with the same color as the arrows.

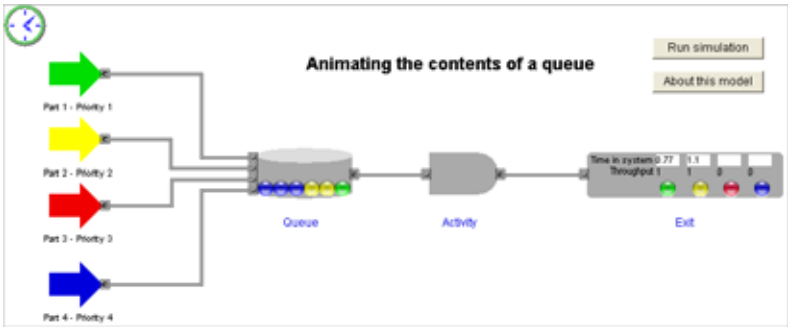
Animating queue contents

By default, the number of items in a Queue block are displayed on its icon. However, a more detailed animation showing an animation picture for each item can be obtained by putting the Queue inside a hierarchical block and animating the hierarchical block's icon. To add this type of animation to a model:

- Encapsulate a Queue inside a hierarchical block (right-click the Queue block and select Make Hierarchical.)
- Open the hierarchical block's structure (right-click the hierarchical block and select Open Structure) and add a number of identically sized animation objects from the icon tools in the toolbar.
- On the Item Animation tab of the Queue block, enter the first and last animation object numbers in the Animate H-block objects fields.

Animating Queue Contents model

This example model animates a priority queue. There are four possible priorities, as indicated by the colored arrows on the left. Green circles represent items with a priority of 1, yellow circles indicate an item has a priority of 2, and so forth.



Animating Queue Contents model

Items with a lower number for their priority (a higher priority) will move to the front of the queue. The animation shows this happening as new items with lower priorities will pass other items.

In this model, there are four rows and six animation objects per row on the icon of the hierarchical block to the right of the arrows. The Item Animation tab of the Queue block inside that hierarchical block is set to Animate H-block objects: 1 to 24. This causes all 24 objects on the hierarchical block's icon to be animated, based on what is happening in the Queue.

For more detailed information about animating hierarchical blocks, see "Animating a hierarchical block's icon without programming" on page 113.

Discrete Event Modeling

Routing

Handling items from several sources;
sending items to multiple destinations

When building models, you will frequently encounter situations where you want to manipulate items coming from several sources or send items to several possible destinations. Depending on the purpose, there are several methods for accomplishing this. This chapter will cover:

- Items arriving from multiple sources
  - Merging items from several streams into one stream
  - Balancing waiting line lengths
  - Using the Throw Item and Catch Item blocks
- Items going to several destinations
  - Simple routing to one of several streams
  - Scrap generation
  - Successive, explicit, and conditional ordering of routes
  - Routing based on item properties

☞ The models illustrated in this chapter are located in the folder \Examples\Discrete Event\Routing.

Commonly used blocks

The following blocks will be the main focus of this chapter. The block's library and category appear in parentheses after the block name.

Blocks that route items

Catch Item (Item > Routing)
Receives items sent remotely by a Throw Item block.

Select Item In (Item > Routing)
Selects an input and outputs its item.

Select Item Out (Item > Routing)
Sends each item it gets to a selected output.

Throw Item (Item > Routing)
Sends items remotely to a Catch Item block

Blocks that affect the flow of items

Decision (Value > Math)
Can be used with Item library blocks to control the flow of items in a portion of the model.

Gate (Item > Routing)
Controls the flow of items in a portion of the model (area gating) or based on model conditions (conditional gating).

Math (Value > Math)
Performs a mathematical operation, such as addition or subtraction, that can be used with Item library blocks to control the flow of items in a portion of the model.

Max & Min (Value > Math)
Outputs the minimum or maximum value found among its input connectors. Can be used with Item library blocks to control the flow of items in a portion of the model.

Items from several sources

Depending on your modeling needs, you may want to merge different streams of items into one stream of individual items, select one item from several for routing or processing, or join separate items into a single item.

- To *merge* streams of items from several sources into one stream, where each item remains separate and retains its unique identity, use the Select Item In or Throw Item and Catch Item blocks. You then typically direct the single stream into a queue. For example, you can use this to represent traffic merging into one lane or people accessing one hallway from several offices. A Select Item In block is used to:
  - Merge streams of items in the “Merging Inputs model” on page 297.
  - Direct items requiring more processing in “Cumulative processing time: time sharing” on page 323.
  - Reroute preempted items in “Preemption” on page 330.

  Those models use the Select Item In block to route items. The section “Throw Item and Catch Item blocks for merging item streams” on page 298 illustrates using a Catch Item block to merge multiple streams of parts into one stream.

- To *select* an item for processing from several sources based on some decision, use the Select Item In block. The decision can be a logical decision (choose every other item to route to the top waiting line) or it can be based on some characteristic of the item (get the item with the highest priority). The specifications for the decision are determined by the entries you make in the dialog of the Select Item In block and are modified by blocks connected to its “select” input connectors. Using the Select Item In block to choose specific items is shown in “Balancing multiple input lines” on page 297.

- To *join* items from various sources and process them as one unit, use a Batch block, as described in “Batching” on page 346. This is most common when modeling manufacturing processes or packaging operations where subassemblies are joined together. It is also used when two or more items need to be temporarily associated with each other for processing or routing, such as a clerk processing an order. Note that batching differs from using the Select Item In and Select Value In blocks, which only merge streams of items so that items remain separate and are processed separately.

☞ To merge streams of items from one hierarchical layer into one stream at a different hierarchical layer, you can add connectors to the hierarchical block or use the Throw Item and Catch Item blocks, at shown on page 298.

Select Item In dialog

The Select Item In block chooses an item from one of its input connectors and sends that item to its output connector. The selection is based on settings and options in the block's dialog.

Selection options

The Select Item In block has several rules for selecting an item from its input connectors:

- *Item priority*. Selects the input connector that has an available item with the highest priority (the lowest numerical value for its priority.) For example, you could use this option to select from a group of queues to a single activity. The queue with the item that has the highest priority will be selected. This option always starts and restarts its selection search at the top input.
- *Random*. The inputs are selected randomly based on probabilities entered in the block's selection table. Enter probabilities in decimal format. For example, enter 0.75 for 75%. If the entered numbers do not equal 1.00, the actual sum will appear in red in the bar below the Probability column. If *Select from: all inputs* is chosen, an input will be randomly selected whether or not an item is available at that input. This situation can potentially cause starving, as discussed below. If *Select from: only inputs with available items* is chosen, the block will only select from inputs with available items.
- *Select connector*. The value received at the *select* connector determines which input is chosen. The block's dialog has an option for setting which value chooses the top input; the default is 0. The lower connectors will be numbered sequentially after the top connector. That is, if the top input is chosen by a select value of 1, the second input will be numbered 2, the next lower input would be numbered 3, and so forth. In that case, a value of 3 at *select* would cause the item from the third connector from the top to be selected. Note that, even if items are available at the other inputs, the block will wait for an item at input 3, potentially causing starving as discussed below.

☞ See "Item library blocks" on page 413 for some precautions when using this option with a Get block.

- *Sequential*. Selects the inputs in strict sequential order starting at the top; this is also known as a "round robin" selection. This option could cause starving (discussed below), since the block will wait for an item to become available at each selected input.
- *Merge*. Items are taken as they become available through any input. Generally, this option is used to combine the flows of items where there is no blocking of items arriving at the Select Item In block. Inputs are selected in a "round robin" fashion starting from the top; once a selection has been made the selection search will resume at the next lower input.

☞ Residence blocks such as the Activity have variable item input connectors. This works the same as using the Select Item In block in Merge mode to merge multiple item streams.

Starving conditions

If an item is not available from the selected input of a Select Item In block, the following options will cause a starving condition:

- Random (if *Select from: all inputs* is chosen)
- Select
- Sequential

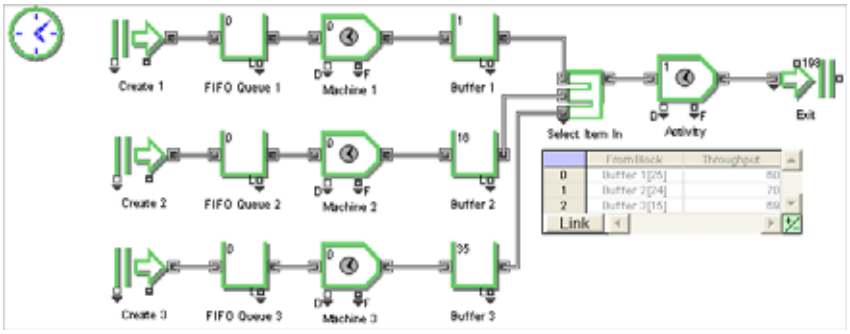
Merging several item flows into one stream

The Select Item In block can combine the inputs from any number of sources into one stream of output items.

☞ An alternative is to use the variable item input connectors on an Activity or other residence block.

Merging Inputs model

In the Merging Inputs example model, the Select Item In block will accept items from any of the three inputs. Its dialog is set to *Select input based on: merge*. If the Select Item In's output is blocked, the block will force items to wait in the Queues (labeled Buffers 1-3). When the Select Item In becomes unblocked, it will check each input in turn to try to pull an item through for processing by the Activity. As you can see in the table that has been cloned from the Select Item In block, when it is ready to process items, the Activity gets whichever item is available. This can cause some queues to have longer waiting lines than others, as you can tell from their Results tabs.



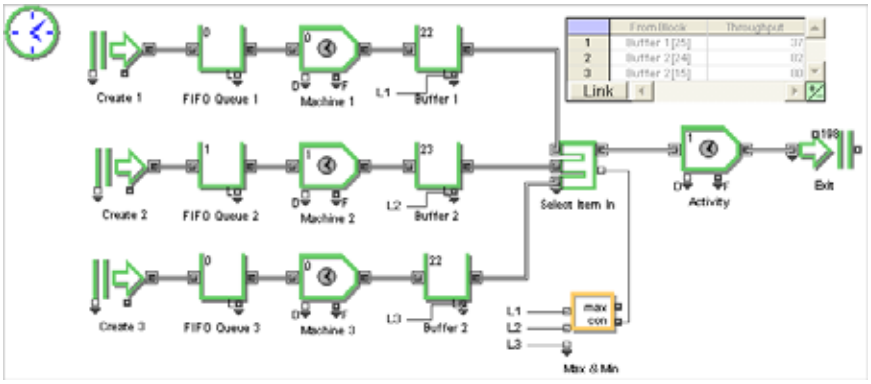
Merging Inputs model

Balancing multiple input lines

To even out the queue lengths of multiple input lines, use a Select Item In block controlled by a Max & Min block (Value library), which checks the length of each queue. An example of this would be three loading docks that fill up as trucks unload, and you want items to come first from the dock that is most full.

Input Line Balancing model

The Input Line Balancing model is the same as the Merging Inputs model, except a Max & Min block looks at the length for each of the queues and sends that information to the Select Item In block.



Input Line Balancing model

On the Max & Min block, the *con* output connector tells which of the inputs has the largest value, in this case it indicates the longest queue. This tells the Select Item In block which queue to retrieve the next item from. In the dialog of the Select Item In block, *Select input based on: select connector* and *Top input is chosen by Select value: 1* have been selected. As you can see from the cloned Throughput table, items are drawn in a balanced manner from each line, and the queue lengths are almost equal, as opposed to what happened in the Merging Inputs model, earlier.

Throw Item and Catch Item blocks for merging item streams

The previous examples discussed routing items using connections to blocks that are nearby and at the same level of hierarchy. Sometimes, especially in large models, it is necessary to send an item to a different hierarchical layer. The Throw Item and Catch Item blocks are especially useful when there are items from various locations in a model (even from various hierarchical levels) that need to be sent to one place. Note that these blocks are used as an adjunct to routing, not a replacement for the methods described previously.

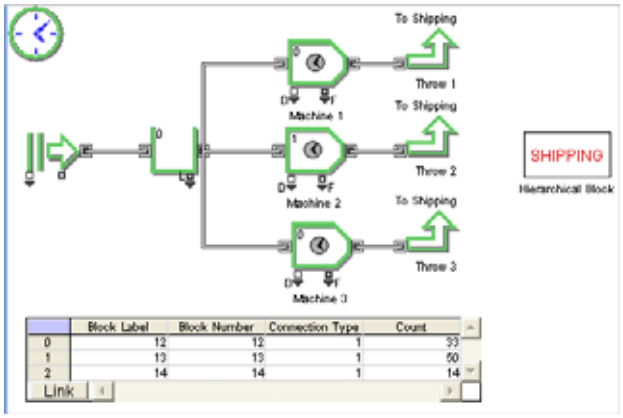
Throw Item and Catch Item blocks pass items without connections and can even be used deep within nested hierarchical blocks to send items to other hierarchical blocks. For that reason, they are sometimes used instead of the Select Item In and Select Item Out blocks.

☞ Throw and Catch blocks should only be used when named connections will not be sufficient. For instance, to pass items through different levels of hierarchy or to use the routing features on the Throw and Catch blocks.

Throw & Catch model

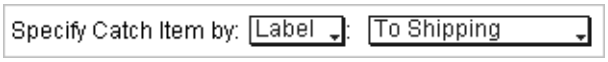
The Throw Item and Catch Item blocks can also be used to merge several item flows into one stream. In the following example, three Throw Item blocks route items to Shipping, which is a

hierarchical block containing two blocks, a Catch Item and an Exit. The Catch Item block is labeled *To Shipping* and is designated as belonging to *Catch group 1*.



Throw & Catch model

A popup menu in the Throw Item block's dialog displays the labels of the possible Catch Item blocks. You have the option of routing all items to the Catch Item block specified in the popup menu (as shown to the right) or routing items to different Catch Item blocks depending on the value of a specified attribute or priority (see "Throw and Catch Attributes model" on page 307).



Selecting Catch Item block labeled "To Shipping"

☞ You must enter text in the label field at the lower left corner of each Catch Item block's dialog. Only labeled Catch Item blocks will appear in a Throw Item block's popup menu.

Catch Item groups

If you are working with a large number of Catch Item blocks, you may want to organize them into groups. To do this, select or create a group name using the *Catch Item group* popup menu in the Catch Item block, shown at right. Then use the *Catch Item group* popup menu in a Throw Item block to select the desired group. Once a group is selected in the Throw Item block, the block's *Specify Catch Item by: Label* popup menu will only contain the labels for the Catch Item blocks in the selected group.

Catch Item group popup menu

☞ Groups can only be defined in a Catch Item block.

Items going to several paths

In many cases, you will need to route items from one stream to one of many different streams:

- Taking a stream of items and routing them to different activities or operations is called *parallel processing*. In parallel processing, each item is handed off to one of several activities, such as an Activity block. The logic that determines which operation the item is routed to can be simple (the part is machined at the first available station) or it can be complex (bottle

type A is filled at Machine 3). Different methods of routing items to parallel processes are described in detail throughout this chapter. See also “Processing in parallel” on page 318.

- For situations where one item is *unbatched* or separated into its component items, use the Unbatch blocks. For example, you might receive a shipment of furniture consisting of 8 desks, 20 chairs, and 7 typewriter returns, or a mail cart with 1000 pieces of mail. You use an Unbatch block to disassemble that item into its individual components, then route the items to appropriate destinations, as described in “Unbatching” on page 353.
- To *select the path* an item should go on, use the Select Item Out or Throw Item blocks. The Select Item Out block is useful for routing a stream of items to several paths based on some decision. For instance, you can send all the parts that need rework to a rework station, and ship the remaining parts. Or direct patients requiring immunizations to the Injection Clinic. The use of these blocks is described in “Sequential ordering” on page 304, “Explicit ordering” on page 305, “Routing decisions based on Item properties” on page 306, and “Select Item Out dialog” below.

Select Item Out dialog

The Select Item Out block is appropriate for routing items onto one path or another. Its dialog contains several options for determining which route an item should take.

Selection options

The logic in the dialog of the Select Item Out block chooses which output connector an input item should be routed to. The selection options are:

- *Property*. The appropriate output is determined using the item’s property–its attribute or priority. Values to represent the outputs are entered in the table’s Select Output column; the default is that 0 selects the top output. For each item, the block finds the value of the specified property in the table’s second column (which is named for that property), and determines the corresponding output connector for that value in the Select Output column. Since the block will hold the item until there is downstream capacity, this option can cause blocking.
- *Connector priority*. An attempt is made to send the item out each connector, in the order of the *connector’s* priority, until the item is accepted by a connected block. The priority for each connector is entered in a table in the block’s dialog. The top priority is the lowest number, such that an output with a priority value of 1 has a higher priority than an output with a priority value of 3.

☞ Note that this is different from assigning a priority to an item and selecting the output based on the item’s priority, as can be done with the block’s *Property* option. With the *Connector priority* option, the Select Item Out block essentially prioritizes the output path, not the item.

- *Random.* Outputs are selected randomly based on settings in the block’s probability table. Enter probabilities in decimal format. For example, enter 0.75 for 75%. If the entered numbers do not equal 1.00, the actual sum will appear in red in the bar below the Probability column. When the option *If output is blocked: item will try unblocked outputs* is chosen, the block will randomly try to find an output that can accept the item. When *If output is blocked: item will wait for blocked output* is used, the block will select an output and the item will wait until that output is able to accept the item; this can cause blocking.

- *Select connector*. The value received at the *select* connector determines which output is chosen. The block's dialog has an option for setting which value chooses the top output; the default is 0. The lower connectors will be numbered sequentially after the top connector. That is, if the top output is chosen by a select value of 1, the second output will be numbered 2, the next lower one would be numbered 3, and so forth. In that case, a value of 3 at *select* would cause the item to go to the third connector from the top. Since the block will hold the item until there is capacity downstream from connector 3, this option can cause blocking.

☞ See "Item library blocks" on page 413 for some precautions when using this option with a Get block.

- *Sequential*. Outputs are selected one after the other in sequential order starting from the top; this is also known as a "round robin" selection. When the option *If output is blocked: item will try unblocked outputs* is chosen, the block will try the next connectors sequentially. When *If output is blocked: item will wait for blocked output* is used, the block will select an output and the item will wait until that output is able to accept the item; this can cause blocking.

☞ The Select Item Out block expects integer values for comparison and will truncate non-integer values. For example, if *select connector* is chosen as the selection condition, the numbers 0.001 and 0.999 received at the SelectIn input would both be truncated to a 0.

Blocking conditions

The Select Item Out block is a decision-type of block; its default is to pull in the item and then determine the path that the item will take. In some situations, the selected output path may be blocked and the selected item will have to wait to leave. Some selection conditions can cause the items behind a selected item to be blocked:

- Property
- Random (when *If output is blocked: item will wait for unblocked output* is chosen)
- Select
- Sequential (when *If output is blocked: item will wait for unblocked output* is chosen)

For the *random* and *sequential* selection conditions, the ability to choose what happens if the output is blocked is useful for certain modeling problems. For instance, in the Simple Routing model shown below, if the top Queue block is designated to get the item, but it is blocked, the Select Item Out block will route the item to an unblocked Queue.

Predicting the path of the item before it enters the block

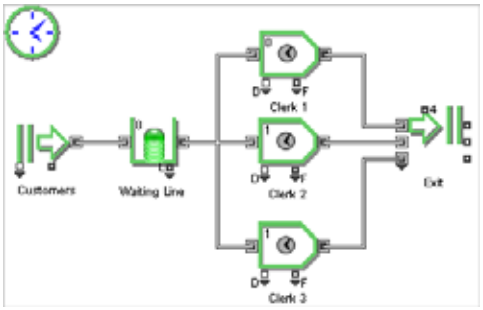
As mentioned above, an item can be pulled into a Select Item Out block but not be able to proceed because the downstream path is blocked. An alternative to this situation is to cause the item to wait in an upstream queue, rather than in the Select Item Out block. This is accomplished by checking *Predict the path of the item before it enters this block*. When this is enabled, the Select Item Out block will query upstream to determine the properties of the next item to arrive. It then checks to see if the appropriate downstream path is clear. Only if the item can be sent out the desired output will the item be pulled in. This guarantees that the item will not get "stuck" in the Select Item Out block.

☞ This setting requires that any properties used to make the selection have to be set before the item begins to move into the Select Item Out block. For example, a Queue is necessary between a Set block and a Select Item Out block.

Implicit routing

The simplest, but not necessarily the best, way to route items is by creating connections between the output of the collection point and the inputs of each activity-type block. This causes ExtendSim to pass items to the first available activity.

However, if more than one activity-type block is free when an item is ready, it is not obvious which block will get the item. For instance, a Queue that holds items for three Activity blocks would look like the model below.



Simple Connections model

If two or more machines are free when an item comes out of the queue, the machine that was *first connected* will get the item. With these types of simple parallel connections, even just disconnecting and then reconnecting a connection line could change the order of activities getting items. This implied routing may not be reflective of the actual system and is usually not want you would want.

⚠ Unless it is completely unimportant in the model, you should always use the Select Item In and Select Item Out blocks to explicitly state how items should be routed.Otherwise, the order in which their connections were made will dictate the routing.

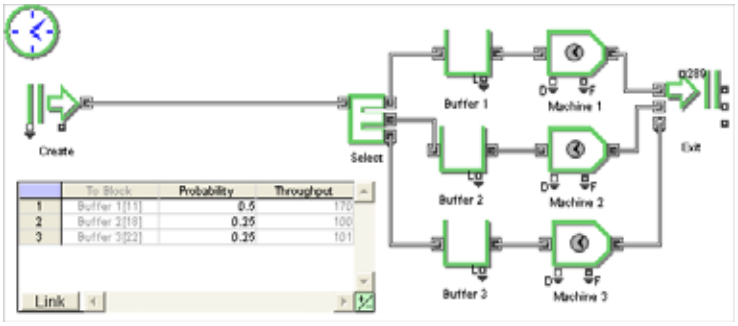
Simple routing

It is most common to route a random number of items to one section of the model, while the rest are routed to another. An example of this is an intersection where a number of cars will turn to the left, some will go straight, and some will turn right.

Simple Routing model

In this model, items are routed randomly to one of three machines, as indicated by the setting *Select output based on: random*. The probability table in the dialog of the Select Item Out

block indicates that *0.50* (50%) of the items will go to the top Queue while the remainder will be distributed equally between the remaining two Queues.



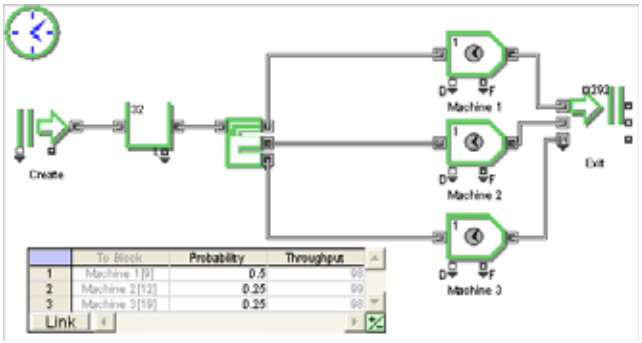
Simple Routing model

Notice that there are queues in front of the machines. If you omitted the queues, there is a possibility that items arriving from the Create block could be blocked. For example, if the second item were destined for the top machine, but that machine was still processing the first item, the other machines would have to wait for items until the top machine finished processing and pulled in its item.

☞ The previous model routed items based on probabilities. To distribute an input item to *any* available output, in the Select Item Out dialog choose *Select output based on: sequential* and *If output is blocked: item will try unblocked outputs*.

Simple Routing One Queue model

A model similar to Simple Routing would be if there were only one queue and it was placed before the Select Item Out block. As mentioned above, however, if two sequential items are destined for the same activity they will block items that arrive behind them. The option *If output is blocked: item will try unblocked outputs* allows the Select Item Out block to try other outputs if the first choice is unable to accept the item.



Simple Routing One Queue model

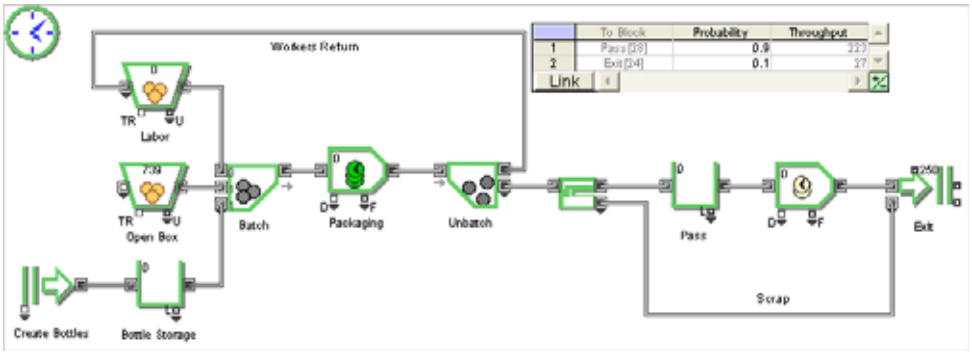
As seen in the model, even though 50% of the items should be going to the top machine, item distribution is almost even. Because all the machines process items for the same amount of time, the top machine is often busy and, rather than cause the system to be blocked, its intended item is routed to a different machine.

Scrap generation

An important aspect of some systems is modeling the generation of *scrap* or simulating a *yield rate*. For instance, many manufacturing processes create an expected but irregular quantity of waste or bad items. This can be accomplished in ExtendSim by randomly routing some items out of the normal processing stream.

Scrap Generation model

This example is similar to the model that is described in “Simple unbatching” on page 355, except it has a Select Item Out block that determines whether items coming from the Unbatch block should continue for processing or be discarded, and the Activity block processes two items at a time. The Select Item Out block is set to *Select output based on: random*. By setting a probability that *0.90* of the items will exit through the top connector and *0.10* through the bottom (scrap) connector, the Select Item Out block causes one out of every ten items to become scrap.



Scrap Generation model

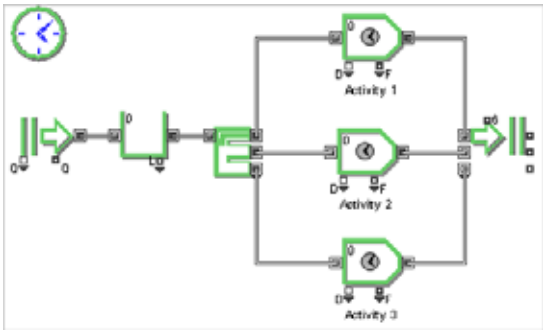
As an alternative, you can also set and check attributes to represent items that need to be scrapped. This will be shown later in this chapter.

Sequential ordering

To hand items to operations in successive order regardless of whether another operation is free, use the Select Item Out block set to *Select output based on: sequential*.

Sequential Ordering model

With the sequential setting, the Select Item Out block will choose outputs in successive order starting from the top output. The block's dialog is set to *If output is blocked: item will wait for blocked output*.



Sequential Ordering model

The first two activities are set to process for 1 time unit while the third activity takes only 0.5 time units. For this model, even if the third activity is the first one ready to accept an item, it will only get an item after the first and second activities have pulled in an item. Also note that an item is only pulled from the Queue block when an activity has finished processing, potentially causing blocking in the system. The example "Balancing multiple output lines" on page 309 shows a solution for this.

☞ To distribute an input item to any available output, choose *Select output based on: sequential* and set the block to *If output is blocked: item will try unblocked outputs*.

Explicit ordering

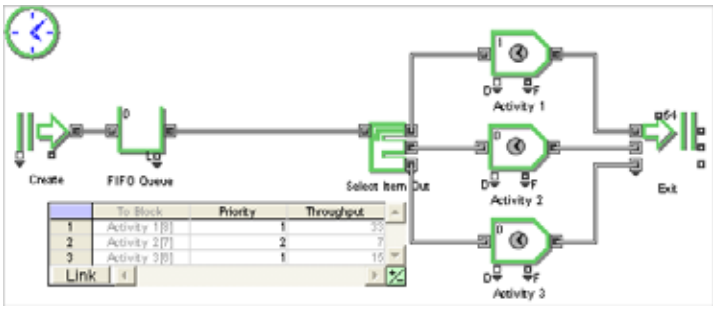
If there are several operations, and you prefer certain ones to be active more than others, you can explicitly state which operations have a higher priority for items. This is common when you want to avoid using an operation because it is not as efficient (such as an older piece of equipment) or because it is an uneconomical use of a resource (having a supervisor wait on customers.)

Choosing the selection condition *connector priority* in its dialog allows the Select Item Out block to be used to specify the priority of each output.

☞ Note that this is different from assigning a priority to an item, since the Select Item Out block essentially prioritizes the output path, not the item.

Explicit Ordering model

For example, assume you want Machines 1 and 3 to get most of the items for processing, and Machine 2 to only get items if Machine 1 and 3 are busy. The model is:



Explicit Ordering model

The dialog of the Select Item Out block is set to *Select output based on: connector priority*. In the table, the highest priorities (which are the lowest numbers) are assigned to the top and bottom outputs, and the next lowest priority is assigned to the middle output. In this case, Activity 1 has first priority on items. If Activity 1 is busy, Activity 3 will get the item. Only if Activity 1 and 3 are busy will Activity 2 get the item.

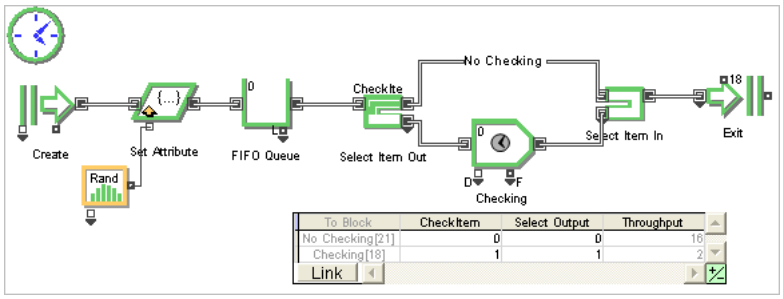
☞ As seen in this model, multiple outputs can have the same priority. However, the item will go to the topmost output that has the highest priority and is free. If that output is not free, the next lower output with the same priority will be checked to see if it is free, and so forth. If this is not what you want, set the priority values explicitly. For instance, you could set the output priorities to 1, 2, and 3 rather than to 1, 2, and 1 as was done in the example model.

Routing decisions based on Item properties

You may want to route items based on some characteristic of the item, such as its priority, size, quality, age, or state. To do this, assign an attribute or priority to the item and read that property value to route the item.

Attributes for Routing model

To specify whether or not an item must have a process performed on it, set an item's attribute to a yes-or-no value using the Random Number block (Value library) as shown in the Attributes for Routing model, below:



Attributes for Routing model

The Empirical distribution in the Random Number block specifies that 75% of the items do not require checking (0 value for the CheckItem attribute) and 25% do (1 for attribute value). The Select Item Out block, set to *Select output based on: property*, reads the attribute value to determine which of two routes the item will take, one through the checking line (value = 1) and the other around it (value = 0). The Select Item In block is used to combine both lines into one stream, exiting the simulation.

This method is especially useful if the checking process takes more than one step. For instance, you may need to transport the item to the checking station using transportation blocks, but only if checking is needed. All those steps would be between the Select Item Out block and the Select Item In block.

With this model, an item that needs to be checked can be pulled into the Select Item Out block but not be able to advance because there is already an item in the Activity. To prevent this, you can cause the block to predict the path of an item before it enters, as discussed page 301.

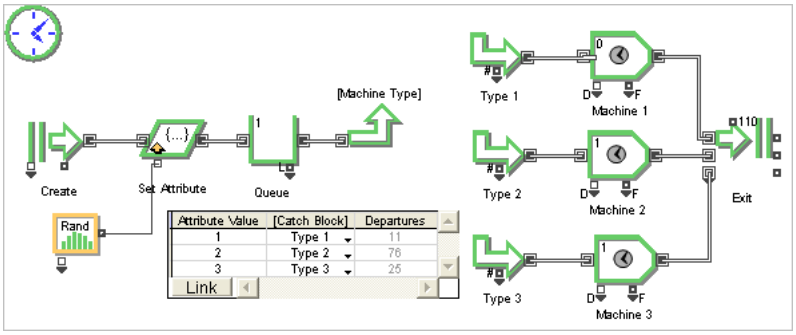
☞ An item that requires checking that is blocked in the Select Item Out block will also block other items that arrive after it, even if they do not need to be checked. If this is not how your process works, insert a Queue before the Activity to hold items that need checking.

The example "Machines that can only process certain types of items" on page 312 is another instance of using attributes to route items. For a very different approach, the DB Job Shop model located in the folder Examples\Discrete Event\Routing uses information from the ExtendSim database to route items.

Throw and Catch Attributes model

As described in "Throw Item and Catch Item blocks for merging item streams" on page 298, the Throw Item block can be used to route items to a specific Catch Item block that is identified by its label. Throw Item blocks can also be used to route items to different Catch Item blocks depending on the value of an item's attribute or priority. A modification of the Attri-

butes for Routing example, built using Throw Item and Catch Item blocks rather than the Select Item Out block, is shown below.



Throw & Catch Attributes model

In this example, The Throw Item block is set to *Specify Catch block by: Property: Machine Type*, where Machine Type is a value attribute. The Throw Item block reads the Machine Type attribute and routes the items to the appropriate Catch Item block according to the table in the throwing block's dialog, which is cloned onto the model worksheet.

To cause an attribute or priority value to be associated with a specific Catch Item block, type the value into the "Property Value" column and select the appropriate Catch Item block using the popup menu in the "[Catch Block]" column.

State Action model

Another routing example is the State Action model where items are routed to operations depending on their state. For complete information, see "State/Action models" on page 16.

Conditional routing

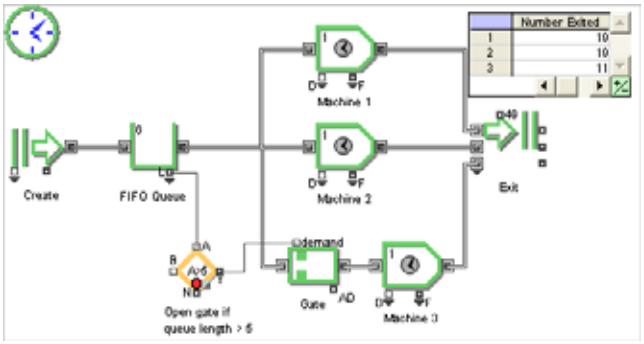
Sometimes you will want to route items based on the current conditions of the model. For example, monitoring queue lengths to determine whether or not an activity will be brought on-line or balancing the use of parallel waiting lines.

Bringing a system on-line

Most of the examples in this manual show items being passed to operations where all the operations are on-line and running. In many situations, particular operations are only started when they are needed. You can bring another system on-line based on the time of day (such as in "Scheduling activities" on page 325) or based on some other factor such as the backlog of work.

Conditional Routing model

For example, you might have a factory where most of the processing is done by two machines but excess work is handled by a third machine. ExtendSim can simulate this easily using the Decision block (Value library) and the Activity block (Item library).



Conditional Routing model

The *L* output of a queue that is feeding one or more machines outputs the number of items waiting to be processed. If this value is greater than a certain threshold, you can route some of the items to another machine or activate another process.

In the model, the dialog of the Decision block specifies that the Y connector outputs a true value (1) when the value at the A input is greater than 5. This activates the Gate block's *demand* connector so that it lets items through to the third machine (until then, it will not accept items). When the Queue holds 5 or fewer items, the Gate closes.

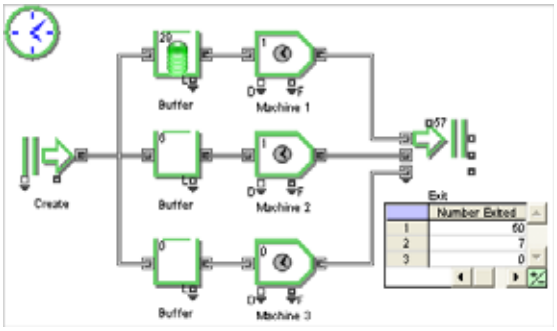
You can also model this situation in the opposite manner, by having all the operation blocks process items and then shut one or more of them down under certain conditions. If you do this, items may be trapped in the shutdown operation until processing resumes.

When you bring a system on line, it may cycle on and off too frequently. See "Bringing an activity on-line" on page 325 for some methods for avoiding this.

Balancing multiple output lines

Operations are often preceded by queues before each operation, such as a staging area for each machine (as compared to the single staging area for all machines as in the "Explicit Ordering model" on page 306.) The location and ordering of placement of queues in a model can affect how the model performs.

Buffering Operations model
A model with queues before each operation could look like:



Buffering Operations model: Parallel processing with buffering

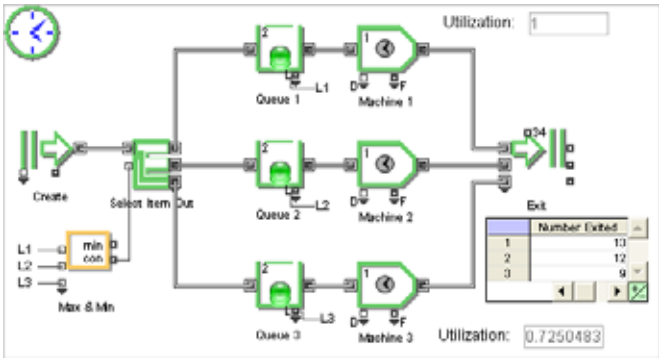
In the model, the queues have a maximum queue length of 30 each. The first queue that was connected will receive all the items until that queue reaches its maximum, then the next queue will start to fill (unless the first machine kept up with the flow of items, in which case the next queue will never receive any items). In the Buffering Operations model, for example, most of the items go to the top queue and get processed by Machine 1 and none of the items go to the bottom queue to be processed by Machine 3. This is rarely what you want.

To even out the use of the machines, use a Select Item Out block set to select machines sequentially, as shown in the “Sequential Ordering model” on page 305, and place a queue before each machine. Note however, that if the machines work at different speeds, this will cause the queue of the slowest machine to fill more rapidly than the other queues.

☞ A green bar across its top indicates that a Queue is set to something other than infinite capacity. For instance, the Queues in this model are set to hold a maximum of 30 items and therefore have green bars across their tops.

Output Line Balancing model
A better method than the Buffering Operations example would be to check the length of the waiting line in each queue and give the next item to the queue that is shortest, causing the

queue lines to be balanced. The Max & Min block (Value library) connected to a Select Item Out block is excellent for this.



Output Line Balancing model: Choosing the shortest queue

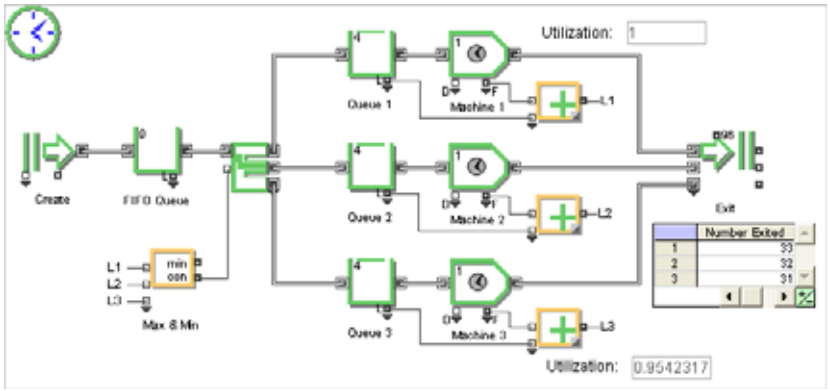
In the model, the Max & Min block tells the Select Item Out block which queue line is shortest and thus which queue to hand the next item to. The Min & Max block is set to *Output the: minimum value* and *Top input connector # is: 1*. With these settings the block's top input (L1) is number 1 and the con output reports which of the inputs (L1, L2, or L3) has the lowest value, indicating the shortest queue. The dialog of the Select Item Out block is set to *Select output based on: select connector* and *Top output is selected by select value: 1*.

Compared to the Buffering Operations model shown earlier, in this model the number of items in the queues tend to be more balanced. However, the system is not as efficient as it could be since an item often goes to the queue for Machine 1 even though Machine 3 is idle. This happens because if all the queue lengths are equal, the Max & Min block will report the first connector as having the shortest queue length. In this case, Queue 1 has first priority for items and the Max & Min block is just looking at the queue lengths and is not considering whether or not the Activity is occupied.

Line Balance with Activities model

The previous model showed how to balance the queues. A more useful model would be to include information about the items being processed when selecting the shortest queue. You can do this by adding the value of an Activity's F (full) connector to the queue length. The full connector is 1 when the activity is at capacity and 0 otherwise. The resulting model, Line Bal-

ance with Activities, prevents items from going to a queue followed by an occupied activity when other activities are idle.



Line Balance with Activities model

As in the previous model, the queue lines tend to be more balanced than the Buffering Operations model; but this method makes more efficient use of the Machines.

☞ Yet another option would be to use a Workstation block to replace the Queue and Activity blocks. The Workstation can report all of the items in its internal queue and activity through its Length connector.

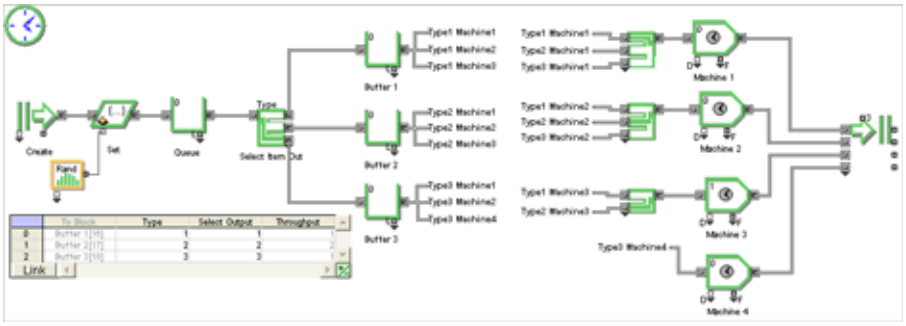
Machines that can only process certain types of items

A typical assembly line can handle more than one type of item at a time. For example, you may have three stereo models being assembled on a single line. Most of the assembly is identical, but a few different parts are used at different points. Unfortunately, some machines in such a heterogeneous assembly line cannot work on particular models being assembled. The method for accomplishing this is a combination of splitting items into different paths to establish the different "types" of items, then recombining the paths appropriately for the different machines. The Select Item Out block and the Select Item In block are handy in such situations.

Processing by Type model

Assume there is an assembly line where each item has an attribute called *Type* that is either 1, 2, or 3, depending on the type of item it will be. At one step of the assembly process, there are four machines. Two of the machines can work on all three types, but one of the machines is old

and can only work on types 1 and 2, and the other machine can only work on type 3. The model is shown below:



Processing by Type model

A Set block assigns a Type attribute to each item. The empirical table in the Random Number block (Value library) indicates that there is a 50% probability that the item will be Type 2, and 25% probability that it will be either Type 1 or 3.

The Select Item Out block is set to *Select output based on: property*. It looks up the value of the Type attribute (1, 2, or 3) and selects the appropriate output (1, 2, or 3) based on entries in the block's options table, shown at right.

	To Block	Type	Select Output	Throughput
0	Buffer 1[16]	1	1	1
1	Buffer 2[17]	2	2	2
2	Buffer 3[18]	3	3	1

Link

Options table in Select Item Out block

Notice the use of the queues as buffers in the above model. They are used to store the items by type, with the top queue for Type 1, etc. Without the queues, the whole line could be blocked, depending on the order in which items arrive. For example, if the first three machines are all processing a Type 1 item, and a Type 1 item is the next to exit the Select Item Out block, blocking occurs until one of the machines is finished with its item. The fourth machine will not be able to pull in an item until one of the other machines finishes processing and pulls in the new Type 1 item. Even then, it will have to wait until a Type 3 item is output before it can process anything.

☞ Named connections are used to simplify the look of this model. Without these, there would be a spaghetti of connection lines connecting the Queue blocks to the Select Item In blocks. Another option for organizing the model would be to use Throw Item and Catch Item blocks to route the items to the appropriate machines.

Discrete Event Modeling

Processing

Using activity-type blocks to cause and control processing

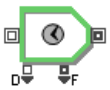
This chapter will discuss different ways to use activity-type blocks and how to control processing time and the availability of items and resources. It will cover:

- Processing in series and in parallel
- Setting the processing time
- Bringing an activity on-line
- Interrupting processes: preemption and shutdowns
- Multitasking
- Kanban systems
- Material handling and transportation blocks

☞ The models discussed in this chapter can be found in the folder Examples\ Discrete Event\Processing. That folder also contains some subfolders, as indicated in the relevant sections of this chapter.

Commonly used blocks

The following blocks will be the main focus of this chapter. The block's library and category appear in parentheses after the block name.



Activity (Item > Activities)
Processes one or more items simultaneously; outputs each item as soon as it is finished. Can also be used for multitasking.

Convey Item (Item > Activities)
Moves items on a conveyor from one block to another. Has dialog settings to define whether the conveyor is accumulating or not, what the length of the item is, and how far and how fast the item moves.

Create (Item > Routing)
Creates items or values randomly or by scheduled. Can be used to control shutdowns for the Activity block.

Shutdown (Item > Resources)
Used to control shutdowns for an Activity (Item library) or Valve (Rate library). Useful for setting random or constant time-between-failures (TBF) and/or time-to-repair (TTR).

Transport (Item > Activities)
Moves items from one block to another. Has dialog settings for defining how fast and how far the item moves.

Workstation (Item > Activities)
Acts as a FIFO queue combined with an Activity block, holding and processing items. Takes in one or more items at a time, holds them in FIFO order, processes them simultaneously, then outputs each item as soon as it is finished.

Systems and processes

Systems encompass one or more processes, which are a series of activities that achieve an outcome based on the inputs. Some examples of processes, the events that might drive them, and the items that flow through or are consumed by the process are:

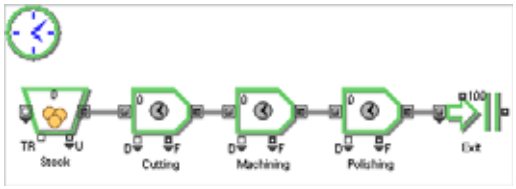
Process	Event	Item(s)	Activity
Planning strategic directions	Plan implementation	Decisions	Planning meetings
Developing a new product	Employee has a new product idea	A prototype	Document the specifications
Manufacturing a product	Receipt of raw materials	Parts, labor	Assemble the parts
Sales fulfillment	Customer orders goods	An order, or the goods themselves	Process the order, ship the goods
Call center support	Customer calls on telephone	Telephone call	Route call to technical support
Processing an insurance claim	Claim is received (or accident occurs)	Claim	Review the claim
Emergency room admitting	Accident	Patients, medical personnel	Assess incoming patients (triage)
Regulating traffic	Traffic light changes	Cars, pedestrians	Cross the street
Computer network	Packet is transmitted	Packet of data	Communication
Material handling	Arrival of AGV	Parts, AGVs	Load part onto AGV
Hiring employees	Company wins contract	Employees	Interview potential candidates
Completing an expense report	Employee finishes trip	Report	Prepare and file report
Writing a contract proposal	Request for proposal is issued	Proposal	Research the requirements
Approving a loan	Customer submits application	Application	Review credit history

☞ The following discussions most often refer to the Activity block for processing. However, the Workstation block can often be used instead of a Queue and an Activity, and the Convey Item and Transport blocks are also useful for simulating processing.

Processing in series

Serial processing occurs when items flow from one activity to another, where each activity performs one required task on the item, out of a series of required tasks. This is most common in manufacturing activities, order entry, or service-intensive situations. A simple example of

serial processing is an assembly line, where several processes are performed on one part prior to shipment.



Serial Processing model

Since there are many machines in series without buffering queues between them, it is possible that items will be not be able to leave one machine because the next machine will still be busy; this is known as *blocking* (as discussed in “Blocking” on page 281. Serial processes can cause the entire operation to be slowed to the speed of the slowest activity. This will cause utilization to increase by the amount of time that the item is blocked. If this doesn’t accurately represent your process, put a queue in front of each machine to represent a holding area, as shown in “Select Item Out dialog” on page 300.

Processing in parallel

It is common in industrial and commercial systems for there to be multiple activities working in parallel, each representing the same task being performed. For example, you might have five machines that can each process parts arriving from the stockroom. Or three bank tellers who are available to wait on customers. With the blocks in the Item library, there are many ways to route items to parallel activities.

Remember that, unless items are purposefully duplicated in the model, they can only follow one path at a time.

Parallel processing using one block

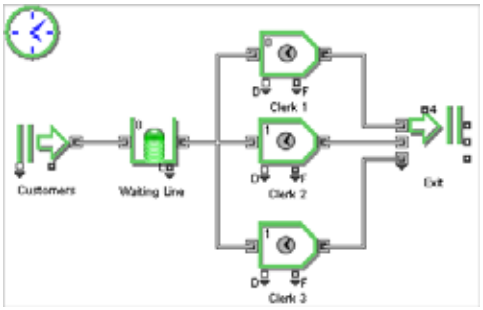
When you do not need to show each activity as a separate block, you can choose that the Activity block represent several operations that occur in parallel. This is accomplished by entering a number greater than one for the *Maximum items in activity* field in the block’s Process tab.

The Activity block can take in items (up to the specified maximum) and process them for a specified time starting from when they arrive. The item with the shortest time in the block (based on the item’s arrival time and how long it takes to process) is passed out first. For example, you could use the Activity block to represent a supermarket where customers arrive at different times and take varying amounts of time to shop. Customers who arrive early or who only shop a little will leave first; customers who arrive later or shop a long time will leave later.

Simple parallel connections

You can also use multiple Activity blocks in a model, each of which represents a process that can accept items in parallel with the other Activity blocks. The simplest way to hand out items to separate parallel activities is by creating connections between the output of the collection point and the inputs of each Activity. This causes ExtendSim to pass items to the first available Activity block. However, if more than one block is free when an item is ready, it is not obvious

which block will get the item. For instance, a Queue that holds items for three Activity blocks would look like the model below.



Simple Connections model

☞ If two machines are free when an item comes out of the queue, the machine that was first connected will get the item. With simple parallel connections, even just disconnecting and then reconnecting a connection could change the order of activities getting items.

⚠ Unless it is completely unimportant in the model, you should always explicitly state the ordering for parallel activities using the Select Item In and Select Item Out blocks. See “Items going to several paths” on page 299 for examples of how to control the flow of items to parallel processes.

Setting the processing time

Activities involve a processing time or delay that is the amount of time it takes to perform a task. Processing time can be static or can vary dynamically depending on model conditions. It can be random, scheduled based on the time of day, customized depending on the item that is being processed, or any combination of these.

You model the processing or delay time explicitly using the Activity or Workstation blocks or implicitly by specifying the length and speed in the Transport or Convey Item blocks. The Activity block is most frequently used to represent a process or operation, and is illustrated in most of the examples for this module

☞ The models discussed in this section can be found in the Examples\Discrete Event\Processing\Time folder.

Processing time for an Activity

In the Process tab of the Activity block you can select that the delay is:

- *A constant*. This uses whichever number is entered in the Delay (D) field. See also the discussion of “Fixed processing time” on page 320.
- *From the D connector*. The processing time is the value at the D input, overriding any value initially entered in the Delay (D) field. For example, see “Fixed processing time” on page 320 and “Scheduled processing time” on page 320.
- *An item’s attribute value*. With this option, you can select an attribute to control the processing time. This is illustrated in “Custom processing time” on page 322.

- *Specified by a distribution*. This choice provides a random processing time, based on the distribution and its arguments selected in the dialog. An example of this is shown in "Random processing time" on page 321.
- *From a lookup table*. This choice allows you to use an attribute value to specify the parameters of a random distribution. With this option, two popup menus and a table appear. The first popup is for choosing a distribution; the second is for selecting an attribute. The table is where the processing time for each type of item is characterized. with each row containing a different set of arguments for the selected distribution. As each item enters the block, its attribute value identifies which row in the table the item is associated with, and thus what its processing time is based on. This option is illustrated in the "Simulate Multitasking Activity model" on page 337.

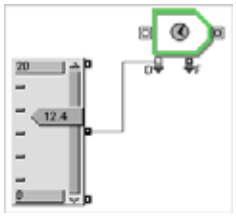
Processing time for other activity blocks

The Workstation block works much like a Queue combined with an Activity block. It has most of the same options (shown above) for setting the processing time as the Activity. (Because of its internal queue, the Workstation does not have a D input connector.) The Convey Item and Transport blocks calculate an implicit processing time based on the settings for speed and length entered in their dialogs. They are discussed in "Transportation and material handling" on page 338.

Fixed processing time

Set the delay in the Activity dialog if the delay doesn't change and you know how long it is. For example, if Machine A always takes 5 minutes to process parts, enter the value 5 as the processing time in the Activity's dialog. This is most common in the early stages of model building when you use constant parameters to get repeatable results.

Another method for having a fixed or constant processing time is to connect to the D input of the Activity block. For instance, you can connect from a Constant block (Value library) or a Slider control, as shown at right. If you use the Slider, you can manipulate it with each simulation run, or within a simulation run, to see the effect of various processing times.



Slider control used to set Activity's processing time

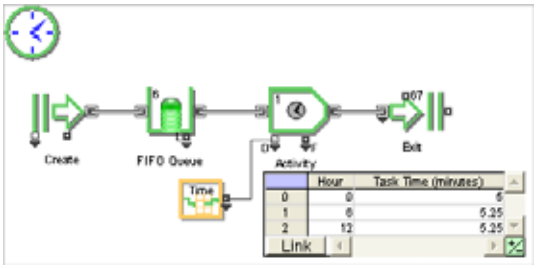
Connecting to the Activity's D connector overrides any manual entry in the Delay (D) field.

Scheduled processing time

If a process takes a specific amount of time under most conditions, but takes another amount of time if the conditions are different, you can schedule the processing time. This is common when simulating worker performance, where output could be a factor of time.

Scheduled Time model

For example, assume that a worker normally takes 5 minutes to perform a task, but takes 5.25 minutes to do the task after doing it for 6 hours. To do this, connect a Lookup Table block (Value library) to the D input of the Activity block as shown in the following model.



Scheduled Time model

The Lookup Table block is set to *Lookup the: time*. Data that represents the worker's day is entered in the *Hours* column; the time to perform the task is entered in the *Task Time* column. As indicated in the Lookup Table's dialog, a portion of which is cloned onto the model worksheet, the time to perform the task changes at hour 6.

Lookup Table block's time units

Notice that the time unit for the Hours column in the Lookup Table's dialog is hours and that the time the worker takes to perform the task (Task Time column) is in minutes. The time units for the model are hours, so the first column must be in hours. However, since the output of the Lookup Table block is connected to the D connector of the Activity block, the Task Time column should be defined in the same time units (minutes) that are used for the Activity's Delay parameter. For this model, at a particular hour of the day (Hour column) an activity will take a specified number of minutes to perform the task (Task Time column). Time unit consistency is discussed in "Choosing time units for the columns" on page 261.

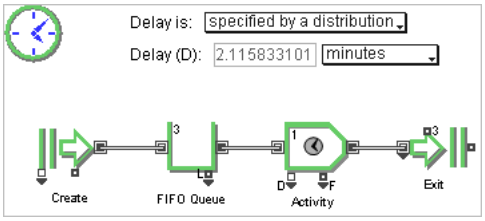
Random processing time

A common requirement for activities is to set a random processing or delay time. This is easily accomplished by selecting a distribution in an Activity block.

Random Activity model

In the dialog of the Activity block, select the appropriate distribution, for example Normal, and specify the value of the parameters, such as a mean of 2 and a standard deviation of 0.2. The

processing time will then be normally distributed and the Activity block will process each item for approximately 2 time units.



Random Activity model

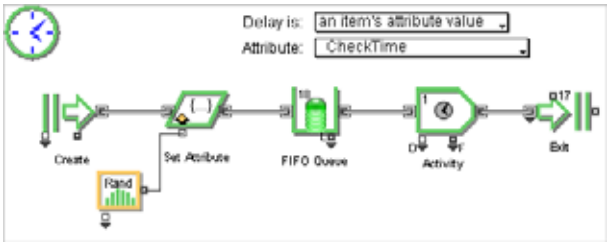
For more information, see “Constants and randomness” on page 7, “Random numbers” on page 197, and “Probability distributions” on page 198.

Custom processing time

Attributes can be used to specify how long a specific item will be processed. This is a very powerful feature since the Activity block can recognize each item’s processing time and behave accordingly.

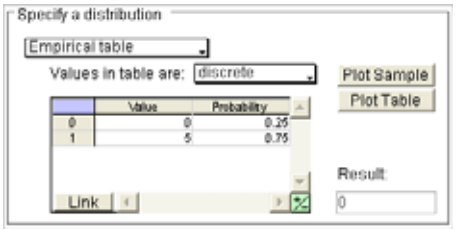
Custom Time model

In the simplest case, set an item’s attribute value to the desired amount of processing time, then use the Activity block to read the attribute value and process the item for that period of time.



Custom Time model

The Custom Time model uses the Set block to set the value of an attribute called CheckTime to the amount of time it takes to check the item. Items that need a final check have a CheckTime attribute value of 5, for instance, and items that ship unchecked have an attribute value of 0. That value (0 or 5) is provided by the Random Number block (Value library) using an Empirical distribution where 25% of the items have a value of 0 and 75% have a value of 5.



Custom processing time

All items then go through the checking step, easily represented by an Activity block. In its dialog, this block indicates that the *Delay is: an item’s attribute value* and the attribute is *CheckTime*.

☞ Although items with a CheckTime value of 0 will not be processed by the Activity block, they may be delayed in the Queue (which is set to FIFO order) while a preceding item undergoes checking.

Implied processing time

Some ExtendSim blocks allow you to specify distance, speed, or other factors that indirectly result in a processing time. For example, the Convey Item block allows you to enter an item length and the length (in feet or meters) and speed (in feet or meters per time unit) for an accumulating or non-accumulating conveyor.

These settings in Convey Item and Transport blocks result in delay times for items. For example, if you set the Transport to be 10 feet long with a speed of 1 foot per time unit, it will take 10 time units for an item to travel to the next block.

For more information about using the Convey Item and Transport blocks, see "Transportation and material handling" on page 338.

☞ If the *Metric distance unit*s preference is selected in the Edit > Options dialog, length units in these blocks are set by default to meters. Otherwise, their length units are in feet.

Cumulative processing time: time sharing

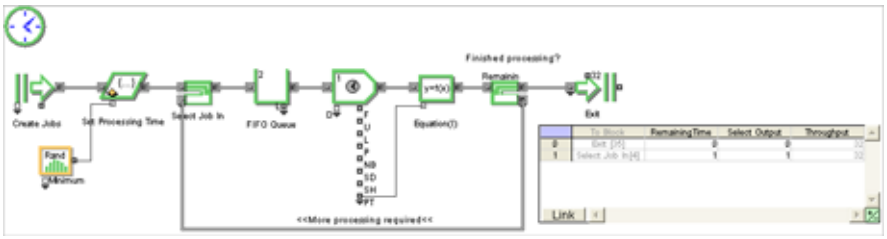
The prior section discussed setting an item's attribute to the time required by a specific activity. You can also set its attribute value to the total processing time required, then route the item to a series of activities, each of which performs one part of the processing, until the attribute value is reduced to zero and the item is fully processed. This is common when there are several stations with different processing times, any of which can process the item. Or when there is one machine that processes each item for a specified time, then passes the item to another section for further processing, and the item must be returned to the original machine for finishing.

You can do this using an activity block, building the model such that the attribute value is decreased by the amount of processing time. In this situation, each activity subtracts its processing time from the attribute value, so that the value represents the remaining processing time. Use a Get block after each activity to determine if the item was fully processed or not, and therefore whether it should proceed to the next activity or be routed out of the line.

Time sharing occurs when an activity processes an item, sends it back to a queue for a short period, then processes it again until the required processing time is completed. This is common for computer networks and telephone communication systems. In these systems, time is specified in small fractions of a second, there are a lot of jobs that must be processed at the same time, and there are only a limited number of processors to do the work. In time sharing, instead of each job being processed sequentially, all jobs are processed at what appears to be the same time. However, each job is processed a small bit at a time, and a given job may have periods in between where nothing is happening to it. Since the time units are so small, the periods when there is no processing of a specific job are typically not noticed, and each job appears to be processed continuously.

Cumulative Time model

A simple time-sharing model has one activity that processes each job for a short period of time, then sends the job back to the queue so it can be processed again, until the total required processing time for that job has elapsed.



Cumulative Time model

In the model, items (jobs) are generated randomly and the Set block attaches a RemainingTime attribute to each job generated. The Random Number block determines the initial value of that attribute, 1, 2 or 3 milliseconds, which represents the total processing time required for each job. The Activity block processes the job for a fixed time (1 millisecond). An Equation(I) block then subtracts the amount of time spent processing (Process Time, or PT) from the RemainingTime attribute.

	To Block	RemainingTime	Select Output	Throughput
0	Exit [35]	0	0	32
1	Select Job In[4]	1	1	32

Link

Table in Select Item Out block

Once the remaining processing time has been calculated, the decision to route the item back to the Queue or to the Exit is made in the Select Item Out block. Jobs with a RemainingTime of 0 are routed through the top output and exit the simulation; items that have 1 or more millisecond of processing time left are routed back to the Queue for further processing. Notice that the table in the Select Item Out block (shown above) only indicates explicitly what happens if the remaining time is 0 or 1. However, the block's dialog is set to *Invalid Select value: chooses bottom output*. With these settings, an item with a RemainingTime value of 0 will exit the top output. If the value is 1 the item will exit the bottom output. And any value other than 0 or 1 will also exit from the bottom output.

☞ The time units for this model are integers representing milliseconds, because the Select blocks expect integer values for comparison and will truncate non-integer values. (For example, the value 0.003 would be truncated to a zero.) When using non-integer values for the processing time, convert the attribute values to integers before they go to the Select Item Out block. You can do this with a Lookup Table block (Value library).

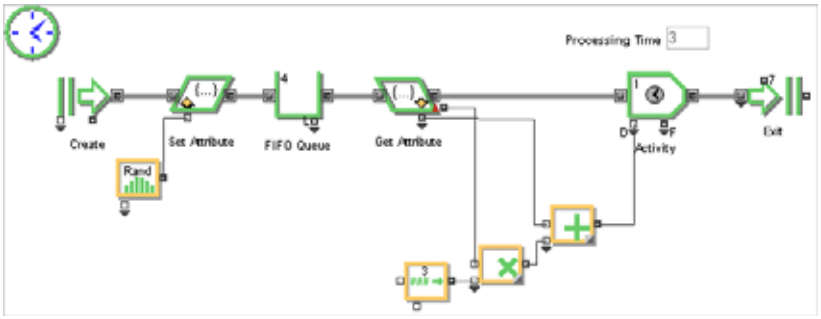
Adding setup time

Every process is not 100% efficient. For instance, it is common in manufacturing for a machine to be reconfigured when the type of item it is processing changes. This reconfiguration usually takes additional time beyond the normal processing time. In the example below, the processing time (and the part type) is determined by the attribute values, similar to what was shown in "Custom processing time" on page 322. However, this model requires an additional *setup time* whenever the type of item changes.

☞ Setup time can add significant delay to the processing of items. For an example showing how to minimize setup time, see "Minimizing setup" on page 286.

Setup Time 1 model

In the Setup Time 1 model, a Process attribute is assigned to each item. The attribute value represents how long items should be processed for – either 1, 3, or 5 minutes, depending on probabilities entered in the Random Number block (Value library). The Δ (delta) output connector on the right of the Get block signals when the value of the Process attribute has changed, indicating that the current item is of a different type than the previous item. This information is used to add a setup time to that item's processing time, resulting in a longer total delay time for the first item of a new type, each time the type changes.



Setup Time 1 model

The Δ (delta) connector outputs 0 (for False) as long as the value of the Process attribute stays the same and outputs 1 (for True) when the attribute value changes, indicating the arrival of a new type of item. The Constant block (Value library) specifies a setup time of 3 minutes. As long as the attribute value does not change, the Constant block is multiplied by 0, adding nothing to the normal processing time. When the attribute value changes, the Constant value is multiplied by 1, and the 3 minute setup time is added to the value of the Process attribute to determine the processing time for that new item. You can see this if you run the simulation – the processing time is cloned onto the worksheet and will be 1, 3, or 5 minutes for most items but 4, 6, or 8 minutes for the first item that is of a new type.

Notice that each item still has its original attribute value. You do not change the attribute value in this model, it is only used to determine whether the item type has changed and thus whether the item requires a setup time. The processing time (whether equal to the attribute value or equal to the attribute value plus the setup time) is input at the D connector. The Activity block processes based on the value at the D connector, not directly based on the attribute value.

☞ While the model above shows the mathematics explicitly, the Equation block (Value library) can also be used to specify the setup time. This is illustrated in the model Setup Time 2.

Bringing an activity on-line

As discussed in the following sections, many systems, activities, or operations can be brought on and off-line based on a schedule or on the current conditions of the system.

☞ The models discussed in this section can be found in the Examples\Discrete Event\Processing\Bring On-Line folder. The Shift block is discussed starting on page 372 and models using the Shift block are located in the folder Examples\Discrete Event\Resources.

Scheduling activities

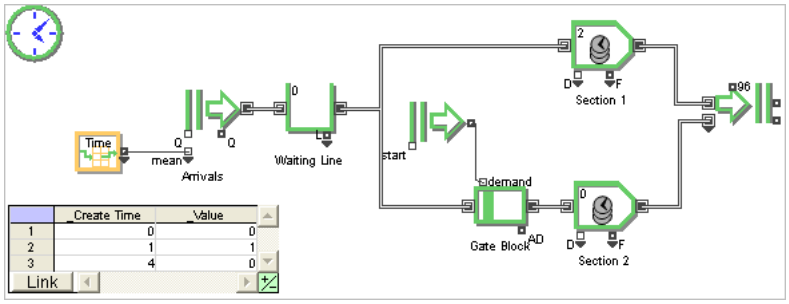
Activities don't always occur randomly; they can be scheduled. This is common when you bring an activity on-line based on the time of day. This could be represented by a schedule in

the Create (Item library) or Lookup Table (Value library) block connected to and controlling a Gate block (Item library), by using a Create block to schedule the capacity of an Activity, or by using a Shift block to control an Activity. As seen below, the Scheduling Activities 1 model uses a Gate block and the Scheduling Activities 2 model schedules an Activity's capacity. Shift blocks are discussed in "The Shift block" on page 372.

☞ So that you can compare both the Scheduling Activities 1 and Scheduling Activities 2 models, they have the same random seed, as seen in their Run > Simulation Setup > Random Number tabs.

Scheduling Activities 1 model

The following example shows how to schedule the availability of a portion of an operation using a Gate block. The Scheduling Activities 1 model represents a diner that opens at 10 AM and closes eight hours later. Customers arrive exponentially throughout the day, with most customers arriving during the lunch period, which is from 11 AM until 2 PM (from hour 1 to hour 4). There is 1 dining section that can service 5 people at a time throughout the day. A second dining section that can also service 5 people at a time is available only for the lunch shift.



Scheduling Activities 1 model

The scheduling of random customer arrivals is accomplished by entering values in a Lookup Table block (Value library), set to *Lookup the: time*. The output of the Lookup Table block is connected to the *mean* input of a Create block, which generates customers exponentially. As seen in the table to the right, the Lookup Table outputs a smaller value from time 1 to time 4. The output represents the average time *between* arrivals, so arrivals will occur more frequently from time 1 to time 4.

	Time	Output 1
0	0	0.1
1	1	0.05
2	2	0.04
3	3	0.08
4	4	0.2
5	8	0.2

Link

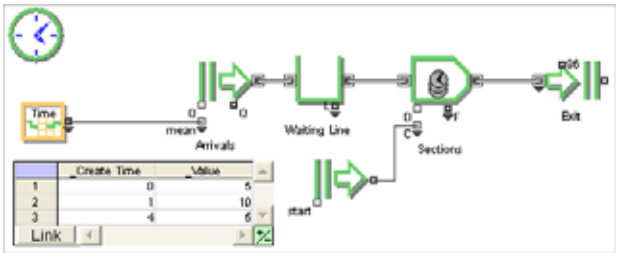
Customer arrivals

Opening the second dining section is accomplished by connecting a Create block to a Gate block, allowing customers access only during certain hours.

The Gate block is set to set to *Mode: conditional gating with values*, so that it only allows items through when its demand connector is activated by a value. This is accomplished by connecting a value connector, such as the one on the Create block when it is in *Create values by schedule* mode, to the demand connector. As long as the value connector is true (outputs 1), the Gate stays open; when the value is 0, for false, it closes. Running the model with animation on shows that, even though the queue length is increasing, the Gate shuts down after three hours.

Scheduling Activities 2 model

As an alternative to the preceding example, you could have used the C connector on the Activity block to control its capacity, simulating the opening of the second dining section.



Scheduling Activities 2 model

Each dining section has the capacity to serve five customers at once. By doubling the capacity of one Activity block during the period between 11 AM and 2 PM you can model both dining sections being open. This is accomplished by connecting the value output of a Create block, set to *Create values by schedule*, to the *C* input on the Activity. In the model, the portion of the Create block that controls the capacity of the Activity is cloned onto the model worksheet.

Shift block used to schedule

Yet another approach to scheduling an Activity would be to use a Shift block to control it. For instance, the Shift block could contain the same information as the Create block that is connected to the C input on the Activity in the Scheduling Activities 2 model, above.

For more information, see the section titled “The Shift block” on page 372.

Controlling the flow of items to an activity

As discussed in “Scheduling activities” on page 325 you can have an activity start based on the time of day. Some methods of adding an additional activity to a model can cause the new activity to cycle on and off frequently. You may not want this to occur, as it can result in higher start up costs, increased machine wear and scrap production, and excess energy consumption. Instead, you can add some *hysteresis* and have the activity stay on to process a number of items, or stay on for a period of time.

When bringing a system on-line, there are two main ways to control the flow of items to an activity:

- Specify the number of items that will be processed
- Specify the amount of time the activity will be on-line

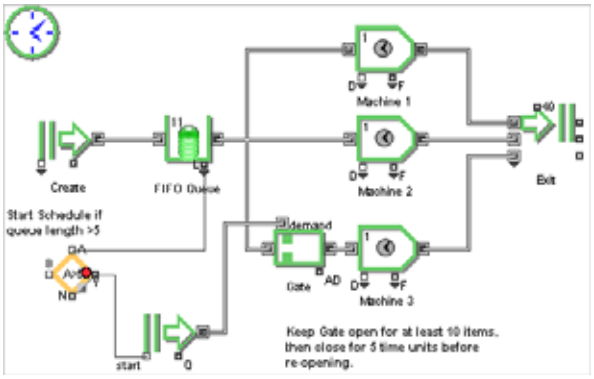
☞ You can view and report the item contents of Activity blocks at any time. Please see “Item Contents of queues and activities” on page 210.

Fixed number of items

Instead of having a system cycle on and off, you may want to keep the optional activity running. For instance, you can keep a machine on to process a particular number of items, even if the waiting line for the other machines is below the threshold that originally activated it. This reduces the number of times the machine turns on and off.

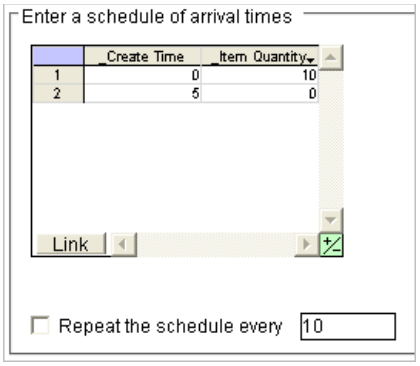
Fixed Items model

To keep an activity running until it has processed a fixed number of items, add a Create block between the Decision and Gate blocks to the model discussed in "Conditional routing" on page 308. (The Conditional Routing model is located in the Examples\Discrete Event\Routing folder.) The resulting model is named Fixed Items, and is shown here.



Fixed Items model

In this model, the optional machine will be brought online if/when the queue exceeds five items. The Y output of the Decision block (Value library) is connected to the start connector of a Create block, which is set to *Create items by schedule*; the schedule is shown at right. The Decision block outputs a true value (1) at Y if the queue length is 5. When that happens, the Create block (Control the Gate) starts its schedule and puts out a single item with a quantity of 10 to the demand input on the Gate block. This causes the Gate to stay open until 10 items pass through the block.



Activating "demand" with an item

There are two items of special note in this model: how the *start* and *demand* connectors are activated.

- Since the Create block's *start* connector is connected, and since the block is in *Create items by schedule mode*, the block's schedule runs in relative simulation time (begins its schedule relative to when start is activated), as explained in the section "The Start connector" on page 262. Once start is activated (gets an value ≥ 0.5) it causes the entire schedule to happen; messages from the Decision block to the start connector are ignored until the schedule is complete. The second line of the schedule means that the Create block will also ignore any start messages for 5 time units after the schedule has been completed. This provides hysteresis and allows the machine to process items (and hopefully reduce the buffer length) before the sequence is activated again. If the schedule did not include this pause, the Activity block could be activated constantly.

- The demand connector is activated when it gets an item, causing the Gate to open and allow items through. The number of items allowed through before the Gate closes is determined by

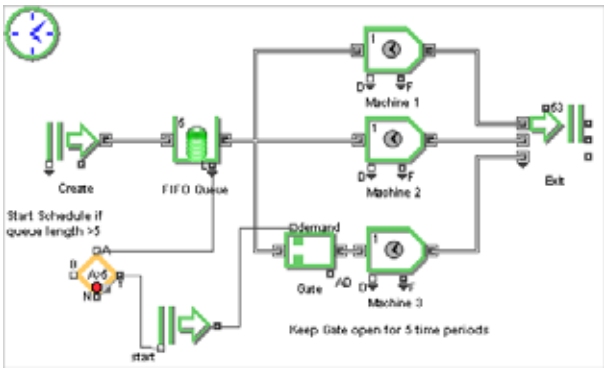
the quantity of the item at *demand*. For instance, each item with a quantity of 10 creates a demand for 10 items before the Gate will close.

Fixed period of time

You may want to keep the optional machine on for a particular length of time instead of for a certain number of items.

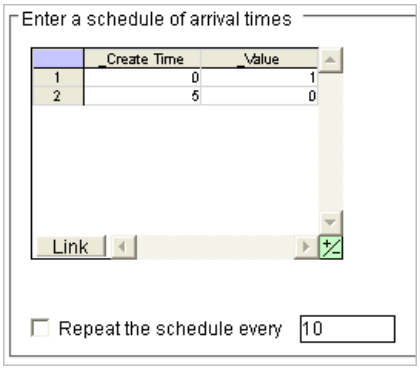
Fixed Time model

To do this, use the Create block, set to *Create values by schedule*, to output *values* to the demand connector on the Gate block.



Fixed Time model

In the table in the Create block's dialog, the first line has 0 for the output time and a value of 1. The second line has the time you want to turn off the optional machine, and a 0 for the value. For example, to keep the optional machine on for five minutes, you enter the values as shown at right.



Activating "demand" with a value

Once start is activated, the demand connector will receive a value of 1 (True). After 5 time units have passed, the value at demand will change to 0 (False). Because you are connecting a value output to demand, the Gate block will stay activated as long as the value the demand connector receives is greater than or equal to 0.5, which in this example is for 5 time units.

Interrupting processing

In discrete processes, it is common for interruptions to occur. This could happen for any number of reasons, such as the arrival of an item that has a higher priority for processing, random machine failures, planned shutdowns, the occurrence of a higher priority event, and so forth. Interruptions are of two kinds, preemption and shutdown.

- *Preemption* occurs when an Activity block is told to prematurely end one or more items' processing. When this occurs, the Activity immediately sends the preempted items out of the block through an alternate item output connector.

- *Shutdown* occurs when processing is suspended for one or more items currently in an Activity block. Items that have been shut down may or may not have their processing completed when the shutdown ends. In any case, they are either discarded or leave through the normal item output connector.

The Activity block has a Preempt tab for specifying what to do when there is preemption and a Shutdown tab for controlling what happens when the block gets a message to shutdown. The Convey Item block can be shutdown by reducing its speed to zero. The Shutdown block is most commonly used for shutting down an activity.

☞ Preemption and shutdown are discussed in the following two sections. The models for those sections are located in Examples\Discrete Event\Processing\Preemption and Shutdown folder.

Preemption

Preemption occurs when a signal is received at an Activity block's PE (preempt) input, prematurely ending an item's processing by forcing it to leave through an alternate output.

In the Preempt tab you can specify that preemption occurs only if the block is already processing its maximum number of items and that the preempted item's remaining processing time be stored as an attribute for subsequent processing. Once preempted, the item's processing can be finished by another Activity, finished later by the original block, or never finished at all, depending on how the item is routed in the model.

PE input connector

Once preemption is enabled in the Preempt tab, the PE (preempt) universal input connector and an alternate item output connector appear on the Activity's icon. The connection to the PE connector can either be a value input or an item input, depending on what is selected in the Preempt tab. The type of connection to the PE input determines how preemption is controlled:

- Value connection. Based on which of the first four preemption options (discussed below) is chosen, the selected item or items will be preempted whenever a true (0.5 or greater) value is received at the PE input.
- Item connection. When a "preemption item" arrives at the PE connector, the Activity looks up the specified attribute value on the preemption item. The Activity then searches all items currently in processing, and any of those items with an attribute value equal to the one on the preemption item will be required to leave the block. In addition, the Activity block has an option to transfer attributes from the preemption item to the preempted items.

Preemption options

As discussed above, preemption occurs when a signal is received at an Activity block's PE (preempt) input. Settings in the block's Preempt tab determine which item or items must leave. Depending on whether the preempt signal is sent by a value or an item connector, the preemption options are:

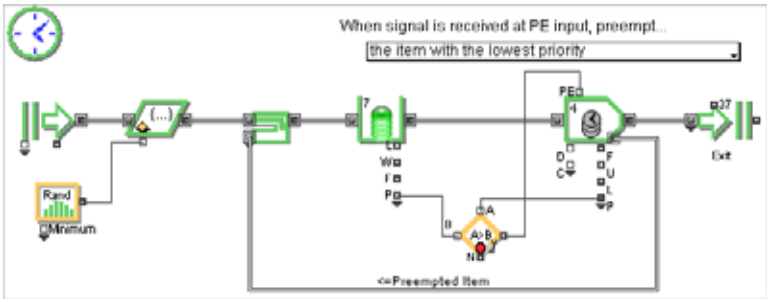
- The item that is closest to finishing
- The item that is furthest from finishing
- The item with the lowest priority
- All items currently being processed
- Only items with a particular attribute value

The first four options are only available when a value connection is made to PE; the last option is only available, and is the only choice, if an item output is connected to PE.

Preempting model

For instance, an Activity and Queue (set to *Sort by: priority*) can be used in conjunction with a Decision block (Value library) in such a way that lower priority items being processed *may* be preempted to make room for higher priority items as they arrive at the Queue. Note that it is not guaranteed that a lower priority item will be preempted. If "Preempt only if block is full" is checked, items will be preempted only if the Activity block is full.

In the Preempting model, the Queue reports the priority of the item that is about to leave and the Activity reports its lowest priority item; this information is sent to the Decision block. If it is determined that there is a higher priority item in the Queue than is being processed by the Activity, a True signal (a value greater than 0.5) is sent to the Activity's PE input. Notice that the Activity's dialog is set to *When signal is received at PE input, preempt... the item with the lowest priority* and to *Preempt only if block is full*.



Preempting model

Unless they are preempted, items arriving to the Activity block are processed for the time indicated in the block's dialog. In block's Preempt tab, *Store remaining time in attribute: remainingTime* is selected. If an item is preempted, the Activity attaches the remaining processing time to the item as an attribute named *remainingTime*. Since the Activity also has *Use this attribute as delay* checked, when the preempted item returns to the Activity block it will process only for the time indicated by the remainingTime attribute.

Shutting down

Employee breaks, equipment maintenance, inventory-taking closings, and tool failures all involve interruptions in activities for a period of time called *downtime*. If interruptions are significant, models should include provisions for shutting down activities to avoid overly optimistic predictions.

Shutdowns involve a temporary or permanent halting to the processing of items currently in the Activity block. The block's Shutdown tab has settings to determine which items should have their processing shut down, how long to interrupt the processing, and what to do with the items in an Activity when the shutdown occurs.

You can shut down activities at a scheduled time, such as for vacations or machine maintenance, or it can be a random occurrence, such as for equipment failures or emergency leaves. Activities can also be shut down based on some factor in the model, for instance when a downstream Queue is full. Like shutdown occurrences, the duration of the downtime can be a con-

stant value or a random number. A shutdown can also be used to block the entry of additional items while the shutdown is in effect.

SD input connector

Once shutdown is enabled in the Shutdown tab, the SD universal input connector appears on the Activity's icon. It is common to connect from a Create or Shutdown block to the SD input but connections can be made from other blocks as well. The input to the SD connector can either be a value connection or an item connection, depending on what is selected in the Shutdown tab. The type of connection to the SD input determines both which items are shut down and for how long:

- Value connection. This acts like an on/off signal. The entire block will be shutdown whenever a true (0.5 or greater) value is received at the SD input. This suspends the processing of all items in the block and stops new items from entering it. The Activity will stay shut down until the SD input gets a false (less than 0.5) value.
- Item connection. When a "shutdown item" arrives at the SD connector, the Activity will shut down the item or items currently being processed, as specified by the shutdown options discussed below. The duration of the shutdown is determined by an item's attribute or quantity as specified in the Shutdown tab; the value of that property on the shutdown item determines how long the shutdown will be in effect.

The Create block is used to schedule the shutdown for the "Scheduled Shutdown model" on page 333; the Shutdown block provides random shutdowns for random durations for the "Random Shutdown model" on page 335.

Shutdown options

As discussed above, shutdown occurs when a signal is received at an Activity block's SD (shutdown) input. Settings in the block's Shutdown tab determine which items currently in processing will be shut down. Depending on whether the shutdown signal is sent by a value or an item connector, the shutdown options are:

- All items currently in processing
- A randomly chosen item
- Items whose attribute matches the attribute at SD
- Entire block

The first three choices are only available if an item connector is connected to SD; "Entire block" is only available, and is the only option, if a value output is connected to SD. If "Items whose attribute matches the attribute at SD" is selected, the Activity looks up the specified attribute value on the shutdown item. The Activity then searches all items currently in processing, and any of those items with an attribute value equal to the one on the shutdown item will be shutdown.

Item options

Once a shutdown is in affect, the shutdown items are handled in one of four ways, as specified in the Shutdown tab of the Activity's dialog:

- The item can be discarded, such as when food is spoiled by the machine going down.
- The Activity can resume processing the item after the shutdown ends.
- An Activity can restart processing the item after the shutdown ends.

- The Activity can finish processing the item prior to shutting down, such as when the shutdown is part of scheduled maintenance and can wait until the item is finished.

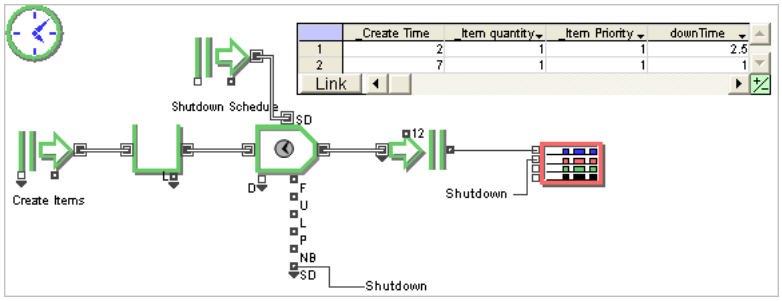
Items that are not discarded leave through the Activity's normal item output.

SD output connector

The Activity block's variable output connection contains an SD connector that can be used to relay shutdown status. Depending on how the Activity has been configured, this connector either outputs a 1 (one) while the Activity is down and a 0 (zero) when it is up, or it outputs the number of items that are currently shutdown. If a value connection has been made to the SD input, the SD output connector is set to 1 or 0; if an item connection has been made to the SD input, the SD output connector reports the number of items currently shutdown.

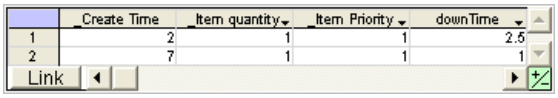
Scheduled Shutdown model

The Create block is often used to schedule an Activity to shut down. For example, the Scheduled Shutdown model schedules downtime for a machine by connecting a Create block's item output to the SD input on an Activity.



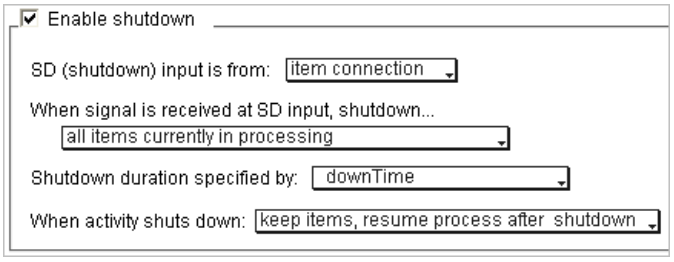
Scheduled Shutdown model

The "Shutdown Schedule" from the Create block's Shutdown tab is cloned onto the model worksheet and shown at right. It indicates that two items will be created, one at time 2 and one at time 7. Each item has a downTime attribute as seen at the top of the fourth column; the attribute's value is 2.5 for the first item and 1 for the second item. In the Create block's dialog, this maintenance schedule has been set to repeat every 10 time units.



Schedule in Create block

The Activity block is set to process one item at a time. Its Shutdown tab, shown at right, indicates that *all items currently in processing* will be shutdown when a "shutdown item" is received at SD, *downTime* is the name of the attribute that determines the duration of the



Settings in Activity block

shutdown event, and the block will *keep items, resume process after shutdown*.

☞ With this information and the settings in the Activity's Shutdown tab, processing will shut down for any item that is already in the Activity at time 2 and that item will not resume processing until time 4.5. Likewise, any item in the Activity at time 7 will be shut down and not resume processing until time 8, and so forth.

⚠ When shutdown is triggered by an item connection to SD, the selected items being processed by the Activity block will be shut down at the time and for the duration specified. However, because the block supports parallel processing, if items arrive to the block after the shutdown has been triggered, those items will be processed normally. For complete shutdown of the block, use a value connection instead of an item connection.

When the model is run, the plotter will show both the number of items processed (obtained from the Exit block) and the timing and duration of the shutdowns (obtained from the SD output on the Activity.) The SD output gives a value of 1 while an item is shutdown and a value of 0 while it is not.

The Shutdown block

The Shutdown block provides for dependent or independent failure policies and can be used to model the failure or scheduled downtime for a single or multiple components.

A shutdown event can either be random (e.g. a failure) or intentional (e.g. scheduled maintenance). The Shutdown block manages both types of events and broadcasts the overall down status as either a value or item signal. Typically these signals are used to directly control either the Activity (Item library) or the Valve (Rate library).

- Random failures are modeled on the Shutdown block's first tab where distributions can be selected for the Time-Between-Failure (TBF) and Time-To-Repair (TTR).
- Scheduled shutdowns are modeled either by connecting to the Shutdown's "sched" input or by going to the block's Options tab and linking the Shutdown block to a Shift block. Note that there are policy options allowing the modeler to specify the impact an intentional down event will have on random downs. For example, when a maintenance event is scheduled, should progress towards the next failure event be reset?

Whether random or scheduled, there are two types of down signals:

- Value down signals by default are "1" for down and "0" for up. However, on the Shutdown's first tab you can control what the up and down values should be. For example, if the Shutdown block is connected to the "R" connector on the Valve block, it would make more sense to have the down = 0 in order to close the Valve during a down state.
- An item down signal is generated by the Shutdown block once for every shutdown event. Each item has two attributes that indicate to the Activity which items to shut down and for how long. Typically, this option is used when you want to target specific items to shut down inside the Activity.

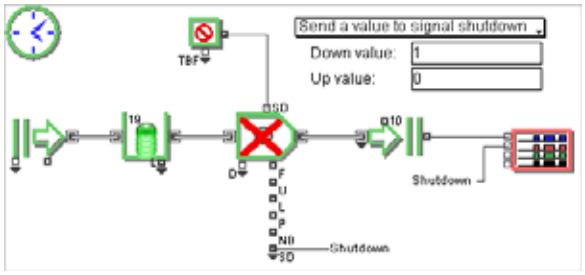
When a database table is used to specify TBF and TTR distributions, one Shutdown block can be used to model any number of component failures. The components (and their respective fail behaviors) that are modeled in one particular Shutdown block are typically related in some way. For example, you would want to use one Shutdown block to model the failure of a bicycle's tires, chain and brakes because all three components are only wearing out while the bike is being ridden, and if one component fails, the wearing on the other two stops. It would be

unnecessarily difficult to model the correct reliability behavior of this bicycle with three separate Shutdown blocks.

☞ For more detail, click the “Help” button in the lower left of the Shutdown block’s dialog.

Random Shutdown model

While the Scheduled Shutdown model on page 333 used a Create block to schedule when an Activity would be down, the Random Shutdown model uses the Shutdown block to halt processing on a random basis for a random amount of time. It does this by connecting the Shutdown block to an Activity block’s *SD* input.



Random Shutdown model

In this model the dialog of the Shutdown block is set to:

- Send a value to signal shutdown
- Output a Down value: 1 and an Up value: 0
- Use an Exponential distribution with a Mean: 9 for the Set time between failures (TBF) parameter
- Use a Triangular distribution with the settings Minimum: 1, Maximum: 3, and Mostly likely: 2 for the Set time to repair (TTR) parameter.

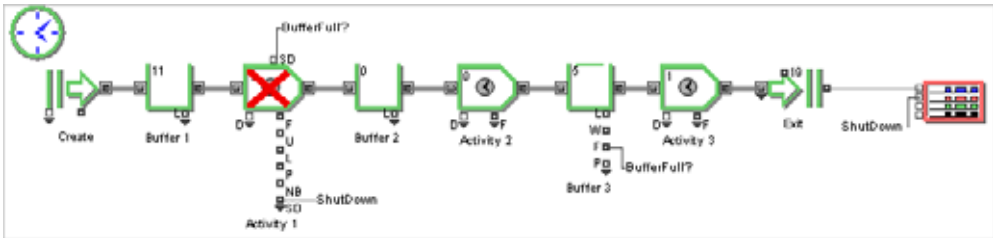
With these settings both the TBF and TTR are random. A true signal (a value of 1) from the Shutdown block will cause the Activity block to shut down the entire block, halting all items currently in processing and blocking any new items from entering. It is only when the signal switches from true to false (a value of 0) that the block will resume processing. In this case, the Activity will remain down for approximately 2 hours before coming back online.

Model-related shutdown

The Scheduled Shutdown and Random Shutdown examples presented earlier showed how to shut down a process in isolation from other events in the model. You can also shut down activities based on model factors such as the length of a waiting line or whether another activity is in process.

Explicit Shutdown model

The Explicit Shutdown model represents a buffer downstream from a machine that reaches a limit. So that the queue length will control shutdown events, the *F* connector from the Queue block is connected to the *SD* input connector on the Activity block.



Explicit Shutdown model

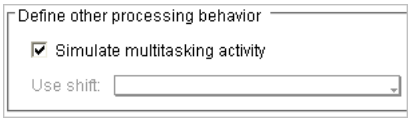
For this model the Queue's limit is specified in its dialog; whenever the Queue is full, the *F* connector outputs 1. The Activity's Shutdown tab indicates that the *SD (shutdown) is: value input connection* and that therefore the *Activity stays down until SD value < 0.5*. This causes the Activity to stay shut down for as long as the Queue is full.

This is an example of how to have downstream factors affect upstream activities. If you examine the model closely, you see that the last machine is processing so slowly that Queue 3 quickly reaches its limit of 5 items. Since a Queue cannot take in any more items while it is full, the middle Activity is blocked (cannot process a new item until an item is removed from Queue 3). However, the first Activity continues to process items, filling Queue 2. By explicitly shutting down the first Activity, you affect where items are stockpiled and which Activities are shut down when one of them is blocked.

☞ Please also see "The Shift block" on page 372 for examples of activity and resource allocation that are tied together and scheduled as "Shifts."

Multitasking

The Activity block has a checkbox in its Process tab that allows the block to simulate multitasking. Choosing this option means that the block's available time to process items (its Delay time) must be divided between each of the items in the block. This causes each item to take longer to finish processing and leave. Examples of multitasking include computer processors or a person who is working on multiple tasks at the same time.



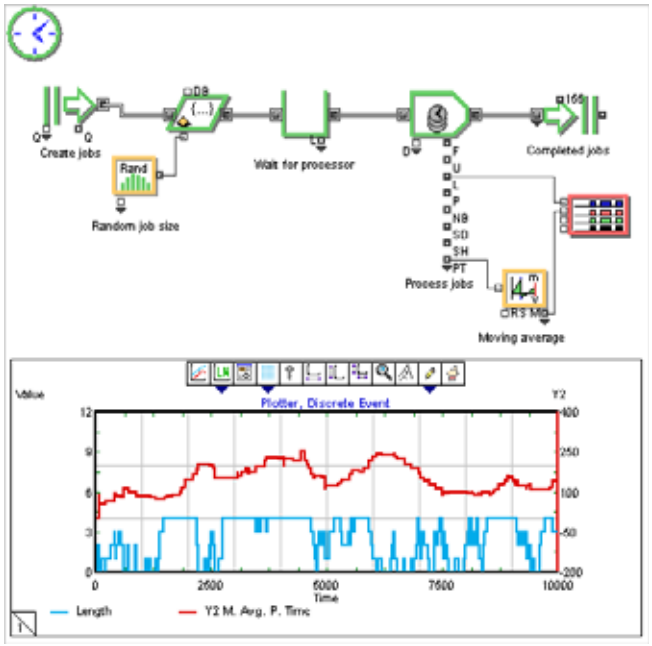
Choosing multitasking in an Activity

With multitasking, if only one item is in the block the actual processing time will be exactly the same as the original delay time specified by the block. If two items are in the block, each item will take twice as long as the original specified processing time. If three items are in the block, their processing times will be multiplied by three, and so forth. This is equivalent to situations where a single server or operator has to divide their available time between multiple customers or tasks.

The changes to the processing time occur dynamically as items enter and leave the block. When new items enter the Activity, the remaining delay times for all of the items in the block will become progressively longer. As processed items leave the block, the delays for the remaining items will become shorter.

Simulate Multitasking Activity model

In this example model, there are three kinds of jobs that are being processed on a computer. The computer must share its time between all the jobs it is simultaneously working on.



Simulate Multitasking Activity model

The Simulating Multitasking Activity creates a random number of each type of job (small medium and large), as determined by probabilities entered in the Random Number block (Value library). Each job has a Job Size attribute with the string value small, medium, or large. The job's processing time is defined by a distribution and entries in a table in the dialog of the Activity block, which is set to *Delay is: from a lookup table*. (For a description of this setting, see "Processing time for an Activity" on page 319.)

Running the model shows that the moving average processing time increases as the number of items in the Activity increases.

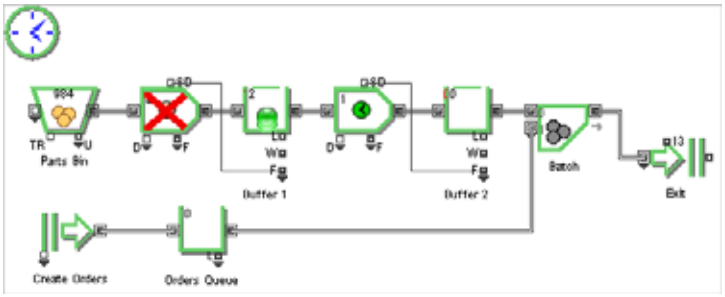
☞ Changing the capacity of the Activity changes the number of items allowed in the block, but it does not change the calculation for the delay time. Thus, if "Simulate multitasking activity" is enabled, increasing the capacity will not necessarily increase the throughput rate of the Activity block.

Kanban system

A kanban just-in-time (JIT) inventory system limits the amount of inventory between processing stations with a controlling "kanban" card. In this type of system, a station is only authorized for processing if a kanban for that part is available. When processing is complete, the kanban moves with the part to the next station. As the next station consumes parts, it returns the kanbans to the previous station to authorize additional processing.

Kanban model

A Kanban system is modeled in ExtendSim by monitoring the queues between machines and having that information regulate processing. To do this, set the Queue block capacity to the number of kanbans and connect the *F* (full) output from the block back to the preceding Activity's *SD* (shutdown) output connector. When the queue has remaining capacity, its *F* connector will output 0 (zero) and the preceding Activity block will be authorized to produce parts. If the Queue block is full, its *F* connector will output 1 and the preceding machine will be shut down until the queue length is reduced.



Kanban model

If you run this model with animation on, the Activity blocks will be struck through in red while they are shut down.

Transportation and material handling

The following Item library blocks are used to represent transportation and material handling:

- Transport
- Convey Item
- Resource Item

The Convey Item and Transport blocks represent ovens, conveyors, and so forth to provide fixed-path routes with a specified travel time for items. The Resource Item and Transport blocks are used with the Batch and Unbatch blocks to simulate AGVs and other independently moving vehicles.

☞ The models discussed in this section can be found in the Examples\Discrete Event\Processing\Material Handling and Transportation folder.

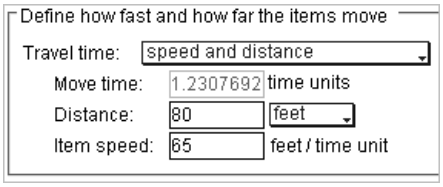
Travel time

In a discrete event model, items move from block to block as dictated by the connections. These connections indicate the direction of movement, but they don't provide any delay for the items. If travel time is significant, it is common to either:

- Increase the delay time of destination blocks to compensate for the travel time.
- Specify a minimum wait time in a Queue block's Options tab to simulate travel time.
- Set an explicit travel time in a Convey Item or Transport block, as shown below.

Transport blocks

Transport blocks (Item library) move items from the start of a path to the end based on distance and speed information. When the model is animated, these blocks can display multiple items travelling a certain distance simultaneously. There is also an option to specify that items cannot pass each other.



Transport block: Behavior tab

☞ While you can choose either feet or meters as the distance unit in the block's dialog, the default is feet. If *Metric distance units* has been selected in the Edit > Options command, the default will be meters.

Travel time options

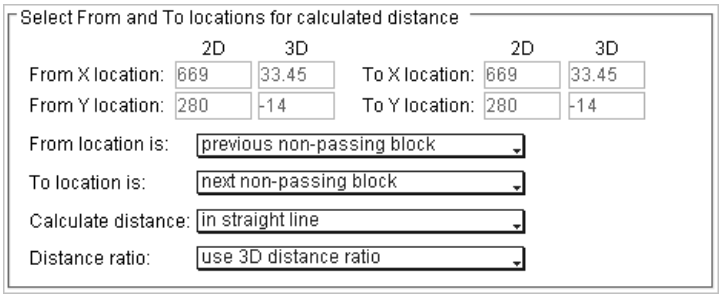
The block's Behavior tab has options that specify how to calculate travel time – the time it will take an item to travel from the starting point to the ending point. Travel time can be based on:

- Move time. Each item will take the amount of simulation time that is entered in the *Move time* field or received at the D input connector. This acts just like a delay in an Activity block; length and speed are ignored.
- Speed and distance. How fast the item is traveling, and how far the item must travel to reach its destination, are entered in the fields in the Behavior tab or received at input connectors. The calculated move time is displayed in the dialog. This option is most often used if the transportation pathway is centered around the block.
- Speed and calculated distance. The item's speed is entered in the *Item speed* field or received at an input connector. The distance is determined from information entered in the frame labeled "Select From and To locations for calculated distance", as discussed below.

☞ If *Speed and calculated distance* is selected, the starting and ending locations (which determine the distance) must be defined in the tab.

Calculated distance

If *Speed and calculated distance* has been selected as the travel time, a frame appears at the bottom of the dialog for entering the distance information. If the *from* and/or *to* locations are set to anything other than *Entered X and Y location*, the relative positions of the blocks in the model, and their connections, determine the distance.



Frame for entering information to be used to determine the distance

The factors considered in the calculation are: the "from" location, the "to" location, how the distance is calculated, and the distance ratio.

From location options

- Previous non-passing block (the default)

- Entered X and Y location
- Block location
- Enclosing hierarchical block
- Previous block

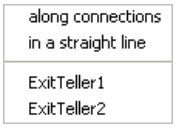
  (See explanations following the To location options.)

To location options
- Next non-passing block (the default)
- Entered X and Y location
- Block location
- Enclosing hierarchical block for next block
- Next block

  Note: In the Item library *non-passing blocks* are either residence or decision types of blocks. If *Entered X and Y location* is selected, the numbers can either be entered in the dialog or defined by the location of block in the model. The *block location* option means the current block; this choice is especially helpful when the *from* location is a previous block (non-passing or not).

☞ If the selected *from* and/or *to* locations are blocks, the distance starts at the *from* block's output connector and ends at the *to* block's input connector.

Calculate distance options
The popup menu provides two methods for calculating the distance from the start to the end of the path:



- Along the connections between the *from* and the *to* locations
- In a straight line between the *from* and the *to* locations

Notice that the shape of the conveyor will not visually change with these choices, but the information is included in the calculation of the conveyor's length. For instance, if there is a series of right-angle connections between the *from* and the *to* locations, the conveyor's length will be longer than if the straight line option had been selected.

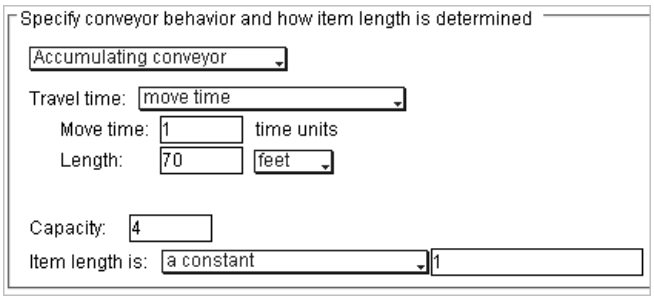
Distance ratio option
This popup menu is for specifying the ratio between pixels in the 2D model. It will control how the distances defined in the Behavior tab affect 2D animation. The choices are:

- Use speed and distance directly. Specifies a ratio of 1 pixel to 1 meter.
- Use 3D distance ratio. Note: 3D animation not available in ExtendSim 10.
- Use distance ratio of: This is for entering the ratio directly. A value of "x" means a ratio of "x" pixels to "x" meters or feet.

☞ These options are explained more in the comprehensive example discussed in "How the length is calculated" on page 341.

Convey Item blocks

The Convey Item block (Item library) moves items along a conveyor, oven, cooling unit, moving walkway, or any other type of moving path. The items travel along the length of the conveyor, from its start to its end. The Behavior tab is similar to that for a Transport block, discussed above. The differences are:



Convey Item block: Behavior tab

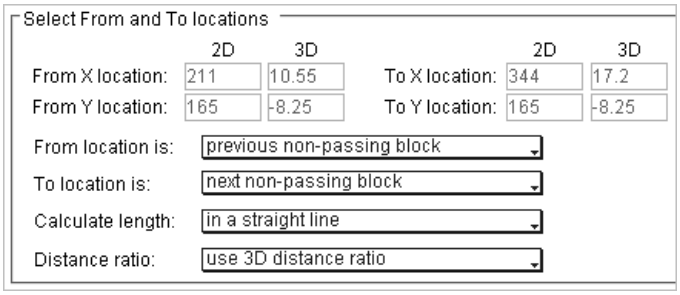
- The Convey Item block can be:
  - Accumulating. If the block's ability to pass items through exceeds downstream demand, any items delayed from exiting will begin piling up at the outflow end of the block, up to the *Capacity* setting.
  - Non-accumulating. If downstream demand exceeds the block's ability to pass items through, the conveyor will stop until the item at the end moves into the next block.
- Instead of the Transport block's reference to the *distance* from one point to another, the Convey Item block is concerned with its *length*.
- If *Travel time: move time* is selected, a *length* entry is also required. In combination with the item length (discussed below), the conveyor's length determines how quickly items can move onto the conveyor.
- The length of the items that pass through this block must be defined. They can be defined as:
  - A constant
  - From an attribute
  - Based on length and capacity. (For instance, if the conveyor's capacity is 1,000 units and its length is 50 units, the length of each item will be 50/1000, or 0.05 units.)

  Item length does not visually change the item picture or object but it is included in calculations for accumulation, capacity, and the timing of when items are pulled onto and released from the block.

How the length is calculated

When *Travel time: speed and calculated length* is selected in a Convey Item block, the relative positions of blocks in the model, and their connections, determine the path's length. The following example explains how the length is determined.

The bottom potion of the Behavior tab for a Convey Item block is seen at right. In this frame:



Length calculations for Convey Item block

- The 2D *From x location* is 211 pixels (the position of the previous non-passing block) and the *To x location* is 344 pixels (the position of the next non-passing block).
- By default, the length is calculated *in a straight line* between the *from* location and the *to* location. (The alternative is to calculate the distance along the connections.) The shape of the block's object will not visually change with either choice, but the choice affects the determination of the path's distance. For instance, if there were a series of right-angle connections between the *from* and the *to* locations, and *along connections* was selected, the distance would be longer than if the straight line option had been selected.

How the settings affect the length
The distance between the previous non-passing block (the *from* location) and the next non-passing block (the *to* location) is calculated in a straight line between the output connector of the *from* block and the input connector of the *to* block. This is 133 pixels (344 pixels - 211 pixels).

The distance ratio of 20 pixels per meter is used to convert the 133 pixels into meters. The result is the length of the conveyor: 6.65 meters. as can be seen in the block's Behavior tab.

☞ If the *from* and/or *to* locations are blocks, the determination of the length starts at one block's output connector and ends at the other block's input connector.

Moving this Convey Item block along the connection line between the *from* and the *to* blocks will not have any affect on the conveyor length. This is because the distance between the *from* and *to* locations stays the same. However, moving the *from* block away from the Convey Item will change the length of the conveyor. This is because the *Next non-passing block* is now further away from the *Previous non-passing block*.

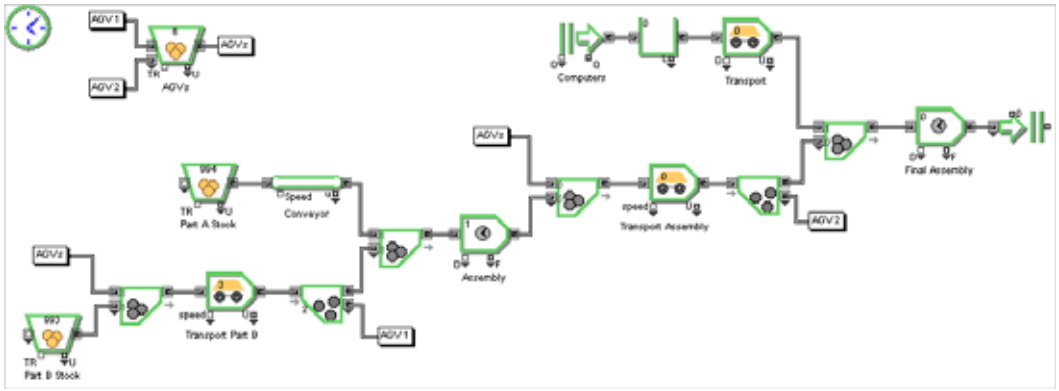
Transportation models

The following models use Convey Item, Resource Item, and Transport blocks to simulate item movement along fixed paths and material handling using AGVs.

Transportation 1 model

To model vehicles such as AGVs in ExtendSim, use a Resource Item block to provide items that represent the vehicles, a Batch block to attach the vehicle with whatever it is transporting,

one or more Transport blocks to provide the transportation delay, and an Unbatch block to separate the vehicle from its load at the end of the route.



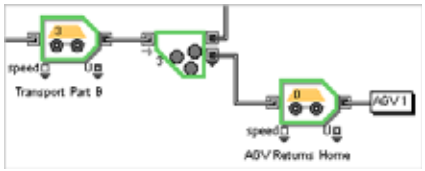
Transportation 1 model

The Transportation 1 model shows how two parts are assembled, inserted into a computer, then moved to a loading dock. Part A is moved by a Convey Item block to the assembly machine while Part B is moved there by a Transport block. The two parts are joined by a Batch block for processing. After assembly, they move by a Transport block to the machine that will put them into the computer. Since the computer is heavy, it is moved to the final assembly machine by a Transport block that represents a small crane.

To simulate movement, AGVs from a Resource Item block are batched to the parts/items and then moved via the Transport block. If the AGV is released at the end of each route, as it is in the model above, the part is left behind where it can then be processed before moving on to another section of the model. To model a situation where the Resource Item transports the item to an activity, then waits there to continue transporting the item again, don't release the Resource Item until the last stop in the route.

Transportation 2 model

In the Transportation 1 model there is no time associated with the return of the AGVs to the Resource Item block. To model how long the return path takes, insert Transport blocks after the Unbatch blocks and enter a distance and speed for the AGV return trip. This is shown in the model segment shown at right.



Adding time for AGVs to return

Discrete Event Modeling

Batching and Unbatching

Joining items or separating them

In a discrete event model, items pass through the system and something is done to or with them. The process often involves temporarily or permanently joining, or *batching*, resources or other items with the original item. For instance, in a manufacturing plant, precursors of the final products come into the process as raw materials, subassemblies, and packaging that are joined in various combinations. During the manufacturing process they are often batched with other precursors and require additional resources such as pallets and workers for processing. These batched items move through the process together.

These same concepts apply to other discrete processes. For example, in an emergency room model, doctors are temporarily batched with their patients during medical diagnosis. In the same model, a technician, a diagnostic machine, and a patient would be batched for the duration of x-ray treatment. In a communication system, multiple packets might be batched together to create a single message. For a retail store model, customers could be shown arriving and selecting merchandise, then be temporarily batched with a sales person to make the purchase.

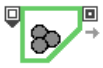
This chapter discusses:

- Blocks for batching and unbatching
- How to batch and unbatch items
- Dealing with item properties when items are batched or unbatched
- Delaying the batching of items until all requirements are present
- Preserving unique item properties as items are batched and unbatched

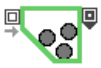
☞ The models discussed in this chapter can be found in the folder \Examples\Discrete Event\Batching.

Blocks of interest

The following blocks will be the main focus of this chapter. The block's library and category appear in parentheses after the block name.



Batch (Item > Batching)
Joins multiple items into a single item for use in the model. This causes the original input items to be destroyed and replaced by one output item. A batched item may be unbatched at a later point in the model, but that is not required.



Unbatch (Item > Batching)
Outputs multiple items for each input item. Depending on selections in the dialog, this block can separate items that were previously batched or make duplicates of items that were never batched.

Batching

Batching allows multiple items from different sources to be joined as one new item for simulation purposes (processing, routing, and so on). The Batch block accumulates items from each source up to a specified count, then releases a single item that represents the batch. In this process, the original input items are destroyed and replaced by one new output item.

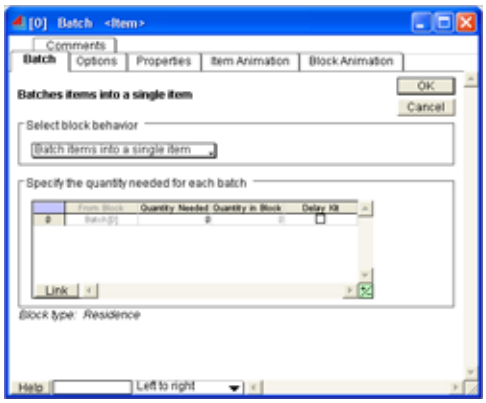
☞ Items may have properties, such as attributes, before they are batched. To specify what will happen to the properties of items that have been replaced by a new batch item, see "Properties when items are unbatched" on page 356.

The number of items required for a batch is called the *batch size*. In some situations you know in advance how many of each item is required to make one item; in other situations the number of items batched depends on model factors and changes dynamically.

Items can be permanently batched together as one new item that flows through and exits the model, or they can be temporarily joined for some specific purpose and unbatched at a later point in the process. For example, two manuals could be batched with three promotional pieces and one CD to make a software package that is shipped as one product. Or a ship attempting to dock might be temporarily batched with two tugboats resources to guide it through the docking process, after which the tugboats and the ship are sent on separate paths.

Batch dialog

The Batch block has three tabs for determining when items should be batched, how many items to include in a batch, and what to do with the properties of items that are batched. The Batch and Options tabs are discussed below; options in the Properties tab are described on page 351.



Batch tab in Batch block

Batch tab

The top of the Batch block's Batch tab has a popup menu with two options that determine how the block behaves. These are summarized below and illustrated in models later in this chapter.

- *Batch items into a single item* creates a batch using items on a first-in, first-used basis as they arrive at the item input connectors. The quantity of items required from each input is entered in a table or (if this choice is selected on the Options tab) determined by the value at a BatchQuantityIn connector.
- *Match items into a single item* creates a batch of items that have a common attribute value. For instance this could represent a process where items were combined together based on a serial or order number. For this choice, it doesn't matter which input connector the items arrive from.

The table in the Batch dialog is for entering the number of items from each input that are required to make a batched item; the size of the batch can also be set using value input connectors as discussed later in this chapter. The first column shows the block label or name the input is connected to. The Quantity column is for specifying the number of items required from the input and the next column reports the number of items available. Checking the fourth column determines if Delay Kit is activated; as discussed on "Delaying kits" on page 353, this causes the specified item or items to not be pulled into the block until certain conditions are met.

Options tab

Among other choices, this tab has options for setting the size of a batch through value input connectors and determining when to start a batch.

BatchQuantityIn connectors
Checking *Use quantity input connectors* on the Batch block's Options tab enables the BatchQuantityIn variable connector. Each BatchQuantityIn value input connector corresponds to an item input connector and controls the batch size for that item connector. If a value input connector has been connected, it will set the number of items required at its adjacent item input connector. If a value input connector is not connected, the number of items for the adjacent item connector will be set by the value in the Batch dialog.

When *Use quantity input connectors* is checked, there are two options that affect the batch size. With either option, the *initial* size of the batch is the value at the BatchQuantityIn connector when the first item on its corresponding item input connector arrives to the Batch block. The options determine what happens if the input value changes:

- Dynamically as batch is created. If the value at a BatchQuantityIn connector changes before that item connector's batch is released, the number of items required for that batch will change as well. This enables the size of a batch to be changed dynamically.

☞ The number of items to be batched from each input connector can never be less than the number of items that have already arrived to the block from that input. That is, if 10 items have already been pulled in through an item input connector, and the BatchQuantityIn connector changes to 8, the batch size for that item input connector will be set to 10, not 8.

- By first item at each connector. The size of the batch does not change after the first item for the batch has arrived, even if the value of the BatchQuantityIn connector changes. Once that batch is released, a new batch size can be set.

The demand connector
To enable the demand connector, check the *Show demand connector* box in the Batch block's Options tab. Then choose one of the following options:

- Start batch when value at demand ≥ 0.5. No items are brought into the Batch block until the value of the demand connector equals or exceeds 0.5. For instance, while the Batch block sees a 0 at demand, no items will enter for batching. When it sees a 1 at demand, the required items currently available will enter the Batch block to be joined together. Depending on how the model is constructed, selecting this behavior can cause blocking of upstream items.
- Create batch when value at demand ≥ 0.5. Items are allowed into the Batch block as they are available, up to the required number. However, the batched item will not leave the block as long as the demand connector has a value < 0.5. When the demand connector becomes ≥ 0.5, the batched item leaves the block. With this option, a batch can consist of fewer items than the number in the Quantity Needed column, because the batched item will have been created by joining whichever items were available when the demand connector got a value ≥ 0.5.

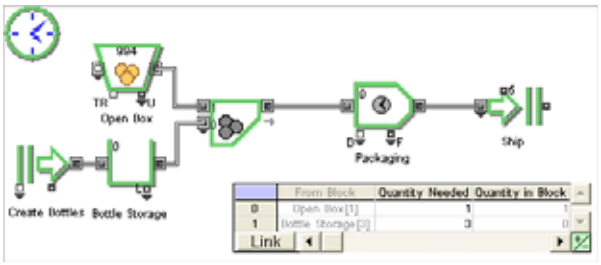
A value output connected to the Batch block's demand input is used as a true/false indicator, triggering batching. The actual value from the value connector is ignored; what is considered is whether or not it equals or exceeds 0.5.

Simple batching

The simplest batching method is to cause multiple items to be joined as one new item, replacing the original items in the model. The batched item may or may not be unbatched at a later point, depending on model requirements.

Simple Batching model

The model shown at right joins one "Open Box" with three "Bottles"; they then travel as one item to be shipped. For this model, the Batch block's behavior is set to *Batch items into a single item*. The items are joined by the Batch block according to settings in the table in its dialog. The Batch block will not release the batched item until it has received one item (a box) from the top input and three items (bottles) from the bottom input.



Simple Batching model

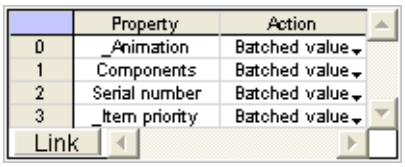
Batching by matching items

Selecting the *Match items into a single item* option in the Batch dialog allows you to specify an attribute the items must match and a different attribute that determines the batch size. With this option, each batch will be composed of items whose matching attribute value is the same; the batch size attribute of the first item in the batch determines how many items are in the batch. With this option, it does not matter which input connector the items arrive at. As items arrive to the Batch block they are segregated based on their matching attribute value until the total number of items in that group equals the batch size attribute. When this occurs a batch is created and the item representing the batch leaves the block.

Matching Items model

The Matching Items model shown below simulates a refurbishment process for police cars. As the cars arrive, they are assigned a consecutive "serial number" by the Information and Set blocks. The Information block counts each car in order and outputs that number to the Set block, which assigns the count number as the value of the Serial Number attribute. For instance, this causes the Serial Number attribute for the second car to be assigned a value of 2.

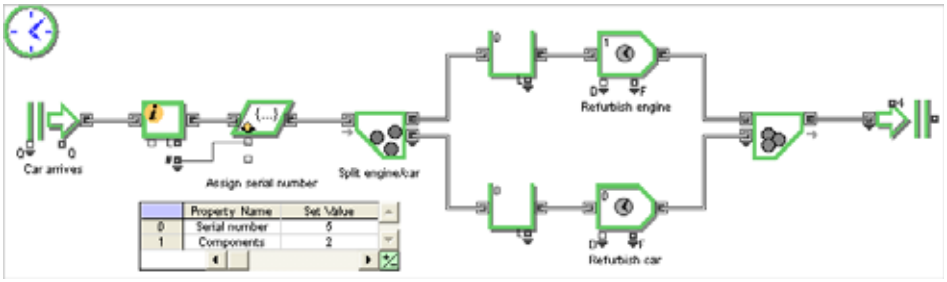
The engine is then separated from the car by an Unbatch block. (When items are unbatched, you can specify what the block should do with their properties. This is accomplished by selecting an Action for item properties in the Unbatch block's Properties tab, as shown at right. Actions are discussed in "Properties when items are unbatched" on page 356.)



Properties tab of Unbatch block

These two components (the engine and the car) are refurbished individually. When both components are finished being refurbished, the engine is reassembled into its original car by matching them together in a Batch block set to *Match items*

into a single item, Match on attribute: Serial Number, and *Get batch size from attribute: Components*.



Matching Items model

For a similar model that uses the Queue Matching block to match items based on an attribute value, see “Matching items using the Queue Matching block” on page 288.

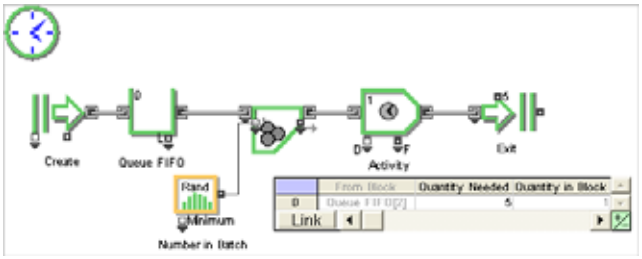
Batching a variable number of items

Sometimes the number of items required to create a batch changes during the course of the simulation. For instance, outside factors could determine how many items go into each batch, or you may want batches to be made in a time-dependent fashion. As shown in the following two models, the Batch block’s Options tab allows you to manipulate the size of batches through quantity input connectors or through a demand connector.

☞ For additional examples of dynamically setting batch size, see also the “Batch and Unbatch Variable model” on page 356 and the advanced batching models “Equation(I) Controls Batch” and “Queue Eqn Controls Batch” that are located in the folder \Examples\Discrete Event\Batching.

Batching Variable model

This example model uses a Batch block’s quantity input connector to determine the size of a batch. A Random Number block (Value library) sets random batch sizes of two to five items. In the Batch block’s Options tab, *Use quantity input connectors* is checked to enable the BatchQuantityIn connector and *Set batch size: dynamically as batch is created* is selected. The output of the Random Number block is connected to the BatchQuantityIn connector on the Batch block. These settings cause batches to be created that require a variable number of items. The information about the size of the batch can be saved on an attribute specified in the block’s Options tab.



Batching Variable model

☞ The BatchQuantityIn connectors will not be visible on the Batch block's icon unless *Use quantity input connectors* has been checked in the block's Options tab.

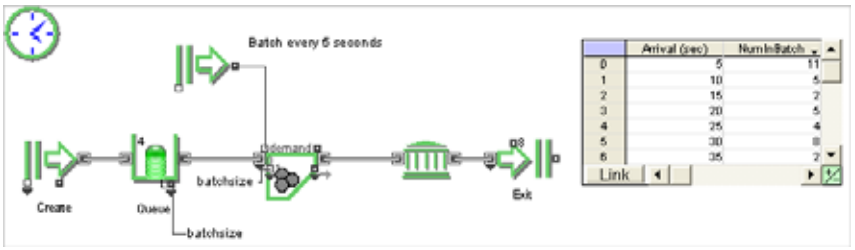
Batch on Demand model

You may want batches to be made in a time-dependent fashion, such as based on the time of day or on a periodic basis. The Batch block's demand connector can be used to control when items are pulled into or sent out of the block. For example, if the model represents filling a truck with boxes, you can signal the demand connector to stop the batching at the end of the day or when another truck arrives at the loading dock. The batched item (the truckload of boxes) is then created and released. In this example model:

- The top Create block sends a value that triggers the Batch block's *demand* connector at scheduled times, every 5 seconds.
- The Options tab of the Batch block is set to *Use quantity input connectors. Set batch size: by first item at each connector* and *Start batch when value at demand ≥ 0.5*.
- The Batch block's quantity input (BatchQuantityIn), is connected to the L (length) output on the Queue block.

With these settings and connections, the Batch block will create a batch every 5 seconds, the size of the batch is dependent on how many items are available to the Batch block when it creates the batch, and items will be allowed into the block only when demand is triggered.

The information about when items arrive and the size of the batch is recorded by the History block and displayed in its cloned table.



Batch on Demand model

☞ By default the History block clears its information each time the model is saved. A choice on the block's dialog allows you to *Save item history with model,* but that option can cause the model to become quite large.

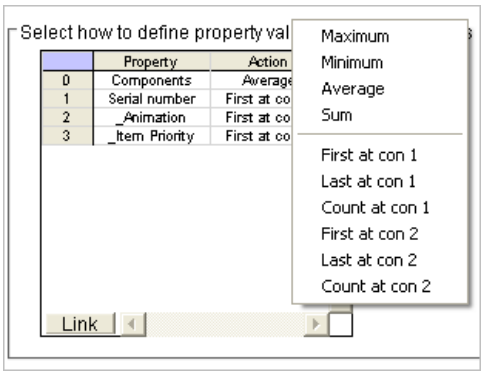
Properties when items are batched

As described in "Item properties" on page 258, items can have properties such as attributes and priorities. When items are combined into a batch, their properties need to be combined as well.

Property options

A table in the Batch block's Properties tab allows you to define what happens to item attributes and priorities. The table's first column lists every property for the items in the model. The second column, Action, gives potential options that can be taken for that property's values. This allows you to select how properties get transferred from the original items to the batched item.

The Properties tab for the Batch block in the Matching Items model looks like the screenshot at right. It shows two user-defined attributes (Components and Serial Number) and the item properties _Animation and _Item Priority.



Action options for properties

Attributes and priorities

The options that can appear in the Action column for attributes and priorities are:

- Maximum. Sets the value of the property to the largest number found on any of the items that formed the batch.
- Minimum. Sets the value of the property to the smallest number found on any of the items that formed the batch.
- Average. Sets the value of the property to the average value of that property for all of the items that are part of this batch.
- Sum. Sets the value of the property to the sum of that property's values for all of the items that are in this batch.
- First at con X (the default). Sets the value of the property to the value of that property of the first item that entered on connector X. Con 1 is the topmost item connector.
- Last at con X. Sets the value of the property to the value of that property of the last item that entered on connector X. Con 1 is the topmost item connector.
- Count at con X. Sets the value of the property to the number of items that entered on connector X. Con 1 is the topmost item connector.

Other item properties

An Item's quantity property (*_Item quantity*) determines its count toward the batch size. For example, if an item arrives with a quantity of 2 and two items are required at that input, then that input is full and no further items are required for that batch from that input. In most cases, the item quantity of the items going into the batch will be 1.

Some models have an animation attribute (*_Animation*) that stores the indexes of the animation pictures for the items moving through the model. Note that the animation attribute can only be set to *First at con X* or *Last at con X*.

Batch size attribute

By creating a new attribute or selecting an existing attribute for *Store number of items in batch in attribute:* in the Batch block's Properties tab, an attribute can be set to the total number of items in the batch as it is released.

Delaying kits

You might not want to begin forming a batch until some or all of the items required for each part of the batch are available. This is most common when you do not want a resource item from the Resource Item block to flow into the Batch block until all the items requiring that resource are available. A good example is a manager who does not want to wait for everyone else to arrive at a meeting.

The Batch block's *Delay Kit* feature restricts specified items from entering the block until all of the other input connectors have the items they need. Delay Kit is enabled through checkboxes in the fourth column of the table in the Batch dialog. Each item input for which Delay Kit is checked will have its items wait outside until all required items for the unselected inputs are in the block.

☞ Delay Kit is only available when the Batch block's behavior is set to "Batch items into a single item."

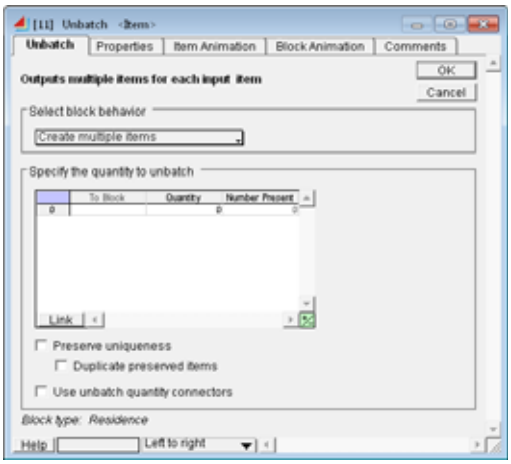
When the kitting starts

If the number of items required at a Delay Kit input is one, and all the other required items have been pulled into the block from their input connectors, the batched item will be created as soon as the Delay Kit item is available. If the number of items required at a Delay Kit input is greater than one, the block will start a kit as soon as all the other required items are available and there is at least one of the Delay Kit items available. This causes items with Delay Kit to be pulled into the block as they become available; the batched item will not be created until all the items with Delay Kit are available.

Unbatching

Unbatching can be used to separate items that were previously batched or to duplicate items that have not been batched. Some examples of when you would use unbatching are:

- Returning an item resource to the Resource Item block
- Ungrouping items that had been temporarily grouped so that they could be processed at the same time
- Creating items based on a single "seed" item
- Creating a logical item to trigger some action in the model while allowing an item that represents the physical part to continue processing



Unbatch dialog

If items were previously batched with *Preserve uniqueness* enabled in the Batch block's Options tab, the Unbatch block can be used to restore the items that formed the batch with their original properties. This is accomplished if *Preserve uniqueness* is also checked in the Unbatch block's dialog. For information about this feature, see "Preserving the items used to create a batch" on page 357.

⚠ Be careful when using any property-setting blocks in the path between a Batch and an Unbatch block. Those property modifications could be lost, depending on selections in the Unbatch block's Properties tab, as discussed on page 356.

The top section of the Unbatch tab in the Unbatch block has two options that determine how the block behaves when costing is involved. Each option causes the block to output a number of items, specified in the dialog, for each item that is input. The options are:

- *Create multiple items*. This is the default choice. If the model has costing, costing attribute values are distributed to the output items as specified by settings in the table in the Unbatch block's Properties tab. The section "Simple unbatching" on page 355 shows how to use the block to separate batched items.
- *Release cost resources*. Resources can have a cost associated with them. If an item is batched with a resource, cost information is maintained with it. If the model has costing and the *Release cost resources* option is selected, the block releases the resources out of the same connector that they were originally batched and updates costing information for the items accordingly. For more information, see"Combining resources with cost accumulators" on page 390.

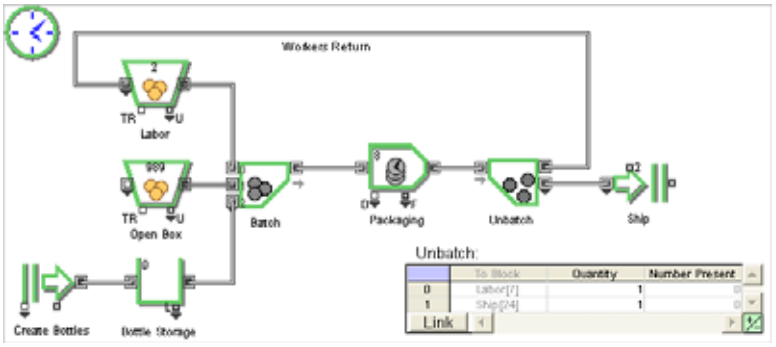
☞ The Unbatch block will not behave differently when either of these two options are selected unless the model calculates costs.

The Unbatch tab also has a table for specifying the number of items that will be sent through each output connector. The first column displays which blocks the Batch's output connectors are connected to. The Quantity column is for entering the number of items that will be output for each input item the Batch block gets, while the next column displays the number of items present.

⚠ When preserving items that have been batched or when releasing cost resources, it is important to physically match where items enter a Batch block with where they leave an Unbatch block. For instance, items that arrived to the Batch block on the top input should be released from the top output of an Unbatch block.

Simple unbatching

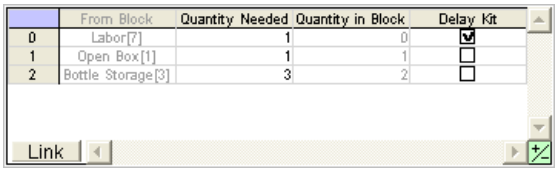
The Batching and Unbatching model is an extension of the example “Simple batching” on page 348, with the addition of laborers and unbatching. Since there is no costing in this model, the Unbatch dialog is set to *Create multiple items* (the default option).



Batching and Unbatching model

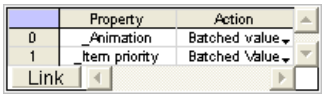
In this model, the packaging process must be performed by a laborer. There are 10 laborers available and each is represented by an item in the Resource Item block. Each worker item is temporarily batched with the bottles and cartons to represent the requirements of the packaging process.

The dialog of the Batch block (shown at right) indicates that one laborer is needed for each package assembly. Since *Delay Kit* is checked for that row, the labor resource will not be drawn into the Batch block until the other items required for the batch (1 box and 3 bottles) have arrived.



Binding a worker

The worker is returned to the pool when the task is finished, while the boxed bottles exit the simulation. This is modeled by the Unbatch block, which takes a single item (the output from the Activity) and creates two items, as shown on the right. One item represents the worker who is sent back to the Resource Item block through the top output and the other represents the assembled package that is shipped.



Unbatching 1 worker and 1 package

Variable batching and unbatching

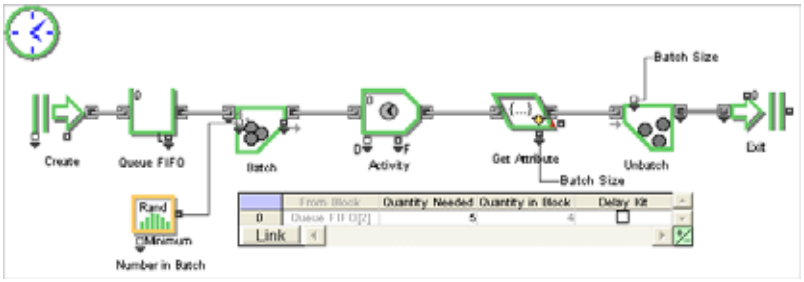
The previous model showed how to batch and unbatch a fixed number of items from each input. This example shows how to batch a variable number of items and unbatch that same number of items.

To keep track of the number of items a batch is composed of, select an attribute in the Batch block’s Properties tab to store the number of items in the batch. When the batch is created this attribute's value will be the number of items that arrived to create the batch.

☞ As discussed later in this chapter, if both the Batch and Unbatch blocks are set to preserve the uniqueness of items, batch size does not need to be saved in an attribute. Instead, just check *Use preserved items to determine unbatch quantities* in the Unbatch block's dialog.

Batch and Unbatch Variable model

An example of batching a variable number of items and unbatching that same number, is shown in the model below.



Batch and Unbatch Variable model

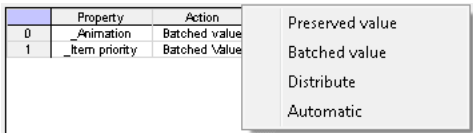
In this model, the Batch block's Options tab indicates that the size of each batch is stored on an attribute named *batchsize*, which is accessed by a Get block. Attaching the Get block's value output to the Unbatch block's *UnbatchQuantityIn* connector sends information about the size of the batch to the Unbatch block, causing each batch to separate into its original number of items.

☞ Since the Batch block is set to *Set batch size by first item at each connector:*, the size of the batch is locked when the first item for that batch arrives in the Batch block.

Properties when items are unbatched

As happens when items are batched, when items are unbatched you can specify what the block should do with their properties. This is accomplished by selecting an Action for item properties in the Unbatch block's Properties tab.

For example, assume a Batch block combines six bottles that have a Weight attribute, and that the bottles are then filled with liquid such that the batch weighs 12 pounds. When these bottles are subsequently unbatched, you can select one of the following actions for the Weight attribute:



Unbatch block property actions

- Preserved value. This option causes the bottles to retrieve their preserved value, if *preserve uniqueness* is turned on. (See the following topic for more information about preserving uniqueness.) In this case the weight of each bottle will be what it was before batching and the 12 pounds of weight acquired after batching is discarded.
- Batched value. With this choice, the 12 pounds of weight will be copied to each of the resulting bottles.
- Distribute. The 12 pounds of weight will be divided among each item equally, 2 pounds to each bottle.

- Automatic. Behaves as *Preserved value* if "Preserve uniqueness" is checked; otherwise behaves as *Batched value*.

Preserving the items used to create a batch

Before items are batched they may have properties such as attributes attached to them. By default, a reduced set of those properties is transferred to the new batched item, according to actions selected in the Properties tab of the Batch block. (For more information, see "Property options" on page 352.)

If it is important in your models to retain the attributes and priorities of the items that were batched, select *Preserve uniqueness* in the Batch block *and* in the Unbatch block. This marks the items in the batch as unique so that an Unbatch block can restore all of the items' properties.

☞ For those properties that you want to retain, select the Preserved Value action in the Properties tab of the Unbatch block, as discussed in "Properties when items are unbatched" on page 356.

⚠ Do not select *Preserve uniqueness* unless the items have unique information attached to them, the items are not just temporarily batched, and you need to restore the items and their properties at a later point. Preserving uniqueness requires a lot more memory and slows processing time.

Both blocks choose to preserve uniqueness

The consequences of selecting, or not selecting, Preserve uniqueness in both the Batch and Unbatch blocks are:

- If the Preserve uniqueness option *is* checked, the batch is temporary. The original members of the batch are stored when the batch is created and they can be restored when the item representing the batch is unbatched. Examples include batching a group of parts together to process them as a group and later unbatching them to continue processing individually, or batching an item with a resource and later returning the resource item with an Unbatch block.

- If the Preserve uniqueness option is *not* checked, the items used to make up the batch are destroyed when the batch is created. The batch, represented by a new item, is permanent and the original items cannot be restored. Examples include batching items together into a box for final shipping, or batching an order with the required inventory.

☞ If a Batch block is set to preserve uniqueness, the unique identity of the items will only be restored upon unbatching. While batched, the attributes of the unique items will be combined into one set of attributes, as specified by settings in the Batch block's Properties dialog.

Either block chooses to preserve uniqueness

To restore the items with their properties intact, the option to "Preserve uniqueness" must be selected in both the Batch and Unbatch blocks. There are special outcomes if "Preserve uniqueness" is selected in *either* the Batch *or* the Unbatch block, but not in both blocks:

- If Preserve uniqueness is selected in the Batch block but not in the Unbatch block, the behavior of the Unbatch blockdepends on whether or not "Duplicate preserved items" is selected. If "Duplicate preserved items" is not selected, the preserved items travel with the first item that leaves the Unbatch block's top output connector. All the other items leaving the Unbatch block will be identical and not contain any information about the preserved

items. If “Duplicate preserved items” is selected, each item leaving the Unbatch block will have a duplicate copy of the preserved items. In this case, the Unbatch block copies the set of preserved items and attaches them to each unbatched item.

- If Preserve uniqueness is not selected in the Batch block but is selected in the Unbatch block, it is the same as if preserve uniqueness is not checked at all. Because the original information about the batched items was lost when they were batched, the Unbatch block will unbatch identical copies of the items that arrive to it.

Neither of these conditions is typically desirable.

Additional models

The folder located at \Examples\Discrete Event\Batching contains additional batching models not discussed in this chapter:

- Equation(I) Controls Batch
- Queue Eqn Controls Batch

Both models show advanced concepts for dynamically changing the size of batches.

Discrete Event Modeling

Resources and Shifts

Modeling resources and controlling them with shifts

Items will sometimes require resources before they can proceed to the next step in a process. For example, a car might need an attendant to drive it through the car wash, a vendor's invoice could require a receiving report before payment is made, or a piece of equipment might need to be assembled by a worker. Resources provide a service to the items in a model; their lack of availability can cause constraints on the flow of items.

One of the main reasons to model a process is to analyze resource availability and utilization and to determine the impact of resource constraints on the system's capacity. This tells how efficiently current resources are being used and what happens if they will not be available or when there is a wait for them to become available. Often the objective is to try to improve resource utilization without causing overly long waiting lines or to determine how to reduce waiting lines without adding more resources.

This chapter discusses:

- Modeling resources with the Resource Pool block
- Modeling resources using the Resource Item block
- Other methods for modeling resources
- Closed and open systems
- Ways in which resources can be scheduled
- Controlling resources and activities with the Shift block
- Advanced Resource Management and setting up resource requirements

☞ The models illustrated in this chapter are located in the folder ExtendSim/Examples/Discrete Event/Resources and Shifts.

Blocks of interest

The following blocks are the main focus of this chapter. Each block's library and category appears in parentheses after its name.

Resource pool blocks

Resource Manager (Item > Resources)
Manages and controls the behavior and utilization of advanced resources. Use this block to create, modify, view and delete Advanced Resources. This block creates internal databases where it stores all of the information needed to manage and control Advanced Resources. See "Advanced Resource Management" on page 378.

☞ The Resource Manager block is an advanced tool that is only available with some ExtendSim products.

Resource Pool (Item > Resources)
Stores a count of resources for the model. The resources are taken by the Queue block (in "resource pool queue" mode) and released by the Resource Pool Release block at some later point in the model.

Queue (Item > Queues)
When the Queue type is set to "resource pool queue", items wait here for required resource pool units from the Resource Pool block. Once the needed resource units are available, the block checks for downstream capacity before releasing items.

Resource Pool Release (Item > Resources)
Releases the specified number of resource pool units, making them available for re-use and causing the count in the Resource Pool block to increase.

Other resource blocks

Resource Item (Item > Resources)
Unlike the resource pool method, this block stores resources as items for use in the model. Resource items are usually batched with items that require them; they may or may not be unbatched at some later point in the process.

Shift (Item > Resources)
Generates a shift schedule that can be used to change the capacity or stop the activity of other blocks in the model.

Modeling resources

Resources are the means by which process activities and operations are performed. Different parts of a model can share the same resource, just not at the same time. While a particular resource is being used in one place in a model, it is not available for any other part of the model. Thus the availability or lack of availability of resources causes constraints in a model.

Many approaches to modeling resources

As seen in the following sections, there are many ways to simulate resources when building models in ExtendSim– some are explicit and some are implicit or conceptual.

- Resources can be explicitly modeled using specialized resource management blocks; this has the advantage of direct access to features like automatic costing and utilization calculations.
- In some situations, however, it could be simpler or provide more control to model resources just as any other item in the model (conceptual) or by limiting block capacity (implicit).

☞ Explicit methods should only be used in a model if the presence of those resource-type blocks is required for the system. Otherwise, use an implicit method.

Explicit methods

The ExtendSim discrete event architecture supports three explicit methods for modeling resources:

- Resource Pool method
- Resource Item method
- Advanced Resource Management (ARM) method

☞ These methods are compared and contrasted in the table on page 362.

Implicit methods

A resource can be implied in a model by restricting or scheduling the capacity of residence type blocks like the Activity and Queue. These blocks are useful for implicitly modeling resources.

For instance, the “Simple Resource Pool model” on page 366 illustrates how to model resources using the resource pool blocks. A simpler method would be to use the Activity block

to represent a limited resource, without using explicit resource blocks. In this case, you would remove the Resource Pool and Resource Pool Release blocks from the model, set the Queue as a FIFO sorted queue, and set the maximum items in the Activity block to three. The Activity's capacity limitation would have the same constraining effect as the Resource Pool block in the original Simple Resource Pool model.

Another advantage of modeling resources implicitly is that the Activity block can be shutdown and brought back online using the Shutdown block, as shown in "Shutting down" on page 331. This is common when modeling random failures.

Conceptual resources

The concept of what is a resource is not limited to the explicit (resource pool and resource item) or implicit (capacity-constrained) methods of representing resources. Theoretically, a resource is anything where its availability can restrict items flowing from point A to point B. Some examples are:

- Any item can conceptually represent a resource. For example, batching a "bus" item with "people" items, where the bus is required before the batched bus/people item can be released from a Queue (see "Delaying kits" on page 353). Note that the bus is created as any other item, not as a resource item from the Resource Item block.
- Using the ExtendSim database or global arrays to track resource availability, limiting the flow of items in a model. For example, item availability would be regulated by how data in the database changes during the course of a run.
- Using block combinations to control item movement as model status changes over time, such as a Queue followed by a Gate that is connected to a Read block.

Your cleverness and expertise with ExtendSim can probably lead to even more ideas.

Comparison of the three explicit methods

Technique	Resource Item Method	Resource Pool Method	Advanced Resource Management (ARM)
Description	Models resources as *resource items* that are available to another item. This involves batching resource items from the Resource Item block with the items that require them and, typically, unbatching the resource when it is no longer needed.	Models resources as a *count* of the resources that are available in a pool. By keeping track of the available resource pool units, this method controls the flow of items that require those resources.	Models resources as *records in a table* in an ExtendSim database. Each record represents a resource with individual properties and statistics. The Resource Manager block provides the interface for creating managing advanced resources.
See	"Resource Item method" on page 368	"Resource Pool method" on page 365	See page 364 and the document "Advanced Resource Management Tutorial and Reference"

Technique	Resource Item Method	Resource Pool Method	Advanced Resource Management (ARM)
Main Model Components	Resource Item, Batch, and Unbatch blocks	Resource Pool, Queue (in *resource pool queue* mode), and Resource Pool Release blocks	Resource Manager block, Queue and Resource Pool Release blocks (in *advanced resource* mode), and the ExtendSim database
Resource Representation	Resources are represented by items that are created and reside in Resource Item blocks. Each Resource Item block maintains statistics. Attributes can be used to store properties and statistics for individual resources.	Resources are represented as a quantity in a pool in the Resource Pool block. Individual resources cannot have properties and are thus indistinguishable within a pool. Each Resource Pool block maintains statistics.	Each resource is represented as a record in a table in an internal database. The fields in the table are used to store property values and statistics for individual resources.
Requirements Specification	There is no dedicated user interface for specifying resource item requirements. Typically, Queue Equation or Queue Matching blocks are used to hold items until the required resource items are available. Then, released items are batched with resource items.	Requirements are specified using a data table in the Queue. The table specifies the required quantities from one or more pools. The required pools are selected using the settings in the dialog. The required counts can be specified using attribute values or connectors.	Requirements are specified using logical expressions created in the dialog of the Resource Manager block or alternately in an Excel worksheet. These expressions consist of components that specify required quantities and filtering conditions that limit the set of resources to those wanted.

Technique	Resource Item Method	Resource Pool Method	Advanced Resource Management (ARM)
Advantages	1. Because resources are items, they can have properties that allow them to be distinguished individually. 2. Modelers have full control over the logic governing how resources are utilized.	1. Does not require connections to blocks. 2. Does not need complex routing logic to control the allocation and release of resources. 3. Resources automatically know which items are waiting for them. 4. Provides the capability to globally prioritize the allocation of resources to the highest ranked items.	1. Does not require connections to blocks nor complex routing logic. 2. Resources automatically know which items are waiting for them. 3. The allocation of resources to the highest ranked items is globally prioritized; multi-tiered ranking is supported. 4. Resources have properties that allow them to be distinguished individually. 5. Allows for very complex resource requirements and provides a flexible mechanism for creating resource release rules. 6. All information about resources is globally available to the model; details can be logged in the database.
Disadvantages	1. The Resource Item block must be connected in the model such that the resources it outputs can be batched with the items that require them. 2. A resource item cannot "see" the items waiting for it. Routing blocks must be used to direct the resource item to the correct Batch or Unbatch block.	1. Does not allow the use of attributes or other properties to track information about individual resources. 2. It is more difficult to control the complex scheduling of competing resources across a number of different queues using resource pools.	1. The complexity of this method requires more learning time compared to the other methods. 2. Advanced Resource Management is only available with certain ExtendSim packages.

Advanced Resource Management

Advanced Resource Management (ARM) is an integrated ExtendSim system for organizing resources, distinguishing between them, and allocating them as required throughout a model. ARM provides a convenient and straightforward method for defining complex resource requirements for items as well as a flexible set of rules for how resources get allocated to them.

And it provides automated methods for quickly changing resource information and generating statistical reports.

☞ This sophisticated and powerful feature is only available in some ExtendSim products; it is discussed fully in the separate document *Advanced Resource Management Tutorial and Reference*.

Resource Pool method

The resource pool blocks in the Item library (Resource Pool, Queue, and Resource Pool Release) cause restraints to be placed on the flow of items in the model based on the availability or lack of resources. The Resource Pool block maintains a count of the number of resources that are currently available for use. When an item enters a Queue block that is in resource pool queue mode, the Queue will query the resource pools to determine if the required number of specified resources are available. If so, the number of resources currently available will be decremented in the appropriate Resource Pool block, and the item that requires the resource will be released from the Queue. If the required number of resources are not available, the item will wait in the Queue until resources become available.

In a closed system, the resources are returned to the Resource Pool block by passing the "resourced" item through a Resource Pool Release block. In an open system, such as for a consumed resource, the resource is not returned to the pool but is removed from the system when the item exits. Closed and open systems are discussed on page 370.

☞ Since, in an open system, resources are not returned to the originating block, statistical calculations such as utilization cannot be accurately determined.

Advantages and disadvantages of using resource pools

Advantages

- The Resource Pool block does not require any connections to other blocks in a model. Because of this, using resource pools to model resources (as opposed to using the resource item/batching method that will be described later) is more flexible when the same resource can be used in many different places or when an item can use any one of a group of resources.
- The resource pool method does not require complex routing of resource items because the resources are not actual items but merely constraints on the flow of items through the model.
- When items wait for resource pool units, they can be ranked by priority or FIFO order. The Resource Pool block is able to globally allocate the resource pool unit to the highest ranked item.

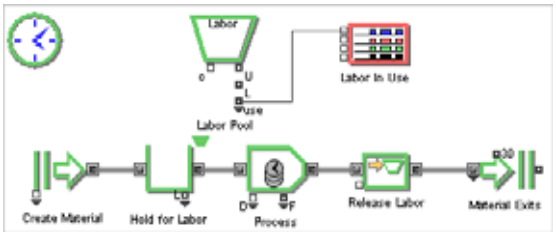
Disadvantages

- The resource pool method does not allow the use of attributes to track information about the individual resources. To use attributes, you must use resource items; this is shown in "Resource Item method" on page 368.
- It is more difficult to control the complex scheduling of competing resources across a number of different queues using resource pools.

☞ After learning about resource pools, see "Advanced Resource Management" on page 223 for a more rigorous method of creating resource requirements, scheduling, and tracking resources.

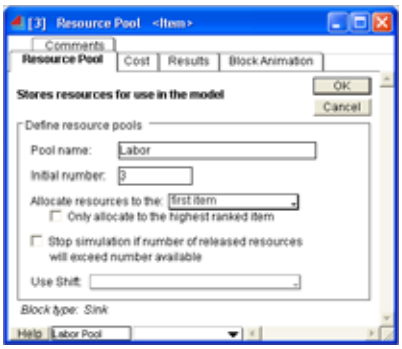
Simple Resource Pool model

This example shows how to use and release resources from resource pools:



Simple Resource Pool model

The Simple Resource Pool model represents a flow of material where each piece requires one laborer for processing. In the dialog of the Resource Pool block (labeled Labor Pool), the pool of resources is called *Labor* and the initial number of Labor units is **3**, as shown at the right. One piece of material is generated by the Create block about every two minutes. Since this model uses resource pool units, the Queue block's type is set to *resource pool queue*. This causes generated material to wait in the Queue until the required Labor is available.



Resource Pool dialog

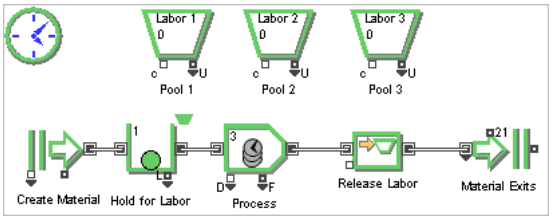
In the Activity block, processing takes 5 minutes and the capacity is infinite, so any number of pieces can be worked on at a time. Within the Queue and Resource Pool Release blocks, the quantity of resources required/released is 1 and the name of the resource pool (Labor) is listed.

Running the model shows that the amount of processing that can occur is constrained by the number of laborers available. Although the Activity has an infinite capacity, the cloned plotter graph shows that there are not enough workers available for much of the simulation.

☞ So that the focus is on the constraining effect of labor resources, the Activity block is set to infinite capacity. This causes the availability of labor, but not the processing of material, to affect the flow of items in the model.

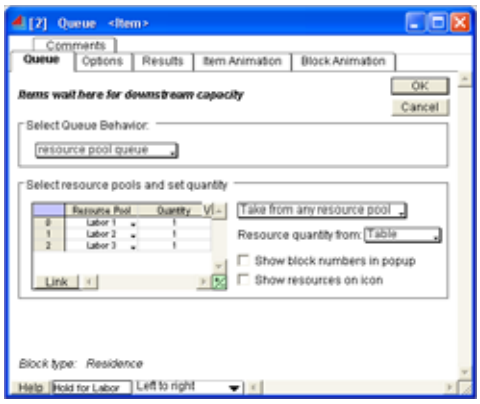
Resources required from different pools

In the Multiple Pools example, there are three Resource Pool blocks, each with their own labor resource. Items require *either* Labor 1, Labor 2, *or* Labor 3.



Multiple Pools model: One laborer per piece required from any of the three pools

Because it will hold items that require resource pool units, the Queue's type is set to *resource pool queue*. As shown in the Queue's dialog, the three resource pools have been selected from popup menus in the table and the block is instructed to take the resource from any one of those pools. When an item enters the Queue, it will query each of the resource pools in the order that they are listed in the table (from top to bottom). In other words, if no resources are available in Pool 1, the Queue block will try Pool 2, and then Pool 3. If a resource is still not found, the material will be held in the Queue until the resource requirement is met by whichever pool first has an available resource.



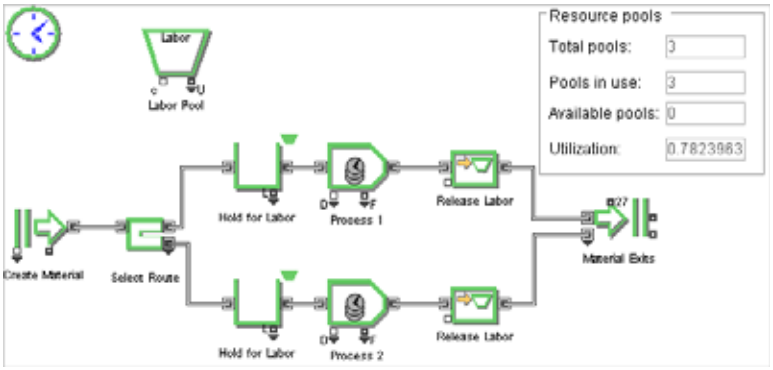
Queue block dialog

Note that the Queue block (Options tab) stores the information about which pool the resource came from, and how many resources were used, in an attribute that attaches to the items processed. In this model, the information is stored in the attribute "Resource Name." This attribute is used by the Resource Pool Release block to inform the appropriate pool when a resource is no longer in use.

The Multiple Pools model is similar to the Simple Resource Pool model in that each piece of material requires 1 laborer. However, in this model the laborer can come from one of three pools rather than from a pool of three laborers. As in the earlier example, modeling this process using the resource item/batching method would require complex logic to correctly route the resources.

Same resource used in multiple places

The Multiple Uses model has two parallel processes, each requiring laborers. Items waiting in both Queues (set to *resource pool queue*) require a labor resource from the same Resource Pool block, which has 3 laborers initially available. When an item enters one of the queues, a request is sent to the Resource Pool block for a labor resource. The requests are satisfied in the order in which they were received (or in order of the requesting item's ranking as specified in the Resource Pool dialog).



Multiple Uses model

Resource Item method

Another method for explicitly modeling resources is by using the Resource Item block. With this method, each resource is represented by an item whose purpose is to provide a service for other items in the model. The number of resources that are initially available are entered in the dialog of a Resource Item block. For an item in the model to use this type of resource, the item must be batched with the resource (see also "Batching and Unbatching" on page 345.) While the resource is batched with an item, it cannot be used elsewhere in the model. If a resource is not available, the batch will not be able to be completed, and the item will have to wait until a resource becomes available. As with the resource pool method, the movement of items in the model is restrained based on the availability or lack of resources.

If you are modeling a closed system, the resource must be unbatched from the item when it is no longer being used. Once it is unbatched, it should be routed back to the resource-type block so that it may used again. In an open system, for example where the resource is a consumable product, the resource can stay batched with the item. Closed and open systems are discussed on page 370.

Advantages and disadvantages of using resource items

A resource item can have properties such as attributes, a priority, and a quantity like any other item. For this reason, this method of modeling resources is preferred if you need to track information about resources.

☞ See "Items, Properties, and Values" on page 257 for a complete explanation of attributes and other item properties.

Some limitations of using resource items are:

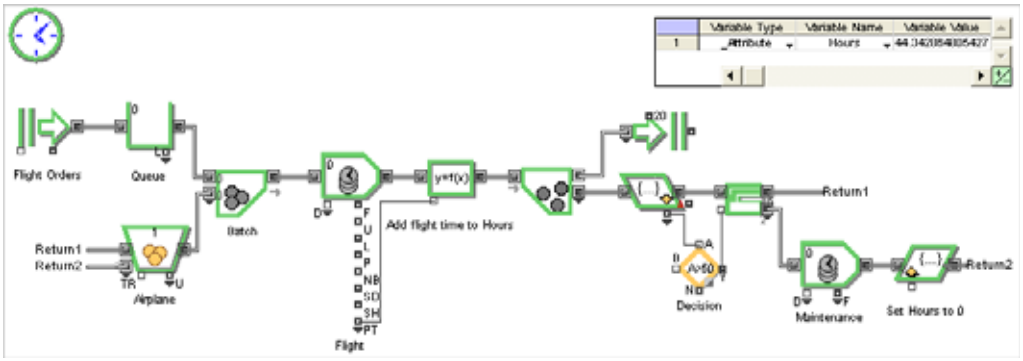
- The Resource Item block must be connected in the model and the connection must be such that the resources it outputs can be batched with the items that require them.
- The resource item cannot "see" the items waiting for it. You must use routing blocks to direct the resource item to the correct Batch or Unbatch block.

☞ After learning about resources, see "Advanced Resource Management" on page 223 for a more rigorous method of creating resource requirements, scheduling, and tracking resources.

Air Freight model

An example of using the attributes of resource items to track information is shown in the following model of an air freight company. An airplane receives orders for flights, but regulations require that airplanes must undergo maintenance after every 50 hours of flight time. Thus the model needs to track the airplane's accumulated flight time (hours). Once an airplane accumu-

lates 50 hours of flight time, it is sent for maintenance and the accumulated hours are reset to 0. The model looks like:



Air Freight model

The Create block generates orders which are batched with an airplane from a Resource Item block. While the airplane is batched with a flight order, it is not available for other flight orders; the order will wait in the Queue (set to *type: sorted queue*, and *Sort by: first in, first out*) until the plane becomes available.

The Resource Item block attaches an *Hours* attribute to the airplane. Settings in the Batch block's Properties tab, shown at right, cause the airplane's Hours attribute (the first attribute from the item arriving at the second connector) to be attached to the batched item.

	Property	Action
0	Hours	First at con 2
1	_Animation	First at con 2
2	_Item Priority	First at con 1

Link

Batch block properties

☞ For more information about item attributes, see "Item attributes" on page 263.

An Activity block (labelled Flight) uses a random distribution to determine how long each flight will take, then outputs the flight time to its Process Time (PT) connector. The Equation block gets the airplane's flight time and adds it to the plane's Hours attribute.

After the flight, the airplane is unbatched from the order; the airplane returns to the Resource Item block for reuse and the order exits the simulation. The Get block reads the value of the airplane's Hours attribute and the Decision block determines if the accumulated flight time is greater than 50 hours. If it is, the airplane will be routed to the maintenance group for processing. After maintenance the Set block re-initializes the Hours attribute to zero. If accumulated time is not greater than 50, the airplane is returned to the Resource Item block where it will wait for another flight order.

When the simulation is run, a clone of the output from the Equation block's dialog shows the value of the airplane's Hours attribute. With animation on, it is easy to see that once the airplane has an Hours value greater than 50, it routed to the maintenance group.

For this model it is essential to be able to track information about the airplane's flight time. Therefore the ability to assign attributes to the airplane resource is critical.

Stripping attributes from resource items

As described in "Properties when items are unbatched" on page 356, an item returning to a Resource Item block after batching may have many attributes that are irrelevant to the returning item. The Resource Item block provides the option of stripping attributes or keeping them

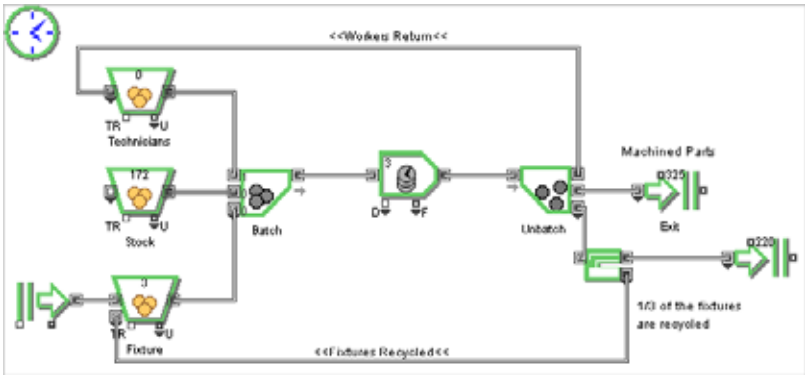
with resources that are recycled; the default is to strip them. When tracking resource information using attributes (as in the above model), you will not want to strip the attributes, so the block is unchecked. However, in cases where you are not concerned with attribute values after the item has been recycled, you may want to strip the attributes so that the item will be "clean" when it comes out of the Resource Item block again.

☞ The Queue Matching block (Item library) holds different types of items until the requirements for each type have been met. This can be useful when modeling the release of resources and items into a Batch block. For more information, see "Matching items using the Queue Matching block" on page 288.

Closed and open systems

As discussed in the Discrete Event Quick Start guide, blocks that provide a finite number of resources can be part of closed or open systems. In a closed system, resources are recycled back to the originating block. In an open system, resources are not recycled but instead exit the system. Systems can also be partially closed, for example when some of the resources are recycled back and others are not.

The following model uses three Resource Item blocks to illustrate a closed system (Technicians), an open system (Stock), and a partially closed system (Fixtures).



Closed and Open Systems model

The model assumes that about one third of the fixtures are consumed in the process; they are restocked at periodic intervals by the Create block.

Scheduling resources

To accurately characterize the impact of resources in a simulation model it is common to model resource scheduling logic, both in terms of where and when a resource should be assigned. For example, if one resource item is required in multiple places, such as an item that could be routed to two or more Batch blocks, then scheduling logic needs to be added to the model.

There are several ways resources can be scheduled. Some methods apply to using either resource pools or resource items and some apply only to scheduling resource items.

Scheduling resource pools and resource items

As discussed in the following sections, there are two systems available to schedule resource pools and resource items:

- Use the *TR* (total resources) connector on individual resource blocks, as described in the following section.
- Use Shift blocks to control aspects of one or more resource blocks. This is discussed in “Resources model” on page 375.

Using the TR (Total resources) connectors

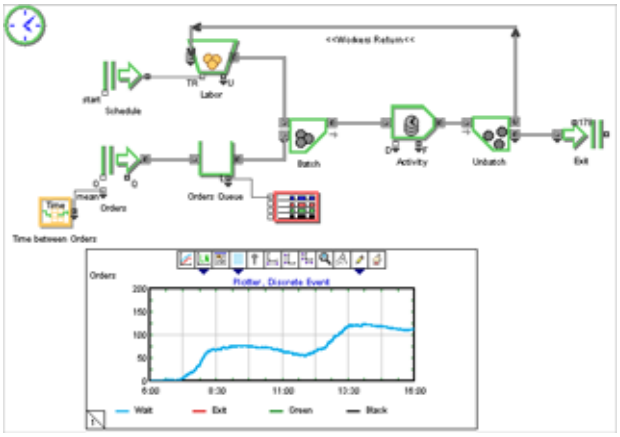
Blocks that provide resources (whether as items or as resource pool constraints on items) have value connectors labeled *TR* (total resources). You use the TR input connector to change the total number of resources available. This change can be scheduled, such as when workers take breaks, or unscheduled, such as an equipment failure.

The value at the TR connector determines how many resources the block has and can result in an increase or a decrease in resource availability. For example, if the initial number in a Resource Item block is 10, and the block gets a value of 3 at its TR input connector, the block will eliminate 7 resources from its availability list. If the block doesn't have enough resources to dispose at the time of the change, it will dispose of them as they return.

Scheduling Resources model

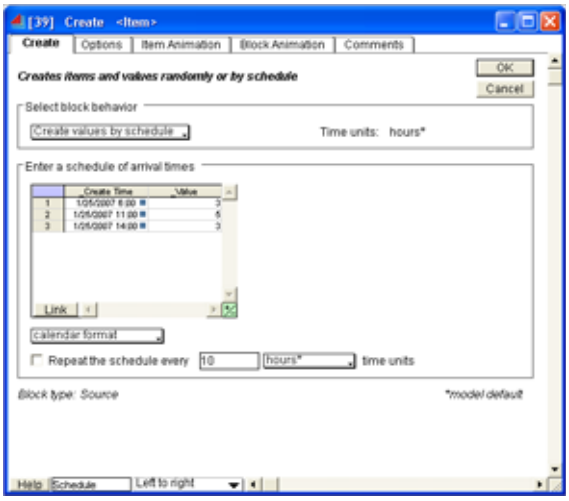
It is common to schedule the availability of resources based on some factor in the model, typically time. For example, in the model discussed in “Scheduling activities” on page 325, you could have scheduled workers in the diner depending on the time of day using the Resource Item and Create blocks.

The new model is shown at right. It assumes there are three workers initially available when the coffee shop opens at 6am. Two additional workers arrive after 5 hours and remain just for the lunch period, from 11am to 2pm (1400 hours). The coffee shop closes after 10 hours.



Scheduling Resources model

The schedule for workers is entered in the dialog of the Create block, as shown at right. The block is set to *Create values by schedule*, and its value output is attached to the TR (Total resources) input of a Resource Item block.



Arrival times for workers

☞ This model is set to use Calendar dates. The Simulation Setup dialog and block options (such as selecting *calendar format* when a Create block is set to schedule its outputs) facilitate the display of times and dates in Calendar format. For more information, see "Calendar dates" on page 95.

In this model the workers are part of a partially closed system. Some are recycled back to the Resource Item block through its item input connector, while other workers are added to or removed from the block through its TR connector.

Scheduling resource items

There are at least four additional ways to schedule resource items in ExtendSim:

1) A Resource Item block followed by a Gate can control when resource items are released.
2) A Resource Item followed by a Select Item Out block controls where resource items are routed.
3) A combination of the Gate and Select Item Out blocks can be used to control both where and when items are scheduled.
4) The Queue Equation block is useful for controlling both where and when items are scheduled for use when the scheduling logic is more complex. With this block, ModL logic statements can intelligently control the scheduling of items based on their properties, information in the ExtendSim database, or even the status of other sections of the model. For more information, see "Sorting items using the Queue Equation block" on page 283.

The Shift block

The Shift block (Item library) is used to schedule both the magnitude and availability of capacity in other blocks in a model. This is useful for simulating situations where a system's resources follow a pattern of coming on and off line over time. For example, the Shift block could be used to model workers in a factory following a repeated daily pattern of reporting to work in the morning, taking a break for lunch and going home at some point in the evening.

Each Shift block represents a named shift and its schedule that can be referenced by other Item library blocks. The Shift controls the capacity of the blocks that reference it, based on the schedule that is defined in its dialog table. If a shift schedule is changed, all blocks using that named shift will receive the same modified shift pattern. In addition, shifts may be repeated at regular intervals if the *Repeat schedule every* checkbox is selected. This is useful for modeling

repeated shift patterns, e.g., an eight-hour workday each day of the week or breaks that occur every four hours.

☞ It would be quite easy to define a complex shift schedule that includes all breaks, holidays, weekends, and so forth. However, before adding such complexity to a model, carefully consider whether such detail adds to the validity of the model. If, for example, nothing at all happens during the weekend, a better solution would be to simply assume one week is 5 days long (or specify that in the Simulation Setup dialog) rather than adding a Shift block to model the weekends. Shifts should only be used when they will make a significant difference in the results of the simulation.

Shift types and what they control

A Shift can schedule capacity in a number of Item and Rate library blocks. For example, it can turn an Activity on or off, or specify a maximum number of items the block can process at a time. The schedule depends on whether the selected Shift type is On/Off or Number.

- An *On/Off* type of shift acts like a binary switch that turns associated blocks on or off at specific points in time. For example, an On/Off shift might be used to shut down an Activity block during lunch and evening hours, to open or close a Gate block, or to turn a Valve (Rate library) on and off according to a schedule.
- The *Number* shift type explicitly defines the size of a block's capacity over time. For instance, it might set the size of a Resource Pool to 3 for the morning shift, 0 over lunch, 5 for the afternoon shift, and 0 overnight.

The following table shows which Item library blocks can be controlled by a Shift block, which types of shifts those blocks support, and what aspect of a block, if any, the Number type of shift controls:

Item Blocks That Can Use Shifts	Shift Type: On/Off or Number	Number Type Setting on Shift Controls This Aspect of Item Library Block
Activity	Both	Maximum number of items in the Activity at a time.
Convey Items	Both	N/A
Create	On/Off	N/A
Gate	Both	In "area gating" mode, the number of items allowed in the gated area at a time. In "schedule gating with Shift", the Gate is always open when the Shift value is >0.
Resource Item	Both	Total number of resources available, per the TR (Total resources) connector.
Resource Pool	Both	The total count of resources available, per the TR (Total resources) connector.
Transport	Both	Block capacity.
Workstation	Both	Maximum number of items in process.

In the Rate library, the Convey Flow, Interchange, Tanker, and Valve blocks can use a Shift set to On/Off type. See the Rate module for more information about using the Rate library.

☞ If a Shift block is used in a model, statistics in the Queue blocks will probably not accurately reflect utilization, etc.

Status connectors

The Shift block's value input connector (*StatusIn)* can be used to override the shift schedule. If the input connector is less than 0.5, the Shift is considered off shift; this will override any value found in the Shift block's dialog table. If the input connector is greater than 0.5, the shift schedule from the dialog table is used.

The Shift's value output connector (*StatusOut*) reports the current shift status (ON = 1 or OFF = 0 for On/Off type Shifts, or the number for Number type Shifts).

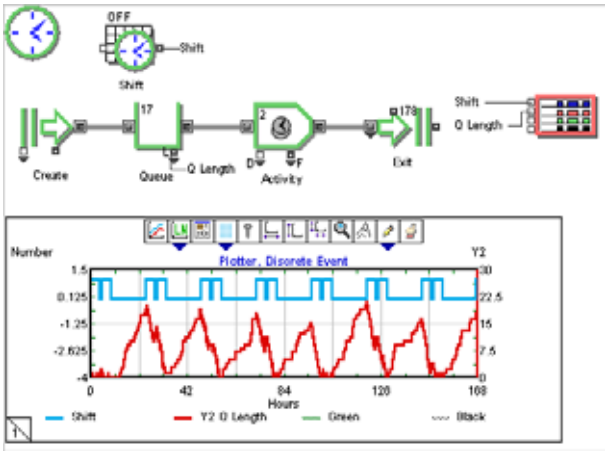
Shift models

The following models show how to use the Shift block in typical modeling situations. They also illustrate how the two types of Shift, On/Off and Number, are used in simulations.

☞ The models are located in the \Examples\Discrete Event\Resources and Shifts folder.

On/Off type example

In the "Shift On and Off" model at right, an On/Off type of shift turns the Activity block on for 4 hours in the morning, off for 1 hour at lunch, back on for 4 hours in the afternoon, and off again in the evening until the next morning. The schedule, named "Day Shift", is set in the Shift block; the dialog of the Activity block is set to *Use Shift: Day Shift*, which causes the Activity to go on or off according to that schedule.



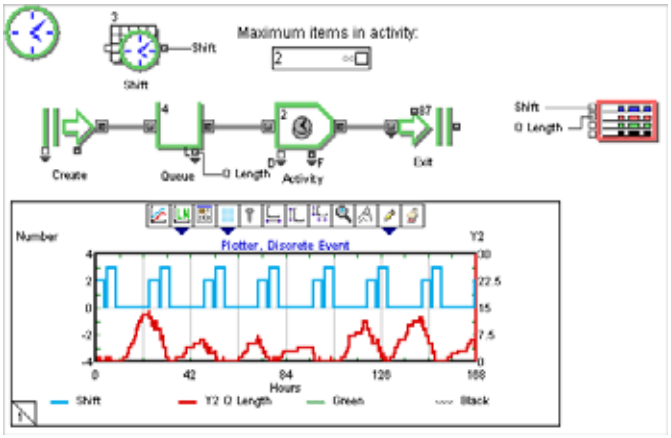
Shift On and Off model

The model contains a cloned graph from the plotter. On the graph the upper (blue) line reflects this 24-hour shift cycle for seven days, a total of 168 hours. Observe how the lower (red) line, which is plotted against the Y2 axis, reflects the queue length's dynamics during the various on and off shift periods.

Number type examples

Shift Capacity Change model

In the model "Day Shift Capacity Change", shown at right, a Number type of shift is used to control the maximum number of items in the Activity block. As determined by the Shift block, the Activity is limited to 2 items during the morning shift, 0 during lunch, 3 items during the afternoon shift, and 0 overnight. The schedule, named "Day Shift", is set in the Shift block; the dialog of the Activity block is set to *Use Shift: Day Shift*. Since this model uses a number type shift, the Shift block's table controls the maximum number that can be allowed in the Activity at one time. When the model is run, the cloned Activity dialog item "Maximum items in activity" reflects the Shift table's schedule of allowed items. Shift behavior over the course of seven days (168 hours) is reflected in the plotted upper line of the graph. The lower graph line reflects the queue length's growth when activities are limited.

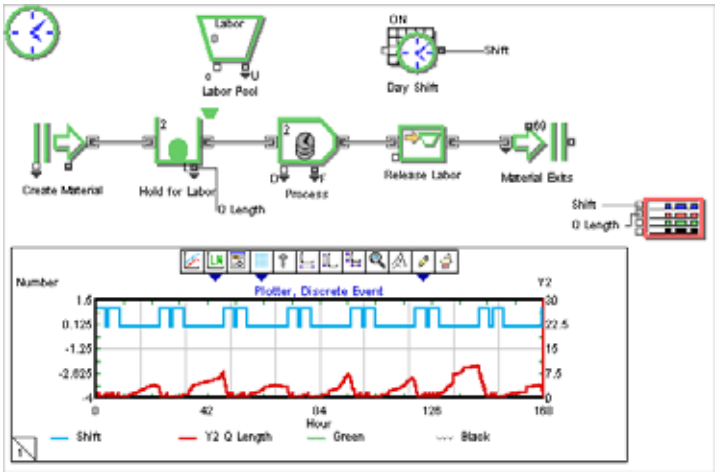


Shift Capacity Change model

Resources model

The next example also uses a Number type of shift. It illustrates a Shift block controlling Resource Pool availability so that workers start their shift, work 4 hours, take a lunch break, work 4 more hours, and then leave for the day.

When workers are not available, the backlog starts building up in the Queue.



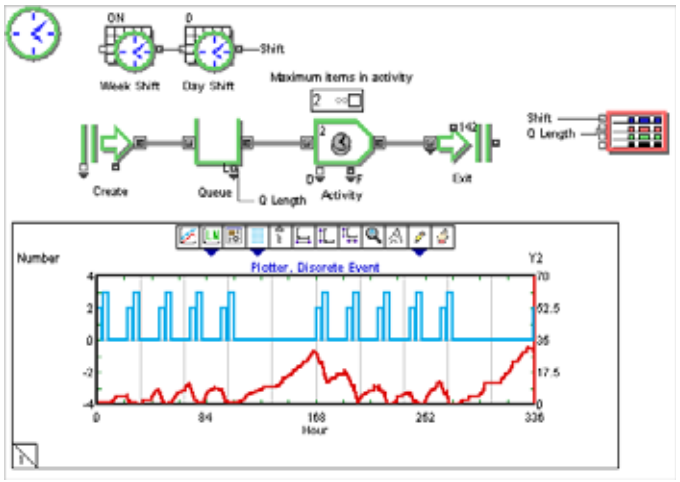
Resources model

Complex patterns

Shift blocks may be configured serially, controlled by other blocks, or used in other patterns to create more complex shift patterns.

Weekly and Daily Shifts model

In the "Weekly and Daily Shifts" model, shown at right, a 40-hour work week with two days off during weekends is modeled by feeding a Week Shift (set to On/Off) into a Day Shift (set to Number).



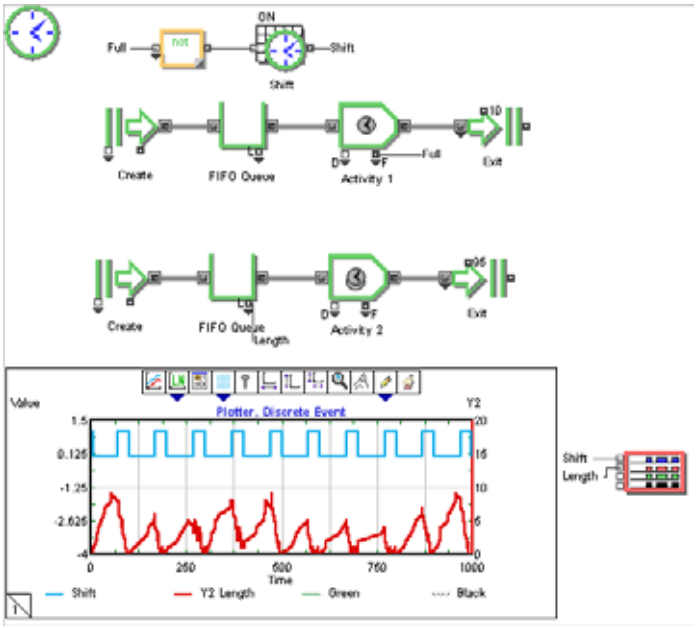
Weekly and Daily Shifts model

The Week Shift is On for the five weekdays and Off during weekends. Consequently, during weekdays, the Week Shift block sends a value of 1 to the StatusIn connector of the Day Shift block, allowing it to observe its own daily schedule. However, during weekends, the Week Shift block overrides the Day Shift schedule by sending it a value of 0. The plotted line in the cloned graph shows two work weeks (336 hours); the first and second weeks are separated by a weekend.

The cloned dialog item shows the same daily schedule as in the preceding model, with activity halted during lunch and at night.

Controlling Shifts model

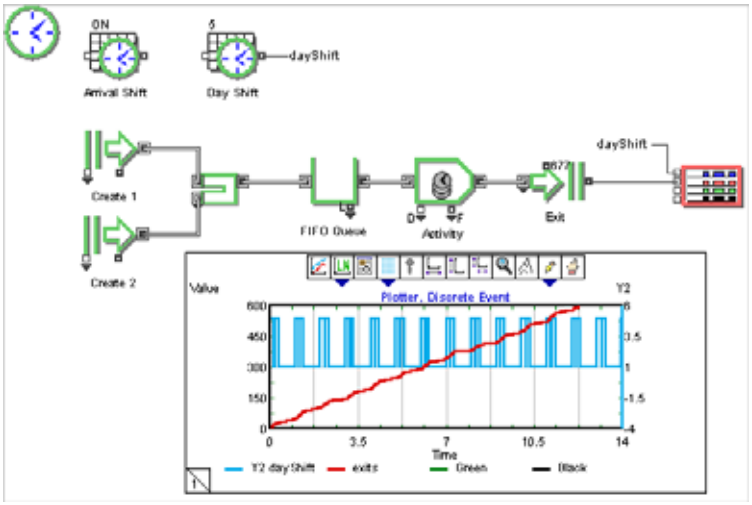
Another example of a more complex shift structure is the Controlling Shifts model where a Math block (Value library) monitors the processing at the Activity 1 block. While Activity 1 is processing, the shift is turned off, stopping any processing in Activity 2. An example of this would be if a worker is required to monitor two processes, but can only monitor one process at a time and must stop the second process while the first process is active.



Controlling Shifts model

Arrivals and Activity model

The Arrivals and Activity model illustrates how Shift blocks can shut down portions of a model without connections. This example shows the day shift dynamically setting the maximum capacity of an Activity to either 1 or 5 items and the Arrival shift turning the topmost Create block on and off at specified times.



Arrivals and Activity model

Advanced Resource Management

Advanced Resource Management (ARM) is an integrated ExtendSim system for organizing resources, distinguishing between them, and allocating them as required throughout a model. ARM provides a convenient and straightforward method for defining complex resource requirements for items as well as a flexible set of rules for how resources get allocated to them. And it provides automated methods for quickly changing resource information and generating statistical reports.

The ARM system is an advanced tool that is only available in certain ExtendSim packages. The manual titled *ARM Tutorial and Reference* explains how, why, and when to use Advanced Resources.

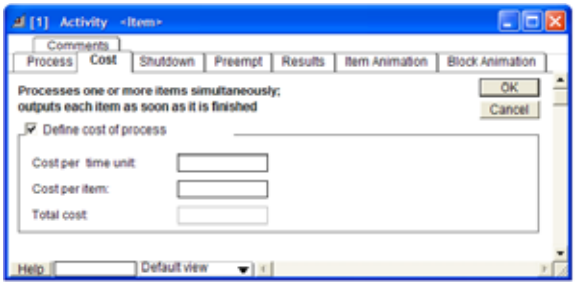
Discrete Event Modeling

Activity-Based Costing

Identifying and tracking fixed and variable costs
to determine operating costs

“The cost of a thing is often more
than the sum of the cost of its parts.”
— The Rev. P.N. Wallis

Activity-based costing (ABC) is a method of identifying and tracking the operating costs directly associated with processing items. It is the practice of focusing on some unit of output, such as a purchase order or an assembled automobile, and attempting to determine as precisely as possible its total cost based on the fixed and variable costs of the inputs. ABC helps identify, quantify, and analyze the various cost drivers (such as labor, materials, administrative overhead, rework, etc.) and determine which ones are candidates for reduction.



Cost tab of Activity block

Once a model has been built, the discrete outputs of the system, as well as the processes and resources that are involved in creating those outputs, have already been identified. To add ABC to models, you enter costing information into dialogs of blocks in the model. Blocks that generate items or provide resources, and blocks that process items, have fields and Cost tabs for specifying costing data. Enter variable cost rates per time unit and fixed costs per item or use. After the cost information has been defined, costs will automatically be tracked as the items in the model change state.

This chapter covers:

- Identifying cost accumulators and resources
- Defining fixed and variable costs
- How ExtendSim tracks costs

☞ The models illustrated in this chapter are located in the folder \Examples\Discrete Event\ABC.

Blocks of interest

The following blocks are the main focus of this chapter. Each block's library appears in parentheses after its name.

Cost by Item (Item)
Calculates the cost of every item in the model, as well as the average and total cost of the process.

Cost Stats (Report)
Records the input costs and total cost generated in each costing-based block. Determines total model cost based on a specified confidence interval.

In addition to the two Cost blocks, many of the Item library blocks have cost fields or a Cost tab for entering and reporting cost information, or have cost-handling capabilities, as shown in the table below:

Blocks with Cost tabs or fields	Blocks with cost-handling capability
Activity	Batch
Convey Item	Get

Blocks with Cost tabs or fields	Blocks with cost-handling capability
Create	Equation(I)
Queue	Set
Resource Item	Unbatch
Resource Pool	
Transport	
Workstation	

The Interchange block (Rate library) can also define costing information for items in discrete rate models.

Modeling with activity-based costing

To include activity-based costing in models, you need to know how to define cost rates, how to properly combine cost resources with items, and how to gather and work with cost information. Although an understanding of ABC is important, most of the work will be done by the ExtendSim architecture.

Item types

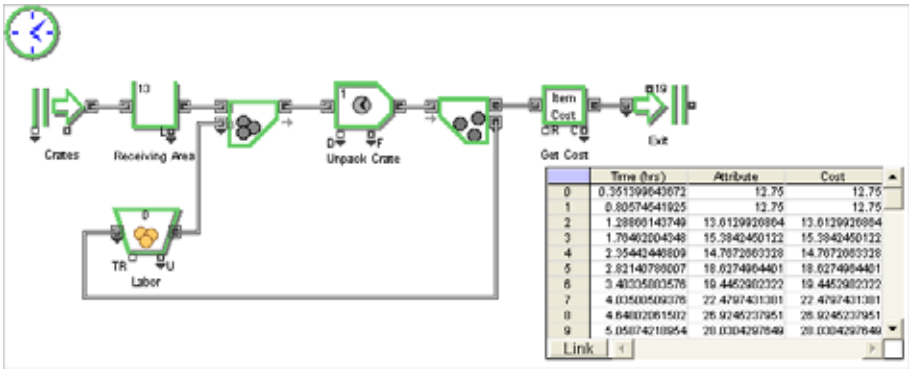
For purposes of costing, every item in a model can be categorized as either a *cost accumulator* or a *resource*. Understanding the difference between cost accumulators and resources is important, because ExtendSim treats them differently, as you will see in "Combining multiple cost accumulators" on page 392.

☞ As is true of items that are resources, non-item resources from the Resource Pool block do not accumulate their own costs.

Cost accumulators

You perform ABC to determine the costs associated with storing or processing an item. The item being stored or processed is called the *cost accumulator* and will accumulate costs as it waits, gets processed, or uses resources. Cost accumulating items can be introduced into a model using the Create and Resource Item blocks, as you will see in "Costs for cost accumulators" on page 383. The following example shows how costs are assigned to cost accumulating items.

Receive Inventory model



Receive Inventory model

Assume you want to determine the cost associated with receiving crated inventory at a warehouse. There is a one-time docking fee of $3.00 for every shipment that is received, and it costs $0.15 an hour any time the crate waits for processing (such as in the receiving area.)

As each shipment arrives, a labor resource takes an average of 30 minutes to unpack the crate and stock the contents on the appropriate shelves. In this case, the crate is being processed and is therefore the cost accumulator. The half-hour processing time and the hourly wage of the laborer is used to automatically calculate the cost of unpacking and shelving. That cost is then added to the accumulated cost being tracked with the crate. As the crate progresses through the steps of being received and unpacked, ExtendSim will add the cost incurred at each step to the accumulated cost.

Resources

As discussed in “Modeling resources” on page 361, resources can be modeled using resource pool blocks or by batching resources from a Resource Item block with other items. In the resource pool method, resource units act as constraints on the flow of items throughout the model. In the batching method, resource items are required to be batched with other items before the items can proceed to the next process.

Whether resource pool units or resource items, resources provide a service for the items in a model; they do not accumulate their own costs. Whenever a cost accumulator uses a resource, the resource’s cost rates are used to calculate costs which are then added to the total cost of the cost accumulator. (Cost rates are discussed on page 384.) For instance, in the Receive Inventory model described above, the laborer is a resource item that is batched with the crate. Since the laborer is a resource, it will not accumulate its own cost. Rather, the cost rate of the laborer (the hourly wage), and the time it takes the laborer to unload the crate, is used to automatically calculate the cost of unpacking the crate.

☞ The default is that the Resource Item block outputs resource items. However, you can select in the block’s Cost tab to output items as cost accumulators. This is discussed at “Resource Item block” on page 384.

Defining costs and cost rates

To include ABC in a model, simply enter information in the costing section of the dialogs of cost-aware blocks. There are two types of cost information:

- A direct or fixed cost, which is entered as the *Cost per item* or *Cost per use* in block dialogs.
- The variable cost rate, entered in block dialogs as the *waiting cost/time unit* or *processing cost/time unit*.

You do not need to define all cost information in order to perform ABC. However, if even one cost field is defined as a positive, non-zero number, ExtendSim will automatically track costs when the simulation is run.

Cost accumulating items have their own fixed costs and variable cost rates. As they use resources, wait for processing, and are processed, they acquire additional costs from resources, queues, and activities.

The following information describes how and where to define costs.

Costs for cost accumulators

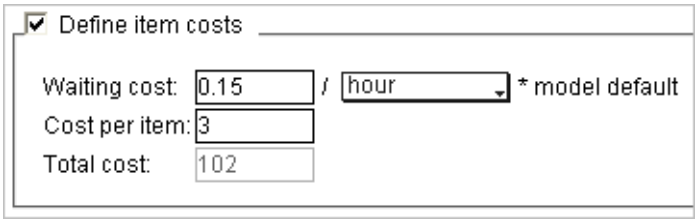
You specify costing information for a cost accumulator in the cost section or tab of the block that originates the item. Each cost accumulator can have a fixed cost per item, such as its direct materials cost, and a variable waiting or processing cost rate, which causes it to accumulate costs as it is stored or waits for processing.

Cost accumulators are usually generated by the Create block. They can also be provided by a Resource Item block, depending on a setting in its Cost tab.

Create block

Costing information is entered differently in the Create block depending on whether the block is set to *Create items randomly* or to *Create items by schedule*.

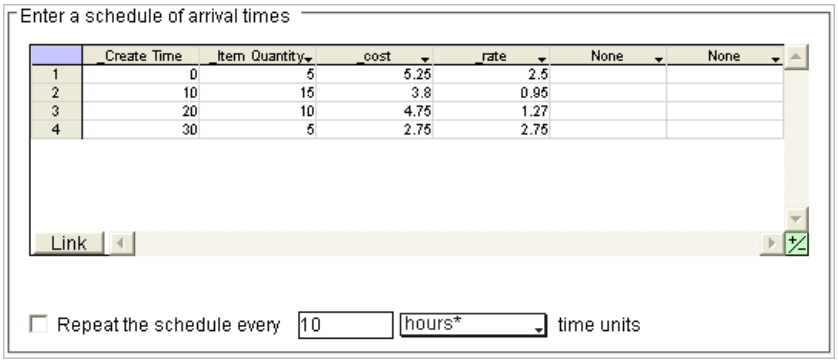
- In the Receive Inventory model described on page 382 the Create block generates crates randomly. The *Waiting cost/hour* and the *Cost per item* for each crate are defined in the cost section of the block's Options tab, as shown above.



Cost section of Options tab; block set to "Create items randomly"

- When the Create block generates items by schedule, you must explicitly set *_cost* and *_rate* system attributes (discussed in "Working with cost data" on page 387) for each cost accumulating item. The value of the _cost attribute should be set to the cost accumulator's fixed cost.

The value of the _rate attribute should be set to the cost accumulator's variable cost rate (waiting cost per time unit), as shown below.

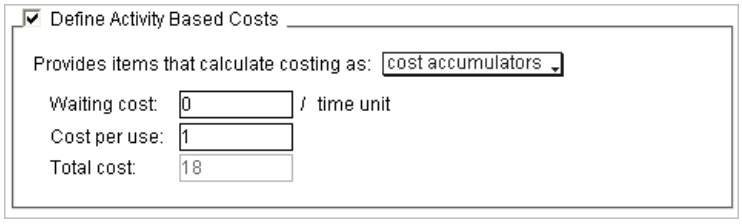


Create tab; block set to “Create items by schedule”

☞ The _rate attribute must be defined using the same time unit as the model’s default global time unit. In the example, the time unit is hours, so the _rate attribute is the hourly rate.

Resource Item block

Cost accumulators can also be provided for a model using the Resource Item block. To do this, you must choose that the block *Provide items that calculate costing as: cost accumulators* in the block’s Cost tab, a portion of which is shown at right.



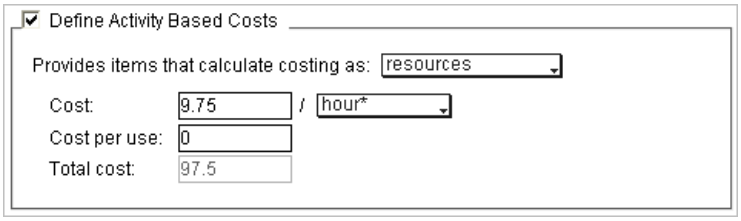
Cost tab of Resource Item block, providing cost accumulators

Costs of resources

The costs that will be assigned to items that require resources are defined in the Resource Pool and Resource Item blocks. The Cost tab in those blocks has fields for entering the following information:

- The cost per time unit rate, used to calculate and assign a time-based cost to the cost accumulator while it uses the resource.
- The cost per use is a one-time cost assigned to the cost accumulator for the use of that resource, such as a fixed service charge.

In the Receive Inventory model described earlier, cost rates for the laborer are defined in the Resource Item's Cost tab. The block's dialog indicates that items stored in the block are resources, as seen at right.



Cost tab of Resource Item block, providing resources

☞ For the Resource Item block to provide resources, the Cost tab must be set to *Items are: resources*, the default choice. Otherwise the items will be cost accumulators, as discussed earlier.

Activities

You can also define cost information in activity-type blocks; those costs are accumulated by each item the block processes. The activity-type blocks are the Activity, Convey Item, Transport, and Workstation. Within the Cost tab of these blocks, enter a cost per time unit and a cost per item:

- The processing cost per time unit is used to calculate the time-based processing cost of each item that passes through the block.
- The cost per item is a fixed cost added to every item that passes through the block.

☞ The Activity block has a PC (Processing Cost) input for targeting different processing costs on different items as well as a CPI (Cost per Item) input to attach a unique CPI to each item.

Combining resources with cost accumulators

As discussed in "Modeling resources" on page 361, there are two ways that items can use resources. One method is to batch a resource item with another item. While batched, the resource item is in use and cannot be used by another item until it is unbatched and returned to the Resource Item block. The second method is to use the resource pool blocks, which act as a constraint on the flow of items throughout the model. This section discusses how to properly use these two methods when performing ABC.

Batching and unbatching resources with cost accumulators

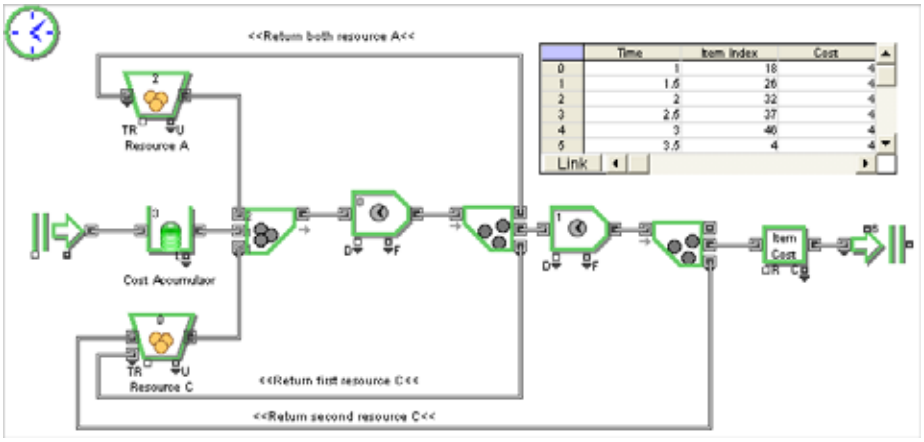
The Resource Item block holds resources for use in the model. When batching a resource item with a cost accumulator, the resource's cost rates are automatically stored with the cost accumulator and used in any subsequent cost calculations.

☞ A maximum of two separate types of resource items can be combined with a cost accumulator at a time. For instance, one or more worker resources from a Resource Item block and one or more cart resources from a different Resource Item block.

To unbatch a resource and remove its cost rate information from the cost accumulator, you must select *Release cost resources* as the Unbatch block's behavior. This choice tells the block to modify the information stored with the cost accumulator to indicate that the resource has been released.

☞ If the Unbatch block's behavior is set to *Create multiple items*, the items released by the block will be identical to the item which entered the block. In other words, there would be multiple copies of the cost accumulator, and each copy would still be joined with the resource.

Multiple Resources model



Multiple Resources model

In the example model, the cost accumulator is initially batched with 2 of Resource A and 2 of Resource C. When multiple resources are batched with a cost accumulator, they may be released all at once (as with Resource A), released incrementally (as with Resource C), or remain with the cost accumulator. Whenever a resource is batched or released, the cost array of the cost accumulator is updated to reflect the current number of resources in use. (The cost array is described in “Combining resources with cost accumulators” on page 390.)

There are three things to remember when batching resource items with cost accumulators:

1) A resource will be released from the output connector that corresponds to the input connector originally used to batch it to the cost accumulator. For example, if the resource entered the Batch block through the “ItemsIn(2)” connector, it will be released through the Unbatch block’s “ItemsOut(2)” connector.

☞ As in the Multiple Resources model, this could mean that one or more of the Unbatch block’s outputs will be unconnected and you will need to define that there will be zero items output through that connector.

2) If an item is simultaneously batched with different types of resources, you must use different connectors for each resource type when creating the batch. In the Multiple Resources model above, Resource A uses connector “ItemsIn” and Resource C uses connector “ItemsIn(2)”.

3) When performing ABC, you are limited to two different *types* of resource items batched with a cost accumulator item at one time. This limitation is not true, however, when modeling resources using the resource pool blocks, as you will see below.

Cost accumulators and the resource pool blocks

As described in “Resource Pool method” on page 365, as cost accumulating items pass through a Queue block in Resource Pool mode, resources are allocated to the items. When this happens, the cost rate of the resource pool unit is automatically stored with the cost accumulator and used in any subsequent cost calculations.

When a resource pool unit is released using the Resource Pool Release block, the information stored with the cost accumulator is modified to indicate that the resource has been released.

Unlike what happens where resource items are batched with other items, there is no limit to the number of different types of resources a cost accumulator can use when using the resource pool blocks. Furthermore, the two methods of modeling resources (batching resource items and resource pools) may be used in conjunction with each other.

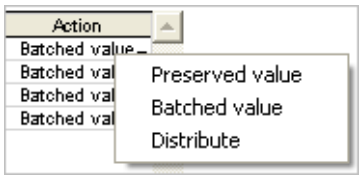
Combining cost accumulators

The Batch block can be used to combine cost accumulators arriving from one or more paths. This may be used in conjunction with an Unbatch block, for instance to combine cost accumulators for processing and separate them after processing has been completed. In the Properties tab of both the Batch and Unbatch blocks you can specify what the block should do with the cost values. This is accomplished by selecting an Action for the _cost and _rate attributes.

Costing attributes when items are unbatched

For example, assume a Batch block combines three cost accumulators together and that while batched, these items accumulate an additional $9.00 due to processing. When these cost accumulators are unbatched, you can select one of the following actions for the *_cost* and *_rate* attributes in the Properties tab of the Unbatch block:

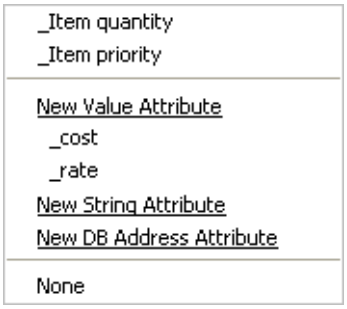
- *Preserved value*. This option causes the cost accumulators to retrieve their preserved value, if "preserve uniqueness" is turned on. In this case, the $9.00 is discarded.
- *Batched value*. With this choice, the $9.00 will be copied to each of the resulting cost accumulators.
- *Distribute*. The value will be divided among each item equally. In this case, $3.00 to each.



☞ For more information, see "Preserving the items used to create a batch" on page 357 and "Properties when items are batched" on page 351.

Working with cost data

To provide access to cost information, ExtendSim creates two attributes (*_cost* and *_rate*) for every item in models that have costing. Since these are automatically created, they are considered *system* attributes. If a cost is defined somewhere in the model, these attributes will appear in attribute popup menus, shown below:



Attribute popup menu

The information that is stored in these attributes depends on whether the item is a cost accumulator or a resource, as described in the following table:

Item type	_cost attribute	_rate attribute
Cost accumulator	The accumulated cost of the item	The item's waiting or storage cost (cost per time unit defined using the model's global time unit)
Resource	The cost per use of the resource	The resource's cost per time unit (defined using the model's global time unit)

The attribute handling blocks in the Item library (Get, Set, and Equation(I)) can be used to read, set, or manipulate these attributes. In addition, two statistics blocks in the Item library (Cost By Item and Costs Stats) can be used to gather cost data.

Viewing Cost Data

You can use a Get block to read the _cost and _rate attributes of any item, then plot the data or use the attribute value to perform additional calculations. For example, you can use a Get block to read the _cost attribute of cost accumulators and connect the Get block's _cost output connector to a Plotter Discrete Event to plot the accumulated cost of each item that passes through. This is shown in the model discussed in the following section.

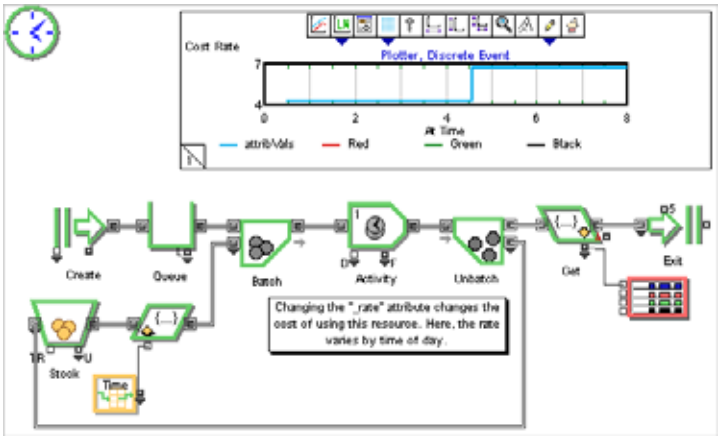
Changing Cost Data

In most cases, it is sufficient to define the cost rates of the various cost drivers in a model and allow ExtendSim to automatically calculate and track costs. However, there may be times when you need to manipulate the cost values generated. The attribute handling blocks in the Item library (Get, Set, and Equation(I)) can be used to accomplish this.

Change Rate model

For example, suppose the cost rates of a resource vary throughout the day. During peak times the demand for the resource is high and the cost per time unit increases. This can be modeled using the Set block (Item library) and the Lookup Table block (Value library) to explicitly set

the _rate attribute of the resource as it exits a Resource Item block, as shown in the model below:



Change Rate model

The table in the Lookup Table block has a different rates for the period between hour 4 and hour 6.

☞ A change in the rate will only affect resources as they exit the Resource Item block. Resources currently in use will not be affected until they are recycled back through the Resource Item and Set blocks.

In the above model, a Get block reads the _cost attribute before the items exit the model. The accumulated cost of each cost accumulator is then plotted. The plot (cloned onto the worksheet), shows that the cost of the items increases during the period of time that the resources' cost rates are higher.

Gathering and Analyzing Cost Data

The Create, Resource Item, and Resource Pool blocks, as well as queue and activity-type blocks, are capable of generating costs that get tracked with cost accumulating items. Additionally, each cost-generating block displays the total cost it generated in its *Total Cost* dialog item.

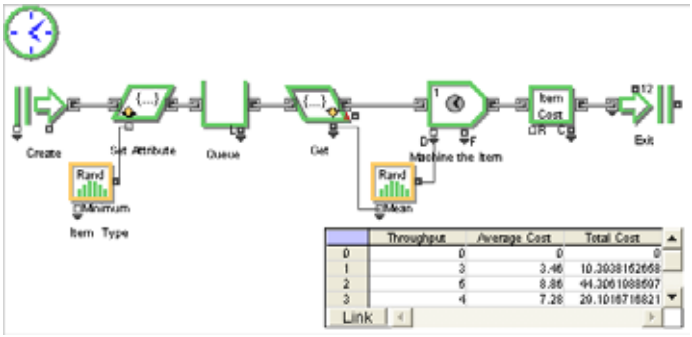
You can also use the Cost By Item (Value library) and Cost Stats (Report library) blocks to gather summarized cost information. The Cost By Item block reads and stores the _cost and _rate attributes of all the cost accumulating items that pass through it. The Cost Stats block collects and displays the total cost for each cost-generating block in a model.

Cost By Item block

Depending on selections in its dialog, the Cost By Item block lists the accumulated cost of each item that passes through it, the time the item passed through the block, and the total and average cost of all the items that have passed through. This block can also be used to list the cost of the items sorted by type.

In the Sort By Type model (shown below) three different item types are generated by randomly assigning a *Type* attribute of 1, 2, or 3. It costs $5.00 per hour to run the machine. The machine's processing time for, and therefore the cost of, each item varies by type. The Cost By

Item block lists the costs of the items sorted by the Type attribute. As an item passes through the block, the row corresponding to the value of the Type attribute (1, 2, or 3) is updated.



Sort by Type model

Cost Stats block
The Cost Stats block is useful to determine which blocks are contributing the most to the total cost of the items being processed. See the help text of the block for a detailed description of how to use the Cost Stats block.

How ExtendSim tracks costs

☞ The previous sections discussed how to perform ABC in ExtendSim. This section provides a more detailed look at how ExtendSim tracks costs and is included mainly for informational purposes.

Setting the _cost and _rate attributes

When you define the cost or cost rate for a cost accumulator or resource (as discussed in “Defining costs and cost rates” on page 382), ExtendSim will assign the value to the appropriate costing system variable for that item. The fixed cost of the item is assigned to the *_cost* attribute and the variable cost rate is assigned to the *_rat*e attribute.

Combining resources with cost accumulators

Whether you use the resource item and batching method or the resource pools method to model resources, two things happen when a resource is attached to a cost accumulator:

1) The value of the resource’s fixed cost is automatically added to the cost accumulator’s *_cost* attribute.

If using the batching method, the resource’s fixed cost comes from the resource’s _cost attribute. If using resource pools, the resource’s fixed cost comes from the *Cost per use* dialog item of the corresponding Resource Pool block.

2) The resource’s variable cost rate and the number of resources used are stored in an internal program structure called the *cost array*.

The cost array stores costing information for each cost accumulator in the model. ExtendSim uses the data in the cost array to calculate the time-based cost contributed by any resources that are combined with the cost accumulator. If using the batching method, the resource’s variable cost rate comes from the resources _rate attribute. If using resource pools, the resource’s vari-

able cost rate comes from the *Cost per time unit* dialog item of the corresponding Resource Pool block.

When a resource is released by an Unbatch or Resource Pool Release block, the information stored in the cost array is updated to indicate that the resource is no longer combined with the cost accumulator.

Calculating costs

As previously mentioned, the Create block and activity, queue, and resource-type blocks are all capable of generating costs. As these blocks process cost accumulators, they will automatically calculate the cost and add it to the item's _cost attribute. In addition, each cost-generating block will update its *Total Cost* information. This dialog item displays the total cost contributed by that particular block only. The following sections briefly discuss how these calculations are performed.

In the Create block

When a cost accumulator is generated, ExtendSim will add the fixed cost (*Cost per use)* of the Create block to the cost accumulator's _cost attribute.

For each cost accumulator generated, ExtendSim also will add the fixed cost of the Create block to its *Total cost* dialog item.

In activity-type blocks

When a cost accumulator enters an activity-type block, ExtendSim will add the activity's fixed cost (*cost per item*) to the cost accumulator's _cost attribute. In addition, it will calculate the variable time-based cost (the *processing* or *transportation* cost of the activity and the waiting cost of any resources currently combined with the cost accumulator), and add it to the _cost attribute of the cost accumulator.

For each cost accumulator that passes through the block, ExtendSim also will add the fixed and variable cost contributed by that activity-type block (not including costs contributed by any resources combined with the cost accumulator) to that block's *Total cost* dialog.

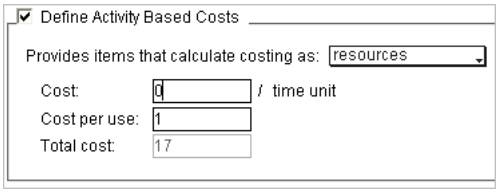
In queue-type blocks

Queue-type blocks have a checkbox labelled "Calculate waiting costs". If that checkbox is selected when a cost accumulator enters a queue-type block, ExtendSim will calculate the time-based cost. This is composed of the waiting or storage cost of the cost accumulator as calculated from the cost accumulator's _rate attribute and the variable cost of any resources currently combined with cost accumulator. The time-based cost is added to the _cost attribute of the cost accumulator.

For each cost accumulator that passes through a queue-type block, ExtendSim also will add the waiting cost calculated from the cost accumulator's _rate attribute (not including costs contributed by any resources combined with the cost accumulator) to that block's *Total cost* dialog item.

In resource-type blocks

The Resource Item block is capable of providing items that are either cost accumulators or resources, depending on selections in its Cost tab, as shown at right.



Cost options in Resource Item block

If the block is providing cost accumulators, it will generate costs similar to a queue-type block.

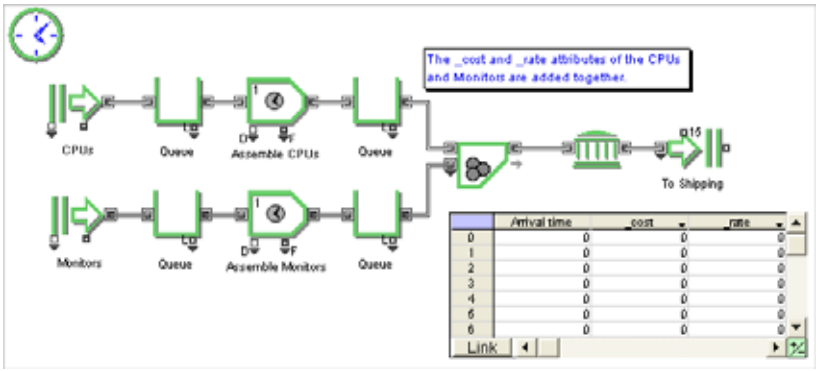
If the block is providing resources, the total cost of using the resources is calculated and displayed in the block's *Total cost* dialog item. The calculations are based on the resource's utilization rate and cost rates defined in the block's Cost tab.

Combining multiple cost accumulators

In manufacturing processes, different parts of the product may be worked on in parallel, then combined later to form the final product. In these cases, multiple cost accumulators will contribute to the cost of the final product.

Multiple Cost Accumulators model

For example, when a computer manufacturer prepares a system for an end user, the CPU and the monitor must each be assembled then combined into one shipment. The CPU and monitor may be worked on in parallel, then combined using a Batch block, as shown in the model below:



Multiple Cost Accumulators model

When the two cost accumulators, the CPU and the monitor, are batched together, two things will happen:

- The _cost and _rate attributes of the input items are added together. The resulting cost accumulator will have an accumulated cost equal to the combined accumulated cost of the input items and a waiting cost rate equal to the combined waiting cost rates of the input items.
- Any resources, whether from batching or from a resource pool, that are combined with the input cost accumulators will be combined with the cost accumulator that is output from the batching block. Note that any rules or limitations associated with batching resources with

items will apply to the resulting cost accumulator (see “Batching and unbatching resources with cost accumulators” on page 385).

Discrete Event Modeling

Statistics and Model Metrics

Statistically analyzing discrete event models

Remember that, by itself, simulation does not provide exact answers or optimize a system. Instead, a well-built model will capture important data and report statistical results. These metrics should provide the information needed for the analysis and decision-making process.

This chapter discusses specific methods for statistically analyzing discrete event models, such as:

- Gathering statistics for specific types of blocks
- Clearing statistical accumulators after a warm-up period
- Using the History block to get item information
- Tracking change-of-state information about items
- Using attributes to accumulate information about items
- Determining cycle time by timing the flow of items
- When to use time weighted statistics

For a more generalized discussions of statistical analysis, see also the following chapters:

- "Math and Statistical Distributions" starting on page 187
- "Analysis" starting on page 133

☞ The models illustrated in this chapter are located in the folder \Examples\Discrete Event\Statistics.

Commonly used blocks

The following blocks will be the main focus of this chapter. The block's library and category appear in parentheses after the block name.

Clear Statistics (Value > Statistics)
Clears statistics in other blocks, eliminating the statistical bias of the warm-up period.

Display Value (Value > Outputs)
Displays and outputs the value that is input.

History (Item > Information)
Records information about items and their properties, such as the value of an attribute, the item's arrival time, its priority, and so forth.

Information (Item > Information)
Reports item statistics such as a count of the number of items, the throughput rate, cycle time, and the time between item arrivals.

Item Log Manager (Report > Information)
Creates a log of item information from data found in History blocks and other blocks with Contents tabs.

Mean & Variance (Value > Statistics)
Calculates a mean, variance, standard deviation, and confidence interval.

Statistics (Report > Statistics)
Summarizes statistics for a particular type of block, such as activities or queues. Reports results in a table. Information is calculated using a specified statistical method, which can be customized.

Gathering statistics

The Statistics block (Report library) accumulates data and calculates statistics for a particular type of block using a specified statistical method. In addition to the block number, block name, and the time the information was observed, this block displays metrics that are specific to the block type, such as utilization or average wait time for activity-type blocks or the mean, variance, and standard deviation of all the Mean & Variance blocks in the model.

The Queue Statistics model, located in the folder \Examples\Discrete Event\Statistics, uses the Statistics block to gather information about queues.

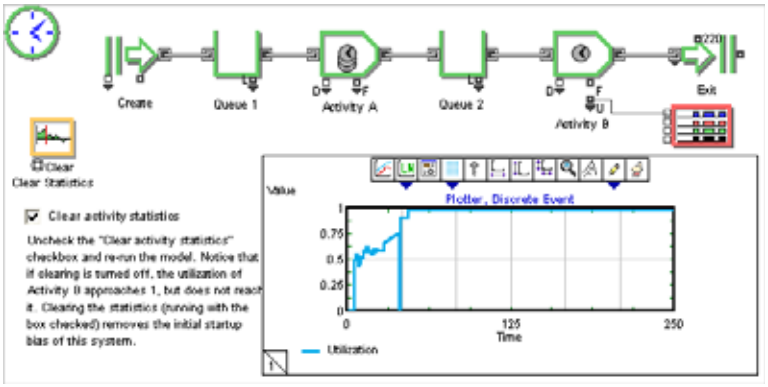
Since the Statistics block is used to gather information in continuous, discrete event, and discrete rate models, it is discussed fully in “Statistics” on page 134.

Clearing statistics

At the start of a simulation run the queues are often empty and operations have nothing to process. After the model has been running for a while, it gets to the point where it is functioning more like the real system at normal operating levels. The interval from when the model starts to when it is functioning in a steady or normal state is called the *warm-up period*.

The Clear Statistics block (Value library) is used to reset statistical accumulators for the blocks specified in its dialog, eliminating the statistical bias of the warm-up period. For more information about this block, see “Clear Statistics” on page 136.

Clearing Statistics model



Clearing Statistics model

In the Clearing Statistics model, statistics are cleared after 40 seconds, removing the warm-up period for the model. This is seen by the utilization of 1 for Activity B when the model is run. Unchecking the *Clear activity statistics* checkbox on the model worksheet causes the utilization of Activity B to approach, but never actually reach, 1. This is due to the effect of the initial idleness of the Activity B block at the start of the simulation run.

Using the History block to get item information

When building a model, it is important to start small, verify that the section you have built is working as expected, then enhance that model section. The History block is particularly useful for verifying model data because it provides important information about each item as the simulation runs.

There are two ways to add a History block to a model:

- Connect it in series by dragging a History block from the Item library and connecting it between other blocks so that items pass through it.
- Connect it in parallel by right-clicking an item output connector and selecting *Add History block*. This automatically connects a History block to the original block's item output connector. If a History block is added in this manner, only its input connector is used. (Caution: Be sure there is an Item library block connected to the original block's item output connector, otherwise its item will have no place to go.)

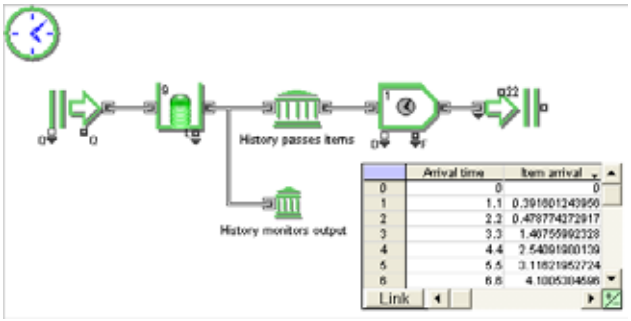
Each item that passes through the block (if it is connected in series) or is viewed by the block (if it is connected in parallel) is allocated a row in the History block's table. The table's first column displays the item's arrival time. Popup menus at the top of the other columns are for selecting additional information to display, such as the value of an attribute, an item's property, and so forth. You can choose to save item history with the model, show string attributes, and display Calendar dates.

☞ History blocks can use a lot of memory. To temporarily disable the recording of messages during a run, check the Disable Recording box in the Options tab. For blocks that have been added to the model by right-clicking, right-click to disable message recording or to delete the blocks.

History model

The History model shows two History blocks: one has been physically placed in series between a Queue and an Activity and one has been auto-created and placed in parallel to the Queue block.

Both blocks report the same information (the item's arrival time and the value of an attribute called *Item arrival*), as shown in the cloned table in the model window.



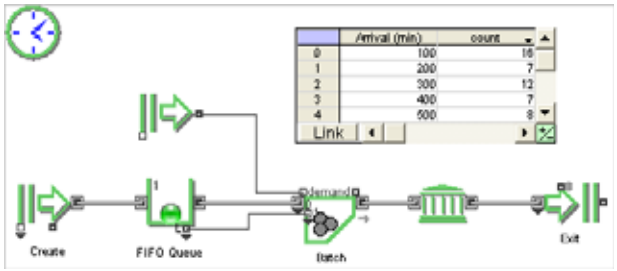
History model

☞ If a History block has been auto-created and placed in parallel to another block, there must be subsequent blocks that can pull the items in. This is discussed at "Pulling and viewing" on page 405.

Verifying Information model

The example shown at the right illustrates the use of the History block to verify that batches are created at the correct time with the correct number of items.

The topmost Create block (labeled Schedule Batches) schedules when the batch is created and the Queue's length (L) output determines the batch size. This causes all items within the Queue to be batched.



Verifying Information model

Using the Item Log Manager to get item information

The Item Log Manager (ILM) block (Report library) collects customized information about how item states change over time, then packages this information into a customizable database report. The block allows you to customize which blocks will contribute to the report, what information will be included in your log, and in what order the information will appear. For example, you may only want blocks with the label "Process 52" to collect data for attributes "SKU" and "waitTime". You can place as many different Item Log Manager (ILM) blocks in your model as you want.

The ILM tracks how items change states during a run by doing two things:

1) During the simulation run, it directs other blocks in the model to collect data on items as they pass through. These "remote" data collecting blocks include History, Activity, Queue, Queue Equation, and Resource Item. Each of these blocks can be remotely instructed to collect specific item property information for the ILM through its "Data Collection" tab.
2) Once the remote blocks have collected the data, the ILM grabs a subset of the collected data and packages it into a report. You can control which types of data are included in the report and the order in which the data appears. This is done on the ILM block's "Configure Log" tab.

Once the run is complete, any number of differently configured log reports can be generated from the data that has been collected in the remote blocks. This allows you to do all kinds of analysis on the data collected from just one run. These reports can be generated after each run either automatically or manually. You can have one or more ILM blocks in a model with each one responsible for generating its own report. The ILM also supports generating reports across multiple runs.

See the "Air Freight with Item Log" model (Examples/Discrete Event/Resources and Shifts) for an example of using item logging. Also see the help of the Item Log Manager block for a detailed example showing how it works.

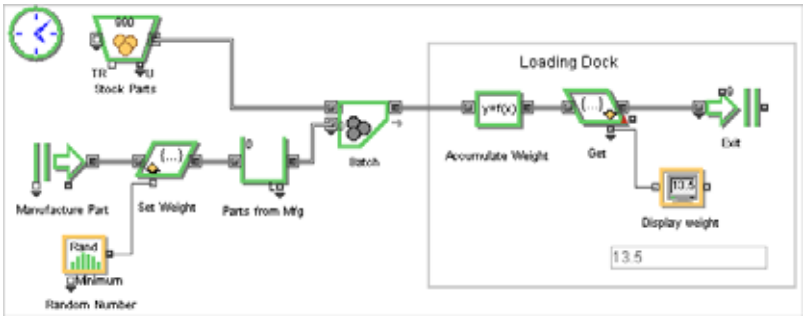
Accumulating data

There are various methods you can use to accumulate data. Attributes can be used to hold cumulative values, such as the total weight of an assembly or the number of parts in a box. And the Holding Tank block (Value library) can accumulate total processing time to determine equipment refurbishment schedules. Data can be accumulated at any step in the model, even when the item is not being processed.

⚠ It is important to not make the error of assuming that you can combine attribute values and then accumulate them.

Non-Processing model

In the Non-Processing model, one part from Stock and another from Manufacturing are combined into an assembly. The stock part weighs 10 pounds and the manufactured part weighs between 1 and 3 pounds. The model uses an attribute called Weight to track the weights of the separate parts.

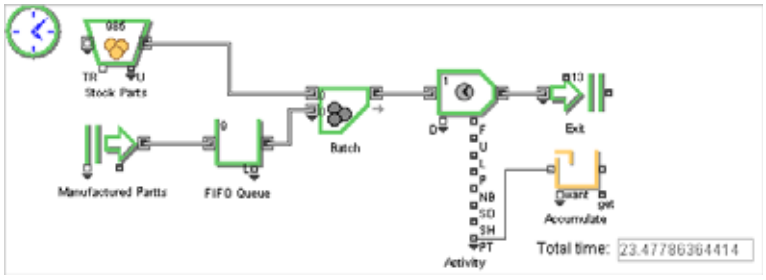


Non-Processing Model

The Properties tab of the Batch block is set to *sum* the values of the Weight attribute for the completed assembly. After the parts are batched, an Equation(I) block increments the Weight attribute by 0.5 pounds. At the loading dock, the weight of the current item is displayed as it leaves.

Processing model

If the data to be accumulated is dependent on processing, you can accumulate values using a Holding Tank block (Value library) connected to the *PT* (process time) connector on an Activity block. For example, to accumulate the total amount of processing time parts required, as an indication of when the processing equipment needs to be refurbished.



Processing model

In this model, each time an item leaves the Activity, its processing time will be added to the value in the Holding Tank.

Time weighted versus observed statistics

The Mean & Variance block (Value library) can calculate either a time weighted or observed statistic:

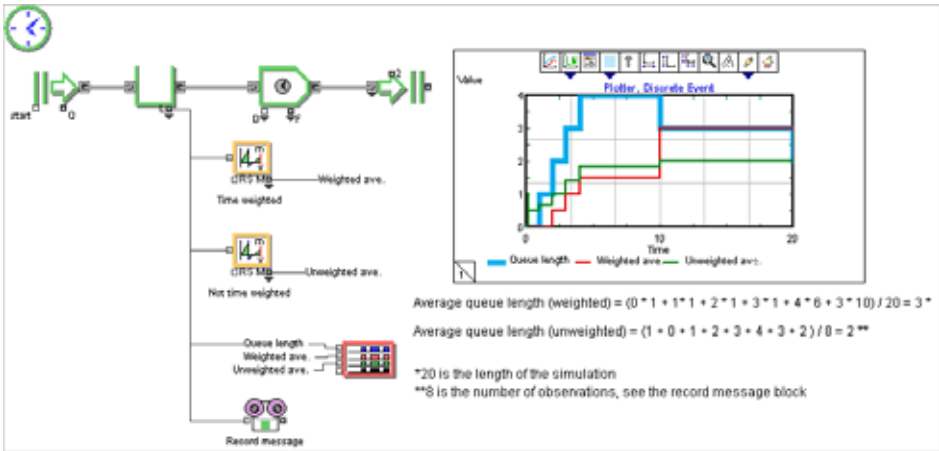
- If *Use time weighted statistics* is checked in the Mean & Variance block, the mean, variance, and standard deviation are calculated by weighting the input value based on the simulation time. This is derived by dividing the input value by the duration of that input value and then summing these over the course of the simulation.
- If *Use time weighted statistics* is not checked, the sum of the input values will be divided by the total number of input values received, resulting in an observed statistic.

When using the Mean & Variance block, carefully consider the type of statistics that you want to calculate. Some guidelines for whether or not to select the time weighted statistics option are:

- If the value that you are collecting statistics on has a value at every point in the simulation, enable time weighed statistics. A good example of this is the number of items in a block (reported by the L connector). At any point in the simulation, there are a certain number of items in a block. To determine the average value, weight it over time.
- If the value that you are collecting statistics on only has a value at specific events, do not use time weighted statistics. An example of this is the W or wait time connector. This connector only has a value when an item leaves the block, which is a specific event. In that case, time weighted statistics should not be used.

Time Weighted Statistics model

The Time Weighted Statistics example shows the difference between the two methods of calculating statistics and how they are calculated.



Time Weighted Statistics model

Comparing the weighted and un-weighted approaches to the average queue length reported in the Queue block's Results tab, it is clear that not using time weighted statistics would give an incorrect answer for this model.

Timing the flow of items in a portion of the model

In addition to performance information that is directly available in a model, you may want to determine *cycle time* – how long it takes an item to go from one part of the model to another. For example, you may want to know how long a customer waits in line to place an order, or how long it takes that customer to get served once the order is placed.

To see how to determine cycle time in a model, see “Cycle timing” on page 412.

Discrete Event Modeling

Tips and Techniques

Helpful information for when you build discrete event models

This chapter provides some tips, techniques, and information you may find helpful when building discrete event models. The chapter covers:

- Moving items through a simulation
  - How items move: holding and pushing. viewing and pulling
  - Implications of connecting to multiple item inputs
  - An item's travel time
  - Using scaling for a large number of items
  - Preprocessing
  - Restricting items in a system
  - Connecting to the *select* connector
- Issues for continuous blocks in discrete event models
  - Setting time-based parameters using a Random Number or Lookup Table block
  - Varying a distribution's arguments for the Create block
  - Accumulating values using a Holding Tank block
- Cycle timing
- Item library blocks
  - The Executive
  - Types of blocks: residence, passing, decision, and non-item
  - Common connectors on Item library blocks
  - Templates for common situations
- Event scheduling
- Messaging in discrete event models

☞ The models for this chapter are located in the folder \Examples\Discrete Event\Tips.

Moving items through the simulation

In general, item input connectors on discrete event blocks will pull an item in, do something with it, wait for the block connected to the item output connector to pull the item out, then pull in another item. For example, the Activity block will pull items from preceding blocks, process those items, and hold them to be picked up by another block.

It is important that you understand the ExtendSim discrete event behavior so you can avoid making modeling errors.

How items move through the simulation

It is important to understand how items move through specific blocks so that you can avoid two rare but possible pitfalls: losing items from the simulation and having items stop moving in the simulation.

To avoid the problems discussed below, you should probably connect Create blocks to queues so that items do not get lost and connect the History block in parallel with other blocks that will actually pull in the items.

Holding and pushing

Item library blocks treat their output items in one of two ways:

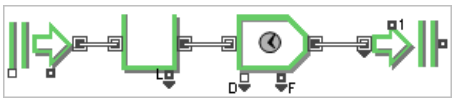
- Most blocks hold the item and it leaves only when another block pulls it in.
- When set to create items randomly, by schedule, or infinitely, the Create block pushes the item from the block when it is generated, regardless of whether it will be picked up by another block. The Create block has to push items out, because those items are created within the block and are *arrival time* related.

Avoid this pitfall

When a Create block pushes an item and it is not picked up, the item disappears from the simulation. Generally this would only be used in certain very specific types of models. In most situations where the Create block is set to create items randomly, by schedule, or infinitely, follow the block with a queue to collect the items and hold them, so that all the items generated are available for the rest of the model.

Unwise:

Wise:



Of course, if the Activity has an infinite capacity, it is not necessary to place a queue after the Create block.

⚠ A Create block set to *Create items infinitely* should *never* be connected to an infinite capacity queue, since generating an infinite supply of items would overwhelm the system.

Pulling and viewing

There are two ways a block's item input connector can have access to an item: it can pull an item from the preceding block (as most connectors do), or it can simply *view* an item that is waiting at the item output of the preceding block. If an item input connector pulls an item in, it has access to the item for processing. However, if an item input connector only views items, it does not have direct access to them, it can only sense their presence at the preceding output connector.

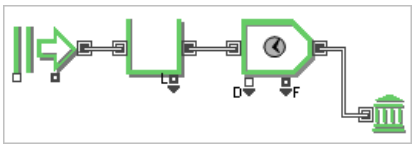
The particular connectors that only view items (not pull them) are:

- The Gate block's *sensor* connector when it is set to *Type: area gating* or its *demand* connector when the block is set to *Type: conditional gating with item*s.
- The item input connector on the History block, if the block has been added in parallel to another block. This is shown below and described in "Using the History block to get item information" on page 398.

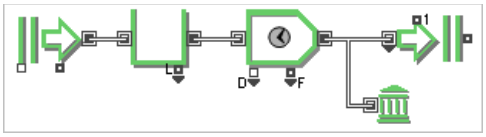
Avoid this pitfall
When one block holds an item and that item is *only* viewed by another block, the item does not move through the simulation and is blocked. This is never desired.

For example, both screenshots to the right show a History block that has been auto-created and is viewing items in an Activity block. In the top (error) screenshot, after the Activity block has finished processing its first item, the item will have nowhere to go since it is blocked.

Error:



OK:



Connections to multiple item input connectors

You can connect from one item output connector to as many item input connectors as you want. However, since items can only be in one place at a time, the first connector to pull in the passed item gets it and the other connectors do not.

Furthermore:

- If more than one input on a single block is free, the item will arbitrarily go to any available input. (Note that the selection is arbitrary – not random.)
- If more than one block is free to accept the item, the item will go to the block that was first connected in the model. This is shown in “Implicit routing” on page 302 and “Simple parallel connections” on page 318.

It is more typical that you would want to specify which input connector, or which block, would get an item. For more information, see “Items going to several paths” on page 299.

Unless it is completely unimportant in the model, you should always use the Select Item In and Select Item Out blocks to explicitly state how items should be routed. Otherwise, the order in which their connections were made will dictate the routing.

An item’s travel time

In a discrete event model items travel from block to block as dictated by the connection lines. The lines between blocks indicate the path of the movement, but they don’t provide any delay to the items.

In most cases, travel time is insignificant and can be safely ignored. Where an item’s travel time is significant to the model, you can:

- Increase the delay time of destination blocks to compensate for the travel time
- Specify a minimum wait time in a Queue block’s Options tab
- Explicitly set a travel time in a Transport or Convey Item block, as discussed in “Transportation and material handling” on page 338. (You can easily add a Transport block between two blocks by right-clicking the leftmost block’s output connector and selecting “Add transport block”.)

Using scaling for large numbers of items

In discrete event modeling problems, the number of items that need to be processed through the simulation may be quite large. This will slow down the execution of the model and increase the amount of memory required. It is often possible (and non-destructive to the validity of your results) to scale down the number of items passing through the model. For example, if there is one item representing a single log in a simulation of a lumber mill, the same model could quite possibly run faster, and equally well, with one item representing ten logs.

When you make scaling changes to a model of this nature, it is very important to reflect the changes everywhere in the model. Thus, if an activity that represented a saw in the lumber mill was set to take one time unit to process an item (one log) before, it must now take ten time units to process the same item (ten logs) after the scaling.

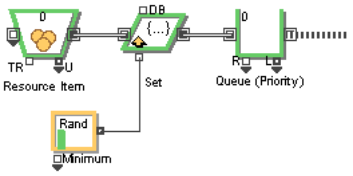
☞ While scaling can sometimes be a useful approach, the Rate library is specifically designed to model high volume and/or high speed systems. In most cases, using the Rate library is superior to item scaling. The Rate library is available with the ExtendSim Pro product.

Preprocessing

You sometimes want to have all the items available at the beginning of a simulation instead of generating them as the simulation proceeds. For instance, if you need some random orders presented to the model in sorted order, you might want to sort them before the simulation starts. This is difficult under normal circumstances since the first order would begin traveling through the simulation as the second one was being created. There is an easy method that will cause ExtendSim to create lots of items, store them in a queue, and release them.

Set the initial value in a Resource Item block to the number of items you want to generate. Connect the Resource Item block to a Set block where you attach item properties (priority, attribute, etc.). Then connect to a Queue that sorts based on the desired item property.

When the model is run, all the items will travel from the Resource Item block to the Queue on step zero. This makes the items, with all their properties, available to the rest of the model at the start of the simulation.



Preprocessing

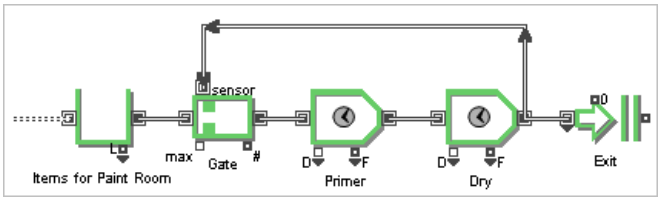
☞ If there are many items in the Resource Item block, the status bar may show the phrase *Initializing Data*. As soon as the preprocessing is done, the timer will settle into a more useful number.

Restricting items in a system

As part of a model you may want to have a section composed of a group of blocks in which only one item (or a limited number of items) can be anywhere in the section at a time. For example, assume you are modeling a manufacturing process with a paint room. There are many blocks that represent the steps in the paint room but only two items are allowed in the entire paint room at a time. New items must be restricted from entering the room until one or more items leave.

When set to *Type: area gating*, the Gate block is perfect for this because its *sensor* connector tells it each time an item has reached the end of a system. The number of items allowed are set in the Gate block's dialog; in this case, two. Put the Gate block at the beginning of a system; at the end of the system, run a parallel connection from the output of the last block to the Gate block's sensor connector.

In this example, the paint room is represented by two Activity blocks. The Gate block will pass the first two items it receives into the paint room, then only let a new item pass when it sees the first item at its sensor connector. As each item leaves the paint room, a new item can enter. Note that the sensor connector doesn't accept any items; it only views them as an indicator of their position in the model.

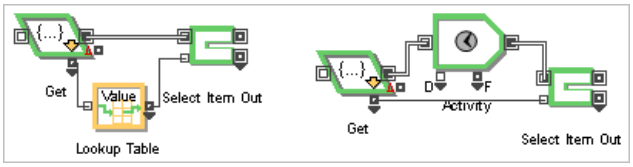


Restricting items

☞ Another, more flexible, approach is to use a Resource Pool block to restrict items in a section of the model. This is useful when you need to track statistics on utilization, or if you have multiple flows of items accessing the same physical space.

Connecting to the select connector

The *select* connector is used to control the behavior of the Select Item In and Select Item Out blocks. If the select connector gets its value from a Get block, you should avoid putting Set blocks, activities, and queues between the Get block and the Select block. These blocks can alter the value sent to the *select* connector or delay the item so that the Select block routes the wrong item.



Safe and unsafe connections to the select connector

For instance, the model segment shown at the left of the above screenshot will work properly. The one shown to its right may not work correctly, because the item to be routed may still be in the Activity block.

Continuous blocks in discrete event models

Value library blocks can be considered passive blocks in discrete event models. In most cases, blocks from the Value library will not recalculate unless told to do so by an Item library block. This has important ramifications for the behavior of Value library and other continuous blocks in discrete event models.

When an Item library block needs a new value at one of its value input connectors, it will send a message out that connector to the connected Value library block, requesting a new value. Likewise, when an Item library block has calculated a new value at one of its value output connectors, it will send a message to the connected Value library block notifying it of the change. Typically these messages will cause the Value library blocks to recalculate. The messages are then propagated to all other connected Value library blocks.

In discrete event models, most blocks from the Value library typically neither post events to the Executive nor receive event messages from the Executive. Furthermore, Value library blocks do not recalculate on each time step in discrete event models as they do in continuous models. Rather, they are only alerted to recalculate if they receive a message from an Item library block. And most Item library blocks are triggered to action only by the arrival of an item. Complications can arise if a Value library block does not get a message from an Item library block when you expect or want it to recalculate.

☞ Some Value library blocks, such as the Clear Statistics and the Lookup Table, do generate events in a discrete event model because they need to perform a specific action at a scheduled time.

To prevent modeling errors, it is helpful to understand this relationship between Item and Value library blocks. Common situations where this is important include:

1) Setting time-based parameters using connections from a Random Number or Lookup table block. This is described on page 409.
2) Varying an argument for a Create block's distribution with a Lookup Table where there is the possibility of a message being ignored. This can cause a lot fewer items to be created than expected.
3) Using a Holding Tank block to accumulate the result of a calculation performed on two or more values coming from Item library blocks. If not modeled properly, the Holding Tank can get duplicate messages and will have incorrect results.

☞ For a detailed discussion about messaging between discrete event and continuous blocks, see "Value input and output connector messages" on page 419.

Setting time-based parameters using connectors

Some time-based parameters can be set using a connector value. In these situations, the value sent to the input connector must be defined in the time unit specified in the receiving block. The following examples illustrate issues you should be aware of.

Random Number block

Assume you want the delay for an Activity block to be approximately 30 minutes and you connect a Random Number block to the Activity's D input connector. If the local unit of time for the Activity is minutes, you would set the Random Number block to generate numbers with a mean of 30. However, if the Activity block used hours as its local time unit, the Random Number block should be set to generate numbers with a mean of 0.5.

☞ It is a modeling error to expect the Random Number block to create random values at each event in a discrete event model. The only time this Value library block will be activated to output a new value is when it receives a message on one of its connectors. In the above example, the Random Number block will get a message each time an item arrives to the Activity block, so each item will get a random delay time. For more information, see "Value input and output connector messages" on page 419.

Lookup Table block

It is possible that the numbers in one column of a block are based on the time unit for that block, and the numbers in another of its columns are based on the time unit for a second block. An example of this is described in "Choosing time units for the columns" on page 261.

Varying a distribution's arguments

It is common to use another block to specify the arrival intervals by varying the argument (such as the mean) of a distribution in the Create block. To avoid unexpected results, it is important to understand what happens in the Create block when you do this. The Create block's default behavior is to send a message to the Executive block giving an arrival time, called "nextTime", for the next item based on the current input parameters. When simulation time reaches nextTime, the Executive block sends a message to the Create block. The Create then releases the item and generates a new nextTime based on the current values of the input

parameters. For the period of time between releasing items, the Create block will not react to changes in the input parameters. If the inputs change drastically, this can cause unexpected results as shown in the following example.

Lookup Table example

Assume you connect a Lookup Table block to the *mean* input connector of a Create bock, varying the interarrival mean according to the schedule in the table at right.

	Input Value	Output 1
0	0	12
1	6	0.5
2	14	12
3	24	12

Link

Lookup Table's table

For this example, both the Time and Output 1 columns of the Lookup Table block are defined using hours as the time unit. The mean for the interarrival time is 12 hours except for the period between hours 6 and 14 where the mean is 0.5 hours. Because the mean is only an average, it is possible for the Create block to generate an item at time 0 with an arrival time of 14 or more. If that happens, the Create block will get the message that the mean should have changed to 0.5 between hours 6 and 14, but it will ignore it. In this case, the number of arrivals will be much less than expected.

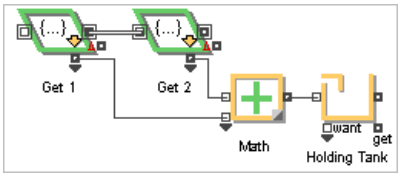
The Options tab of the Create block has a check box labeled *Interarrival time changes occur immediately*. When checked, it will cause the Create block to immediately respond to changes to any input parameter. In the case of the above example, the Create block would recognize that the mean had changed from 12 to 0.5 at time 6. It would then generate a new random number for the arrival time using the new input values.

Using the Holding Tank block to accumulate values

When accumulating data in a model, it is important to not make the error of assuming that you can add attribute values and then accumulate them.

Incorrect approaches

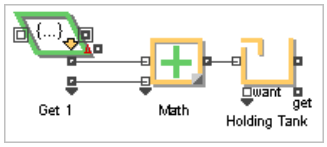
Assume you want to accumulate the sum of two attribute values. Your first intuition might be to add the two attribute values together and send the result to a Holding Tank block (Value library).



Incorrect approach #1

Two incorrect approaches to do this are shown at right. In the first case the attribute values are obtained from two Get blocks; in the second case the attribute values are captured from one Get block. In both cases the Holding Tank will give incorrect results.

Each time an item passes through a Get block, a message is sent out each value output connector. The way this been constructed, the passing of one item will result in two additions being contributed to the Holding Tank block.

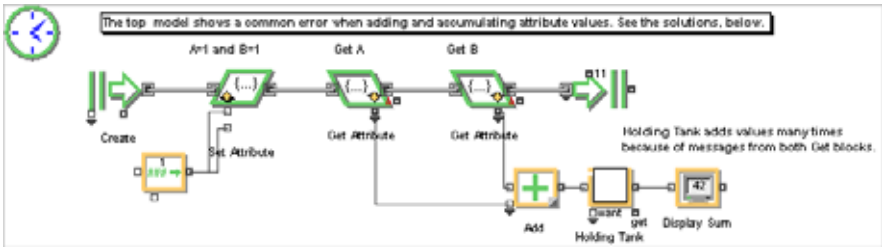


Incorrect approach #2

Attributes Error model

The Attributes Error model illustrates the modeling problem and some potential solutions. The problem with this model is clear if you look at the numbers, 11 and 42 respectively, displayed by the Exit and Display Value blocks. Since the values of attribute A and attribute B are both 1, the accumulated total

displayed on the Display Value block should only be twice the value displayed in the Exit block; clearly this is not the case in this model.



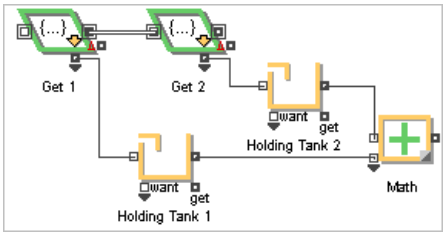
Attributes Error model: Problem

The reason for the modeling problem shown above involves the message passing system in discrete event models. Individual items travel through the Get blocks sequentially. As an item passes through the first Get block, the block sends a message and the value of the attribute to the Math block (Value library). The Math block then recalculates and sends a message and the value to the Holding Tank block (Value library). When the item moves to the second Get block, it will send a message to the Math block again. This causes the Holding Tank block to get two messages and two values for each item that passes through the system. This kind of problem will occur in any discrete event system where there are multiple connections to a Holding Tank block (either directly, or indirectly as shown above) or if one Get block was used with two outputs.

The Attributes Error model includes four examples: the problem and the three solutions discussed below.

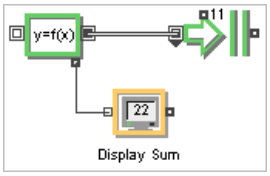
Solution #1: two Holding Tank blocks

One way to solve this problem is to accumulate the attributes' values separately using two Holding Tank blocks. The contents of the Holding Tanks are then added together. This prevents the "double counting" of the previous example, because each Holding Tank block receives only one message and value per item that passes through the Get block it is attached to.



Solution #2: the Equation(I) block

Another solutions is to perform the calculation in the Equation(I) block (Item library). The Equation(I) sums up and accumulates the attributes in one step, so it avoids the double messaging problem altogether. The equation entered is:

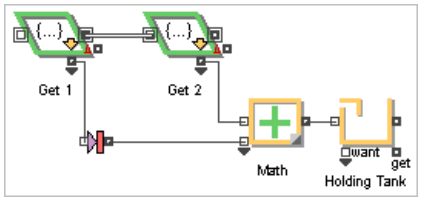



```
Accumulate = Accumulate + a + b;
Result = Accumulate;
```

where *accumulate* is a static variable and *a* and *b* are the two attribute values from the item entering the Equation(I) block. The equation adds the two attribute values to the accumulated value, then sets the output to the accumulated value.

As an alternative, instead of both summing the attributes and accumulating, the Equation(I) could just sum the attributes and output that value to a connected Holding Tank.

Solution #3: the Stop Message block



You can also use the Stop Message block (Utilities library) to prevent the Math and Holding Tank blocks from receiving two messages for every one item. This block stops messages from being passed through a value connection; it is designed to solve problems of this nature.

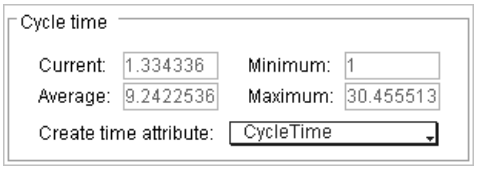
The Stop Message block is connected between the value output of Get 1 and the value input on the Math block. This will prevent the first message from reaching the Math block but will allow the value to be passed.

Cycle timing

The amount of time one block takes to process an item is known as the *delay* or *processing* time. *Cycle time* is the time an item takes to travel through a group of blocks. If there is no blocking in a model (that is, if all items leave their blocks exactly at the end of their delay time), the cycle time is the sum of the delay times for the section being measured. In most situations, this would rarely occur, and cycle time is usually more than the sum of the processing times. For instance, it is common that an item cannot leave a block because the next block is still processing its item.

To track an item's cycle time, use either the Timing attribute feature (if the item is being tracked from its origin) or a Set or Equation(I) block with an Information block (if the item is being tracked from some place other than its origin).

These methods are discussed below. In each case, the Information block reads the attribute and calculates the difference between when the item started the cycle and when it ended. The dialog of the Information block displays the current, average, minimum, and maximum cycle time for all items with the specified attribute. Its output connectors report the count of items, the time between items, their cycle time, and the throughput rate.

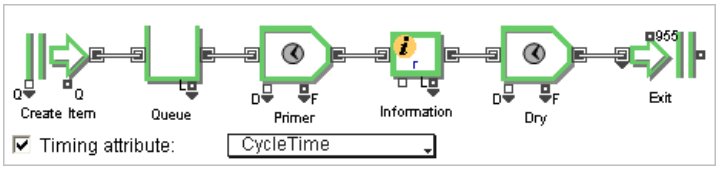


Cycle time portion of Information dialog

Using the Timing attribute feature

If you are tracking the item from its origin, use the *Timing attribute* feature in the Create block's Options tab to create a new value attribute and assign it to all items that are generated by the Create block. Then place the Information block at the end of the section you want to observe and select the name of the attribute as the *Timing attribute* in its dialog.

In the example shown at right, a value attribute named CycleTime has been created in the Create block, and the block's Options tab is set to *Timing: CycleTime*. The Information block is placed after the Primer activity and is set to *Calculate TBI and Cycle Time statistics*, and its *Timing attribute:* is also set to *CycleTime*. For this model, the Information block



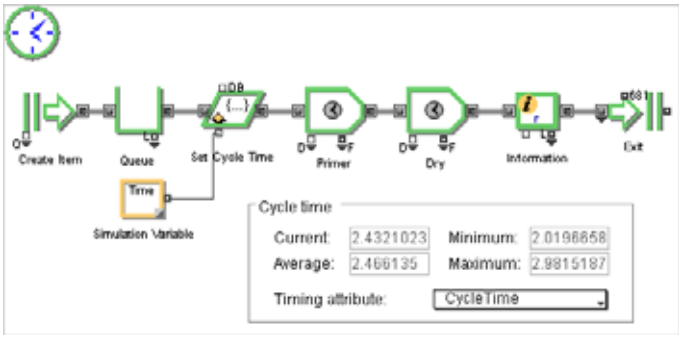
Cycle Time 1

calculates the time from when items were first created to when they finish being primed. This includes the time items wait in the Queue.

Using a Set or Equation(I) and Information blocks

If you are tracking the item from some place other than its creation point, put a Set or Equation(I) block at the beginning of the section you want to observe, create a new value attribute in that block, and set the attribute to the current time. Then place an Information block at the end of the section and enter the name of that attribute as the *Timing attribute* in its dialog.

In the example at right, an attribute named CycleTime2 has been created in the Set block. The Simulation Variable block (Value library) outputs current time and is attached to the Set block's value input connector. For this model, the Information block calculates just the time that the items take to go through the priming and drying processes.



Cycle Time 2

Item library blocks

Executive block

The Executive block controls and performs event scheduling for discrete event and discrete rate models. An Executive block must be placed to the left of all other blocks in a discrete event or discrete rate model. Using it in a model changes the timing from continuous to discrete, and simulation time advances when events occur, rather than periodically.

This block can be used to:

- Manually control when the simulation stops. The default is that a simulation stops at the end time set in the Run > Simulation Setup dialog. You can also choose to stop the simulation when the number at the Executive's *count* input reaches a specified value.
- Allocate item availability. To conserve memory, this number should be something close to and less than the maximum number of items that you expect to see in the simulation. The default is that 10,000 items are initially available and additional items are made available in batches of 10,000.
- Declare and manage string attributes for items in Item library blocks.
    - A table in the Item Attributes tab is used to enter a descriptive text label (string) for each potential attribute value for a selected string attribute. For a tutorial on how this is done, see the Discrete Event Quick Start guide and "Item attribute types" on page 264.
    - Other tables in the Item Attributes tab allow you to select an attribute for renaming or deleting, and display blocks that use the selected attribute.
- Declare and manage string attributes for flow in Rate library blocks

  - A table in the Flow Attributes tab is used to enter a descriptive text label (string) for each potential attribute value for a selected string layer attribute. For more information, see “Flow attributes” on page 431.
  - Other tables in the Flow Attributes tab allow you to select a flow layer attribute for renaming or deleting, and display blocks that use the selected attribute.
- Manage flow units and select global and advanced options for discrete rate models. For more information, see “Global and advanced options in the Executive” on page 514.
- Set information for the LP solver used in discrete rate models. For more information, see “LP technology” on page 526.

☞ Unless you use string attributes, it is rare that you would need to make any changes in the Executive’s dialog. Most of its options are for advanced users.

Templates

The Item Templates library contains templates – submodels within hierarchical blocks – for common modeling situations such as emptying a queue at intervals or selecting the shortest queue.

Block types

As discussed on “Item connector messages” on page 421, Item library blocks pass messages through item connectors. There are three types of item-based blocks that determine how the item connector messages are handled:

- Residence-type blocks are able to contain or hold items for some duration of simulation time. Some residence blocks post events and some do not.
- Passing-type blocks pass item through without holding them for any length of simulation time. These blocks implement modeling operations that are not time-based; they usually do not post future events.
- Decision-type blocks route the items through the model. These blocks choose a route based on an item property, a random value, a sequence, or an input from a connector. Depending on what options are selected in the block, a decision-type block may or may not be able to hold onto items.

Why block types matter

Knowing these categories of blocks and how they relate to the processing of items will help you to build better models. For example an item will not enter a passing block before it has been determined that there is space in the next downstream residence block. And when you debug models it is useful to understand where the items can reside for any amount of time, as well as the time required for an item to move from one residence block in the model to another. In addition, some of the options in the blocks refer to specific block types. An example of this is the Transport block where you can specify that the distance to the next block is from the Transport block to the next non-passing (residence or decision) type block.

Table of block types

Not all of the blocks in the Item library fit neatly into these categories, but it is helpful to use the categories as a framework for thinking about the messaging architecture. Following is a table of the blocks in the Item library and their associated type.

Block	Type	Block	Type
Activity	Residence	Queue Matching	Residence
Batch	Residence	Read(I)	Passing
Catch Item	Passing	Resource Item	Residence
Convey Item	Residence	Resource Pool	N/A
Cost By Item	Passing	Resource Pool Release	Passing
Cost Stats	N/A	Select Item In	Decision
Create	Residence	Select Item Out	Decision/Residence*
Equation(I)	Passing	Set	Passing
Executive	N/A	Shift	N/A
Exit	Residence	Shutdown	Residence
Gate	Decision	Throw Item	Passing
Get	Passing	Transport	Residence
History	Passing	Unbatch	Residence
Information	Passing	Workstation	Residence
Queue	Residence	Write(I)	Passing
Queue Equation	Residence		

* If an item is allowed into the Select Item Out block before the decision is made (see the dialog check box), then it is a residence-type block. If the decision is made before the item enters the block, then it is a decision-type block.

Common connectors on discrete event blocks

Many blocks use abbreviations or acronyms to indicate a connector's purpose. Some of these represent more than one purpose and are context sensitive. The following connector labels appear on many Item library blocks:

Connector	Meaning
Δ	Delta
#	Count
A	Average cost (Cost By Item)
AD	Accumulated demand (Gate)
AS	Activity status
BT	Blocked time

Connector	Meaning
C	Capacity (Activity, Workstation) Current cost (Cost By Item)
CI	Confidence interval
CT	Cycle time
D	Delay (Activity)
DB	ExtendSim database
DT	Down time (Activity)
DV	Down value (Shutdown)
F	Full (Activity, Queue) Field of a database table (Read Item, Write Item)
I	Interval between items
L	Length of waiting line (Queue, Activity, Workstation) Length of line (Information - will be 0 or 1)
LO	Length out (Queue Matching)
MG	Match group (Queue Matching)
NB	Number blocked
P	Priority
PE	Preempt
PT	Process time
Q	Quantity
R	Renege time (Queue) Row (Cost By Item) Record (Read Item, Write Item when ExtendSim database is selected)
RS	Reset
SD	Shut down
T	Total cost (Cost By Item)
TBF	Time between failures (Shutdown)
TP	Throughput rate (Information)
TTR	Time to repair (Shutdown)
U	Utilization
UV	Up value (Shutdown)
W	Wait time for items leaving the queue

Event scheduling

ExtendSim moves items in a discrete event model only when an event happens. Events are controlled by the Executive block and only occur when particular blocks specify that they should. Blocks that depend on time cause events to happen at the appropriate time. For

instance, an Activity block holding an item until a particular time will cause an event to be posted to the ExtendSim internal event calendar. When the time is reached, the event occurs and the model recalculates its data.

Blocks that do not generate events allow the blocks after them to pull items during a single event. Thus a single event can cause an item to pass through many blocks if those blocks do not stop them. For instance, a Set block could set the item's attribute and pass the item to the next block in the same event.

Discrete event and discrete rate simulations use the same method for updating the simulation clock. Simulation models of this type are driven forward by event and the state of the model changes only at event times.

At each event, blocks that have posted an event to the event calendar for the current time receive a message notifying them that the time has arrived. Once all of the blocks have received their messages, the time for the next event is determined. Through this event scheduling mechanism the simulation clock jumps from one event to the next.

Event calendars

ExtendSim utilizes a two-stage event calendaring method – the Executive block maintains a list of all events for the model and time-delay blocks maintain their own event calendars.

☞ This two-stage event calendar is very efficient and flexible. Unlike single stage event calendars, relatively little time is spent by the Executive in maintaining and searching the event list.

The Executive

The Executive block maintains a list of all event times for the model in its event calendar. At the beginning of each simulation event, the Executive locates the next future event and sends a message to each of the blocks in sequence that posted an event for that time. Once a block has completed processing its event, it will post its next event time to the Executive. If the block does not have a future event time, it will post a very large value as its next event time, effectively removing it from the list of pending events.

Blocks may have two or more entries on the Executive's event calendar. This is because they have different types of events that need to be processed. For example the Convey Item block has an event that occurs when an item is able to enter the block and an event for when the item leaves the block.

For more information about the Executive block, see page 413.

Internal event calendars

Each block that has a time delay associated with it (for example the Create, Activity, Pulse, and Shutdown blocks in the Item library) maintains its own, independent next event time.

Blocks such as the Activity, Convey, or Shutdown block can have multiple future events (one event for each item in the block) ongoing simultaneously. In this case, the blocks maintain their own internal event calendar, posting only the earliest of these events to the Executive's event calendar.

Zero time events

As the simulation progresses, there are many times when it is useful to generate a zero time event. This is done to allow an item to complete the process of moving into a block before the block attempts to perform additional actions on the item. For this purpose, the Executive main-

tains a current events list. This is a short list of the blocks in the model that need to receive a message before the simulation clock advances.

A prime example of this is the Queue block. When an item arrives to a Queue, a zero time event is posted so that the Queue can return control to the upstream block that sent the item. The Queue receives another message before the clock advances so that an attempt can be made to send the item to the next downstream block. This feature enhances the efficiency and predictability of discrete event models.

Event Scheduling model

A discrete event model is helpful in understanding how event scheduling works. For this example: items arrive, wait at the first queue, are processed at the first activity, wait at the second queue, are processed at the second activity, and leave through an exit.

Event Scheduling model

In this model, there are three blocks that post events:

- The Create block posts an event for the creation of each item. The time between item arrivals is 0.6.
- The Activity 1 block posts an event for the earliest completion time of an item in the block. The duration of this activity is 1.0.
- The Activity 2 block posts an event for the earliest completion time of an item in the block. The duration of this activity is 0.5

As the simulation progresses through time, the event calendar in the Executive might look like this:

Time	Create posts next event time	Activity 1 posts next event time	Activity 2 posts next event time	Events
0.0	0.0	Infinity	Infinity	Item #1 is created
0.0	0.6	1.0	Infinity	Item #1 begins service at Activity 1
0.6	1.2	1.0	Infinity	Item #2 is created
1.0	1.2	2.0	1.5	Item #1 completes service at Activity 1 Item #1 begins service at Activity 2 Item #2 begins service at Activity 1
1.2	1.8	2.0	1.5	Item #3 is created
1.5	1.8	2.0	Infinity	Item #1 completes service at Activity 2
1.8	2.4	2.0	Infinity	Item #4 is created
2.0	2.4	3.0	2.5	Item #2 completes service at Activity 1 Item #2 begins service at Activity 2 Item #3 begins service at Activity 1
2.4	3.0	3.0	2.5	Item #5 is created

Notice how the next event time is always the lowest of all of the event times for all the blocks; this is how a discrete event simulation works. Also, the table illustrates the concept of event

scheduling but does not show all of the detail of what is happening as the items move through the blocks. For example, the Queue schedules a zero time current event as it moves the item through, but this is not shown in the table.

Messaging in discrete event models

As discussed in "How ExtendSim passes messages in models" on page 101, the ExtendSim architecture allows application messages to be sent from ExtendSim to a model's blocks and block messages to be passed between blocks.

Discrete event models use the same application messages as do continuous and discrete rate models. The block messages sent between Item library blocks are discussed below.

Block messages

Discrete event blocks have a sophisticated messaging structure for communicating with each other and with blocks in the Value and Rate libraries. These messages can be categorized as:

- Event
- Value connector
- Item connector
- Block-to-block

Event messages

Event messages communicate between the Executive block and Item library blocks in a model. In a discrete event model the simulation clock advances from one event to another. Each time the clock advances, the Executive block sends event messages to the Item library blocks that have associated themselves with that event. There are two types of events: future and current.

- A *future* event message occurs when the simulation clock reaches a time posted by a block. For example, when an item enters an activity, the activity will post a future event to the Executive corresponding to the item's "finished time". Once the simulation clock has advanced to this future event, the Executive sends an event message to the activity, alerting it that the item has finished processing.
- A *current* event message occurs when a block wants to be activated before the simulation clock advances, but after it has completed its response to another message. For example, a queue will post a current event message to the Executive as it is pulling in items. After all the items have arrived to the queue, the Executive sends a current event message to the queue. This signals the queue to try and push all the items out of the block.

☞ The only blocks in the Value library that post future events are the ones that provide values or perform actions at specific times. Examples are the Clear Statistics block that resets the simulation statistics at a scheduled time and the Lookup Table block that provides values at scheduled times. Other Value library blocks lie dormant during a discrete event simulation unless they receive an activating message (either directly or indirectly through another block) from an Item library block.

Value input and output connector messages

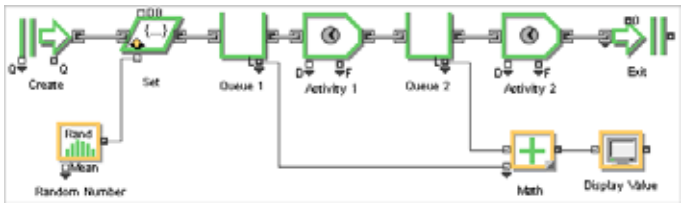
Blocks in a discrete event model send value connector messages either because a new number is needed by an input connector or because the value of an output connector has changed. These messages either request updated information for the input connectors or notify connected blocks that the output value has changed.

- When a message is sent from an input value connector, the sending block requests an updated connector value from the receiving blocks. Messages sent out the input connectors go only to the outputs of the directly connected blocks.
- Whenever an output connector changes, messages are sent to all of the inputs of the directly connected blocks. In this case, the sending block alerts the receiving blocks of a connector value change. Blocks that receive messages at their input connectors may, if appropriate, propagate messages:
  - Out other input connectors to make sure that all input values are current
  - To their output connectors to notify other blocks of the change in value

Through this mechanism, a single value change may cause any number of connected blocks to recalculate, ensuring that any system dependencies are automatically evaluated. For example, if the value of an input connector on an equation-type block changes, messages are first sent out the other input value connectors if they are connected (ensuring the equation will have up-to-date inputs prior to calculation.) Then, with updated inputs, the block recalculates its equation and posts the new results on its output value connectors. Once the new results have been posted, messages are sent out the output connectors, alerting any connected blocks that the results have changed.

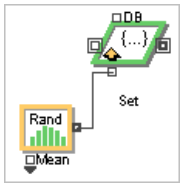
Example of value connector messaging

The sample model shown below illustrates how value connector messages work. In this example, items arrive, an attribute is set to a random number, and the items are then processed at two work areas (Queue and Activity blocks) in series. Three Value library blocks (Random Number, Math, and Display Value) provide a random number for the attribute value, add the two queue lengths, and display the sum of the lengths, respectively.



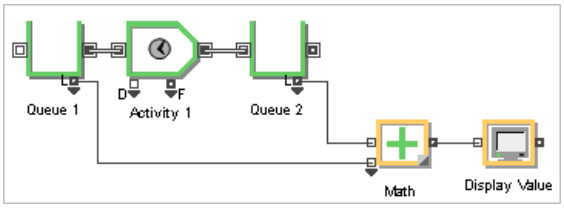
Example of value connector messages

When an item arrives to the Set block a message is sent out its value input connector. The Random Number block responds by providing a new random number each time it receives a message. This simple messaging example is shown at the right.



One message sent

In a discrete event or discrete rate model, the only time most Value library blocks (such as the Random Number block) are alerted to do something is when they receive a message on one of their value connectors. For instance, do not expect the Random Number block to continuously output a stream of random numbers in discrete event or discrete rate models.



Three messages sent

In the more complex messaging case that is shown to the right, when an item arrives at the first queue its length will increase by 1. Since this changes the block's L (length) output connector, Queue 1 sends a message to all inputs connected to L (in this case the Math block). When the Math block receives this message, it sends a message to the L connector at Queue 2, ensuring that both inputs are up-to-date before any calculation is made. The values of the two length connectors are then added together and a third message is sent to the Display Value block, which then updates its animation and dialog.

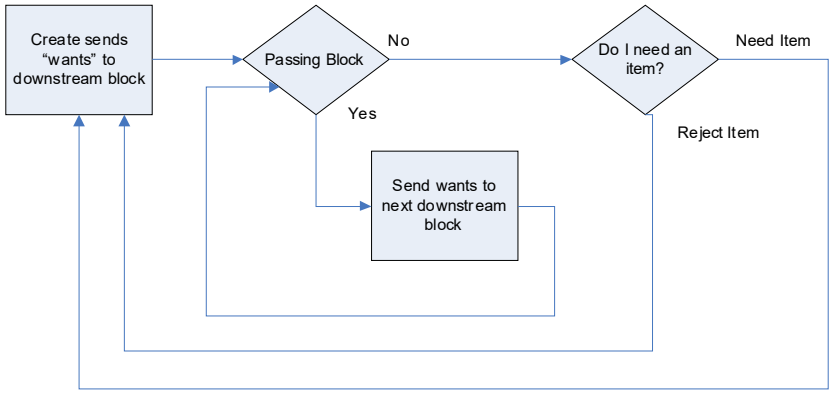
Item connector messages

Item connector messages (primarily *wants*, *needs*, and *rejects*) propel items through the model. These messages use a conversation of messages to move items from one block to another. This mechanism allows for items to be both pushed and pulled from one block to the next. How these messages are handled depends on whether the block is a passing, residence, or decision block (see "Cycle timing" on page 412.)

Pushing items

In the case of pushing, the upstream block first sends a *wants* message.

- If the downstream block is a passing block, it forwards the message to the next downstream block through its output connector.
- If the downstream block is a residence block, it responds with either a *needs* message (if it can accept an item) or a *rejects* message (if it is unable to accept an item based on its status).
- If the downstream block is a decision block, it determines the status of the decision and any downstream blocks. It often does this by sending additional item or value messages and then responding with a needs or rejects.

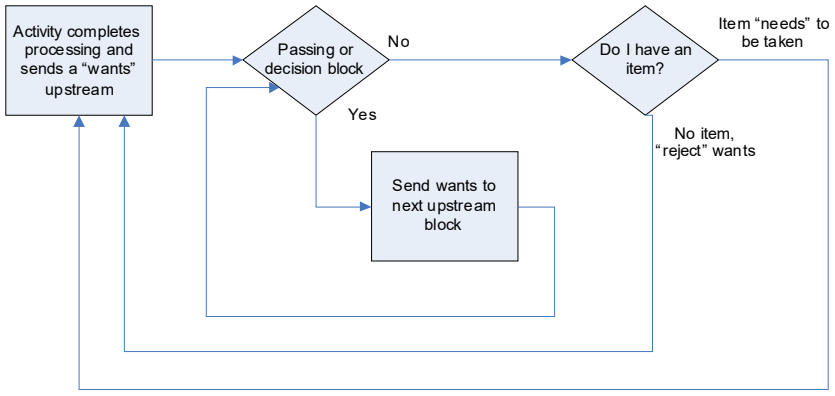


Flowchart for how items are pushed

Pulling items

To pull an item, a residence block sends the wants message upstream. This wants message is passed through the passing and decision blocks until it reaches a residence block. If the residence block has an item that is ready to leave, then a needs message is returned. If no item is available, then the residence block rejects the wants message.

Following is an example of pulling an item from an upstream block to a downstream block:



Flowchart for how items are pulled

This is only the first step in the process of moving an item. A number of messages follow that propel the item through the network of blocks. More details about those messages can be found in the Technical Reference.

Block-to-block messages

Block-to-block messages update the status of other blocks in the model. Sometimes a block needs to communicate with another block in the model, but there is no direct connection between them. For example, if a change in the shift status occurs, a Shift block needs to notify all of the blocks that reference that shift. These messages are sent 'through the air' to the blocks. In most cases, you will not even be aware that these messages are being passed back and forth. The actual operation and context of the message depends on the blocks involved in the conversation.

Discrete Rate Modeling

Introduction

Expanding on the Quick Start Guide

The following section of the User Reference is specific for those doing discrete rate simulations. It shows how to:

- Simulate rate-based flows of product
- Add properties (attributes) to the flow
- Work with flow rates and constraints
- Represent the storage of flow
- Delay and route the flow
- Integrate discrete rate models with discrete event models

Since the following chapters assume a certain level of knowledge, it is very important that you have read the Discrete Rate Simulation Quick Start Guide before these chapters.

Discrete Rate Modeling

Creating Flow

Creating flow and giving it properties

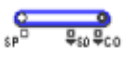
This chapter discusses creating and characterizing flow in a discrete rate model. It will cover:

- Creating flow in the model
- Using indicators to monitor the level of flow
- Working with flow attributes

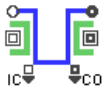
Blocks of interest

The following blocks from the Rate library will be the main focus of this chapter.

Residence blocks for creating flow



Convey Flow
Delays the movement of flow from one point to another. Can have multiple segments, each with an initial contents of flow.



Interchange
Represents a Tank that can interact with discrete event items. The block can have an initial contents of flow and can receive more flow while the simulation runs.

Tank
Acts as a source, intermediate storage, or final storage (sink). The block has a capacity and can provide an initial quantity of flow for the simulation.

Blocks for managing flow attributes

Get(R)
Displays the value of flow attributes. Allows you to select the type of information reported and which attribute you want displayed.

Set(R)
Allows you to assign attributes to the flow.

Diverge
When flow is diverged, this block provides options for how the flow attributes are handled.

Merge
When flow is merged, this block provides options for how the flow attributes are handled.

Creating flow

Flow is what is stored in and moves through a discrete rate system. It can be almost anything: oils and chemicals, manufactured product, transactions, electricity, and so forth.

The Convey Flow, Interchange, and Tank blocks are residence-type blocks; they hold flow and can serve as sources of flow for the model. There are two ways flow can be introduced into a discrete rate model:

1) By setting an initial contents for a Convey Flow, Interchange, or Tank block. These three residence blocks can be preloaded with initial amounts of flow, either a finite amount or, in the case of the Tank and Interchange blocks, an infinite amount. Depending on the block and options that are selected, the initial amount is:
   - Created by entering a constant number, or selecting "infinite", in the block's dialog
   - Entered in a table for each segment (Convey Flow block)
   - The attribute values for any number of initial layers (Tank block)
   - The value of an item attribute (Interchange block)
   - Getting the value from a ICO valueIn connector (Interchange block)
2) Once an initial contents has been set, under certain conditions the contents of the Tank or Interchange block can be changed during the simulation run. An example is the Interchange block when it is set to *Tank only exists while item is in it*. In this case, you can choose to have each arriving item reestablish the initial contents.

Empty and not-empty

If a Tank or Interchange block's initial contents is finite, its status can alternate between the empty and not-empty states. This change of state has an impact on the effective rate calculations:

- If the Tank or Interchange block is not empty, its flow level can fall. Consequently, the effective outflow rate could be higher than the effective inflow rate.
- If the block is empty, it cannot provide more flow than what it concurrently receives. In this case, the effective outflow rate has to be less than or equal to the effective inflow rate.

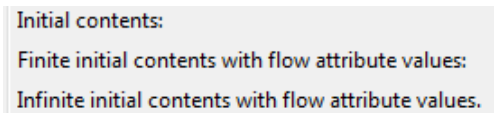
When a Tank or Interchange block changes state between empty and not empty, ExtendSim will calculate a new set of effective rates. For more information about rates, see "Rates, rate sections, and the LP area" on page 449.

☞ The Convey Flow block has a different mechanism for calculating a change of state between empty and not-empty. For more information, see the "Delaying Flow" chapter.

Tank initialization

A Tank's initial contents can be:

- Entered in the dialog as:
  - 0 (the default)
  - A number
  - Infinite
- Specified by flow attribute values and be either finite or infinite. (Flow attributes are discussed starting on page 431.)



Options for initial contents

To simply cause the Tank to have an infinite amount of initial contents, select the *Initial contents* option, then check the ∞ (infinite) check box. For example, in the Yogurt Production model from the Discrete Rate Tutorial, the Tank blocks that represented Liquid Supply and

Fruit Supply had infinite initial contents, while the Tank that stored the Yogurt product had no initial contents.

If the option for *Finite initial contents with flow attribute values* is selected, a check box causes the initialization to be repeated whenever the Tank becomes empty.

☞ If a Tank has no inflow connections, by definition it is being used as a source. If at some point the source reaches the empty state, the effective outflow rate will remain at zero for the remainder of the simulation run.

Interchange block modes

The Interchange block represents a tank, or holding area, where flow can interact with items generated by discrete event blocks. Flow can enter the Interchange block not only through its inflow connector but also through the arrival of an item. Conversely, flow can exit the block through its outflow connector or through the exiting of an item.

The Interchange block has two options that affect how the block's initial contents and maximum capacity are set:

- *Tank only exists while item is in it.* This behavior is analogous to a truck (an item) that arrives at a loading dock (a tank) where the loading or unloading of product can take place at a certain rate. The truck arrives with a capacity and perhaps some quantity of product already in it. As long as the truck is in the dock, loading or unloading is possible and occurs at the specified rate. When the truck leaves, the dock's ability to load and unload product disappears (the inflow and outflow effective rates are set to zero).
- *Tank is separate from item.* This behavior is similar to a truck (an item) that brings product to a holding area (a tank) that may nor may not contain product. The truck empties its load and perhaps takes some of the holding area's product with it when it leaves. This process occurs instantaneously. Whether the truck is at the holding area or not, both the truck and the holding area can have product. The holding area can receive or deliver flow to the system even if there is no truck (the inflow and outflow effective rates can be greater than zero).

☞ For more details about these two modes, see "Interchange modes" on page 504.

Depending on the option selected, the Interchange has different choices for setting an initial contents.

Initialization of contents when "Tank only exists while item is in it"

This choice allows the block to have an initial contents only when an item arrives. To set an initial contents for this block, choose one of the options from the dialog's popup menu, shown here:

Interchange initial contents, default behavior

- *A constant.* Enter a number in the field; the default is 0. To cause the initial contents to be infinite, check the field's ∞ (infinite) check box or set the field to blank.
- *Value at ICO*. The value at this connector will control the initial contents.
- *Value of item attribute*. Select an item's attribute in the dialog. When an item arrives, the initial contents will equal that attribute's value.

In the Yogurt Production model from the Discrete Rate tutorial, the Interchange blocks were set to *Tank only exists when item is in it* and *Initial contents (on item arrival): 0.*

Initialization of contents when "Tank is separate from item"

To set the initial contents for the Interchange block when this behavior has been selected, leave the *Initial contents* field set to the default infinite amount or enter a number.

Interchange initial contents, alternate behavior

With this behavior, arriving and departing items can cause contents to be contributed to or removed from the tank, depending on choices set in the block's dialog.

Convey Flow initialization

The initial contents for the Convey Flow block are set in its Initialize tab. The table allows you to customize a number of segments for the conveyor, each with its own initial contents. Each row in the table represents an individual segment of the conveyor possessing a uniform density that differs from the adjacent segments.

	High Limit	Low Limit	Density	Initial Quantity
0	100 length unit	75 length unit	10 units/length un	250 units
1	50 length unit	25 length unit	5 units/length unit	125 units

Example initial contents for Convey Flow block

The Initialize tab has a *Show Example* button that places example settings in the table. These are helpful for understanding how to make the entries you want; they can also be used as a starting point for entries. Shown above is the example setting for a 100 foot long accumulating-density conveyor that transports containers. The table indicates that the block would have an initial density of 10 containers per foot for the segment from 75 to 100 feet (a total of 250 containers) and 5 containers per foot for the segment from 25 to 50 feet (a total of 125 containers).

☞ In this example, the sections between 0 and 25 feet and between 50 and 75 feet do not hold any product.

Indicators

As the simulation runs, the level of flow in the residence blocks (Convey Flow, Interchange, and Tank) will vary over time.

You might want to establish a set point or indication to report when a block's flow level is within a certain range of values. This is common when monitoring a block to determine if its contents are approaching or have reached one or more important benchmarks. For instance, some emergency procedures might need to take place if a Tank's level reaches the "high" range; they can be discontinued when the contents return to a "normal" range.

For residence blocks, *indicators* are a method of reporting what category or range the current level of flow falls into. With this feature, each range is assigned a name, a lower limit, and an upper limit. When the level of flow reaches a value that falls within a different range, the block reports the change on its *I* (indicator) value output connector and alerts any connected blocks to the change in status.

☞ While the Tank and Interchange blocks report information about the current level of flow from their I (indicator) connectors, the Convey Flow block reports how far (the *accumulation length*) the accumulation point is from the end of the conveyor. (When the amount of product ready to leave exceeds the amount that can be received downstream, flow begins to accumulate

from the end of the conveyor. For more information, see “Distribution of flow” on page 494.

Setting indicators

The Indicators tabs on all three residence blocks have similar interfaces. Each Indicators tab has a table (shown on the right with example settings) for specifying an indicator name for each range of values, entering the low limits, and defining values (an ID number for each indicator) to output when the block's flow level falls within a particular range.

	Indicator Name	Low Limit	High Limit	Value to Output
0	Full	100%	100%	4
1	High	70%	100%	3
2	Medium	40%	70%	2
3	Low	10%	40%	1
4	Empty	0%	10%	0

Link

Example indicators in Tank block

To create indicators, enter your own information or click the Show Example button to populate the table with some example indicator names and settings. In either case, ExtendSim will calculate the High Limit values based on the Low Limit entries.

☞ The top row has to have the highest range; the bottom row must have the lowest range.

☞ To add or delete table rows, use the +/- button in the table's lower right corner. For instance, to delete the example settings, change the number of rows to 0.

The screen shot above shows a Tank’s names, limits, and ID values to output after the Show Example button has been clicked. Each indicator name corresponds to a range of flow contents defined by the Low and High Limits for that row. (The High Limit column is presented for clarity only, since those numbers are calculated using the values entered for the Low Limits.)

Unless the block has infinite capacity, the indicator limits can be expressed in absolute numbers (shown above) or as percentages.

See “Bucket Elevator 2 model” on page 507 for an example of how indicators are used in an Interchange block to control a Valve’s effective rate.

☞ If the block has infinite capacity, the limits must be expressed as absolute numbers. If the block's initial contents are set to infinity, the indicators are disabled.

Getting information about levels

There are two types of events that will cause a new indicator to be reported:

	Indicator Name	Low Limit	High Limit	Value to Output
0	Full	250000 pints		4
1	High	175000 pints	250000 pints	3
2	Medium	100000 pints	175000 pints	2
3	Low	25000 pints	100000 pints	1
4	Empty	0 pints	25000 pints	0

Example indicators in a Tank

- When the level is increasing and the block’s contents reach the next indicator’s Low Limit.
- When the level is decreasing and the block’s contents reach the next indicator’s High Limit.

In each case, the new output ID is used to update the I value output connector, and any connected blocks are alerted to the change.

Using the above table as an example, if the level in the Tank increases from 120,000 to 175,000 containers, the block will compare that level to the Low Limit and send the value 3 to

its I (indicator) output connector. However, if the level of the Tank instead decreases from 230,000 to 175,000 containers, the block will compare that value to its High Limit and output the value 2.

☞ The value that is output at the I (indicator) connector depends on whether the level of flow is increasing or decreasing, and where in the range the new indicator level falls. The Tank block has a S (status direction) output connector that reports if the level is going up or down when the event occurs.

Tank Flow Units model

The Yogurt tank in the Tank Flow Units model (Examples\Discrete Rate\Sources and Storage) outputs values that indicate the level of flow in the Tank; those values are displayed on the third line on the plotter.

☞ For more information about the Convey Flow block, including the use of sensors, see the discrete rate chapter on “Delaying Flow”.

Flow attributes

A flow attribute is a characteristic or quality applied to a specific volume or quantity of flow. Flow attributes stay with the flow as it moves through the model, provide information about the flow, and allow you to organize quantities or volumes of flow into layers. For example, a flow attribute could be Flavor, Thick, Product, or any other quality or characteristic of some portion of the flow.

The entire set of attribute values assigned to a particular layer is what distinguishes it from other layers. In other words, all the units of flow in a layer possess the same set of flow attribute values.

While not all Rate models require the use of flow attributes, some uses for flow attributes include:

1) The flow held in a Tank can be organized into individual, distinct layers. This allows you to control where incoming flow is deposited and which type of outgoing flow next exits.
2) Flow rates through the Valve block can be calculated based on flow attribute values.
3) Flow can be routed through the Merge and Diverge blocks based on flow attribute values.

Attributes are assigned to the flow by the model builder. They become linked to sections of flow that pass through the blocks, according to settings in the blocks’ dialogs.

Names and values of flow attributes

As with item attributes (discussed on page 263), each flow attribute is composed of a name and a value.

- A flow attribute's *name* identifies some general characteristic of the flow such as “Product”, “Flavor”, or “Color”. Attribute names are limited to 15 characters.
- A flow attribute's *value* indicates one dimension of the named characteristic. For instance, a flow attribute named Product could have a value of 1 (for Television) or a value of 5 (for Stereo), while a flow attribute named Flavor could have a value of 1 (for Vanilla), 2 (for Berries), or 3 (for Caramel).

Attributes are meant to be unique;. If you attempt to add a new flow attribute with exactly the same name as an existing one, ExtendSim warns you that the name already exists. While attri-

bute names are not case sensitive (*Type* is equal to *type*), spaces are significant and should be avoided.

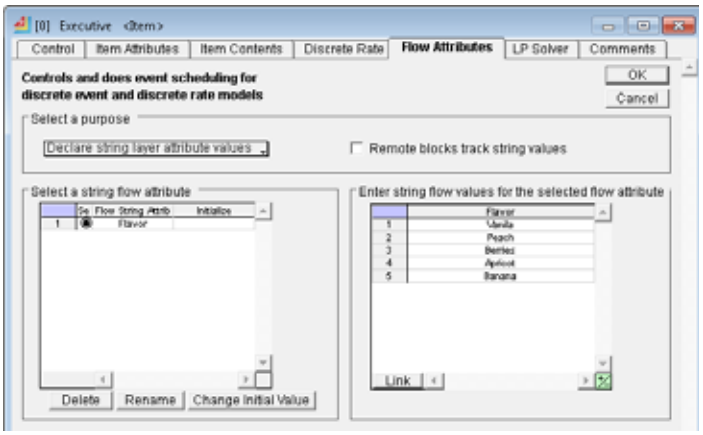
Discrete rate models can have many flow attributes and each attribute can have multiple values.

Types of flow attributes

ExtendSim supports two types of flow attributes for Rate library blocks:

- A *layer attribute* holds a real number as its attribute value.
- The value of a *string layer attribute* is still a number, but it is represented in the model by a string.

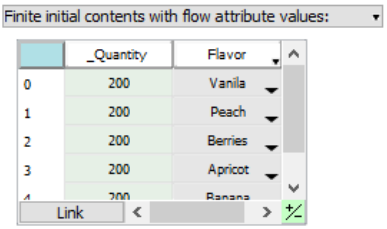
With string attributes you choose a descriptive text label (string) for each potential value of the flow attribute; the information is entered in a lookup table in the Executive block's Flow Attributes tab. The string can then be used in the model in place of the corresponding number. For example, a string layer attribute named Flavor might have three possible values: 1, 2, and 3. Once the lookup table for this attribute has been properly configured, the blocks referencing the Flavor attribute will display the strings Vanilla, Berries, or Caramel instead of the numbers 1, 2, and 3. For instance, see the string layer attribute Flavor in the Executive, below.



Creating and assigning a flow attribute

There are several methods for creating a new flow attribute:

- Use the Tank and set the initial contents (either finite or infinite) to use flow attribute values, as shown to the right
- Use the Set(R) block, as discussed below
- Use the Merge and Diverge and customize how attributes behave at their inflows or outflows
- The Interchange block has a mapping capability such that item attributes can be mapped to flow attributes, and vice versa.



To create a new flow attribute in the Set(R) block, use the green +/- resize button to add any needed rows to the table in its dialog. Then use the pop-ups at the top of the columns to add a new layer or new string layer attribute, give it a name, and (in the case of a layer attribute) assign an initial attribute value. Selecting a string layer attribute type causes the Executive to open to the Flow Attributes tab shown on page 434; this is where strings that correspond to the attribute's values are entered. Values for flow attributes can also be set dynamically through value input connectors on the Set(R) block.

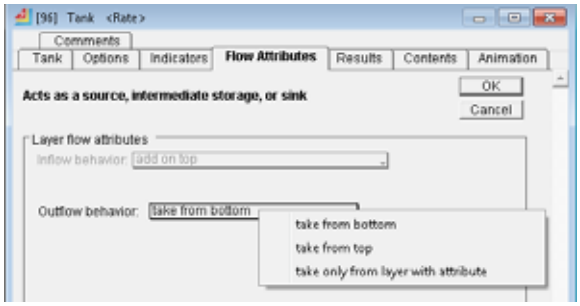
How flow attributes are propagated in the model

Flow attributes attach to sections of flow and are present throughout the model. Furthermore, a discrete rate model can have many flow attributes where each flow attribute can have multiple values. However, the values of flow attributes are location specific. This means that the value of a flow attribute is unique at any time in each section of the model. (For information on sections. see "Rates, rate sections, and the LP area" on page 449.)

Flow layers in residence blocks

As mentioned earlier, flow attributes are used to organize quantities or volumes of flow into layers, where each layer represents an amount of flow that has specific and unique characteristics. By definition, the attribute values for every unit of flow in a particular layer must be the same.

Flow layers are stored, tracked, and distributed by the Tank, Interchange, and Convey Flow blocks in the following ways:



- Tank. The Tank has the ability to store incoming layers of flow by placing them on top or on bottom, or by adding the incoming flow to a pre-existing matching layer. Outgoing flow can be taken from the top, the bottom or from anywhere in between, as shown on the right. This flexibility allows the Tank to mimic LIFO, FIFO, and priority behavior.
- Interchange. The Interchange block can use layers to manage the filling and releasing of items. For example, there's an option to release an item as soon as an attribute on incoming flow changes to a different value. This allows you to fill items with flow that contains a uniform attribute value.
- Convey Flow. The Convey flow will always receive and distribute flow layers in a FIFO fashion.

For example, a layer in a Tank could be a quantity of yogurt with the Flavor attribute value "Vanilla" and the Nuts attribute value "No Nuts". Vanilla yogurt with nuts would be in a different layer, as would Berry yogurt without nuts.

ExtendSim handles the organization of layers within the residence blocks according to options set in the blocks' dialogs.

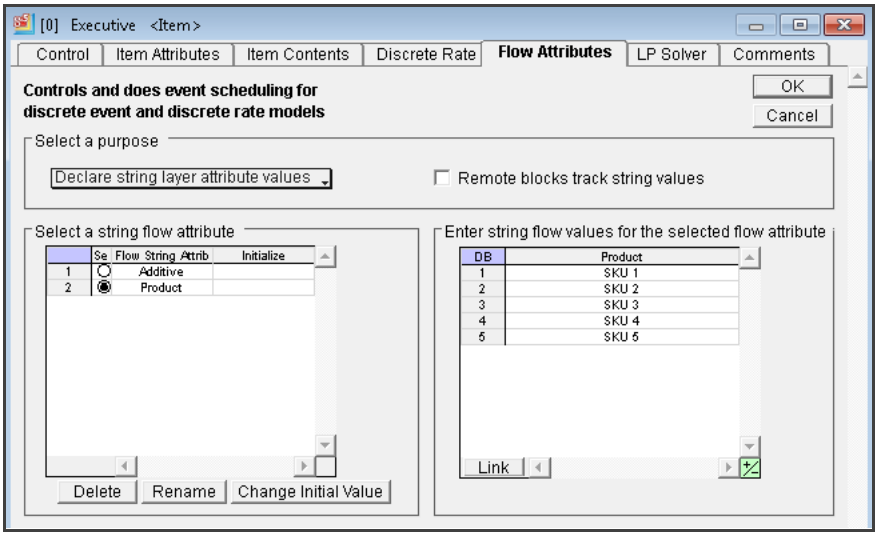
Propagation of attribute values outside of residence blocks

All Rate blocks get attribute values from inflow connectors and set attribute values on outflow connectors. For instance, a Merge block will receive attribute values from each of its upstream branches and, depending on these values and the rules set in the block, it will set the attribute values for the downstream flow.

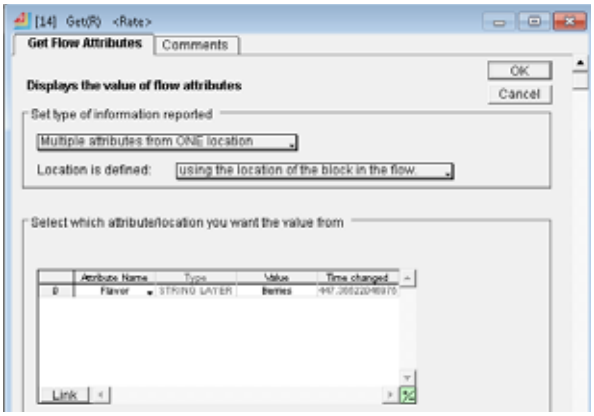
If the upstream flow has the potential to satisfy downstream demand, the connectors in between will reflect the attribute values of the upstream flow even if the effective rate is zero. On the other hand, if there is no upstream flow available to potentially satisfy downstream demand, the attribute values on the connectors in between will be a Blank.For example, if the demand is for Vanilla yogurt and the Tank has Vanilla yogurt but the effective rate is currently zero, the attribute value will be “Vanilla”. However, if the effective rate is zero and the demand is for Berry yogurt, which isn’t currently present in the Tank, the flow attribute value will be a Blank.

Managing and displaying flow attributes

The Flow Attributes tab of the Executive block (shown below) is for managing flow attributes – deleting, renaming, and changing their initial values. It is also where the strings for string layer attributes are defined and where you can search for blocks that use a specific flow attribute.



Unlike an item attribute, which is assigned to individual items, flow attributes are assigned to the flow in a section of a model. Flow attribute values can thus be read at any time and from any model location. Each Rate block that holds or provides flow has a Contents tab showing the attribute values of its flow by volume. The Get(R) block can be used to retrieve the values of flow attributes elsewhere in the model.



In order to manipulate flow based on the attribute, usually to route it or process it, you need to get the flow's attribute value. The most common method for getting attributes is to select the attribute by name from the list in the attribute popup menu in the Get(R) block, as shown on right. When flow passes through the Get(R) block, the block accesses information about the attributes that have been specified in the table in its dialog. It then reports the information in the table and on its value output connectors. What the Get(R) block reports and where, depends on the type of attribute:

- *Layer attributes*. The value for the attribute is posted in the Value column of the attribute table and on the value output connector that corresponds to the attribute.
- *String layer attributes*. The string text is displayed in the Value column of the attribute table and the number that corresponds to the string is posted on the appropriate value output connector.

In addition to the Time Changed column in its dialog, the Get(R) block has a Δ (delta) connector for reporting when a flow attribute's value changes. The Δ connector outputs a 1 when the value of the first flow attribute specified in the dialog changes. Otherwise it outputs 0.

Flow attributes and item attributes

It is common for systems to exhibit mixed-mode behaviors, where items from the discrete event arena intermingle with discrete rate processes. The Interchange block provides a matrix for converting item attributes into flow attributes and vice versa. See its Flow Attributes tab.

Example models

Two models that use flow attributes are located in the ExtendSim\Examples\Tutorial\Discrete Rate folder:

- Yogurt Production with Flavors
- Yogurt Production with Flavors Plus

Both models are a continuation of the Yogurt Production model discussed in the tutorial in the Rate Module Quick Start guide. They use attributes to simulate the production of five different flavors of yogurt. The model named Yogurt Production with Flavors Plus uses the item/flow attribute mapping capability of the Interchange block to preempt items, avoiding mixed pallets of flavored yogurt.

Discrete Rate Modeling

Storage and Units

Storing flow and the use of flow units

As discussed in the Rate Module Quick Start guide, quantities of flow are located in one or more parts of a discrete rate model. During the simulation run, the flow moves from one location to another at the effective rate. In order for the flow to move, one or more of the model's blocks need to have the capacity to hold flow as time advances.

The Convey Flow, Interchange, and Tank blocks are residence type blocks – they have capacity and can hold defined amounts of flow.

Flow units describe what is flowing from one Rate library block to another. Blocks that are connected together through flow connections and share the same flow unit are part of the same unit group. The Change Units block is used to create a new unit group. This causes the blocks downstream of the Change Units block to be in a unit group different from its upstream blocks.

This chapter discusses providing and storing flow and the use of flow units in a discrete rate model. It will cover:

- Defining a block's flow capacity
- Indicators that provide information about a block's level of flow
- Defining and selecting time, flow, and length units
- Using the Change Units blocks to create a different flow unit group

This chapter focuses on setting capacity for the Convey Flow, Interchange, and Tank blocks. Other aspects of those blocks are covered in different chapters:

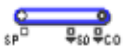
- The Convey Flow block is most often used for delaying flow and will be discussed more fully on page page 491.
- The Interchange block is mainly used for interacting with items from discrete event portions of the model and will be discussed more completely starting on page 502.
- Setting maximum inflow and outflow rates for the Tank and Interchange blocks is described in page 456.

☞ The Tank Flow Unit model is located in the folder \Examples\Discrete Rate\Sources and Storage. The Yogurt Production model is located at \Examples\Tutorials\Discrete Rate.

Blocks of interest

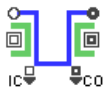
The following blocks from the Rate library will be the main focus of this chapter.

Residence blocks for holding flow



Convey Flow

Delays the movement of flow from one point to another. Can accumulate flow to a maximum density, accumulate flow to fill empty sections, or act as a non-accumulating conveyor.



Interchange

Represents a Tank that can interact with discrete event items. The block has two behaviors: the Tank only exists while an item is in it; the Tank is separate from the item.

Tank
Acts as a source, intermediate storage, or final storage (sink). The block has a capacity and can have an initial quantity of flow for the simulation.

Changing the flow unit group

Change Units
Changes the flow unit from one unit to another, resulting in a new flow unit group. The dialog has a field for entering the conversion factor and a popup menu for indicating the direction of the change.

Capacity

The Tank, Interchange, and Convey Flow blocks are considered *residence* blocks. This means that they have capacity and can hold defined amounts of flow as time advances.

A residence block's maximum capacity can be a specific number or, in the case of the Tank or Interchange blocks, it can be set to infinite.

Full and not-full

When a residence block's capacity is finite, its status can alternate between the full and not-full states. This change of state has an impact on the model's set of effective rates:

- If a residence block with finite capacity is not full, there is room for the flow level to rise. Consequently, the effective inflow rate can be greater than the effective outflow rate.
- If a residence block with finite capacity is full, the flow level is not permitted to rise; the effective inflow rate will be less than or equal to the effective outflow rate.

Any time a residence block with a finite capacity changes state between full and not-full, ExtendSim will calculate a new set of effective rates.

☞ A residence block with infinite capacity can never be full during the simulation run. In this situation, it is similar to a residence block with finite capacity that is not full; its effective inflow rate can be greater than its effective outflow rate.

Tank block's capacity

A Tank can be a source of flow, an intermediate storage for flow, or a final storage for flow (sink). A Tank's capacity can be infinite, a finite but non-zero number, or zero.

Default setting for Tank's capacity

- By default, the Tank has an infinite capacity to hold flow, as indicated by the *Maximum capacity: infinite* ∞ setting in its dialog. In this state it will never be full.

☞ A blank is the same as checking the infinite setting.

- A Tank's maximum capacity can be changed in the block's dialog by entering an amount in the *Maximum capacity* field (which unselects the ∞ checkbox). It can also be changed dynamically through the block's *C* (capacity) value input connector. If the C connector is used, it overrides any entries made in the dialog. With a non-zero finite capacity, the Tank can be in either the full or not-full state at any point in time.

- If a Tank's capacity is set to zero, flow can still move through the block but the flow will not stay in the block for any length of time. In this case, the Tank is neither full nor not-full, and the effective inflow rate will equal the effective outflow rate.

☞ If a Tank has no outflow connection, by definition it is being used as a sink. If at some point the sink reaches the full state, its effective inflow rate will be set to zero for the remainder of the simulation run.

Interchange block's capacity

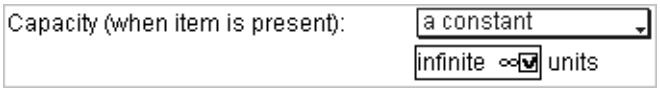
The Interchange block has two options that affect how the block's initial contents and maximum capacity are set. These options are discussed in "Interchange block modes" on page 428.

Capacity when "Tank only exists while item is in it"

This is the default behavior. With this setting the Interchange block has a capacity to hold flow only while an item resides in the block. With an item present, the block acts like a tank and therefore has a definable capacity, either finite or infinite. If the capacity is finite, as along as an item is present in the block, the block can alternate between the full and not-full states.

The block's capacity is fixed at the moment the item enters the block; it remains fixed until the item leaves. When the item leaves, the Interchange's capacity automatically goes back to zero. Therefore, at the time of item departure any flow currently in the block is loaded onto the item. The timing of when the item leaves the Interchange depends on logic set in the Interchange block. For more information, see "Item release conditions" on page 503.

To define the capacity for an Interchange block when it is set to this behavior, choose one of the options from the dialog's popup menu, shown at right.



Maximum capacity; default behavior

- *A constant*. Enter a number in the field; the default is infinite.
- *Value at IC*. The value at this input connector will control the block's capacity.
- *Value of attribute*. Select an attribute in the dialog. When an item arrives, the tank's capacity will equal the item's attribute value.

In the Yogurt Production model of the Discrete Rate Tutorial, the Interchange blocks are set to *Tank only exists when item is in it* and their capacity is set to the constant value *24*.

Capacity when "Tank is separate from item"

With this option, the Interchange block's behavior is similar to a Tank block – it receives flow from its inflow connector, it holds flow, and it releases flow from its outflow connector. The difference is that an item's arrival can contribute flow to the existing contents and an item's departure can remove flow from the existing contents. The item's impact on the block's contents is entered in the *Define Item behavior* section of the block's Item/Flow tab.

To set the capacity for the block when this behavior has been selected, enter a number in the *Maximum capacity* field or leave it set to the default value of infinite.

Maximum capacity; alternate behavior

Convey Flow block's capacity

The maximum capacity for a Convey Flow block is a combination of two factors:

Maximized capacity

1) The block's length and maximum density determine the *maximized* capacity. By default, ExtendSim calculates a maximized capacity for the Convey Flow block by multiplying its length by its maximum density. This is indicated in the Capacity field by the "maximized" capacity checkbox shown above. In this case, the block's maximum capacity will equal its maximized capacity.
2) A number in the Capacity field in the Options tab can reduce the capacity below the maximized amount. In some cases, it may be necessary to define a capacity smaller than the maximized capacity determined by the length*density calculation. For instance, the Convey Flow block could have structural properties limiting how much weight it can safely support. To do this, uncheck the checkbox in the Capacity field of the Options tab and enter the desired number. In this case, the block's maximum capacity will be less than or equal to its maximized capacity.

☞ A Convey Flow block's *maximum* capacity can never exceed its *maximized* capacity, no matter what number is entered in the Capacity field.

For example, if the Convey Flow block's length is 100 and maximum density is 10, the block's maximized capacity will be 1,000. To reduce the capacity to something less than 1,000, enter a number (for instance 300), in the Capacity field. The block will then only be able to contain 300 units of flow, even though its calculated maximized capacity was 1,000.

Units and unit groups

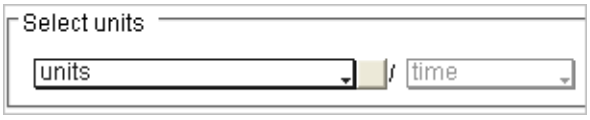
This section focuses on defining, selecting, and changing flow units, discusses the effect of changing time units, and shows how to use the Change Units block to change the unit group.

Definitions

The following sections discuss flow, block, time, and length units. A unit group is two or more blocks connected together through flow connections and sharing one flow unit.

Units and unit groups were introduced in the Rate Module Quick Start guide; they are described fully below.

Flow units



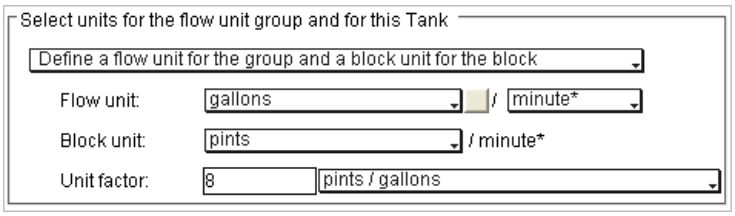
Generic flow and time units in a Valve

The flow unit indicates what is flowing from one Rate library block to another. As is true for ExtendSim time units, flow units can be unspecified generic units (in which case the block dialog will just display the word "units") or they can be specifically defined in the model. For instance, a defined flow unit could be a packet, gallon, transaction, box, liter, and so forth. Existing flow units can be selected, and new units can be defined, in the Options tabs of Rate library blocks. In addition, the Discrete Rate tab of an Executive block (Item library) has a section for managing flow units. This provides a central location where units can be added, deleted, or renamed.

☞ This chapter will be mostly concerned with flow units.

Block units

The Tank and Interchange blocks can have an internal *block unit* that is different than the flow unit. This is an internal representation of volume that is specific to the Tank or Interchange block, and does not affect the flow unit for the unit group. If you select a block unit that differs from the flow units that come into and out of a block, you must enter a conversion factor. The conversion factor represents the ratio of the block unit to the flow unit. Block units are discussed fully in "Defining block units" on page 443.

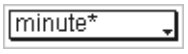


Flow and block units in the Tank

☞ Using block units is optional; the default block unit is the flow unit.

Time units

Time units can be generic, in which case the block will just say "time" or can be specific. Each model can define specific time units, which become the default for the model. You can change a discrete rate block's time unit from the model default time unit to any local time unit using a popup menu in the block's Options tab.



Default time unit

If a local time unit is selected in a discrete rate block, that local time applies to the entire block but only to that block. Changing to a local time unit does not change the global time unit for the flow group or for any blocks in the rest of the model. For complete information, see "Time units" on page 93.

⚠ Calendar dates are not available if months or years have been selected as the specific global time unit for a discrete rate model. Furthermore, if Calendar dates has been selected, Rate library blocks will not be able to select Months or Years as their local time unit.

Length units

For convenience, the Convey Flow block allows you to name a length unit. This is used internally by the block with other settings to determine the block's speed. You can use the default generic unit "length unit" or declare a specific unit of length such as feet or meters.

Unit groups

A *unit group* is two or more blocks connected together through flow connections and sharing one flow unit. Connecting the first block's outflow connector to the second block's inflow connector creates a unit group. Unless the unit group is explicitly changed, all the blocks that are connected through flow connectors use the same flow unit and are in the same unit group. If a flow unit is changed in one of the blocks in a unit group, the unit group does not change but all the other blocks in that group are updated automatically to the new flow unit.

To see the unit group, click the grey square to the right of a block's flow units popup menu, shown above. All the blocks in that block's unit group will be highlighted on the model worksheet.

Unit group selector to right of popup

You can define multiple unit groups, which use different flow units, in portions of a discrete rate model. For instance, one part of the model could be expressed in bottles and another could represent boxes of bottles. The Change Units block can create a different unit group.

Declaring and selecting flow units

As mentioned above, flow units indicate what is moving from one flow connector to another, and a unit group is a collection of connected blocks that share the same flow unit. This section focuses on defining, selecting, and changing flow units.

To convert from one unit group to another in a model, use the Change Units block discussed on page 444.

Where to declare a flow unit

Each block in the Rate library has the ability to select an existing flow unit or create a new one. This is done in the block's Options tab, which has a popup menu for either selecting an existing flow unit or creating a new one. Specifying a flow unit in one block causes that unit to be used by every block within the same unit group, and automatically sets that flow unit for any blocks that are subsequently added to that unit group.

To see the unit group, click the unit group selector button to the right of a block's flow units popup menu. All the blocks in that block's unit group will be highlighted on the model worksheet.

Unit group selector to the right of the popup menu

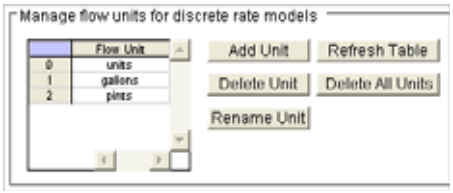
Declaring a flow unit

To declare a flow unit, select an existing unit or create a new unit from the popup menu in the Options tab. The selected flow unit applies not only to the block but also to the entire flow unit group and it will automatically be set in new blocks that are connected within the unit group. For instance, the tutorial in the Rate Module Quick Start guide showed how to create a new flow unit named "gallons". When the fruit processing section was added, the popup menu for the Fruit Supply Tank already included gallons selected as its flow unit.

Managing flow units in the Executive block

The Executive block's Discrete Rate tab has a section for managing flow units in a model. The table displays all the units for a given model and provides buttons for adding, deleting, and renaming the units.

To use this feature, select the flow unit you want to change in the table, then click the appropriate button. ExtendSim will warn you and give options if the unit is being used in the model.



Flow units for Yogurt Production model

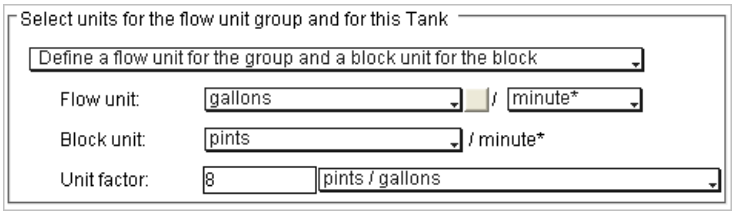
Defining block units

As mentioned earlier, the Tank and Interchange blocks allow you to define a block unit that is different from flow units. This is an internal representation of volume only and is specific to the Tank or Interchange block. It does not change the flow units or the unit group for the flow that has entered or exited the block. If a block unit has been specified, a factor to convert the block unit into flow units must also be entered.

To define a block unit, in the Options tab of the Tank or Interchange block, select *Define a flow unit for the group and a block unit for the block*. In addition to providing fields for declaring a flow unit, this option displays a field for entering an internal block unit. It also has a field for entering the factor to convert between the flow unit and the block unit.

Tank Flow Units model

The Tank Flow Units model is the same as the model described in the Rate Module Quick Start guide. However, the newer model has pints, rather than gallons, as the block units in the Yogurt Tank, causing the process's output to be displayed in the plotter as pints. Clicking the unit group selector button in its Options tab shows that this Tank is still part of the unit group that uses the flow unit "gallons".



Defining a block unit in the Tank

Time units

A popup menu in the Options tab allows a block's time unit to be changed from the model default time unit to any local time unit.

If a local time unit is selected in a discrete rate block, that local time applies to the entire block but only to that block. Changing to a local time unit affects every parameter in the block, but it does not change the model's global time unit or the time units used in any other block in the model. For more information, see "Time units" on page 93.

Changing the unit group

By default all the blocks connected to the same flow stream belong in the same unit group. However, the Change Units block has the ability to create a new unit group, causing connected blocks in a portion of the model to have a different flow unit.

Change Units block

To change the flow units from one part of a model to another, use the Change Units block. While changing a flow unit in most blocks will change the flow unit for the entire group, adding a Change Units block to the model causes a new unit group to be created – the Change Units block's inflow connector will be part of one unit group comprised of those blocks upstream of the Change Units block; its outflow connector will be part of another unit group comprised of those blocks downstream of the Change Units block.

Dialog of Change Units block

The *Change units from:* popup menu defines the flow unit that is entering the block; the *to:* popup menu is for selecting or creating the new flow unit. The block has a field for entering the factor that converts the incoming flow unit into the outgoing unit, and a popup menu to select the direction the conversion should take. You can also change the time units for this block in its dialog.

Yogurt Production model

An example of changing flow units is shown in the tutorial in the Rate Module Quick Start guide, where gallons of liquids were converted into containers of yogurt.

Discrete Rate Modeling

Rates, Constraints, and Movement

Limiting the movement of flow through rates and constraints

As discussed in the Introduction to this module, the movement of flow in a discrete rate model must take some time. It is a modeling error if the flow moves instantaneously throughout a model.

ExtendSim's discrete rate system attempts to move flow through the model as fast as possible. In the absence of any constraints, the effective rate of flow would approach infinity and the flow would move instantaneously throughout the model; this is never correct. For this reason, flow movement must be constrained by rates and conditions that are built into the model, and a lack of appropriate constraints is a modeling error that will stop the simulation run

In order to restrict flow rates:

- Discrete rate blocks are required to define their own sets of constraining flow rules
- Each area of a model must have one or more critical constraint mechanisms

Critical constraint flow rules, such as a block's maximum rate, place an upper bound on the rate of flow, limiting it to a number less than infinite. The blocks' aggregated set of flow rules ultimately defines how fast flow is permitted to move over time throughout the model.

This chapter discusses rates, the blocks that constrain flow, and how model conditions impact the rate of flow. It will cover:

- Rates, rate sections, and the LP area
- Flow rules for defining how a block permits flow to move through it
- The blocks that specify critical constraints
- How to meet the constraint requirement
- A comprehensive example of constraints and rate sections

☞ Most of the models illustrated in this chapter are located in the folder \Examples\Discrete Rate\Rates and Constraints. The tutorial models mentioned are located at \Examples\Tutorials\Discrete Rate.

Blocks of interest

The following blocks from the Rate library will be the main focus of this chapter.

Convey Flow

Delays the movement of flow from one point to another. Can accumulate flow to a maximum density, accumulate flow to fill empty sections, or act as a non-accumulating conveyor.

Interchange

Used to mix flow with items, this block can also limit its maximum rate of inflow and outflow.

Tank

The block most frequently used to store flow can also limit its maximum rate of inflow and outflow.

Valve

Controls and monitors the flow, limiting the rate of flow passing through. This block can also be used to set a goal for the duration or quantity of flow.

Rates, rate sections, and the LP area

One of the most important aspects of a discrete rate model is the rate of flow – the speed of flow movement. The flow rate is represented as the ratio of flow units to the model's time units. This is displayed in block dialogs as units/time, gallons/minute, transactions/second, boxes/hour, and so forth.

Discrete rate models can be thought of as being divided into rate sections and LP areas. The flow connectors within each rate section have the same effective rate, which is the speed at which flow moves through those blocks. The LP area is composed of one or more rate sections whose effective rates might change during the simulation.

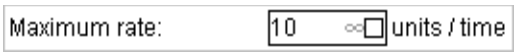
The types of rates considered during the model building process, rate sections, and the LP area, are discussed below.

Types of rates

The following rates are taken into consideration by ExtendSim and by a block's flow rules. (Flow rules will be discussed on page 452.)

Maximum rate

ExtendSim's discrete rate architecture attempts to move flow through the model as fast as possible. In the absence of any constraints, the flow rate would theoretically approach infinity and flow would move from one part of a model to another instantaneously; this would be a modeling error.



Default maximum rate for Valve

The *maximum rate* puts an upper limit on the movement of flow through a block. Six Rate library blocks have the ability to set a maximum rate. You can set an explicit maximum rate in the Interchange, Tank, and Valve blocks. The maximum rate for the Convey Flow block is mathematically derived. Maximum rates may also be implicitly specified under certain conditions in the Merge and Diverge blocks. In each case, the maximum rate is the highest rate of flow those blocks will allow, and hence the highest potential rate of flow for that part of the model.

☞ An inflow connector for a Convey Flow, Interchange, or Tank block can have one maximum rate while the block's outflow connector can have a different maximum rate. The maximum rate for the Convey Flow block's inflow is derived from settings in its dialog; the maximum rate for its outflow is derived from dialog settings and model conditions. The maximum inflow and maximum outflow rates for the Interchange and Tank blocks can be entered directly in their dialogs.

⚠ In order to avoid an error condition, each area of a model must have some mechanism in place to restrict the rate of flow to a number that is less than infinity. If the required minimum set of constraints is not present, ExtendSim stops the simulation and displays an error message.

Effective rate

Effective rate (Valve's Results tab)

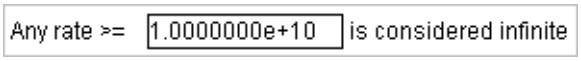
One of the most important reasons for creating a discrete rate model is to determine the actual rate of flow movement. The *effective rate* is the calculated actual rate of flow between Rate library blocks during the simulation run. It is the result of an internal calculation taking into account the maximum rates and all the constraints of the process. In some situations the effective rate is the same as the maximum rate; in others it is lower. One effective rate is associated with each *rate section* in a model, as discussed "Rate sections" on page 451.

In a rate section, the effective rate of flow cannot be higher than the lowest maximum rate for all the blocks in that section. In fact, it can be lower than the lowest maximum rate, and could even be zero (0), depending on model conditions.

While each rate section can have only one *effective* rate, a section can have more than one block that has a *maximum* rate. In fact, it is common to have several Valve blocks, each with their own maximum rate, in a rate section.

Infinite rate

An *infinite rate* is a theoretical rate that would cause the flow to instantaneously move from one location in the model to another.



From the Executive's Discrete Rate tab

The Executive block's Discrete Rate tab specifies that a rate equal to or greater than some number is considered infinite; the default is that a rate ≥ 1e10 is considered infinite, as shown above.

You can change the infinite number to be anything that you want. However, because of the 12 digit precision limitation of the effective rates, the number should be set as close as possible to the highest possible effective rate which would ever reach this limit. (Setting a correct infinite rate is more critical in the case of potential upstream supply and potential downstream demand calculations, an advanced topic discussed on page 533.)

Infinite effective rate

Since instantaneous movement is not possible in the real world, the Rate library does not support an infinite effective rate. The infinity number specified in the Executive establishes an upper limit on the model's allowable set of effective rates. If the simulation calculates an effective rate that equals or exceeds that number, ExtendSim will stop the simulation and generate an error message. This could happen, for instance, if a source tank is directly connected to a sink tank, without any intervening constraint.

Infinite maximum rate

The Executive's infinity number (by default, 1e10) can be used in a block's rule set when conditions are such that it cannot in any way constrain the movement of flow. For example this would be accomplished by checking the ∞ (infinite) checkbox for a Valve's maximum rate. With an infinite maximum rate, the Valve will not limit the speed of flow passing through it.

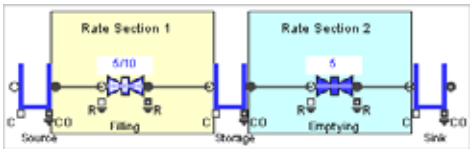
If you set the Valve's maximum rate field to blank or to a number ≥ 1e10, the block would also not limit the speed of flow passing through it.

Upstream supply/downstream demand

This potential rate is considered when using an advanced mode in the Diverge. Merge, and Sensor blocks. It is described fully in "Upstream supply and downstream demand" on page 533.

Rate sections

A *rate section* is defined as a network of connected blocks, all possessing the same effective rate. Each rate section can include a succession of blocks and connections. A rate section always starts with an outflow connector and ends somewhere downstream with an inflow connector. Thus it will always contain at least two blocks sharing at least one flow connection.



Two rate sections in a model

While some blocks always define the boundary between two rate sections, other blocks never define a new rate section boundary:

- Because a residence block (Convey Item, Interchange, or Tank) can hold flow for some period of time, its effective inflow rate can be different from its effective outflow rate. Since the boundaries between rate sections never change, residence blocks always define the boundary between two different sections, even if their effective inflow rate is the same as their effective outflow rate. For example, a Tank's inflow connection ends one rate section and its outflow connection starts another section.

- Some passing (non-residence) blocks define a new rate section and others don't. While flow is not permitted to be held in a passing block for any length of simulation time, some passing blocks are capable of defining boundaries between sections. For example, a Valve is part of one, and only one, rate section. On the other hand, the Change Units block's effective inflow rate will always be different than its outflow rate because it performs a unit conversion. It therefore defines the boundaries between two rate sections.

☞ All residence blocks, and certain passing blocks, always define the boundary between two different rate sections. These boundaries are established internally by ExtendSim at the beginning of the simulation run; they do not change.

The table below lists what role each block in the Discrete Rate library plays in defining the boundaries of a rate section:

Block	Always defines a new rate section?	Comments
Bias	No	The inflow effective rate is the same as the outflow effective rate. The block influences the effective rate associated with the section it is part of.
Catch Flow	No	The outflow effective rate is the same as the catch effective rate.
Change Units	Yes	The outflow effective rate is the inflow effective rate multiplied by the conversion factor.

Block	Always defines a new rate section?	Comments
Convey Flow	Yes	The outflow effective rate can be different than the outflow effective rate because it is a residence block.
Diverge	Yes	The effective rates can be different across all flow connectors – the input and all the outputs.
Interchange	Yes	See Convey Flow comment
Merge	Yes	See Diverge comment
Sensor	No	The inflow and outflow effective rates are the same.
Tank	Yes	See Convey Flow comment
Throw Flow	No	The inflow effective rate is the same as the throw rate.
Valve	No	The inflow and outflow effective rates are the same.

For an example of rate sections, see the “Comprehensive example” on page 461.

Rate precision

The mathematical precision for effective rates is limited to 12 digits. This can become an issue if you separate any two effective rates in an area by more than 12 digits of precision. For more information, see “Precision” on page 510.

LP area

While rate sections don’t change after the start of a run, the boundaries of the *LP area* change dynamically during the simulation. An LP area is composed of one or more rate sections linked together by the fact that their effective rates could change during the simulation run. When an event occurs that causes the effective rate for one rate section to be reevaluated, ExtendSim determines which other rate sections might be impacted. The affected rate sections constitute the LP area and become part of the LP calculation.

Since the LP area is computed internally, and because it is most important for the LP calculation, it is discussed fully in “The LP area” on page 527.

Flow rules

Flow rules completely define how a block permits flow to move through during the simulation run. When calculating rates of flow, ExtendSim's discrete rate architecture tries to maximize throughput throughout the system, subject to a set of constraints. In order to restrict flow, discrete rate blocks are required to define their own sets of flow rules. The aggregated set of these rules ultimately defines how fast flow is permitted to move over time throughout the model.

A block’s particular set of flow rules is derived from four factors:

- The block’s fundamental behavior.
- How its dialog has been configured, such as setting a Tank’s maximum input rate or entering a conversion factor in the Change Units dialog.

- How its value connections have been connected. For instance, the Valve block's R (maximum rate) input connector can be used to dynamically modify the block's maximum flow rate.
- How its flow connections have been connected. A Tank is a source if only its outflow connector is connected; it is a sink if only its inflow connector is connected.

These flow rules completely describe the events or conditions under which a particular block may constrain the movement of flow through it. However, changes in a block's constraints during a simulation cause its effective rates to be reevaluated and can cause a connected block's effective rates to be reevaluated, propagating calculations throughout an LP area. When recalculation is required, the Executive block (Item library) uses the aggregated set of flow rules from all the blocks in the LP area to calculate a new set of effective rates for the area. Thus a particular block's flow rules can be superseded by the global calculations of the Executive.

Critical and relational constraints

There are two primary types of flow rules: critical constraints and relational constraints.

Critical constraints

Critical constraint in a Valve

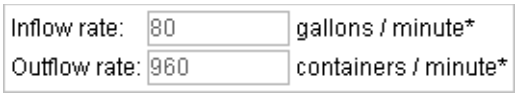
While all flow rules cooperate to constrain the rate of flow, some blocks provide special rules called *critical constraints*. If a rate section contains one or more critical constraints, they place an upper bound to the rate of flow for the blocks within that section. A critical constraint is unconditional – no matter what happens in the simulation, the effective rate of flow cannot be higher than the lowest critical constraint of any block in that rate section. For example, the *Maximum rate* field is a Valve's critical constraint; the entry in that field defines an upper limit on the rate of flow through the block. If that entry is the lowest critical constraint in the rate section, the effective rate for every block in that section cannot be higher.

The blocks with the potential to set a critical constraint for flow are the Convey Flow, Diverge, Interchange, Merge, Tank, and Valve; of these, the Valve is most commonly used. As will be shown in "Meeting the critical constraint requirement" on page 458, these blocks must be placed at critical locations in order for the model to run properly.

ExtendSim's discrete rate system attempts to move flow through the model as fast as possible. Without any mechanism to impede its progress, the effective rate would theoretically approach infinity and the flow would move from one part of a model to another instantaneously. In order to avoid this error condition, each LP area of the model must contain one or more constraints (typically a Valve) to restrict the flow to a number that is below infinity. If the required minimum set of critical constraints is not present in a model, ExtendSim stops the simulation and displays an error message.

Relational constraints

Relational constraints define the way the effective rates of different sections are related to each other, creating dependencies between rate sections. For instance, the relational constraint between one rate section (effective rate x) and another rate section (effective rate y), could be defined as $x \geq y$, $x=y$, $2x-3=y$, or any other expression. Relational constraints get updated when the block reacts to new parameters or to changes in its state, but they don't affect a block's critical constraints.



Different inflow and outflow rates

An example of a relational constraint is the Change Units block, where the use of a conversion factor causes the outflow effective rate to be different than the inflow effective rate. The Change Units block defines the boundaries between one rate section and another; the conversion factor specifies the relationship of the two effective rates.

For another example of a relational constraint, see "Comprehensive example" on page 461. For an advanced discussion of relational constraints, see "The relational constraint calculation" on page 531.

☞ You don't enter relational constraints, they are determined by the behavior of the blocks.

Comparison of constraints

Some blocks can set a critical constraint, some can set a relational constraint, and some can do both. Even for blocks that can set constraints, the block may in some situations place no constraint on the flow.

- The blocks that can set a critical constraint are the Convey Flow, Diverge, Interchange, Merge, Tank, and Valve.
- Relational constraints can be set by the Change Units, Convey Flow, Diverge, Interchange, Merge, and Tank blocks.

For example, a Tank where both the *Maximum inflow rate* and *Maximum outflow rate* are checked will set critical constraints for its inflow and outflow.

If neither *Maximum inflow rate* nor *Maximum outflow rat*e is checked, the Tank will not have any critical constraints but could have relational constraints. If the Tank has a finite capacity but is neither full nor empty, it places no constraints on the flow. However, once the Tank reaches the full state, its inflow rate is required to be less than or equal to its outflow rate; this is a relational constraint.

The effective rate for a rate section cannot be any higher than the lowest critical constraint set for by any of the blocks in that section. Furthermore, because the aggregated set of flow rules also typically contains relational constraints, the effective rate for the section can vary anywhere between zero and the smallest critical constraint.

☞ For a table that lists the blocks and which constraints they can provide, see "Types of information provided to the Executive" on page 529.

Defining a critical constraint

As mentioned earlier, a critical constraint defines the upper limit to the rate of flow through a rate section. While a particular rate section may or may not have a critical constraint, at least one of the rate sections within the LP area must have a critical constraint mechanism to limit the flow rate to a number that is less than infinity.

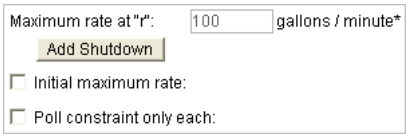
- You can explicitly set a critical constraint in the Valve, Tank, and Interchange blocks. You do this by entering a maximum rate in the block's dialog, obtaining a value for the maximum rate from the block's input connector, or linking the maximum rate field to the value of a cell in a global array or ExtendSim database.
- For the Convey Flow block, the critical constraint is derived from settings in its dialog and sometimes other model values, rather than being entered directly.

- A critical constraint may also be implicitly specified under certain conditions by the Merge and Diverge blocks.

The next sections describe how to set a critical constraint. See also "Meeting the critical constraint requirement" on page 458 for examples of how to apply the constraint requirement in your models.

Valve

The Valve is the block most often used for explicitly setting a critical constraint. You can enter a value in the *maximum rate* field in the block's dialog, link the field to an ExtendSim database or global array, or connect the block's *R* (maximum rate) input connector to some value output.



Defining constraints for Valve

For an example of setting a fixed maximum rate for the Valve, see the tutorial in the Rate Module Quick Start guide.

Dynamically changing the maximum rate

There are two ways to change a Valve's maximum rate during a simulation run:

- Connect to the block's R (maximum rate) input connector
- Link the block's Maximum rate field to an ExtendSim database or global array

Connecting to the Valve's R input connector or linking its maximum rate field to a data source overrides any values directly entered in the maximum rate field. Instead, that field will display the current maximum rate as determined by the simulation run.

☞ The check boxes for "Initial maximum rate" and "Poll constraint every", discussed below, are only used when the Valve's maximum rate is configured to change dynamically.

For an example of using the Valve's R input connector to cause the maximum rate to change dynamically, see the tutorial in the Rate Module Quick Start guide.

☞ As you saw in the Rate Module Quick Start guide, the "Add Shutdown" button in the Valve's dialog automatically connects a Shutdown block (Item library) to the Valve's R input connector. This can be used to stop the flow, or reduce its rate of movement, for a period of time. See also "Shutting down" on page 331 for a description of how to use the Shutdown block.

Initializing the maximum rate

When the Valve's maximum rate is configured to change dynamically, the *Initial maximum rate* check box serves an important role. This is because the first effective rate calculations for a simulation occur just before simulation time starts moving forward. If the Valve's R connector is connected or if the maximum rate field has been dynamically linked, problems can arise at this stage because neither the block connected to the R connector nor the linked data source has yet had a chance to provide an initial value. The *Initial maximum rate* checkbox resolves this issue by initializing the maximum rate.

The initial value entered in the dialog will be used until the Valve gets a different value from its R input connector or from the linked data source.

☞ For multiple runs, the *Initial maximum rate* check box prevents the Valve from using the last maximum rate from the current run as the initial maximum rate for the next run.

Polling constraints

The check box to *Poll constraint every...* can be used when a Valve's maximum rate is configured to change dynamically. This option directs the Valve to request a new maximum rate value at fixed intervals during the run. This causes the Valve to periodically query the block connected to its R (maximum rate) input connector or the cell linked to its maximum rate field for the new values. Any values received between the queries will be ignored.

This checkbox is optional when the maximum rate field is connected to a fixed number in a linked cell in a data source. It is required if the linked cell contains a random number or if the R input connector is connected to a passive block like the Random Number (Value library), since a passive block won't independently generate a new value for the maximum on its own. (The checkbox is not needed if the R connector is connected to block that actively generates values, such as the Lookup Table block set to output values at regular time intervals in the discrete rate tutorial.)

☞ Each time the maximum rate in a Valve changes, effective rates must be re-calculated across multiple sections. If you are using the poll constraint feature in several Valves, consider having them update at the same time. This will dramatically reduce the number of recalculations.

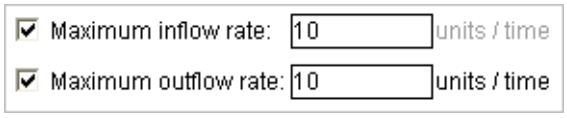
☞ While the polling feature can be handy during the early stages of the model building process, flow rates in real world systems rarely change at fixed intervals. Use this feature judiciously and with caution.

Controlling how and when the Valve applies its maximum rate

The Valve's Control Flow tab has advanced options that allow you to manage how and when that block applies its maximum rate. By setting a goal or using hysteresis, you can explicitly control when the Valve's constraining rate will be observed, when it will be ignored, and for how long either of those will happen. These topics are discussed in the "Delaying Flow" chapter.

Tank and Interchange

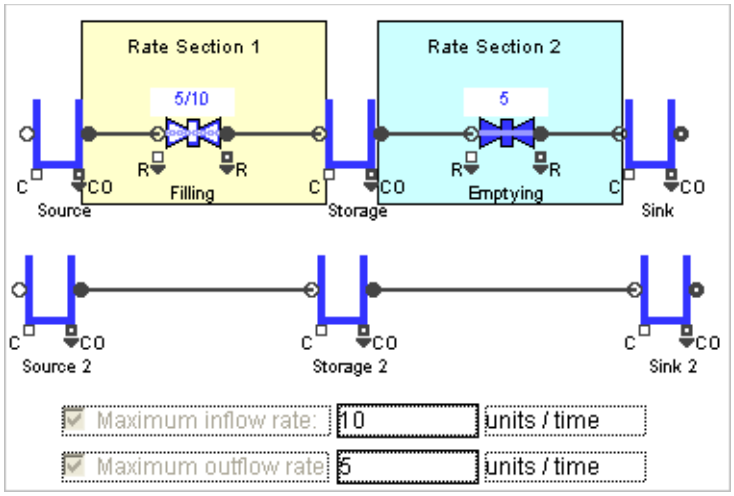
The Tank and Interchange blocks have dialog options for explicitly defining their maximum inflow and outflow rates. Unlike the maximum rate in a Valve, these constraints do not change dynamically during the simulation.



Default maximum rates for Tank

You can enter either a maximum inflow rate or a maximum outflow rate, or both of these.

The Tank Constraint example shows two flow streams with identical behavior. In the bottom flow stream, Tank 2 uses the options *Maximum inflow rate* and *Maximum outflow rate* to replace the filling and emptying valves found in the upper flow stream.



Tank Constraint model. Top stream with two Valves to constrain flow; bottom stream with maximum rates defined in Tank 2.

☞ Instead of using a Valve block to constrain flow, setting maximum inflow and/or a maximum outflow rates in a Tank or Interchange block can be used to satisfy the model's requirements for a constraint.

Convey Flow

The Convey Flow block calculates critical constraints for its inflow and outflow connectors separately. The critical constraints are derived from model conditions and settings in the dialog.

- The critical constraint for the Convey Flow block's inflow is calculated by multiplying the block's effective speed by its maximum density entry.

☞ The effective speed can be less than or equal to the speed set in the dialog. If the block is non-accumulating, or if it is accumulating but cannot accumulate more, and the block's ability to deliver flow exceeds downstream demand, the effective speed will be lower than the entered speed.

- The critical constraint for the block's outflow is the result of the multiplication of the block's speed setting by the density of flow present at the outflow end of the block.

☞ Setting the initial contents or capacity for a Convey Flow block is discussed in the chapter "Storage and Units". The "Delaying Flow" chapter shows how to use the Convey Flow block to delay the movement of flow in a model.

Merge and Diverge

The critical constraint for one or more of a Merge or Diverge block's branches can be implicitly specified under certain conditions. Most often, the result would be a rate of 0 (zero).

When a Merge or Diverge block is set to certain modes, flow can be blocked from moving through one or more of its branches. For example, if one branch of a Diverge block in Distributional mode has been assigned a blank value or a value <=0, flow through that branch is halted. Similarly, flow through all but the selected branch is blocked when the Merge block is in Select mode. In both of these cases, the maximum rate would be 0 for the affected branches.

The “Mode table” on page 465 lists each mode for the Merge and Diverge blocks. The column labeled “Parameter values that always block the flow” indicates which conditions would always cause a branch to have an implied constraint of 0.

Meeting the critical constraint requirement

As discussed earlier, while a particular rate section may or may not have a critical constraint, at least one of the rate sections within the LP area must have a critical constraint mechanism to limit the flow. Otherwise, the rate of flow would approach infinity.

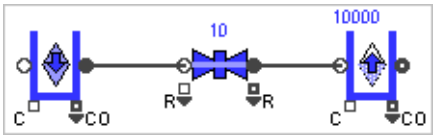
☞ By definition, residence blocks always delineate the boundary between two rate sections. A general rule is that there must be at least one critical constraint between every two residence blocks. (The critical constraints can be provided by the Convey Flow, Diverge, Interchange, Merge, Tank, and Valve blocks. The residence blocks are the Convey Flow, Interchange, or Tank.) The exceptions to the general rule include certain situations where a Merge or Diverge block is between two residence blocks.

The following examples illustrate some ways the required critical constraint mechanism can be met in discrete rate models.

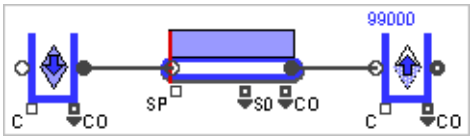
Valve or Convey Flow

The No Merge or Diverge model illustrates two typical ways to provide a critical constraint to the rate of flow between two residence blocks (in this case, Tanks) that don’t have a Merge or Diverge block between them.

The example to the right uses a Valve to constrain the flow between two Tanks. This is the most straight forward and most common situation. In order for the Valve’s maximum rate to provide the critical constraint it must be:



- Greater than or equal to 0 (zero)
- Less than 1e10 (the defined infinite rate)
- Not a blank



The example to the left uses a Convey Flow block to meet the requirement for a critical constraint. A Convey Flow block derives the critical constraint for its inflow from its dialog settings and the critical constraint for its outflow from its dialog settings and model conditions. Because it has critical constraints at both its inflow and outflow connectors, the Convey Flow block limits the rate of flow from the first Tank to the second to a number that is less than infinite.

Tank or Interchange

Instead of using a Valve to provide the critical constraint between two residence blocks, you can specify maximum inflow and maximum outflow rates for an intervening Tank or Inter-

change block. With these maximum rates, the Tank or Interchange will limit the rate of flow between the two residence blocks to a number less than infinite. This is shown in the Tank Constraint example discussed in "Tank and Interchange" on page 456.

Merge or Diverge blocks

If a Merge or Diverge block is between two residence blocks, the inflow and outflow branches may or may not require a critical constraint mechanism.

☞ For any Merge/Diverge mode, if a critical constraint has been placed on a Merge block's outflow branch, no critical constraints are required on its inflow branches. Likewise, a critical constraint on a Diverge block's inflow branch means that no critical constraints are required on its outflow branches. If those constraints have not been placed, the critical constraint requirement depends on the block's mode.

The following table provides an overview of each mode's requirements for critical constraints when neither the Merge block's outflow branch nor the Diverge block's inflow branch has a critical constraint. (In this table, the word "variable branch" means an inflow branch for the Merge block or an outflow branch for a Diverge block.)

Mode	Critical constraint requirements if there is no critical constraint on the non-variable branch
Batch/Unbatch	Only on one of the variable branches
Distributional	Each variable branch
Neutral	Each variable branch
Priority	Each variable branch
Proportional	Only on one of the variable branches (See Note, below)
Select	Each variable branch
Sensing	Each variable branch

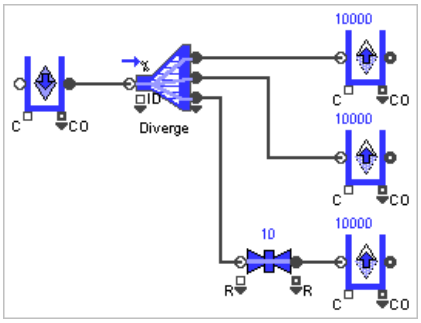
Note: For the Proportional mode, the variable branch with the critical constraint should not have a proportion <=0. Otherwise, that branch will be closed and the other variable branches will have potentially infinite effective rates. This is an error condition.

☞ Merge and Diverge blocks, including their modes, are described fully in the chapter "Merging, Diverging, and Routing Flow".

The two examples that follow use the Minimum Valve model to illustrate some of the table's concepts.

Proportional mode

The top section of the Minimum Value model indicates the critical constraint requirement when a Merge or Diverge block is in Proportional mode. If the block is located between two residence blocks, only one critical constraint is needed as long as the branch's proportion is neither 0 nor blank. (This lower requirement for constraints is an exception to the general rule described on page 458.) The effective rates for the other branches are deduced from the Valve's maximum rate.



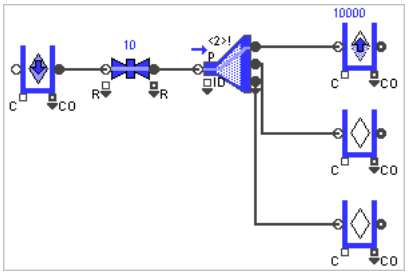
Only one constraint needed

In the Minimum Valve example shown on the right, a Valve is placed on a Diverge block's bottom outflow branch, and that branch does not have a 0 or blank proportion.

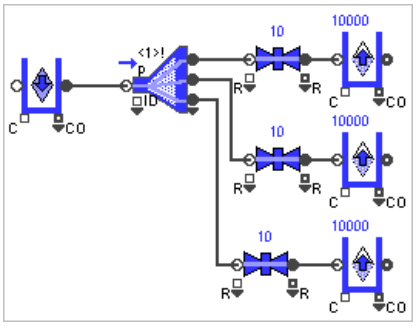
Priority mode

The lower section of the Minimum Value model indicates the critical constraint requirements when a Merge or Diverge block is in Priority mode. In this case, the number of critical constraints that must be placed on the branches between residence blocks depends on where those constraints are placed. These situations are shown in the Minimum Valve model.

If a critical constraint is placed between a residence block and a Diverge block's inflow branch, you do not need to place any other critical constraints on the Diverge block's outflows. Likewise, if you place a critical constraint on a Merge block's outflow branch, you do not need to place any critical constraints on its inflow branches. This is shown on the right, where a Valve with a maximum rate greater than or equal to 0 but less than 1e10 (the infinite rate) is on a Diverge block's inflow branch and there are no critical constraints required on its outflow branches.



No constraint on each outflow



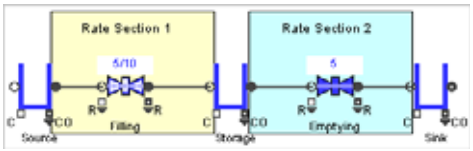
Constraint on each outflow

If you don't place a critical constraint on a Diverge block's inflow branch, you must place at least one critical constraint on each outflow branch. Likewise, if you don't place a critical constraint on a Merge block's outflow branch, you must place at least one critical constraint on each inflow branch.

This is shown in the screenshot to the left, where there is no critical constraint on the Diverge block's inflow branch. This means each outflow branch must have a critical constraint, in this case a Valve with a maximum rate greater than 0 but less than 1e10 (the infinite rate).

Comprehensive example

The following example illustrates many of the concepts from this chapter. The top line of the Tank Constraint model, shown on the right, has two rate sections, two critical constraints, and one relational constraint.



Tank Constraint model

☞ The sections that follow use the abbreviation FPT to indicate “flow units per time unit”.

Rate sections

Rate sections are determined internally by a communication between Rate library blocks and the Executive (Item library). The boundaries between rate sections are established at the beginning of the simulation run; they do not change during the run even if the effective rates change.

At the beginning of the simulation run:

- The Filling valve has a maximum rate of 10, gets its inflow from an infinite Source, and sends its outflow to an empty Storage tank that has a capacity for 100 flow units. The system will thus calculate an effective inflow and outflow rate of 10 FPT for the Filling valve at the start of the simulation run. (This will change once the Tank fills.)
- The Storage tank has a a capacity for 100 flow units, gets its inflow from a valve with a maximum rate of 10 FPT and sends its outflow to a valve with a maximum rate of 5 FPT. At the beginning of the simulation run, its effective inflow rate will thus be 10 FPT and its effective outflow rate will be 5 FPT.
- The Emptying valve has a maximum rate of 5 FPT and sends its outflow to an infinite Sink. Its effective inflow and outflow rate is 5 FPT.

At the start of the simulation run, the Storage tank’s effective inflow rate is different from its effective outflow rate. Thus the first rate section for the Tank Constraint model starts at the Source block’s outflow connector and ends at the inflow connector on the Storage tank. The second rate section starts at the Storage tank’s outflow connector and ends at the Sink’s inflow connector.

Critical constraints

There are two critical constraints in the top line of the Tank Constraint model. The first critical constraint is the 10 FPT entered in the Filling valve’s *maximum rate* field. The second is the 5 FPT entered in the Emptying valve’s *maximum rate* field.

Relational constraint

Relational constraints define the way the effective rates of different sections are related to each other. At the beginning of the simulation run there are no relational constraints – the effective inflow rate is independent of the effective outflow rate. When the Tank (which has a finite capacity of 100 flow units) becomes full, it applies one critical constraint: inflow rate must be less than or equal to outflow rate.

Simulation’s impact on the effective rates

Since it is empty at the start of the run, the Storage tank's initial set of flow rules will not include placing any restrictions on its inflow rate. Consequently, the initial effective rate of flow through Rate Section 1 is limited only by the Filling valve's critical constraint of 10 FPT.

However, this initial effective rate for the first rate section is only temporary. Since the Storage tank's capacity is finite and since Rate Section 2's effective rate is only 5 FPT, the Storage tank will eventually become full. Once this happens, the effective rate of 10 FPT in Rate Section 1 can no longer be maintained. Consequently, the Storage tank introduces a relational constraint that requires its inflow effective rate (Rate Section 1) to be less than or equal to its outflow effective rate (Rate Section 2). Once the Storage tank is full, its relational constraint causes the effective rate through Rate Section 1 to be reduced to 5 FPT.

Discrete Rate Modeling

Merging, Diverging, and Routing Flow

Using the Merge, Diverge, Throw Flow and Catch Flow blocks

When building models, you will frequently encounter situations where you want to route the streams of flow in a model. This is accomplished using the Catch Flow, Diverge, Merge, and Throw Flow blocks.

The Merge and Diverge blocks have similar interface and capabilities. These two blocks send and receive flow through a variable number of inflow and outflow connectors. Their dialogs provide rule-based options to merge or diverge flow in a discrete rate environment.

The Throw Flow and Catch Flow blocks also have similar interfaces. These blocks route flow remotely from point to point.

This chapter covers:

- Blocks for merging, diverging, and routing flow
- Merge and Diverge modes
- Additional features of the Merge and Diverge blocks
- Using the Throw Flow and Catch Flow blocks

☞ The models illustrated in this chapter are located in the folder \Examples\Discrete Rate\Merge and Diverge.

Blocks of interest

The following blocks from the Rate library will be the main focus of this chapter.

Catch Flow

Receives flow sent from Throw Flow or Diverge blocks. Allows you to group blocks that can send the flow into sets, so that the list of possible connections can be filtered.

Diverge

Distributes flow from one inflow branch to one or more outflow branches at a time. The block has several modes for determining how the flow is distributed through the branches.

Merge

Merges flows from one or more inflow branches at a time into one outflow branch. The block has several modes for determining how the inflows should be received.

Throw Flow

Sends flow to Catch Flow or Merge blocks. Allows you to group the blocks that can receive the flow into sets, so that the list of possible connections can be filtered.

Merging and diverging flow

The systems modeled using discrete rate technology typically have multiple flow streams that need to be merged into one stream or, conversely, one flow stream that needs to be diverged to multiple streams. The Merge and Diverge blocks have been designed specifically to model this type of routing behavior.

The Merge and Diverge blocks have seven different rule-based options that determine how they send and receive flow. These *modes* mostly behave as mirror images of each other in the two blocks. The list of modes, and their similarities and differences, are summarized in the “Mode table”, below. The examples that follow the table show how each of the modes can be applied.

☞ For the Merge block, each input connector is referred to as an *inflow branch*. For the Diverge block, each output connector is referred to as an *outflow branch*. Collectively they are known as the *variable branches*.

Mode table

The following table lists the Merge and Diverge modes in alphabetical order and summarizes their main similarities and differences.

Mode	See page	Sum of inputs = sum of outputs?	Fixed rule?	Bias order required?	Parameter value at each branch that will *always* block the flow	Compatible with Sensing mode?
Batch/Unbatch	467	No	Yes	No	None	Yes
Distributional	470	Yes	No	Yes	Blank, <=0	Maybe
Neutral	472	Yes	No	No	None	Maybe
Priority	468	Yes	No	Yes	Blank	Maybe
Proportional	468	Yes	Yes	No	Blank, <=0	Yes
Select	465	Yes	Yes	No	None	Yes
Sensing	471	Yes	No	Yes	Blank, <=0	Yes

Characteristics

Explanations for the mode characteristics are:

- Some modes use a fixed flow rule to obtain or distribute the flow – no matter what happens in the rest of the model, the fixed rule will be respected. For other modes, the flow rules express a preference and are only invoked in specific situations depending on model conditions.
- Competing requests for flow amongst Merge and Diverge blocks that have been set to the Distributional, Priority, and Sensing modes require the use of bias ordering. This is discussed in “Biasing flow” on page 510.
- While other parameter values may block the flow in certain circumstances (for instance, if a number is out of range), for some modes a Blank or zero (0) will always cause the flow to be stopped. Values that are out of range will cause an error message; a zero (0) or Blank will not generate an error message.
- As discussed in their respective sections, incompatibilities can arise if an area of the model has one or more blocks that use the Sensing mode and other blocks that use either the Distributional, Neutral, or Priority mode. These situations should be avoided whenever possible as they can give inaccurate results.

Select mode

When the Merge or Diverge blocks are in Select mode, only one selected branch at a time is open. A table in the block’s dialog allows you to assign a unique ID number to each inflow branch (for the Merge block) or outflow branch (for the Diverge block). The ID connector on the block’s icon is then used to select which branch to open.

Options in the block's dialog allow you to specify what happens if the value at the ID connector doesn't match any of the branch IDs listed in the table:

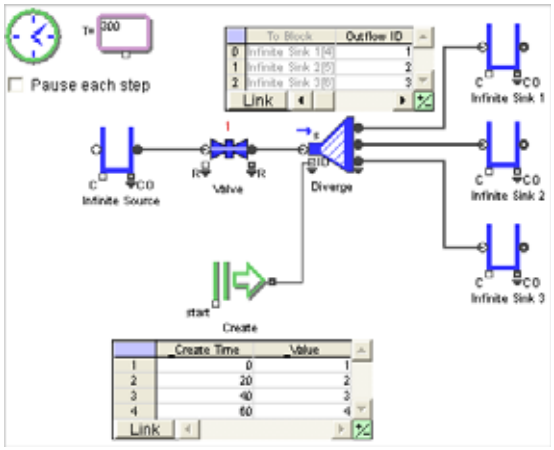
- Choose top connection
- Choose bottom connection
- Stop flow
- Generate error

A blank value received at the ID connector always stops the flow until the connector receives a valid input.

☞ The Select mode uses a fixed flow rule to obtain the set of effective rates for each branch and to determine which branch to route the flow to.

Select Mode Diverge model

In the Select Mode Diverge model, a Create block is set to output a sequential value (1, 2, 3, or 4) every 20 time units. At its ID input connector, the Diverge block receives the value from the Create block, compares that value to entries in its dialog table, and selects the appropriate outflow connector. Three Tank blocks, each with an infinite capacity for flow units, are connected to the Diverge block. The Tank blocks are identified by the ID values entered in the Diverge block's dialog. For example, an Output ID of 1 indicates the Tank labeled "Infinite Sink 1".



Select Mode Diverge model

In this example the Create block is responsible for controlling the Diverge block. When the Create block sends a value of 2 to the Diverge block's ID connector, the flow is routed to Infinite Sink 2, and so forth.

Notice that in this model 4 is an invalid number and the Diverge block is set to *Invalid value at ID: stop flow*. When the Create block sends a value of 4, all flow through the Diverge block stops and a red bar appears on the block's right side. This pause in the flow could be used for a specific purpose, for instance to allow time to empty downstream Tanks.

☞ Try running the model after checking *Pause each step* (in the upper left corner of the model). This will cause the simulation to pause so you can more easily see the effect of the Create block sending values to the Diverge block. Clicking the Pause/Resume button in the toolbar will continue execution to each succeeding event. (There can be more than one event without time advancing.) For more information, see "Stepping through a model" on page 89.

Select Mode Merge model

This model is the mirror image of the Select Mode Diverge model discussed above. While the three source tanks provide an infinite supply of flow, it is the Merge block that controls which tank flow is drawn from. Since the Merge block is in Select mode, the Create block controls

the routing of flow by providing different values (1, 2, 3, 4 sequentially every 20 minutes) at the Merge block's ID connector.

In this model, the Merge block is set to *Invalid value at ID: choose top connection*. This means that when the ID connector gets a value of 4 from the Create block, it will select the flow from its top input connector.

Batch/Unbatch mode

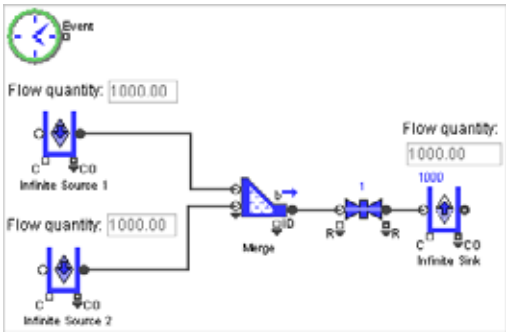
The Batch and Unbatch modes are used to cause a different total amount of outflow than what would be indicated by the total amount of inflow or to change the total amount of inflow into a different total amount of outflow.

- When the Merge block is in Batch mode, each unit of flow from each inflow branch is combined into one outflow unit. The effective rates of each inflow branch and the outflow connector are thus required to be equal. In this mode, the Merge block's behavior is similar to that of the Batch block (Item library).
- When the Diverge block is in Unbatch mode, each unit of flow from its inflow branch is cloned into one unit of flow for each outflow branch. The effective rates for the inflow connector and each outflow branch are thus required to be equal. In this mode, the Diverge block's behavior is similar to that of the Unbatch block (Item library).

☞ The Batch/Unbatch modes are different from all the other modes because the amount of total inflow is *never* equal to the amount of total outflow.

Batch Mode Merge model

In the Batch Mode Merge example, the Merge block is set to *Merge mode: batch*. Each time unit the block takes one unit of flow from Infinite Source 1 and one unit of flow from Infinite Source 2. It then combines them to make one unit of output flow per time unit. Since the model runs for 1,000 time units, the Infinite Source 1 and Infinite Source 2 blocks each provide 1,000 units of flow.



Batch Mode Merge model

Notice the amount of flow (1,000 units) that has entered the Infinite Sink is half the total amount of flow that has left the two source tanks. This is because the effective rate for the Merge block's outflow connector is required to be the same as the rate at each of its two inflow branches.

Unbatch Mode Diverge model

In this example, one unit of flow per time unit from the Infinite Source is unbatched into two flow units per time unit – one for Infinite Sink 1 and the other for Infinite Sink 2. Notice the total quantity of flow (2,000) in the two sink tanks is double the amount of flow (1,000) that exited the source tank.

Proportional mode

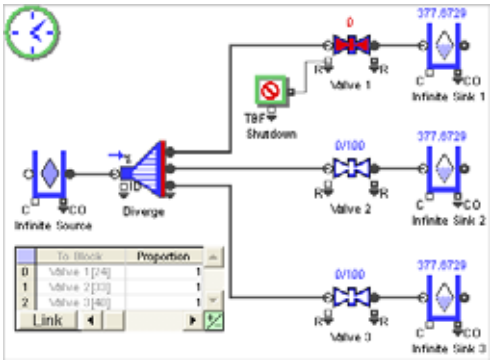
With the Proportional mode, you define in a table what the proportion of flow through each branch will be. The proportion for each branch is defined in the table relative to each of the other branches. For instance, a value of 2 for the top outflow branch and 4 for the bottom outflow branch would indicate that the bottom branch should have twice the amount of flow as the top branch. If a particular branch's proportion has been defined to be blank or <= 0, the effective rate for that branch is set to 0 and the flow is stopped for that branch.

☞ See “Merge blocks in Proportional mode” on page 517 for options when a Merge block is part of an empty loop.

⚠ This mode uses a fixed flow rule where the effective rate at each branch is required to meet the proportion defined by the table. Consequently, if the flow through one or more of the branches is blocked or starved, the effective rates for all branches will be set to zero and all flow through the block is halted.

Proportional Mode Diverge model

In this example, flow coming from the Infinite Source is evenly distributed between the Diverge block’s three outflow branches. This occurs because the proportions in the table in the Diverge block’s dialog have been set to 1:1:1. With this proportion, the effective rate across all three branches is required to be the same – an identical amount of flow must pass through each branch.



Proportional Mode Diverge model

The initial constraining rate for the three Valve blocks is set to 100. However, the Shutdown block forces Valve 1's constraining rate to alternate between 0 and 100 as the model runs. This has an impact on the effective rate for all three branches. When the constraining rate for Valve 1 switches to 0, the outflow from all three branches goes to 0 even though the constraining rate for Valves 2 and 3 is still equal to 100. This is because the Diverge block must enforce its ratio, which is 1:1:1 in this example.

Proportional Mode Merge model

This model is the mirror image of the Proportional Mode Diverge model discussed above. While all three Valve blocks limit the supply of flow from the source tanks at an initial constraining rate of 100, the Shutdown block forces the constraining rate in Valve 1 to alternate between 0 and 100. As in the previous model, when the constraining rate in Valve 1 switches to 0, the effective rates for all three branches become 0 because the Merge block is in Proportional mode and must enforce the 1:1:1 equality it is set to.

Priority mode

The Priority mode allows you to attach priorities to the inflow branches of the Merge block and the outflow branches of the Diverge block. These priorities only impact the effective rates assigned to the branches when discrepancies arise between the upstream flow supply and the downstream flow demand; otherwise they are ignored.

- In the case of the Diverge block, when the upstream supply is greater than or equal to the downstream demand, the block passes as much flow through each branch as the downstream demand will allow and the priorities are ignored. However, when the cumulative downstream demand exceeds upstream supply, the priorities that have been assigned to each branch are used to calculate the appropriate effective rates for the outflow branches.
- In contrast, the Merge block passes as much flow as possible through each inflow branch when downstream demand exceeds upstream supply, ignoring the priorities. However, when the cumulative upstream supply exceeds downstream demand, the priorities assigned to each branch are used to calculate the appropriate effective rates for the inflow branches.

Special cases apply to the use of the Priority mode in a Merge or Diverge block:

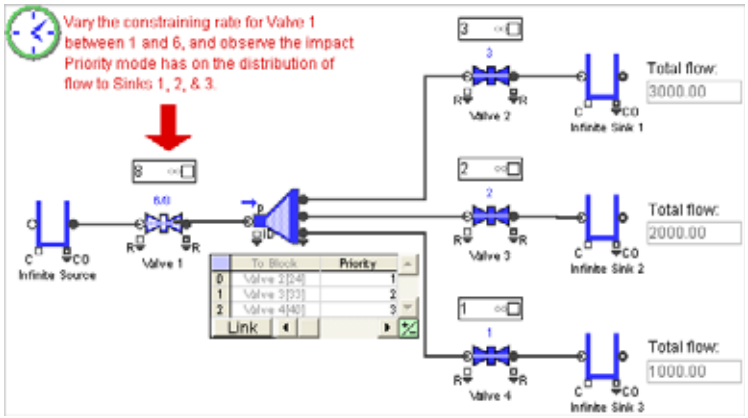
- If the priority for a particular branch has been set to blank, the effective rate for that branch will be zero and the flow will stop for that branch.
- If the priorities of two or more branches are equal, the flow will be divided among them in a "distributional" manner with equal proportions (see Distributional mode, below.)

☞ The priority entries in a Diverge block's dialog are not fixed rules but instead are situational; they are only used to resolve discrepancies when downstream demand exceeds upstream supply. For a Merge block, the entries are used to resolve discrepancies when upstream supply is greater than downstream demand.

⚠ Merge/Diverge blocks in Priority mode are not always compatible with Merge/Diverge blocks in Sensing mode. Consequently, an area of the model with some blocks in Sensing mode and others in Priority mode are prone to error. See "Cautions when using potential rates" on page 533 for more information.

Priority Mode Diverge model

In this example, the constraining rates in the valves have been set such that the upstream supply of 8 flow units per time unit through Valve 1 exceeds the cumulative downstream demand of 6 set by Valves 2, 3,and 4. Because a large enough supply of flow exists to satisfy downstream demand, the priorities in this case are ignored and have no impact on the set of effective rates defined for each outflow branch.



Priority Mode Diverge model

However, if a "supply scarcity" is introduced by changing the constraining rate in Valve 1 from 8 to 4, the Diverge block will calculate a set of effective rates that distributes the now limited supply of flow according to the defined priorities. Since the priorities have been assigned in descending order (top outflow branch has highest priority), the Diverge block will do its best to

satisfy the downstream demand that has been placed on the top outflow branch first. After that, if supply is still available, the Diverge block will attempt to service subsequent branches. This pattern is repeated until every branch has been satisfied or until the upstream supply of flow runs out, whichever comes first.

Priority Mode Merge model

When the blocks are in Priority mode, the difference between a Merge block and a Diverge block (illustrated above) is that the priorities defined in the Merge block's table impact the effective rates for the inflow branches if there is a downstream "scarcity of demand". The Priority Mode Merge model illustrates the use of Priority mode when 1) the downstream demand exceeds upstream supply, and 2) downstream demand is less than upstream supply.

Distributional mode

Similar to the Proportional mode described on page 468, the Distributional mode allows you to define a desired set of proportions for each branch. However, unlike the Proportional mode (but similar to the Priority mode discussed on page 468), these proportions serve as the decision rule for assigning effective rates to the branches *only* when discrepancies arise between the upstream flow supply and the downstream flow demand.

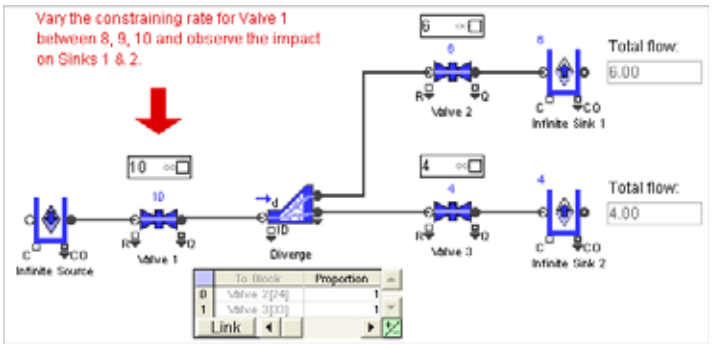
- In the case of the Diverge block, when the upstream supply is greater than or equal to the downstream demand, the block passes as much flow through each branch as the downstream demand will allow and the proportions are ignored. However, when downstream demand exceeds upstream supply, the proportions assigned to each branch are used as a guide to determine how the limited supply should be distributed across the outflow branches.
- In contrast, the Merge block passes as much flow as possible through each inflow branch when downstream demand exceeds upstream supply, ignoring the proportions entered in the dialog's table. However, when upstream supply exceeds downstream demand, the proportions assigned to each branch are used as guides to determine how the limited demand should be distributed across the inflow branches.

☞ The distributional proportions entered in a Merge or Diverge block's table are significant only in certain situations; they are ignored otherwise. Proportions do not follow a fixed flow rule; they only impact the effective rates assigned to the branches when discrepancies arise between the upstream flow supply and the downstream flow demand.

⚠ Merge/Diverge blocks in Distributional mode are not always compatible with Merge/Diverge blocks in Sensing mode. Consequently, an area of the model with some blocks in Sensing mode and others in Distributional mode are prone to error. See "Cautions when using potential rates" on page 533 for more information.

Distributional Mode Diverge model

In this example, the proportions for the Diverge block's two branches are set to 1:1. The constraining rates in the valves are defined such that the upstream supply of 10 flow units per time unit through Valve 1 equals the cumulative downstream demand of 10 set by Valves 2 and 3. Because a large enough supply of flow exists to satisfy downstream demand, the distributional proportions are ignored and have no impact on the set of effective rates defined for each outflow branch of the Diverge block.



Distributional Mode Diverge model

Two examples highlight what happens when the Diverge block is set to Distributional model and there is a "supply scarcity" that causes the upstream supply to be less than the downstream demand:

- If the constraining rate in Valve 1 is changed from 10 to 8, the Diverge block will use the 1:1 proportions that have been defined in its dialog to allocate the now limited supply between the two downstream demanding branches. In this case, 4 units of flow per time unit will move through both the top and bottom branches.
- If the constraining rate in Valve 1 is set to 9 units of flow per unit of time, the situation is different. According to the 1:1 proportions that have been defined in its dialog, the Diverge block should allocate 4.5 units of flow to each of the two downstream demanding branches. However, the constraining rate for Valve 3 is 4 units of flow per time unit and that is all it can accept. The extra 0.5 units of flow will be routed through the top branch because the downstream demand for the bottom branch cannot keep up with the upstream supply (4.0 vs. 4.5) and the Distributional mode will always try to push as much flow as possible.

☞ The Diverge block's Proportional mode is used to resolve discrepancies when downstream demand is greater than upstream supply.

Distributional Mode Merge model

When the blocks are in Distributional mode, the difference between a Merge block and a Diverge block (illustrated above) is that the proportions defined in the Merge block's table impacts the effective rates for the inflow branches only if there is a downstream "scarcity of demand". The Distributional Mode Merge model illustrates the use of the Distributional mode when 1) the downstream demand exceeds upstream supply, and 2) downstream demand is less than upstream supply.

☞ The Merge block's Proportional mode is used to resolve discrepancies when upstream supply is greater than downstream demand.

Sensing mode

Similar to the Proportional mode discussed on page 468, the Sensing modes use proportions to calculate the effective rates for the branches. However, unlike the Proportional mode where

you directly enter or control the proportions for each branch, the proportions for the Sensing modes are derived dynamically from the model as it runs.

- In the case of the Diverge block, Demand Sensing proportions for the outflow branches are calculated as a function of the potential downstream demand. For instance, the downstream demand placed on a particular outflow branch becomes the proportion for that branch.
- Similarly, the Merge block uses the potential upstream supply to define the Supply Sensing proportions for each inflow branch.

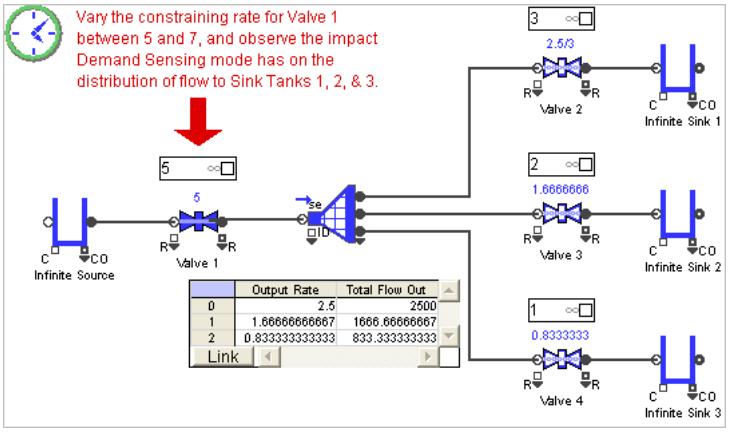
☞ Potential demand and supply rates are advanced concepts that are discussed in "Upstream supply and downstream demand" on page 533.

In the Sensing mode, the block's dialog has a table where you must define the maximum possible rate of flow through each branch. This upper bound is used as a way to limit throughput so that the proportions can be determined if the upstream supply or the downstream demand is infinite.

⚠ The discussion on page 533 provides reasons why the Sensing mode should be used with extreme caution and some situations where it should be avoided altogether. Given the potential problems, and because similar behavior can be achieved using the Distributional mode, the Sensing mode should be used only as a last resort.

Demand Sensing Mode Diverge model

In this example, the constraining rates in Valves 2, 3, and 4 define the demand for flow downstream of the Diverge block. They therefore define the proportions used to distribute the flow across the Diverge block's outflow branches.



Demand Sensing Mode Diverge model

A maximum possible rate of 1,000 for each branch is entered in the Diverge block's table. The block's Results tab (cloned onto the model worksheet) displays each branch's actual outflow rate and the amount of total outflow for the simulation run.

Supply Sensing Mode Merge model

In the Supply Sensing Mode Merge example, the constraining rates in Valves 1, 2, and 3 define the supply upstream of the Merge block. They therefore define the proportions used to distribute flow across the inflow branches.

Neutral mode

Unlike any of the modes discussed previously, the Neutral mode does not allow you to control the effective rates for the branches. This is a passive mode where no branch has a throughput

advantage; the branch that gets chosen cannot be predicted. It is used when the system does not need to control how the flow is routed.

- In the case of a Diverge block, when the upstream supply is greater than or equal to the downstream demand, the block passes as much flow through each branch as downstream demand will allow. However, when downstream demand exceeds upstream supply, the distribution of flow across each branch cannot be predicted.
- In contrast, the Merge block passes as much flow as possible through each inflow branch when downstream demand exceeds upstream supply. However, when upstream supply exceeds downstream demand, the distribution of flow across each branch cannot be predicted.

The Neutral mode should be used carefully but can be handy in certain cases. As a general rule of thumb, if you don't care exactly which branch has priority, but you do want maximum flow, consider using the neutral mode. The Neutral mode can also be used to resolve conflicting decision rules. For example, using the Neutral mode in a downstream Merge block would allow an upstream Diverge block in Proportional mode to control the effective rates of the inflow branches in the Merge.

⚠ Merge/Diverge blocks in Neutral mode are not always compatible with Merge/Diverge blocks in Sensing mode. Consequently, an area of the model with some blocks in Sensing mode and others in Neutral mode are prone to error. See “Cautions when using potential rates” on page 533 for more information.

Features of the Merge and Diverge blocks

Some features available in the Merge and Diverge blocks of particular interest include:

- Bias order
- Managing flow attributes when flow merges and diverges
- Internal Throw and Catch connections
- Dynamically changing parameters

These features are described in the following sections.

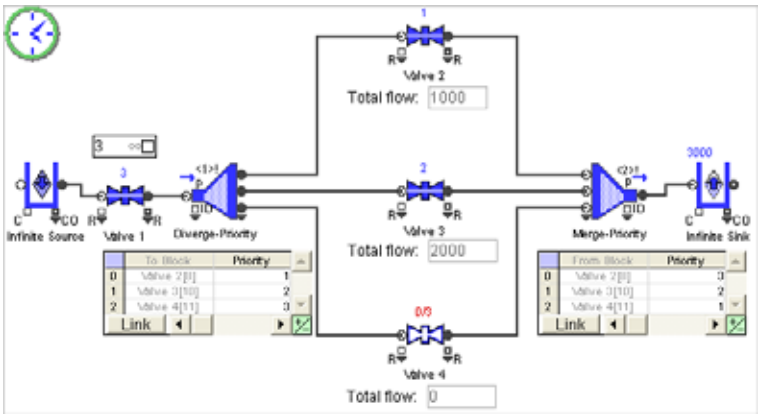
Bias Order – resolving competing requests for flow

As models grow in complexity, it is common for the priorities or proportions defined in one Merge/Diverge block to compete or conflict with the priorities or proportions defined in other Merge/Diverge blocks. This problem of “competing requests for flow” is resolved by assigning a *bias order* to the competing blocks. This is accomplished through entries in either the Model Settings tab of the individual blocks or the Discrete Rate tab of the Executive block. The following example demonstrates one of the many ways competing requests can arise and be resolved.

☞ Because certain modes allow flexibility in the way flow is distributed, Merge or Diverge blocks set to Distributional, Priority, or Sensing modes must specify a bias order to resolve conflicts between competing preferences for flow, as discussed below. For a complete description of the bias concept and bias order, see “Biasing flow” on page 510.

Competing Requests for Flow model

This model demonstrates how the priorities in two routing blocks compete against each other.



Competing Requests for Flow model

In this example:

- The Diverge block's outflow branch priorities have been specified in descending order while the Merge blocks' inflow branch priorities have been specified ascending order.
- The two blocks share common flow streams.

While the Diverge block in this model will try to satisfy its top outflow branch first, the Merge block will oppose that by trying to satisfy its bottom inflow branch. To resolve this conflict, the Diverge block's priorities have been biased over the Merge block. This was accomplished by selecting *Each block defines its own bias order* in the Discrete Rate tab of the Executive block, then selecting the Diverge block in the Executive's table and entering a bias of 1.

Selecting the option "Show bias order on icon" in the Discrete Rate tab of the Executive block causes the bias value to be displayed near block icons as "<x>". In the above model, the bias order is indicated as <1> for the Diverge block and <2> for the Merge block, indicating that the Diverge block has precedence over the Merge block's requests.

☞ To see how the Bias block is used instead of Merge/Diverge blocks to resolve competing preferences for flow, see the "Prioritize With Bias Blocks" model located in the folder \Examples\Discrete Rate\Merge and Diverge and discussed on page 510.

Managing flow attributes

When flow is merged or diverged, the management of flow attributes is based on settings in the dialogs of the Merge and Diverge blocks.

The Flow Attributes tab of the Merge and Diverge blocks has several options that determine how attribute values are handled when flow is merged or diverged. Default behaviors for layer and string layer attributes are specified separately.

Whichever option is selected becomes the default option for that type of flow as it passes through the block. For exceptions to the default, the blocks provide a table for specifying custom behavior for each attribute.

Internal throw and catch

While the Rate library has two blocks, Throw Flow and Catch Flow, specifically designed to transport flow without the use of connection lines, the Diverge and Merge blocks have been

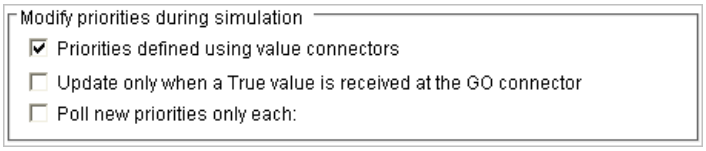
given throw/catch abilities as well. See “Throwing flow and catching flow remotely” on page 476 for a full discussion.

Changing decision rules dynamically

With the exception of the Batch/Unbatch and Neutral modes, decision rule parameters for the Merge/Diverge modes can be changed dynamically during the run. However, this is an advanced feature that requires some caution and extra insight into how ExtendSim works.

There are two ways to dynamically change a decision rule for a Merge/Diverge mode:

- Allow the rules to be controlled by other blocks. To do this, check the appropriate checkbox (such as *Priorities defined using value connectors*, as shown to the right) on the Merge or Diverge block’s Options tab. When this checkbox is selected, a set of value input connectors appear on the block’s icon, with one connector for each branch.



Modifying decision rules dynamically (Priority mode)

- Link the parameter table to an ExtendSim database table or global array. Any changes made to the table or array while the model runs will have the same effect as using a block to dynamically change the values.

Limiting the number of recalculations

While this advanced feature is useful for changing how effective rates are calculated on the fly, there are potential pitfalls. Since computations typically happen sequentially on a computer, new parameter values for each branch are changed one at a time. Ideally, the Merge/Diverge block will not recalculate the new set of effective rates for each branch until after all parameters have been updated. If this is not the case, however, the block will be forced to unnecessarily calculate an entirely new set of effective rates every time a parameter is updated. At the very least this will cause your runtimes to be longer than need be. At the very worst, redundant rate calculations could introduce bugs into your model when effective rates are temporarily calculated using one or more out-of-date parameter values.

There are three approaches that will help avoid this problem:

1) The issue can be bypassed if the inputs on one Merge/Diverge block are controlled by the outputs from one equation-based block, such as the Equation block (Value library). This is because the equation block will update all its outputs with the new results prior to alerting the Merge/Diverge block to the change. For an example of this, see the model “Change Priorities with Equation”.

2) By checking *Update only when a True value is received at the GO connector* in the block’s Options tab. This allows the calculation of a new set of effective rates to be controlled explicitly. In this case, changes to the parameters are ignored until a message is received at the *GO* input connector. This is shown in the example model “Change Proportions with Trigger” which requests a new set of proportions at the beginning of each goal. (A goal represents the production of 1000 units of flow and is repeated over and over until the end of the simulation.) The values at the inputs change every 10 time units, but because

the chosen set of parameter values remains unchanged for the duration of the goal, effective rates are recalculated only at the beginning of each new goal.

3) By checking *Poll new parameters only each: x time units* in the block's Options tab. This causes a new set of parameters to be updated at fixed intervals. In this case, changes to the inputs are ignored until the next interval in time arrives. In the example "Change Proportions Periodically", the model picks a new set of proportions every time another 100 units of time is reached.

☞ The options *Update only when a True value is received at the Update connector* and *Poll new parameters only each: x time units* can be combined together.

Throwing flow and catching flow remotely

The most common way to route flow from one block to another is by drawing a connection from an outflow connector to an inflow connector. This is a powerful mechanism for routing flow because it's simple to implement and it provides a very clear picture of how the flow is being routed in a model. The use of connection lines between connectors, however, can prove to be cumbersome when many streams of flow need to be routed into or out of hierarchical blocks.

The ExtendSim throw/catch mechanism solves this issue by allowing flow to be moved without the use of connection lines. By creating a throw/catch connection via block dialogs, flow can be routed from a throwing block to any catching block in the model.

In the Rate library:

- Flow can be sent from the Throw Flow and Diverge blocks
- Flow can be received by the Catch Flow and Merge blocks.
- Any sending block can throw to any catching block regardless of location.

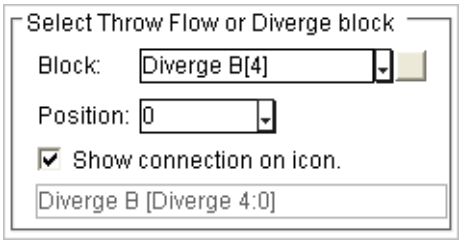
The rules restricting how normal flow connections can be drawn between outflow and inflow connectors also apply to throw/catch connections:

- The flow can go one way only – from throw to catch
- One throw can be connected with one and only one catch
- One catch can be connected with one and only one throw

☞ The advantage of using a Diverge block to throw flow or a Merge block to catch flow, is that each outflow or inflow branch can throw or catch a separate stream of flow remotely. The Throw Flow and Catch flow blocks, on the other hand, are limited to one flow source or destination each.

Creating a throw/catch connection

The creation of a throw/catch connection can be made from either the sending (Throw Flow or Diverge) or the receiving (Catch Flow or Merge) block. Connections are made by selecting the block to catch or throw the flow in a popup menu in a block's dialog. Each eligible block appears in the list with its block label and global block number. Once established, the connection information is automatically displayed in the dialogs of the sending and receiving blocks. In the screenshot above, the Catch Flow block will receive flow from a Diverge block labeled "Diverge B"; the Diverge block's global block number is 4.



Selecting a connection in a Catch Flow block

Choosing the connector position for Merge and Diverge blocks

If a Diverge or Merge block is part of the throw/catch connection, after selecting the connecting block, you must also choose a Merge or Diverge connector position for flow to come from or go to. This is because the Merge and Diverge blocks have multiple inflow and outflow branches. Some of their inflows or outflows may not be used for throwing/catching and some throwing/catching blocks may get flow from or send flow to different branches on a single Merge or Diverge block.

The number that indicates a Merge or Diverge block's particular connector position is displayed in the leftmost column of the table in the Merge/Diverge block's Throw or Catch tab; the number of the topmost inflow or outflow branch is zero (0). You select the connector position from a popup menu to the right of the Position field in the corresponding block. The menu will list all the available connector positions for the named block. In the screenshot in the preceding section, the top outflow connector position (0) for the Diverge block labeled Diverge B is entered in the Position field of a Catch Flow block's dialog.

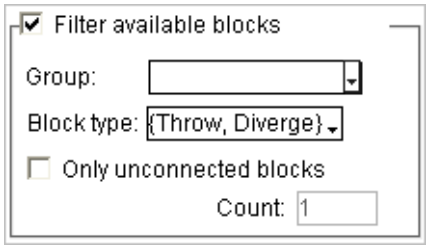
☞ An asterisk to the right of a connector position number in the popup menu indicates that the connector is already being used by some other throw/catch block.

Filter options to facilitate throw/catch connections

In large models, it is possible to have a great number of sending and receiving blocks from which a throw/catch connection can be made. To simplify the popup list of blocks eligible for connection, three types of filters can be applied:

- Group filter
- Block type filter
- Only unconnected blocks filter

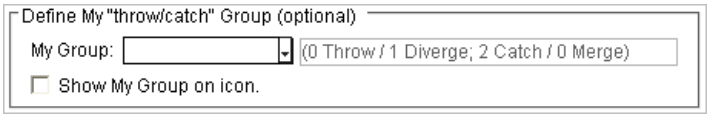
These filters can be used in combination with each other.



Filtering options

Group filter

Each block with throw or catch capabilities can be added to a throw/catch group. Groups can be created or selected through the group popup menu found in the sending and receiving blocks. When this popup is blank, the block does not belong to a group. When a block has been added to a group, its throw/catch options are limited to the blocks currently in that group.

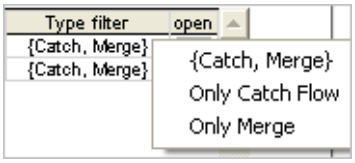


Defining a Group

Block type filter

By default, the block types a throwing block can connect to include both Catch Flow and Merge blocks and the block types a catching block can connect to include by Throw Flow and Diverge blocks.

- For throwing blocks, the list of blocks to select from can be narrowed to only Catch Flow blocks or only Merge blocks.
- For catching blocks, the list of blocks to select from can be limited to only Throw Flow blocks or only Diverge blocks.



Block type filters for a Diverge block

Only unconnected blocks filter

This filter narrows the selection of possible blocks to only those blocks without an established throw/catch connection.

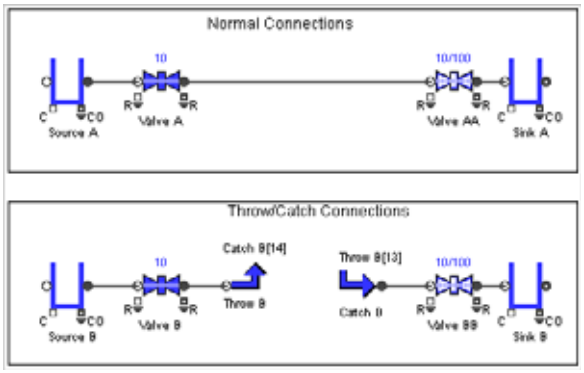
Examples of throw and catch connections

The two example models that follow show how to use catching and throwing in a model. The first model uses Throw Flow and Catch Flow blocks; the second model uses a Diverge block.

Catch Flow and Throw Flow model

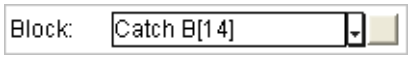
The Catch Flow and Throw Flow model shows two lines, one above the other. They produce identical results even though the flow connections have been created differently:

- The top line uses normal flow connection lines to connect Valve A to Valve AA.
- The bottom line uses a throw/catch connection to connect Valve B to Valve BB. In this line, the flow is sent remotely by a Throw Flow block and received by a Catch Flow block.



Catch Flow and Throw Flow model

The particulars of the throw/catch connection can be viewed from the dialogs of either the Throw Flow or Catch Flow blocks. In the Throw Flow dialog you can select the Catch Flow block and see the throw/catch connection; in the Catch Flow dialog you can select the Throw Flow block and see the throw/catch connection.

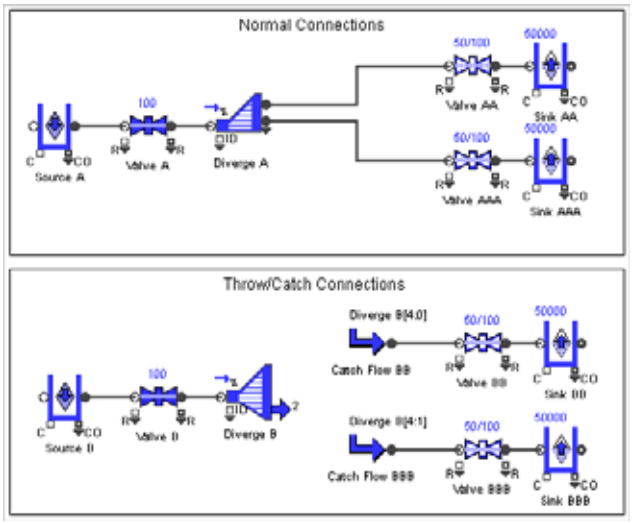


Portion of Throw Flow dialog

Catch Flow and Diverge model

Similar to the previous model, the top and bottom lines for the Catch Flow and Diverge model produce identical results even though the flow connections have been created differently:

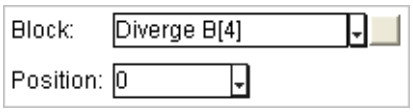
- The top line of the model a uses regular connections to connect the two outflow branches of a Diverge block to Valve AA and Valve AAA.
- The bottom line uses throw/catch connections to connect the outflow branches of the Diverge block to Catch Flow BB and Catch Flow BBB.



Catch Flow and Diverge model

In the bottom line, both Catch Flow blocks indicate in their dialogs that the block labeled Diverge B is remotely sending the flow. Since the flow is being sent by a Diverge block which can have multiple outflow branches, and since each Catch Flow block must receive its flow from a separate source, the Catch Flow blocks must also specify which of the Diverge block's connector positions they will receive flow from.

In the screenshot at right, the Catch tab of Catch Flow BB indicates that it is receiving flow from the Diverge block labeled Diverge B. It is getting that flow from the Diverge block's top outflow branch (outflow connector position 0).



Portion of Catch Flow BB dialog

☞ If the throw/catch connection has been properly defined for a particular branch of a Merge or Diverge block, the "Open" button for that branch will appear in the last column of the table in the block's Throw or Catch tab.

Discrete Rate Modeling

Delaying Flow

For a certain period of time, either
maintaining flow at a certain speed or blocking it.

The “Rates, Constraints, and Movement” chapter discussed how to specify critical constraints (such as a Valve’s maximum rate) and the factors that determine the actual speed of flow moving through a model.

☞ Since the concepts of flow rules and critical constraints are central to the discussion of delaying flow, it is assumed that you have already read the “Rates, Constraints, and Movement” chapter.

This chapter describes how to use advanced methods to delay flow – for a specified period of time or until a specified condition has been met, either blocking the movement of flow or maintaining it at a certain speed. It illustrates several methods for delaying flow, including how to:

- Control how and when a Valve observes or ignores its maximum rate setting:
  - Setting a goal for a quantity of flow
  - Setting a goal for a duration
  - Using hysteresis to control when the block’s maximum rate will be observed
- Use a Shift block (Item library) with a Convey Flow, Interchange, Tank, or Valve to delay flow movement for a specified period of time
- Transport flow over a defined distance at a specified speed with a Convey Flow block

☞ Most of the models illustrated in this chapter are located in the folder \Examples\Discrete Rate\Delaying Flow. The tutorial models mentioned are located at \Examples\Tutorials\Discrete Rate.

Blocks of interest

The following blocks from the Rate library will be the main focus of this chapter.

Convey Flow

Delays the movement of flow from one point to another. Can accumulate flow to a maximum density, accumulate flow to fill empty sections, or act as a non-accumulating conveyor.

Valve

Controls and monitors the flow, limiting the rate of flow passing through. This block can also be used to set a goal for the duration of flow movement or the amount of flow.

Controlling a Valve’s maximum rate

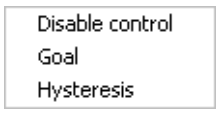
The section “Defining a critical constraint” on page 454 showed how to set a Valve’s maximum rate. Options on the Valve’s Flow Control tab provide advanced control over how and when the Valve applies its maximum rate. These settings determine when the Valve’s maximum rate is observed or ignored, and for how long.

The advanced control options include:

- Setting a goal for a certain quantity of flow to pass through the Valve.
- Setting a goal for how long flow can move through the Valve or for how long it should be stopped from moving.
- Determining what happens when the goal ends.
- Using hysteresis to delay the Valve’s response to system requirements.

Using the Flow Control tab

By default, the settings in a Valve's Flow Control tab are disabled. Use the tab's popup menu, shown at right, to choose either the Goal or Hysteresis option.



Flow Control options

If the Goal option is selected, an additional popup menu appears to the right. Use this second menu to choose:

- Goal as a quantity
- Goal as a duration

Observing the maximum rate for a goal

Whether you select a quantity or a duration goal, a Valve's maximum rate is observed while a goal is On. While the goal is off, you have the option to choose whether the maximum rate is observed or not and whether the flow is stopped. Thus your purpose in using a goal could be to block flow for a period of time and allow it to move through when the goal is off, allow flow for a period of time and block it at other times, accept a certain quantity of flow but stop flow when the goal is off, and so forth. In fact, you can choose to observe the maximum rate when the goal is off. This can be useful for a quantity goal, when you just want the Valve to report when a goal is finished.

- If its maximum rate is 0 (zero), the Valve will block flow while its goal is On.
- If its maximum rate is >0, blank, or infinite, the Valve will allow flow to pass through at that rate while its goal is On. (If the maximum rate is blank, the Valve uses an infinite rate.)

☞ If a Valve's maximum rate is zero and a quantity goal is On, no flow will go through the block and the goal will never end.

Options when goal is Off

Dialog options allow you to choose what will happen when the goal switches to Off:

- Stop the flow
- Ignore maximum rate (do not constrain flow)
- Observe maximum rate

Setting a Valve's quantity goal

Using popup menus in a Valve's Control Flow tab, you can specify that the block has a goal to pass a certain quantity of flow from its inflow to its outflow. Additional options allow you to specify what the Valve should do once that quantity of flow has passed.

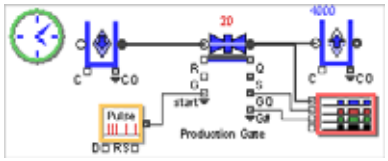
The quantity goal modulates a Valve's critical constraint (its maximum rate) by cycling between On and Off states.

- While a quantity goal is On, the value in the Valve tab's *Maximum rate* field, or the value at the block's *R* (maximum rate) input connector, is observed. The goal remains On until either the amount of flow that has passed through the block reaches the target quantity, or the goal is interrupted. At that point the goal switches to the Off state.
- What happens when the goal switches to the Off state depends on the Off option selected in the block's dialog: stop the flow, do not constrain flow, or observe maximum rate.

☞ To interrupt a goal, send a value to the Valve's *stop* input connector.

Quantity Goal model

This model uses a Pulse block (Value library) to periodically start a new quantity goal. At the start of the simulation run and every 60 minutes afterwards, the Pulse block sends a True value (a number ≥ 0.5) to the Production Gate valve's *start* connector, starting a new goal.



Quantity Goal model

The Valve's maximum rate is 20 gallons/minute and the simulation runs for 480 minutes. The control for the movement of flow through the block is provided by its Flow Control tab. The desired quantity of flow, 500 gallons, is entered in the dialog. While the goal is On, flow passes through the Valve at a maximum rate of 20 gallons/minute. After 500 gallons, the goal goes Off, the flow is stopped, and the effective rate goes to 0. As the simulation runs, a plotter displays the Valve's effective rate, the quantity of the goal and the times when it has been reached, and the number of each new goal.

Options for starting a new goal

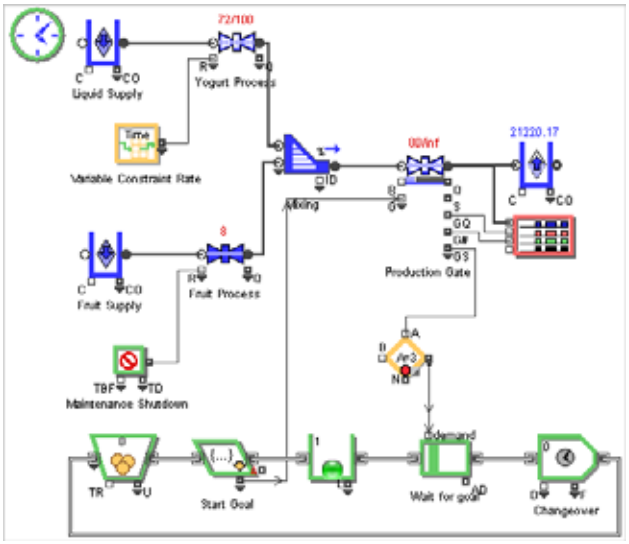
Settings in the Flow Control tab cause the Valve to *Start a new goal when "start" connector value = 1*. The goal-starting options (seen at right) can also be set to start a new goal when the block receives a message at its G (goal) input connector or when the previous goal finishes.

You could have avoided building the Quantity Goal model by instead mentally calculating the effect of the goal on the flow. The rate interactions and results determined by the next model would not be so easy to compute.

Changeover Quantity Goal model

The Changeover Quantity Goal model involves a more complex system than the Quantity Goal model from above. This model, based on the stage of the Yogurt Production model that is discussed in the tutorial in the Rate Module Quick Start guide, shows how a quantity goal in a Valve can be used to control a sequence of production changeovers. The top part of the model

is composed of Rate library blocks and the bottom line is Item library blocks. A Decision block (Value library) transmits values from a Valve (Rate library) to a Gate (Item library).



Changeover Quantity Goal model

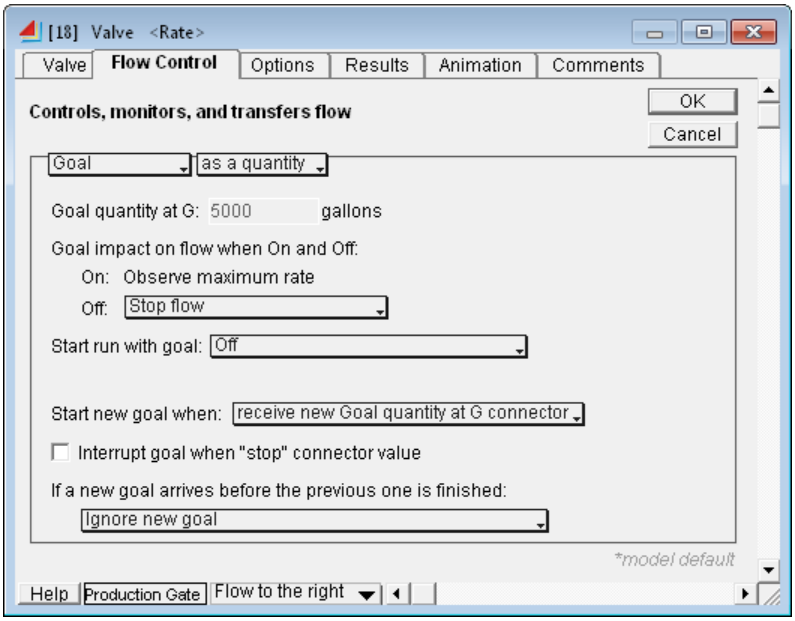
Since its initial goal has been set to "none", the goal status is Off and flow through the Production Gate valve is stopped at the start of the simulation. However, when an item passes through the Get block (Item library) labeled Start Goal, a message is sent to the Production Gate valve's G (goal) input connector. Once that happens, the valve's goal switches from Off to On and the goal quantity is set to the value of the item's Quantity attribute (5000 gallons).

In the bottom portion of the model, when the item leaves the Get block it moves into the Queue, where it is blocked from leaving by a Gate block. The Gate will remain closed until the Production Gate valve reaches its new goal of 5000 gallons. At that time the goal switches from On to Off, the flow stops (because that is the option set on the Control Flow tab), and the value at the GS (goal status) output is set to 3 (indicating that the goal has ended).

As a result, the Gate opens and the item moves into the Changeover Activity block where it is delayed for the amount of time required to perform a changeover. After the changeover has been completed, the item cycles back and initiates the next production cycle.

☞ By default, a Valve's GS (goal status) output connector reports the following values:
0 when there is no goal
1 when a goal is starting
2 when a goal is in progress
3 when a goal has ended
4 when a goal is interrupted

The following screenshot shows how the Production Gate's quantity goal has been configured on its Flow Control tab:



Flow Control tab, goal is quantity

- The value for the Goal quantity is received through the G input connector. (In this model, the goal is 5000 gallons because each item has a Quantity attribute value of 5000.)
- The popup menu option *Start run with goal Off* has been selected. (This causes the goal to be in the Off state at the start of the simulation.)
- The Valve has been instructed to *Stop flow* when the goal is Off.
- The popup menu option "Start run with goal Off" has been selected. (This causes the goal to be in the Off state at the start of the simulation.)
- If a new goal is received before the previous one is finished, the new goal will be ignored. (In this model, a new goal cannot arrive before the previous one is finished.)

An interesting aspect of this model is that the Production Gate valve has been set to have an infinite maximum constraining rate. This means that it will, in and of itself, not limit the rate of the flow moving through it. However, the block's effective (actual) rate will be determined by the two upstream Valves. One of these Valves is connected to a Lookup Table block (Value library) that changes its maximum rate depending on the time of day. The other upstream Valve is connected to a Shutdown block (Item library) that causes the movement of flow to be stopped periodically for specified durations. This sequence causes some interesting effects in the model, and the Production Gate's effective rate ranges between 80 and 0 gallons/minute.

☞ This examples uses the G (goal) connector to control the "when" and "how much" aspects of the goal. Alternately, the Control Flow tab allows you to choose to start a new goal when the *start* connector receives a message or when the previous goal finishes.

Setting a Valve's duration goal

Like the quantity goal, the duration goal is used to modulate a Valve's maximum rate. However, the duration goal cycles between the On and Off states as a function of time rather than the volume criteria used for the quantity goal.

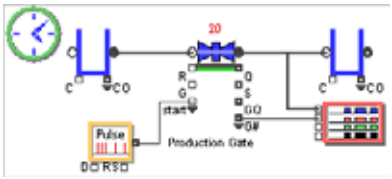
A duration goal remains in the On state for some amount of simulation time before switching to the Off state:

- While a duration goal is On, the value in the Valve tab's *Maximum rate* field, or the value at the block's *R* (maximum rate) input connector, is observed. The goal remains On until either the specified amount of time has passed, or the goal is interrupted. At that point the goal switches to the Off state.
- What happens when the goal switches to the Off state depends on the Off option selected in the block's dialog: stop the flow, do not constrain flow, or observe maximum rate.

☞ To interrupt a goal, send a value to the Valve's *stop* input connector.

Duration Goal model

The Duration Goal model is similar to the Quantity Goal model from above, except the goal allows the flow to move through the Valve for a certain period of time.



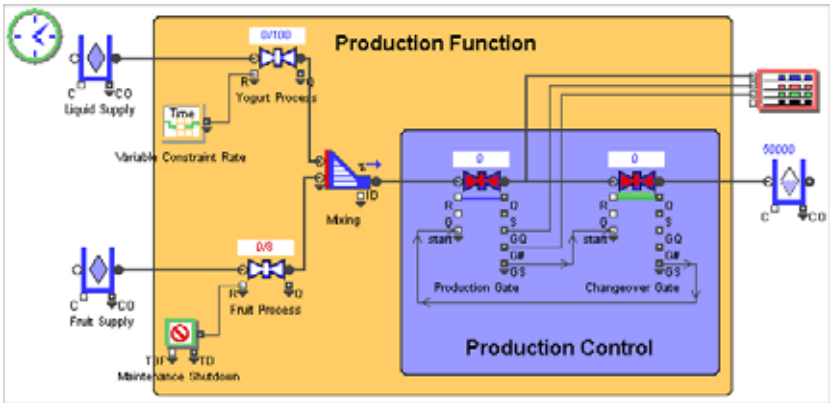
Duration Goal model

In this model, the Valve's Flow Control tab is set to observe its maximum rate of 20 gallons/minute for 45 minutes. When the goal is On, the block passes through whatever amount of flow it can, at the maximum rate of 20 gallons/minute. When the 45 minutes passes, the goal is set to Off and all flow through the block stops. The Pulse block (Value library) restarts the goal every 60 minutes. As the model runs, the block's maximum rate cycles from 20 gallons/minute to 0, depending on whether the goal is On or Off. As a result, the block's effective rate stays at 0 for the period of time (15 minutes) from when the previous goal ends until when a new goal starts.

Changeover With Only Goals model

The Changeover With Only Goals model is equivalent to the Changeover Quantity Goal model (discussed earlier) in terms of behavior. However, while the Production Gate valve still has a

quantity goal in this model, the changeover is controlled using a second Valve with a duration goal, rather than by an item.



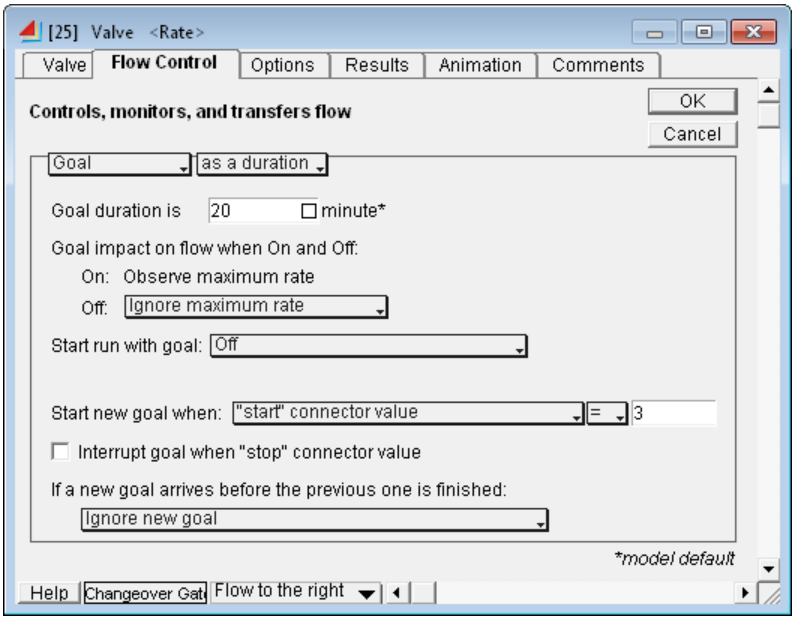
Changeover With Only Goals model

Since it's Flow Control tab specifies that it has an initial quantity goal of 5000 gallons, the Production Gate valve starts the simulation by observing its maximum rate setting – infinity. However, as was true for the Changeover Quantity Goal model, the block's actual effective rate will be impacted by the Liquid Supply and Fruit Supply valves upstream.

The Production Gate's goal state is communicated to the Changeover Gate through the connection between the first block's *GS* (goal status) output connector and the second block's *start* input connector. Because of this connection and because of how the duration goal has been specified in the Changeover Gate, the duration goal starts in the Off state and will switch to On only after the Production Gate's quantity goal is completed. After 20 minutes, the Duration Goal is finished and the Changeover Gate sends a message to the Production Gate to start a new quantity goal.

☞ Unlike the Duration Goal model that allows flow to pass through for a specified amount of time, the duration goal in this model causes flow to be blocked for a certain period. The Changeover Gate's maximum rate is set to 0 and the goal's duration is set to 20 minutes. While the goal is On, the block's maximum rate (0) is observed and no flow passes through, allowing for the changeover.

The following screenshot shows how the Changeover Gate's duration goal has been configured on its Flow Control tab:



Flow Control tab, goal is duration

- The goal duration is a constant 20 minutes. The duration could be made variable by instead choosing *Goal durations is: value at G connector*.
- The 20 minute blocking of flow allows the changeover to occur. The maximum rate on the Changeover Gate's Valve tab is 0. This would cause the flow to be blocked in the absence of any goal being set for this block. Consequently, if the block has a duration goal and it is On, that maximum rate of 0 will be observed and flow will be blocked from entering. When the goal turns Off after 20 minutes, the Changeover Gate doesn't apply any constraining rate on the flow because it is set to *Off: ignore maximum rate*.
- A new duration goal is started only when the *start* input connector receives a value of 3. Consequently, a new duration goal begins only when the upstream Production Gate's quantity goal has finished.
- When the changeover has completed and the duration goal switches from On to Off, a signal is sent from the Changeover Gate to the Production Gate. This results in a new quantity goal starting in the Production Gate.

Setting hysteresis in a Valve

Hysteresis is a property of systems that causes them to not react instantly to a change. The purpose of adding hysteresis in a model is to introduce a delay in the time it takes some part of the system to switch from one state to another.

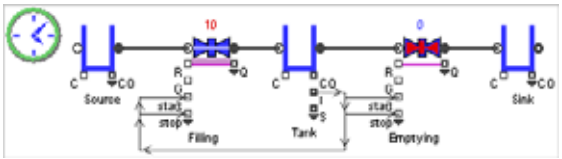
Hysteresis allows you to insert a lag or delay in a Valve's response to system requirements. It is used to avoid oscillations and to achieve better control over flow movement. This is accom-

plished by using model conditions to explicitly control both when a Valve's maximum rate is observed and when it is ignored.

Unlike the quantity and duration goals discussed earlier, where the conditions for applying the Valve's maximum rate were entered in its dialog, hysteresis must always get its control information from outside the block. The hysteresis option always relies on the Valve's *start* input connector to control when the Valve's maximum rate will be observed and its *stop* input connector to control when the maximum rate will be ignored. When the maximum rate is ignored, the Valve's dialog provides a popup menu for choosing if the flow stops or if the Valve does not constrain the flow.

Hysteresis model

In this model, the Filling valve opens when the Storage tank is empty and closes when the tank is full. Conversely, the Emptying valve opens when the Storage tank is full and closes when the tank is empty. As a result, the model repeatedly cycles through the following stages:

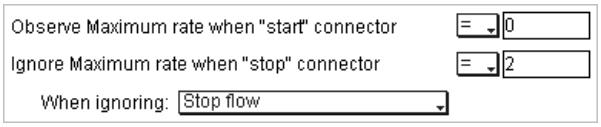


Hysteresis model

- Emptying valve closes and filling valve opens
- Storage tank starts accumulating flow
- Storage tank reaches the full state
- Emptying valve opens and filling valve closes
- Storage tank starts emptying
- Storage tank reaches the empty state

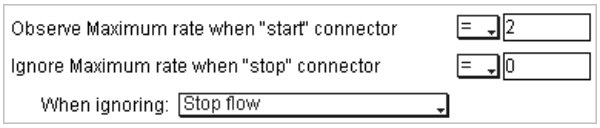
Based on settings in its Indicators tab, the Storage tank's I (indicator) connector outputs a value of 0 when empty and 2 when it is full. (Tank indicators are discussed on page 429.)

The Filling valve's Hysteresis settings are shown at the right. When this valve's *start* connector gets a 0 from the Storage tank's I output connector, the valve observes its maximum rate. When the Filling valve's *stop* input connector gets a 2, the block shuts down.



Hysteresis settings: Filling valve

The Emptying valve's hysteresis settings are the opposite of the Filling valve, as shown at the right. When this valve's start connector gets a 2 from the Storage tank's I output connector, the valve observes its maximum rate. When its stop input connector gets a 0, the valve shuts down.



Hysteresis setting for Emptying valve

Delaying flow with the Shift block

The Shift block (Item library) is discussed fully in “The Shift block” on page 372. It is used to schedule capacity in certain blocks found in the Item and Rate libraries. Shifts come it two types: On/Off and Numeric.

Rate library blocks do not support a Number type of shift. The following Rate library blocks can be controlled using On/Off Shifts:

- Convey Flow. When the shift is Off, the effective inflow rate is set to 0, blocking any new flow from entering. Depending on which option has been chosen in the Convey Flow’s dialog, the block’s speed will either also be set to 0 or it will remain unchanged from what is set in the dialog. If the *Empty when shift is off* checkbox is checked, any flow already on the conveyor at the Off shift time will continue progress towards exiting the block.
- Tank. The effective inflow and outflow rates are set to 0 when the shift is Off, effectively shutting the block down.
- Interchange. Same logic as the Tank.
- Valve. Same logic as the Tank.

Adding a Shift to a model

The easiest way to add a Shift block to a discrete rate model is to click the *Add Shift* button found on the Options tab of a Convey Flow, Tank, Interchange, or Valve block. This automatically does the following:

- Places a Shift block on the model worksheet below the originating block
- Enters the Shift name in the originating block's *Use Shift* field
- Opens the Shift’s dialog so settings can be entered

To use the Shift, enter the required information in the block’s dialog. (The Shift controls the originating block remotely; it does not need to be connected in the model.) Each Shift block starts with a default name for its shift. If you subsequently change the name of the shift, the new name will be reflected in the block that uses that shift.

For more information about using the Shift block, see “The Shift block” on page 372.

Convey Flow block

Setting an initial contents and the capacity of a Convey Flow block is discussed in the chapter “Storage and Units”. The current chapter focuses on the behavior of the Convey Flow block and how to set parameters that affect how flow is delayed through the block.

Flow entering the Convey Flow block is available to leave only after a specified delay that has been defined as a function of length and speed. The flow that enters is required to move some distance at a certain rate of speed before arriving at the block's exit point. The Convey Flow, then, is a residence block with flow distributed across its length at varying densities. (Density is the amount of flow that has accumulated at any one point on the conveyor.)

This block is useful for representing a conveyor, industrial oven, refrigeration system, or other similar piece of equipment with a length component where the position of flow must be taken into consideration.

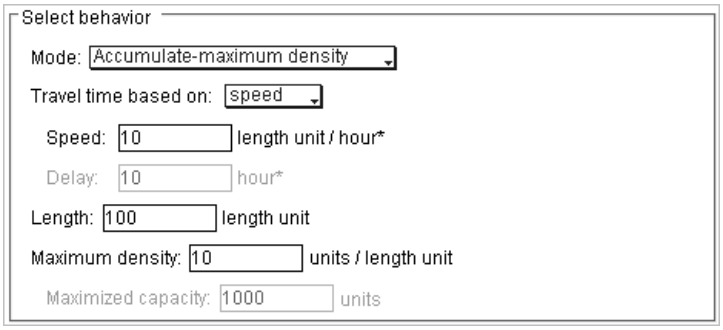
⚠ A Convey Flow block is computationally intensive, so it should be used only if the system you are modeling requires very precise tracking of flow movement and position. For instances when the block should not be used, see page 495.

Dialog settings

Movement of flow across the Convey Flow block is influenced by the dialog settings and parameters.

Determining speed and distance

The Convey Flow block offers two options: *Speed determines travel time* or *Delay determines travel time*. Depending on which of these is selected, the following entries can be made:



Dialog parameters for Convey Flow

- Speed. Specifies the maximum potential speed at which the conveyor can transport flow. However, when the potential supply of flow from the conveyor exceeds the downstream demand, the observed speed of the flow can become something less than the speed parameter.
- Delay. Represents the amount of time flow will spend in the block if there is no downstream blocking. If the block's dialog is set to *Speed determines travel time*, the delay is calculated by dividing Length by Speed. If the dialog is set to *Delay determines travel time*, the flow will take the entered Delay time to travel the stated Length.
- Length. Represents the distance flow must travel before reaching the block's exit point. The length of a Convey Flow block has to be greater than 0.
- Maximum density. Density is the amount of flow that has accumulated at any one point on the conveyor. The observed density of flow on a conveyor is a function of the upstream supply rate, the conveyor's speed, the downstream demand rate, and what settings have been chosen in the block's dialog. The Maximum density setting limits how high the pile of flow can be at any one point along the conveyor. For example, if the upstream supply rate is greater than or equal to the conveyor's maximum inflow rate (speed*maximum density), then the amount of flow entering the conveyor will be equal to the maximum density. In this case, it is the conveyor's capacity to receive flow that limits its effective inflow rate.

☞ The parameters for speed or delay can vary dynamically during the simulation; the length and maximum density parameters remain fixed.

Convey Flow behavior

The Convey Flow block is divided into segments, where the boundaries of each segment are defined by a change in the density of flow. Depending on the options chosen in the dialog, flow

could accumulate or “pile up” along the length of the block any time the amount of flow ready to exit the block exceeds downstream demand.

The Convey Flow block has three options controlling how or if flow is allowed to accumulate:

- Accumulate-maximum density. Allows flow to accumulate up to its maximum density setting. If the conveyor's ability to deliver flow exceeds downstream demand, any flow delayed from exiting will begin piling up at the outflow end of the conveyor up to the maximum density level.
- Accumulate- fill empty segments. Allows flow to fill in any empty segments along the conveyor when the block’s ability to deliver flow exceeds downstream demand. (An empty segment is an area along the block’s length that has a density of 0.) This differs from the first option in that one section of flow is not allowed to pile onto another section of flow.
- Non-accumulating. This option does not allow flow to accumulate. Therefore, the block’s speed slows when its ability to deliver flow exceeds downstream demand.

☞ The Compare Convey Flow model compares the behavior of three Convey Flow blocks, each set to one of these behaviors, under different emptying rates.

Constraining rates

Critical constraints define an unconditional maximum upper bound to the rate of flow. As discussed in the chapter “Rates, Constraints, and Movement”, the Convey Flow block calculates critical constraints for its inflow and outflow connectors separately. The critical constraints are derived from model conditions and settings in the dialog.

- The critical constraint for the Convey Flow block’s inflow is calculated by multiplying the block’s effective speed by its maximum density entry.

☞ The effective speed can be less than or equal to the speed set in the dialog. If the block is non-accumulating, or if it is accumulating but cannot accumulate more, and the block’s ability to deliver flow exceeds downstream demand, the effective speed will be lower than the entered speed.

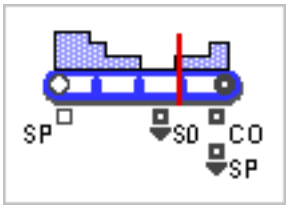
- The critical constraint for the block’s outflow is the result of the multiplication of the block’s speed setting by the density of flow present at the outflow end of the block.

Convey Flow information

The Convey Flow block provides different types of information concerning the distribution of flow along the length of its conveyor. Some of the information is provided by default while other mechanisms for reporting data, like sensors and indicators, must be customized through the block's dialog.

Distribution of flow

If you select Run > Show 2D Animation before you run the simulation, the distribution of flow in the Convey Flow block will be displayed across the top of its icon, as shown on the right.



Flow and accumulation point

On the Convey Flow block's Animation tab, selecting *Show block's flow distribution in table during simulation* causes a table to appear. As the simulation runs, the table displays information about the current distribution of the flow in each segment of its length:

Show block's flow distribution in table during simulation:

	High Limit	Low Limit	Density	Quantity
0	10 feet	9 feet	4 tons/feet	4 tons
1	9 feet	7 feet	2 tons/feet	4 tons
2	7 feet	5 feet	0 tons/feet	0 tons
3	5 feet	3 feet	2 tons/feet	4 tons
4	3 feet	2 feet	4 tons/feet	4 tons
5	2 feet	0 feet	6 tons/feet	12 tons

Link

Table showing example distribution of flow

☞ The length of a Convey Flow block is divided into segments, where the boundaries of each segment are defined by a change in the density of flow. The table above indicates that this Convey Flow block currently has 8 segments along its length.

Accumulation point

If the block is set to be accumulating, any accumulation will start at its outflow and go toward its inflow. The point beyond which no more flow can accumulate is known as the *accumulation point*. This point will probably move between the inflow and the outflow as the simulation runs.

The Results tab reports information about the accumulated flow: the distance from the outflow where the accumulation point is located, the indicator, and the accumulated quantity of flow.

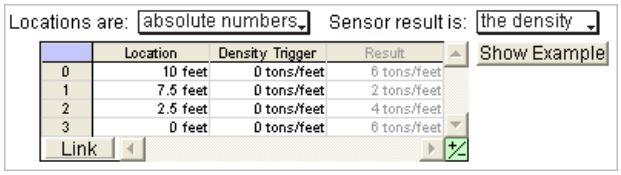
If the command Run > Show 2D Animation is checked while the model runs, and there is accumulation, a vertical red bar will move on the block's icon during the simulation; this indicates the location of the accumulation point. The "Flow and accumulation point" screenshot above shows the red line of the accumulation point.

☞ The accumulation point is located somewhere along the length of a Convey Flow block that has been set to accumulate flow.

Sensors

The Convey Flow block has a Sensors tab for specifying the locations and trigger points of sensors along the length of the conveyor. Each sensor reads and communicates the density of flow over time at a particular point on the conveyor. This information is displayed in a table in the block's dialog and reported by the block's S (sensor) output connectors – one S output for each sensor.

You must specify in the table not only the number of sensors desired but also the location of each one. (Use the +/- button in the table's lower right corner to specify the number of sensors.) The example table shown here indicates that four sensors have been placed along a ten foot section of the conveyor.



Sensor tab table

☞ The S connectors should be used judiciously. Extra events are used to update them at the proper time and the calculation is computationally intensive.

Indicators

As discussed in "Accumulation point" on page 494, the accumulation point indicates the point beyond which no more flow can accumulate. You might want an indication when the accumulation point is within a particular segment of the Convey Flow block.

The Indicators tab on a Convey Flow block is used to define segments to indicate where the accumulation point is along its length. Each segment is assigned a name, a defined range, and an ID number. The ID number is used to update the block's I (indicator) output connector as the accumulation point moves from one segment to the next.

See "Indicators" on page 429 for complete information about creating and using indicators.

☞ The I connectors should be used judiciously. Extra events are used to update them at the proper time and the calculation is computationally intensive.

When to avoid using the Convey Flow block

A Convey Flow block should be used only if the system you are modeling requires very precise tracking of flow movement and position. While the block is very precise, it is also very time consuming, so use it with caution. Following are some usage guidelines:

- The travel time has to be long enough to justify using the Convey Flow block. If the impact on the result of the simulation is small, it is a good strategy to ignore any travel time delays. For instance, the amount of time required for product to traverse a pipe separating two tanks is often insignificant and should usually be ignored.
- The distribution of flow along the Convey Flow block should impact the rest of the model in some significant way. If the effective inflow rate varies significantly during the simulation and/or if the speed changes, the product may be unevenly distributed across the length of the conveyor. This discontinuity impacts the rate at which flow is able to exit the block over time. The greater this variation in availability, the greater the potential impact on the rest of the model and the greater the block's use can be justified.

One alternative would be to use a combination of Tank and Valve blocks to mimic a Convey Flow block. This is far less computationally intensive than the Convey Flow block by itself. While some configurations may not be as precise, you can use a combination of one Tank followed by one Valve block in such a way that the behavior is identical or almost identical to the results when using a Convey Flow block.

Another alternative is shown in the Tank and Valve to Convey model. In this example, the bottom flow stream behaves almost identically to the upper stream without using a Convey Flow

block. Because Valves A1 and B1 vary their constraining rate, the Convey Flow block (Convey A) will get and create a lot of messages, slowing down the simulation. But the Convey B1 tank won't create the extra events. This will be slightly less precise but much more efficient.

☞ Never use a Convey Flow block if the system's behavior can be modeled using a Tank and Valve.

Discrete Rate Modeling

Mixing Flow and Items

Mingling discrete rate flows with discrete event items.

It is common for systems to exhibit a mixture of behaviors, where items from the discrete event arena intermingle with discrete rate processes. In these “mixed mode” cases, a proper understanding of both the Rate and Item libraries and how they can interact with each other is required.

There are two general techniques for integrating discrete event blocks from the Item library with discrete rate blocks from the Rate library:

1) Sending signals and sharing information via value connections. Value connections can be very useful for triggering some type of action as the system moves from one state to another. For example:

   - Value connections can trigger the generation of an item when the level of flow in a Tank reaches a certain level.
   - The value of an attribute on an item passing through some part of a discrete rate model could trigger a change in the constraining rate in a Valve block.

2) Mixing items with flow using the Interchange block, as you saw in the tutorial in the Rate Module Quick Start guide. The Interchange block (Rate library) provides the ability for items and flow to interface with each other. For example, an empty tanker truck (an item) might arrive at a refinery and fill with gas (flow) at a continuous rate using an Interchange block. Once full, the truck might be routed through a series of Item library blocks until it reached its unloading destination. At that point the item would again interface with an Interchange block to discharge its load.

This chapter focuses on the techniques used when the need for “mixed mode models” arises. It will show how to:

- Control flow using blocks from the Item library
- Control items with blocks from the Rate library
- Mix flow with items using the Interchange block

☞ The models for this chapter are located in the folder \Examples\Discrete Rate\Flow and Items.

Controlling flow with items and items with flow

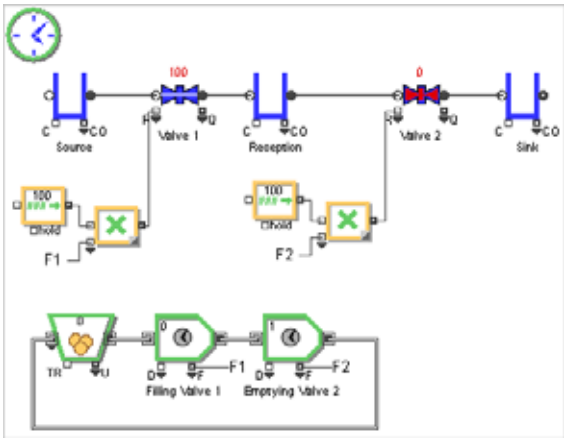
Rate library blocks can control the movement of items and Item library blocks can control the movement of flow. In either case, value connectors communicate state changes to the controlling blocks.

Items controlling flow

The Item Controls Flow model shows one of many ways blocks from the Item library can be used to control the movement of flow in a model. It uses one item to open a valve, allowing a tank to fill, then uses the same item to trigger the emptying of that tank.

Item Controls Flow model

Unless the presence of an item in an Activity block triggers them to open, Valves 1 and 2 are both closed during the simulation run. While an item is in the Activity block (Item library) labeled Filling Valve 1, it causes Valve 1 to open; this allows the Reception tank to fill. When the item moves to the Activity block labeled Emptying Valve 2, it causes Valve 1 to close and Valve 2 to open. This allows the Reception tank to empty into the Sink tank. The closing and opening of the valves is accomplished by detecting which Activity block has the item and for how long it is held there.



Item Controls Flow model

In this model, constant blocks (Value library) provide a potential constraining rate of 100 for the valves. Multiplying the constant value by an Activity's F (Full) output (which will be 1 if the Activity has the item in it and 0 if it does not) gives the valves their actual constraining rate (100 or 0). This causes the valves to open and close depending on where the item is in the model. Both Activity blocks are set to delay the item for a random amount of time, which represents how long the Reception tank can fill and empty.

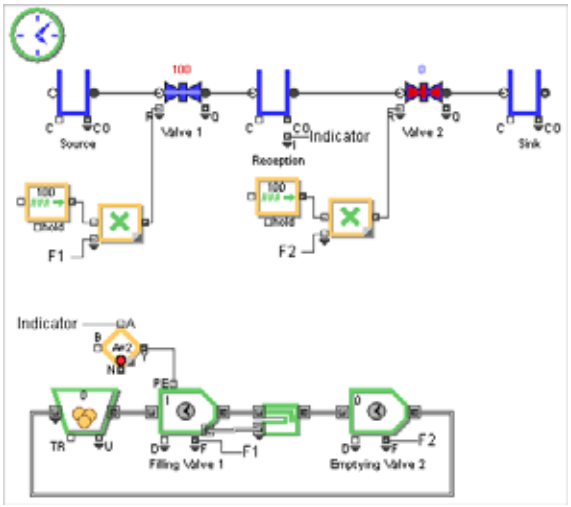
Notice that the Reception tank can be full even while the item is still in the Filling Valve 1 activity. Since the tank can't accept any more flow when it is full, and can't start emptying until the item moves to the Emptying Valve 2 block, the flow stops before the Filling process has been completed. The next model shows how the Reception tank can let the item know that it is full.

Flow controlling items

The example below illustrates one of ways blocks from the Rate library can be used to control the movement of items in a model.

Flow Controls Item model

This model is extends the one above, adding that the state of the Reception tank alters the movement of the item and interrupts the filling process.



Flow Controls Item model

As in the prior model, the Reception tank fills and empties based on the movement of an item and delays set in the two Activity blocks.

In this model, whenever the Reception tank is full, it sends an indicator signal to the Decision block (Value library). This block in turn notifies the Activity block labeled Filling Valve 1, pre-empting the item whose presence opened Valve 1 and allowed the tank to fill. When the item is preempted, it moves to the Activity block labeled Emptying Valve 2. This causes Valve 1 to close, Valve 2 to open, and the Reception tank to empty into the Sink for the duration specified in the second Activity block.

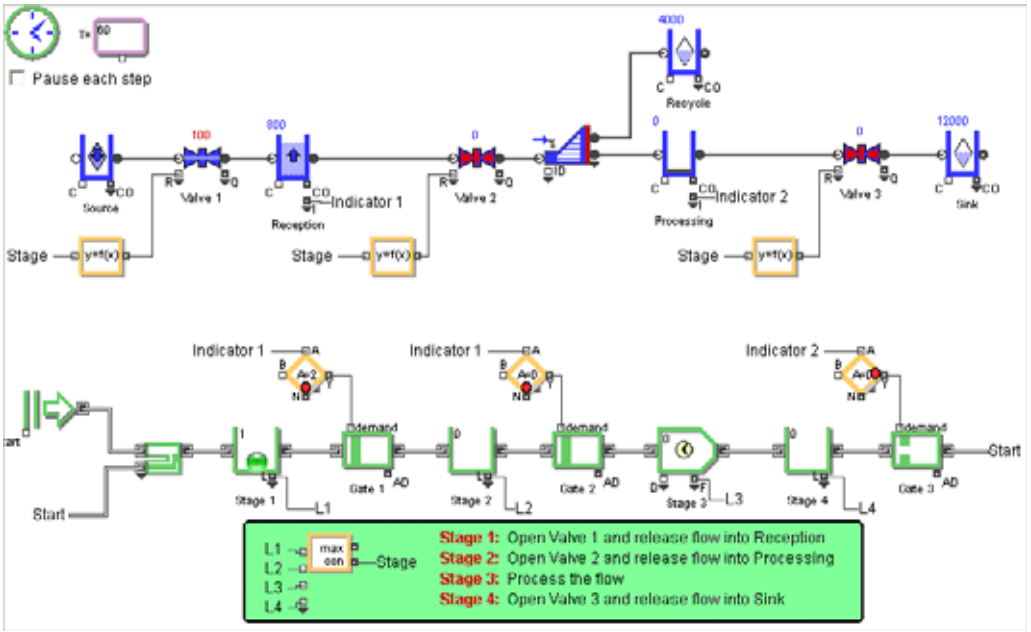
Flow controlling items and items controlling flow

The following model combines both of the previous concepts – flow controlling items and items controlling flow – into an even more complex system.

Step The Flow Process model

In this model the location of a “cycling item” triggers the opening and shutting of valves for the flow, while tank states control the opening and shutting of the gates for the item. The control logic is circular in nature: the item's location defines the current stage; the current stage

controls the opening and shutting of valve blocks; valves impact the level of flow; and the level of flow controls the item's location.



Step The Flow Process model

Flow controlling the item

The top section of the model is where flow controls the item. The I (Indicator) connectors on both the Reception and Processing tanks (Rate library) control the three Gate blocks (Item library) used in the item stream at the bottom part of the model. Gate 1 is open only while the Reception tank is in the *full* state. Conversely, Gate 2 is open only while the Reception tank is in the *empty* state. Finally, Gate 3 is open only while the Processing tank is empty.

Item controlling the flow

The bottom section of the model is where the item controls the flow. In this case, the three Valve blocks (Rate library) in the top part of the model are controlled by the lengths of the three Queue blocks (Item library) in the item stream in the bottom portion. The item stream has only one item which cycles through the item-based blocks in a loop. Consequently, the queue lengths for each Queue in this loop will alternate between 1 and 0. For instance, Valve 1 is open only when the length in the Stage 1 queue equals 1. Similar logic applies to Valve 2 and Valve 3.

The activity in the model is divided into four stages: open Valve 1 and release flow into Reception; open Valve 2 and release flow into Processing; process the flow; open Valve 3 and release flow into Sink

Stage 1: Open Valve 1 and release flow to Reception

The I (Indicator) connector on the Reception tank is controlling Gate 1. At the start of the simulation, Gate 1 is closed because the Reception tank is not full (it's actually empty at this point), and the cycling item is blocked from leaving the Stage 1 queue. Notice the highlighted

area where the Max & Min block (Value library) is used to translate the cycling item's location into a stage number. Since the cycling item remains in the Stage 1 queue, the model is now in a Stage 1 holding pattern. Consequently, Valve 1 is opened (so the Reception tank starts receiving flow) and Valves 2 and 3 are closed.

Stage 2: Open Valve 2 and release flow to Processing

Once the Reception Tank reaches the full state, Gate 1 is opened and Gate 2 is closed. This allows the item to move on to the Stage 2 queue. The result is that Valve 2 opens, Valves 1 and 3 close, and flow starts moving from the Reception tank into Recycling and Processing. Gate 2 remains closed while the Reception tank empties.

Stage 3: Process the flow

Once the Reception tank is completely empty, Gate 2 is opened, Gates 1 and 3 are closed, and the item enters an Activity block labeled “Stage 3”. While the item remains in Stage 3, all three Valve blocks remain closed. The Activity block, which has a delay of 2 minutes, is used to keep the flow in the Processing tank for some period of time so it can be processed.

Stage 4: Open Valve 3 and release the flow

Once processing is completed, the item leaves Stage 3 and moves into the Stage 4 queue. Since Gate 3 is currently closed (because the Processing tank is not empty), the model is now in a Stage 4 holding pattern and Valve 3 opens. Once the Processing tank is empty, Gate 3 opens and the item cycles back to the State 1 queue where the whole processes starts all over again.

☞ While this model is useful for demonstrating how mixed mode models can control item and flow movement, the same behavior can be created without items using the Valve block’s Hysteresis option. For more information, see “Setting hysteresis in a Valve” on page 489.

Using the Interchange block to mix items with flow

The Rate library’s Interchange block is unique in that it allows items and flow to interface directly with each other. Flow can enter the Interchange block not only through its inflow connector but also through the arrival of an item. Conversely, flow can exit the block through its outflow connector or through the exiting of an item.

The use of the Interchange block was introduced in the tutorial in the Rate Module Quick Start guide, and the block’s capacity to hold flow is discussed starting on page 440 of the Flow Sources, Storage, and Units chapter. This chapter will describe how the Interchange block interfaces flow with items.

☞ The Interchange block is where an item can be filled with flow or emptied of flow.

There are a number of occasions where it can be useful to provide items with the ability to store, transport, and empty flow as they move from one section of a discrete rate model to another. For example, the attribute capabilities used to distinguish one item from another can also be used to distinguish one block of flow carried by one item from another block of flow carried by a different item. This can be an especially useful modeling construct since flow units by themselves are indistinguishable from each other.

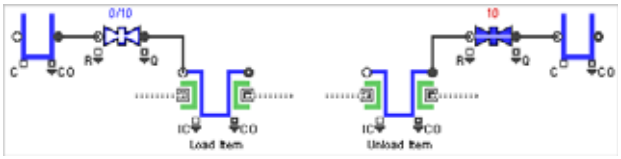
Behavioral rules

The Interchange block has two very different modes (“Tank only exists while item is in it” and “Tank is separate from item”) that affect how the block behaves. (These modes were introduced on page 440 and will be discussed more fully on page 504.) Even so, the Interchange block always follows a fundamental set of rules:

- The item input and item output connectors on the Interchange block must both always be connected.
- At least one of the Interchange block's inflow/outflow connectors must be connected (both may be connected as well).
- The Interchange block's capacity for holding items is permanently fixed at one item.
- The Interchange block loads and unloads the item and its flow instantaneously. (Flow present in the item when it enters the block is instantaneously available to the block; flow leaving the block with an item is instantaneously removed from the block.)

The flow connector configuration

The behavior of the Interchange block is dramatically affected by how the inflow and outflow connectors have been connected. As mentioned above, only one of the two flow connectors needs to be connected. If only the inflow connector is connected, arriving items can only be filled with flow, and if only the outflow connector is connected, arriving items can only unload flow.



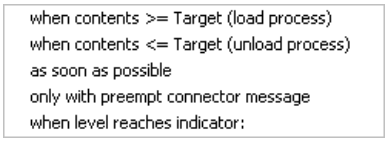
Arriving item is always filled (left) or is always emptied (right)

Item release conditions

The release conditions determine when an item is scheduled to leave the Interchange block; they are the same for both block modes. Releases can also be accomplished at any time using the Preempt connector.

Scheduled releases

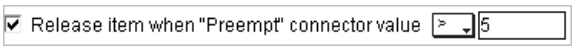
There are 5 options for defining when the item should be released:



Release options for Interchange block

- When contents >= Target. This option requires the item to be filled with a certain amount of flow prior to release. The Target amount is entered in the dialog.
- When contents <= Target. Requires the item's flow level to empty to a certain point prior to release. The Target amount is entered in the dialog.
- As soon as possible. Releases the item whenever there is downstream item capacity for the item, irrespective of the current flow level.
- Only with preempt connector message. Releases the item when a true value is received at the block's PE (preempt) value input connector.
- When level reaches indicator. Requires the flow contents to reach a certain level prior to release. Indicators (segments that indicate the level of contents) must first be entered on the block's Indicator tab for this option to be used.

Preemption



Preemption options for Interchange block

The scheduled release conditions can be superseded at any time by using the preempt connector. Whenever the preempt connector receives a value that corresponds to the preemption options selected in the Item/Flow dialog, seen above, it will trigger a preempt.

☞ To immediately dispose of an item after it releases flow, connect an Exit block (Item library) to the Interchange block's item output connector. To instead have an item present all the time, connect a Create block set to *Create items infinitely* to the block's item input connector.

Interchange modes

The Interchange block has two modes:

- Tank only exists while item is in it
- Tank is separate from item

These are illustrated below.

Tank only exists while item is in it

In this mode, the Interchange's capacity to handle flow is completely dependent upon the presence of an item. The arriving item can be thought of as a "tank" with a capacity to hold flow. This item/tank can move through the item-based blocks just like any other item would. However, once it enters an Interchange block, the item/tank can release flow directly into the block's outflow connection and/or accept flow directly from its inflow connection. In the absence of the item/tank, the Interchange block has no flow capacity.

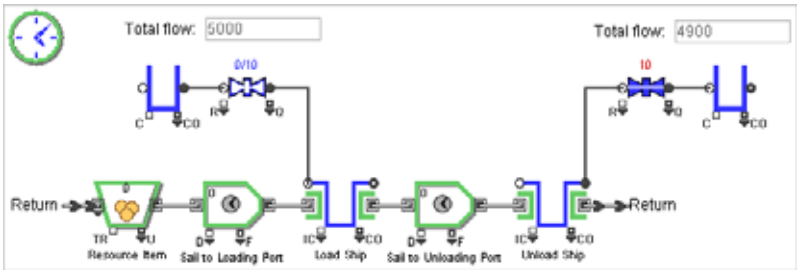
Two very important behaviors result from an item exiting the block when it is set to this mode:

- Once the conditions for item release have been met, the exiting item will always take with it any flow currently residing in the block.
- Until a successor item arrives, the Interchange's inflow and outflow will be blocked.

⚠ In this mode, the absence of an item eliminates not only the Interchange's capacity to hold flow but also its capacity to pass flow from an upstream source to a downstream sink. (This differs from the behavior of a Tank block, which passes flow through even if its capacity has been set to zero.)

Shipping model

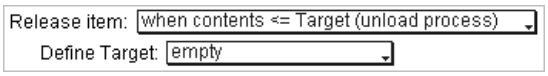
A typical example of how the Interchange block could be used in this mode is illustrated by the Shipping model. In this example, an empty ship (an item) arriving at a loading port (an



Shipping model

Interchange block) where it is filled with cargo according to a filling rate. Once full, the ship sails for a period of time (represented by an Activity block) until reaching the new destination port (another Interchange block). At this point the ship's cargo is unloaded according to the unloading rate.

To simulate the loading process, flow is piped from the Interchange block's inflow connector into the item/ship. Once filled, the item/ship exits the block, taking the flow with it. Conversely, once the item/ship arrives at the second Interchange block, flow is piped from it into the second Interchange block's outflow connector.

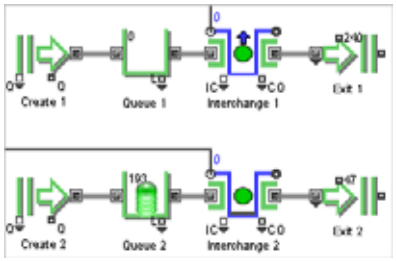


Release options for unloading

The item/ship travels with a *Quantity* attribute that has been set to a value of 1000. This attribute sets the ship's capacity. For the filling process, the Interchange releases the ship when it is full, that is, after 1,000 units of flow have been piped in. The second Interchange block releases the ship when it is empty, that is, after 1,000 units of flow have been piped out. At the end of the simulation run, the ship has not yet been released from the unloading dock because it still contains 100 units of flow.

Yogurt Production model

The Yogurt Production model, located in the folder \Examples\Turorial\Discrete Rate and discussed in the tutorial for the Rate Module Quick Start guide, is an example of using an Interchange block set to *Tank only exists while item is in it.*

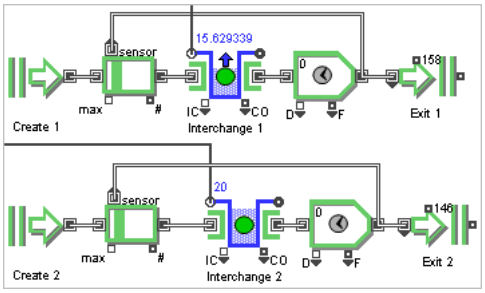


Yogurt Production palletization

In this model, empty item/pallets are generated randomly by a Create block (Item library). The arrival of an item/pallet causes the Interchange block to have flow capacity; the maximum capacity of 24 cartons is entered in the block's dialog. Once the maximum capacity is reached, the full pallet leaves the block. The Interchange block then has no capacity until another pallet arrives.

Yogurt Changeover model

This model is based on the Yogurt Production model from above. The Yogurt Changeover model, however, provides a period of time (the changeover) for an operator to remove each full pallet and replace it with an empty pallet, and there is an infinite supply of empty pallets.



Yogurt Changeover palletization

In this model, each Create block (Item library) can generate an infinite number of items so that empty pallets are available as required. Activity blocks (Item library) represent the two minute changeover time. Once a pallet has been released from an Activity, it notifies a Gate block (Item library) to open, pulling in a pallet from the Create block. This process causes the Interchange to not have a new pallet/item to fill until the changeover from the previous pallet is finished.

☞ The Create block's ability to *Create items infinitely* is specifically intended for this type of situation. It provides an infinite supply of items that are available on demand, creating items whenever there is a capacity for flow.

Tank is separate from item

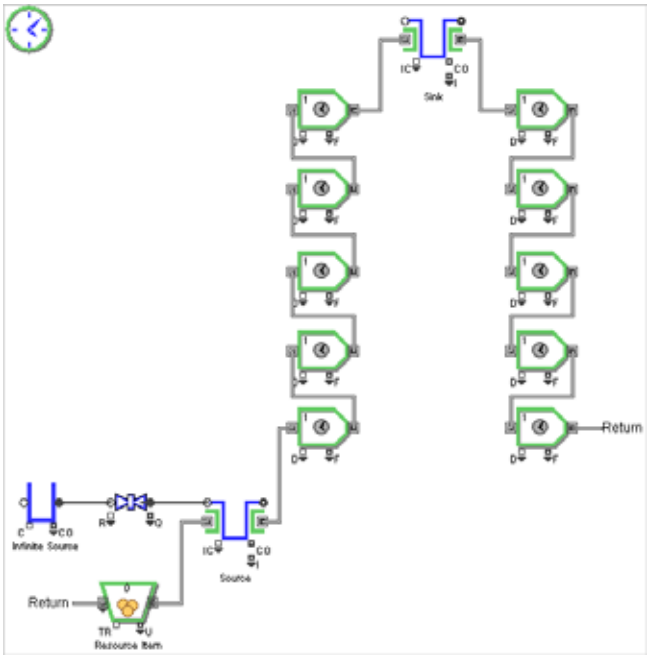
In this mode, the Interchange block's ability to handle flow is identical to the Tank block irrespective of the presence of an item. That is, the Interchange can be set to have an initial amount of flow, its capacity can be set through the dialog or through a connector, and maximum inflow/outflow rates can be defined.

The only difference lies in the Interchange block's ability to pipe flow into and out of items. When the Interchange block is set to *Tank is separate from item*, the block can receive and hold flow that has been provided by its inflow connector or the arrival of an item and it can release flow through its outflow connector or through the exiting of an item.

☞ When the Interchange block is in this mode, an item carrying flow releases its entire load instantaneously upon arrival. However, depending on other settings in the block, the item can also take flow with it upon exiting. If this is the case, the Interchange block's flow level is decremented instantaneously when the item leaves. Furthermore, an item whose load of flow exceeds the block's capacity will be blocked from entering the Interchange.

Bucket Elevator 1 model

The Bucket Elevator 1 model simulates a series of buckets (items) pulling flow out of a source (an Interchange block) located at a low elevation, transporting the water in a series of steps to an infinite sink located at a higher elevation (another Interchange block), and then returning empty through a series of steps to the source. Both Interchange blocks are set to *Tank is separate from item*. With this setting, both the source and the sink have the capacity to hold water even in the absence of an item.



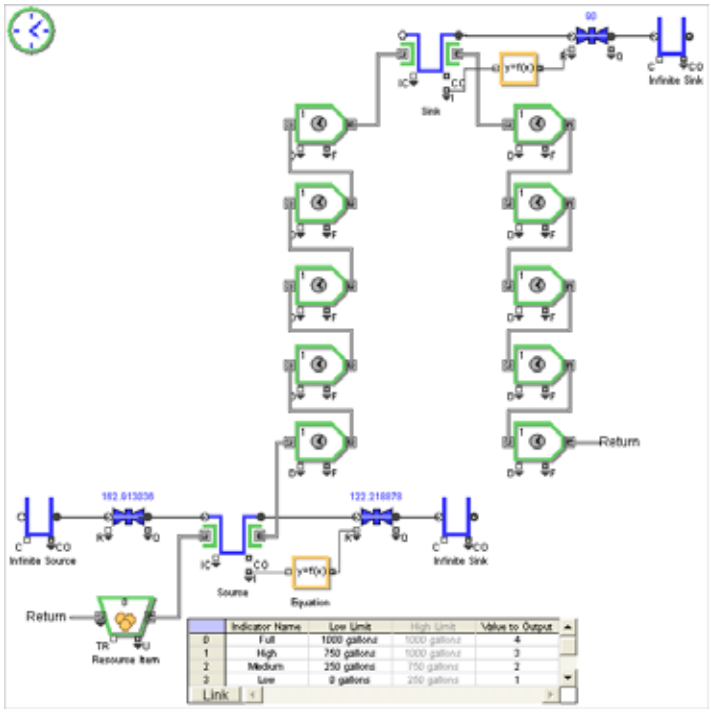
Bucket Elevator 1 model

This model is similar to a continuous loop of buckets drawing water from a well, emptying into a catch basin, and returning to the well. There are ten bucket/items and each bucket has a capacity of 100 gallons; this is defined in the Resource Item block by the attribute Capacity. There are also ten slots in this "pseudo-conveyor" – each represented by an Activity block that can hold one item/bucket. The delay at each slot is 1/10 the sum of all the delays, as determined by the items' Speed attribute.

Even if the source has less than 100 gallons, the buckets keep moving and grab as much water as possible. As each bucket reaches the top, its contents are released instantaneously into the sink, and the journey back down to the sink immediately starts. Running the simulation with animation on shows the buckets as they cycle from the well, up to the catch basin, and then back down again.

Bucket Elevator 2 model

The Bucket Elevator 2 model is similar to the Bucket Elevator 1 model, except there is some added complexity. In this model, the flow blocks at the bottom and top of the model represent streams of water. In the bottom stream, some of the water is removed by the buckets, but the rest continues on at a varying rate of flow. Likewise, after the buckets have unloaded their contents into the sink, water flows from the top stream at a varying rate.



Bucket Elevator 2 model

Both Interchange blocks have a capacity of 1,000. Based on settings in the Interchange block's Indicators tab, the level of water affects the constraining rate of the Valve that follows each of the blocks.

	Indicator Name	Low Limit	High Limit	Value to Output
0	Full	1000 gallons	1000 gallons	4
1	High	750 gallons	1000 gallons	3
2	Medium	250 gallons	750 gallons	2
3	Low	0 gallons	250 gallons	1

Link

Indicators for the source

Indicators are a method of reporting what category or range the current level of flow falls into; the table from the Source block's Indicators tab is shown to the right. An Equation block (Value library) looks at the Interchange block's I (indicator) output to determine the range the water level in the tank falls into. The block then adjusts the Valve's maximum rate depending on that information and an equation.

The upper and lower streams in this model have different equations; the upper stream bases the Valve's maximum rate on a constant while the lower stream bases it on a random distribution. In the lower stream for instance, if the Source tank's level is equal to or less than the Low range, the maximum rate is a Triangular distribution that is most likely 30 units. If the level falls within or above the High range, the most likely maximum rate for the Valve is 150; otherwise, the most likely rate is 100.

☞ For more information about indicators, see “Setting indicators” on page 430.

Converting between item and flow attributes

The Interchange block’s Flow Attributes tab provides a matrix for mapping between item and flow attribute names. The mapping table transfers the values of an item’s attributes to the specified flow attributes and vice versa. The block’s dialog provides options for how the mapping should affect entering and departing items. For an example, see the Interchange block in the Yogurt Production With Flavors Plus model, located at Examples\Tutorials\Discrete Rate.

Discrete Rate Modeling

Miscellaneous

Concepts that don’t easily fit into other chapters

This chapter covers:

- The precision of calculations
- Using bias to give preference to some component or portion of a model and how that affects effective rates
- Global and advanced options in the Executive
- Value connector abbreviations and meanings
- The different types of information that animation displays

Precision

An LP area is made up of one or more rate sections; it encompasses all the rate sections for which the Executive block has been notified that effective rates might change. The LP area has a linear program (LP) that is responsible for calculating an effective rate for each section contained within that area. (The LP area and LP calculations are discussed fully in "LP technology" on page 526.)

The maximum mathematical precision for an LP area is 12 digits. Because one LP can be responsible for calculating multiple effective rates for its rate sections, and because LP precision is limited to 12 digits, precision can become an issue not only for the individual effective rates but also for the effective rates calculated for the entire LP area. For example, if an LP area contains two rate sections where the first rate section's effective rate was 1,000,000 flow units per time unit (FPT), the effective rate for the second section could be no smaller than 0.0001 FPT.

☞ To preserve adequate precision for all rate sections, don't separate any two effective rates within an LP area by more than 12 digits of precision.

Biasing flow

The discrete rate architecture includes a feature called *bias* – a method for stating a preference that flow travel one route rather than another.

☞ Bias is only relevant when the flow is merged or diverged.

The advantages of the bias concept are that it:

1) Gives you a way to specify preferences for how flow circulates in one part of the model compared to other parts.
2) Provides flexibility in resolving conflicts for how flow should be distributed among competing branches.

Bias is present in a model whenever you use one or more of the following blocks:

- A Bias block in a model that contains Merge or Diverge blocks set to a non-fixed mode (discussed in "Merge and Diverge blocks" on page 512).
- A Merge or Diverge block (Distributional, Priority, or Sensing modes only)

Because bias can skew the way flow is distributed, it is taken into consideration by the global LP calculation and can thus have an effect on effective rates. (For specific information on how bias is used in the calculation of effective rates, see the advanced topic "LP technology" on page 526.)

☞ We suggest that you read the chapters "Rates, Constraints, and Movement" and "Merging, Diverging, and Routing Flow" before the Bias section.

Bias order

If a block in a model has bias, it has a *bias order* that indicates its ranking compared to all the other blocks with bias. Each biasing block is listed in order from the top (strongest) bias order to the lowest (weakest) bias order. The block at the top of the ranking list has a bias order greater than 0. Bias orders lower than the top have numbers higher than the top number; bias numbers that are Blank or less than or equal to 0 are ignored.

Since the bias order is used during the LP calculation of effective rates, changing the bias order often results in a different set of effective rates. It is therefore important to understand the concept of bias order and the influence it has on how effective rates are calculated.

☞ The bias of a Bias block is by definition stronger than the biasing effect of any Merge or Diverge block. So Bias blocks will always have higher bias orders than Merge and Diverge blocks.

Bias block

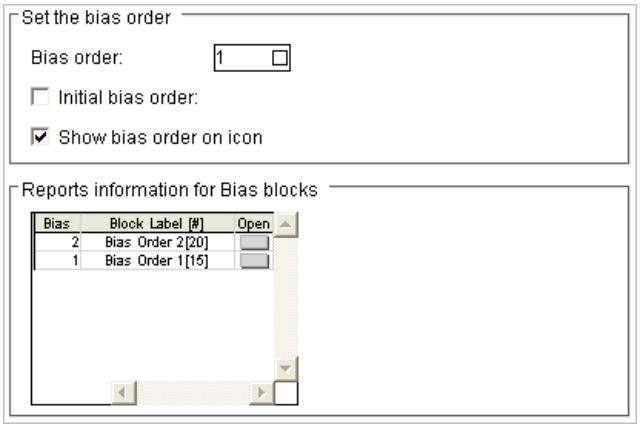
The Bias block allows you to specify a preference for where the flow should be directed. Wherever the Bias block is located in the model, it pulls in as much flow as possible. If a model has multiple Bias blocks, each has its own bias order.

⚠ If the model's Merge and Diverge blocks use non-fixed rules to obtain or distribute flow, there is some leeway in how flow can be biased and the Bias block is useful. If the model's Merge and Diverge blocks all use fixed rules, there is no possibility of biasing the flow with the Bias block. (Fixed and non-fixed rules are discussed in "Merge and Diverge blocks" on page 512.)

Dialog settings and bias order

The block's bias order can be set directly in its dialog or it can be modified dynamically through the B (bias) input connector or by linking the *Bias orde*r dialog parameter to an ExtendSim database. You can enter an initial bias order that has effect until the block gets a bias order value dynamically; you can also show the block's bias order on its icon.

The Bias dialog shown at the right is for one of two Bias blocks in a model. The block shown has the highest preference for flow, as indicated by the setting *Bias order: 1*.



Bias dialog

The dialog table reports information about each Bias block in the model, its bias order, block label or name, and block number. You can use the table's *Bias* column to change the bias order for any of the listed blocks.

Calculation of the effective rate

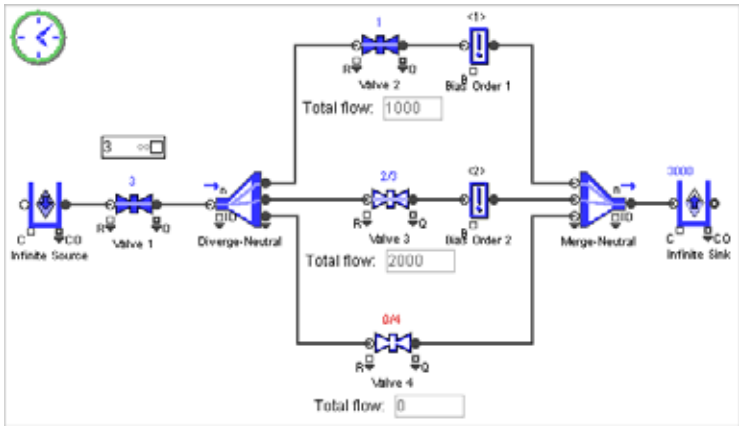
The preferences for flow defined by Bias blocks has an effect on the calculation of the effective rate. If there is more than one Bias block in the model, each block's preference is expressed in turn based on its bias order. As each Bias block takes its turn, it calculates the

maximum effective rate which could circulate at its location, without taking into consideration the preferences expressed by blocks with a lower bias order. When its maximum effective rate has been determined, that rate will be fixed for the succeeding calculations involving blocks with lower rankings.

- The bias order of Bias blocks can change dynamically during the simulation
- If multiple Bias blocks have identical bias orders, the way the flow is distributed between the effective rates cannot be predicted. The model will use one of the possible solutions.
- If a Bias block has a bias order that is Blank or is less than or equal to 0, the block does not express any preference.

Prioritize With Bias Blocks model

This model illustrates how Bias blocks can be used to indicate preferences for flow. It is similar to the Competing Requests for Flow model discussed on page 474. However, this model uses Bias blocks to indicate the preferred route for flow, rather than bias order settings in Merge and Diverge blocks set to Priority mode.



Prioritize With Bias Blocks model

In the Prioritize With Bias Blocks model, the Merge and Diverge blocks at set to Neutral mode and the Bias blocks indicate flow preferences. The results are identical to the Competing Requests for Flow model.

☞ This model is located in the folder \Examples\Discrete Rate\Merge and Diverge.

Merge and Diverge blocks

As shown in the “Mode table” on page 465, some Merge/Diverge modes use a fixed rule to obtain or distribute the flow. For other Merge/Diverge modes, flow rules are only invoked in specific situations depending on model conditions.

Because it influences the way flow is distributed, the bias concept only applies to Merge or Diverge blocks set to non-fixed rule modes: Distributional, Priority, or Sensing. To avoid confusion when Merge or Diverge blocks have competing requests for flow, blocks with these modes must specify a bias order.

Fixed rule modes

The Batch/Unbatch, Proportional, and Select modes all use a fixed flow rule to obtain or distribute flow. For example, the way flow is distributed between the branches for a Merge/Diverge in Proportional mode will follow the proportions set in the block’s dialog. No matter what happens in the rest of the model, the proportions will be respected.

The bias setting options in these block’s Model Settings tabs will be disabled.

☞ If the model contains Merge and Diverge blocks that are only set to the Batch/Unbatch, Proportional, or Select modes, bias has no impact on the effective rates.

Non-fixed rule modes

The Distributional, Neutral, Priority, and Sensing modes do not use a fixed rule to obtain or distribute flow. Instead, they provide a certain degree of freedom about where the flow can be directed.

A Merge or Diverge block in Priority mode, for example, impacts the flow as follows:

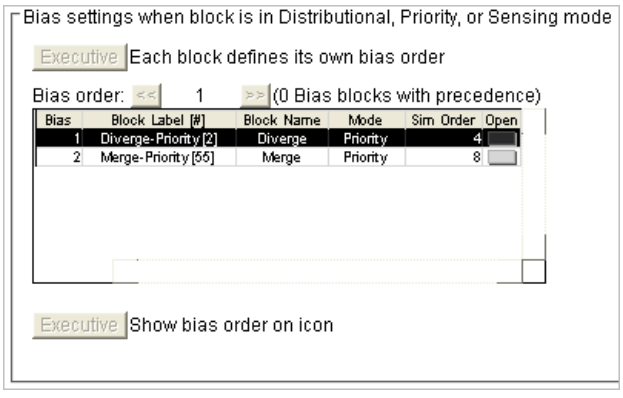
- Taking into consideration how much flow it can get, the block will do its best to direct as much flow as possible to its top priority branches.
- However, the block just expresses a "preference" for where to send the flow; model conditions determine how well those preferences can be achieved and the top priority branches may not actually get the most flow.

☞ If the model contains any Merge or Diverge blocks set to the Distributional, Priority, or Sensing modes, the bias order must be specified. (There is no bias order required for the Neutral mode.)

Setting a Merge or Diverge block's bias order

When set to the Distributional, Priority, or Sensing modes, the Merge and Diverge blocks must express a bias order. This is accomplished as follows:

1) By default, the Executive is set to *Bias order: defined by Simulation Order.* With this setting, the bias order for each Merge and Diverge block is automatically determined based on Simulation Order. Simulation Order is set in the command Run > Simulation Setup > Continuous tab; the default is Flow order. It would be unusual to change the simulation order from the default Flow order.
2) You can also directly enter a bias order for a biasing Merge or Diverge block. To do this, first change the Executive's default setting from *Bias order: defined by Simulation Order* to *Bias order: each block defines its own*. Then do one of the following:
   - In the block's Model Settings tab, use the two array buttons ("<<" and ">>") to change the block's bias order. This also changes the block's position in the tab's bias order table.
   - Or, in the Executive's Discrete Rate tab, select the row that contains the desired block, then use the << and >> arrows to change its position in the table. The bias order changes when the position of the block in the table changes.



Model Settings tab

Bias order table

The bias order table in a Merge or Diverge block's Model Settings tabs displays each biasing Merge and Diverge block, its bias order, block label or name, mode, and Simulation Order.

- If the blocks are set to *Bias order: defined by Simulation Order*, the table will be inactivated since bias order is determined automatically.
- If the blocks are set to *Bias order: each block defines its own*, the table is active only in Merge or Diverge blocks in the Distributional, Priority, and Sensing modes. In blocks with those modes, it can be used to change the blocks' bias order, as discussed above.

The Model Settings tab also reports the number of Bias blocks in the model; Bias blocks always have a higher bias order than any Merge or Diverge block.

Competing preferences

"Bias Order – resolving competing requests for flow" on page 473 discusses how competing preferences between Merge and Diverge blocks is resolved using Merge and Diverge blocks.

Global and advanced options in the Executive

The Executive block (Item library) oversees the global discrete rate system. It is responsible for calculating a model's effective rates, centralizing and coordinating the information from Rate library blocks as discussed in the advanced topic "LP technology" on page 526.

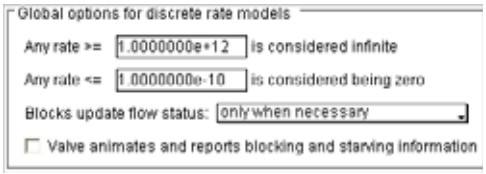
The Executive's Discrete Rate tab is used as a central location for setting options used throughout a discrete rate model. These options are divided into global and advanced options, as discussed below.

Global options

The Executive's global options are:

- Defining the "infinite" rate
- Defining a "zero" effective rate
- Setting options for how often blocks should update their flow status
- Choosing that the Valve animate and report blocking and starving information
- Managing flow units

The first three options are listed in a global options frame at the top of the tab (shown below); the fourth option is located at the bottom of the tab. They are all discussed in the following sections.



Global options in Executive

Infinite rate

The Discrete Rate tab specifies that a rate equal to or greater than some number is considered infinite; the default setting is that a rate ≥ 1e10 is considered infinite. This information is important when setting critical constraints and for the determination of the effective rate. It is discussed fully in "Infinite rate" on page 450.

Zero effective rate

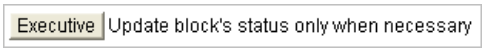
The Discrete Rate tab specifies that an effective rate less than or equal to some number is considered zero, resulting in no flow. The default setting for that number is 1e-10. It is strongly

suggested that you do not change the default setting unless you have an excellent reason to do so.

Update flow status

In the Rate library, some values change continuously over time. The frequency of updating those values can be customized by setting options for the Convey Flow, Diverge, Interchange, Merge, Tank, and Valve blocks. They define how often the following information is updated:

- The level of flow in residence blocks (Convey Flow, Interchange, and Tank)
- The amount of flow passing through the Convey Flow, Diverge, Interchange, Merge, Tank, and Valve blocks



Valve's default update setting

At a minimum, the discrete rate global system updates flow status only when it needs to. The updating options provide two additional opportunities to have calculations performed. The option that has been selected in the Executive is displayed in each block's Options tab (shown in the screenshot), along with any choices associated with that option. The choices in the popup menu are:

- Only when necessary. This default setting is computationally the most efficient option because flow status is updated only when needed by the system. With this setting, the information is updated:
  - When a block is determined to be part of an LP area
  - Whenever a block creates an internal event such as reaching a new indicator
  - When a block receives an active message at one of it value output connectors
- Each block defines how often. If this option is selected, each block's Options tab will give you the choice to check the option *Update animation and results at each event*. If that is checked, the calculation of the flow status will occur at each step for that block.
- Each block at each step. With this choice, every block will update at each step. This option has to be used cautiously because it is computationally demanding to update the information this frequently.

Valve animates and reports blocking and starving information

This option only affects the animation and reporting of Valve blocks. A Valve can be limiting, not-limiting, blocked, starved, or blocked and starved. By default, the Valve's Results tab and *S* (status) output connector only report whether the block is limiting (0) or not limiting (1). The Results tab also reports cumulative information regarding the percentage of time the block was limiting or not limiting.

When the global option is checked, the block's Results tab and its *S* output report all selected status information as a value:

- limiting only (0)
- starved only (1)
- blocked only (2)
- starved and blocked (3)
- and so forth.

There are many options that can be selected for reporting, as shown below.

	0	Advanced Status	(Limit,Rate,Starve,Block)	itput Val	Cumulative time
0		Limit only (r>0)	(Limit,rate>0, --- , ---)	0	0
1		Starved only (r>0)	(--- ,rate>0,Starve, ---)	1	0
2		Blocked only (r>0)	(--- ,rate>0, --- ,Block)	2	0
3		Starved and Blocked (r>0)	(--- ,rate>0,Starve,Block)	3	0
4		Limit and Starved (r>0)	(Limit,rate>0,Starve, ---)	4	0
5		Limit and Blocked (r>0)	(Limit,rate>0, --- ,Block)	5	0
6		Limit, starved, Blocked (r>0)	(Limit,rate>0,Starve,Block)	6	0
7		Limit only (r=0)	(Limit,rate=0, --- , ---)	7	0
8		Starved only (r=0)	(--- ,rate=0,Starve, ---)	8	0
9		Blocked only (r=0)	(--- ,rate=0, --- ,Block)	9	0
10		Starved and Blocked (r=0)	(--- ,rate=0,Starve,Block)	10	0
11		Limit and Starved (r=0)	(Limit,rate=0,Starve, ---)	11	0
12		Limit and Blocked (r=0)	(Limit,rate=0, --- ,Block)	12	0
13		Limit, starved, Blocked (r=0)	(Limit,rate=0,Starve,Block)	13	0

Link

☞ A Valve's complete status information is helpful during the early stages of model construction and for debugging purposes. However, it can slow the simulation, so by default the option is not checked.

The differentiations are animated on the Valve's icon as discussed in "Valve" on page 521.

Manage flow units for discrete rate models

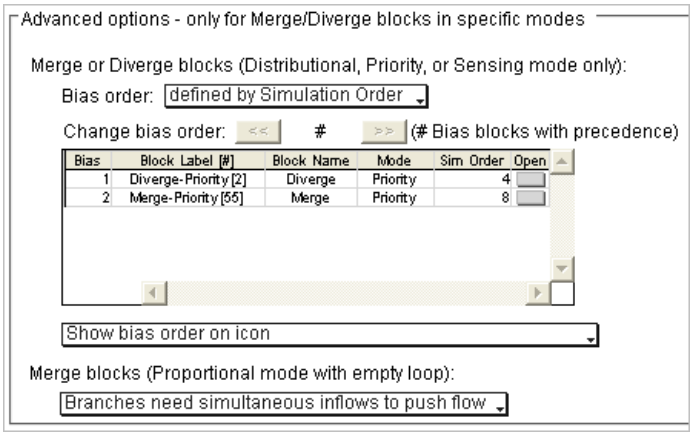
This section of the Discrete Rate tab provides a central location where flow units can be renamed or added to or deleted from a model. To delete or rename a flow unit, select it in the table and click the appropriate button. Flow units are discussed on page 441.

Advanced options

The advanced options in the Discrete Rate tab only apply to specific situations:

- Merge or Diverge blocks in the Distributional, Priority, or Sensing modes
- Merge blocks in Proportional mode when there is an empty loop

☞ Merge and Diverge modes are discussed in the chapter "Merging, Diverging, and Routing Flow".



Advanced options in Executive

Merge or Diverge blocks in Distributional, Priority, or Sensing modes

This first set of options determines how bias order is set for certain Merge and Diverge blocks and whether the bias order is displayed on the block's icon.

Bias order determination
For a Merge and Diverge block in the Distributional, Priority, or Sensing mode, the choices are that the block's bias order number is defined by:

- Simulation Order. This is the default; it causes the bias order to be automatically calculated by the simulation order of the blocks in the model.
- Each block. This option allows the user to customize the bias order of each Merge and Diverge block (Distributional, Priority, or Sensing mode only) in the model.

For detailed information about the requirement for a bias order and how it is set, see "Merge and Diverge blocks" on page 512.

Displaying the bias order
For Merge and Diverge blocks in the Distributional, Priority, or Sensing mode, a popup menu provides choices for displaying their bias order:

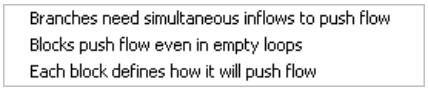
- Show bias order on icon
- Don't show bias order on icon
- Each block decides whether to show bias order on icon

If the bias order is displayed on a block's icon, it will be in the format <#>, where # is the bias order number. If the third option is chosen, a checkbox will appear in each Merge or Diverge block's Model Settings tab. Check the *Show bias order on icon* checkbox if you want the block to show the bias order number on its icon.

Merge blocks in Proportional mode

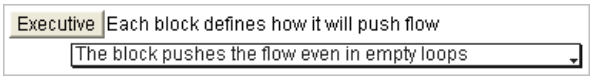
For Merge blocks in Proportional mode, a conflict can result if the block is part of an empty loop (can't get flow). If a Merge block's proportions are 1:2 for instance, what should happen if the top inflow branch is part of an empty loop and cannot get any flow from that loop?

The Executive provides three options for how empty loop situations should be resolved:



Executive options for empty loops

- Branches need simultaneous inflows to push flow. With this default setting, flow is stopped at all inflow branches if one or more of them are part of an empty loop.
- Blocks push flow even in empty loops. This choice allows the branches with flow to send it through. The result is that the branch that is part of the empty loop will then get some flow.
- Each block defines how it will push flow. This setting causes an additional popup menu to appear in the Model Settings tab of any Merge block in Proportional mode. The two choices in each Merge's dialog are:
  - The block pushes flow even in empty loops. This is the default setting; it allows branches with flow to send it through.



Model Settings tab of Merge block

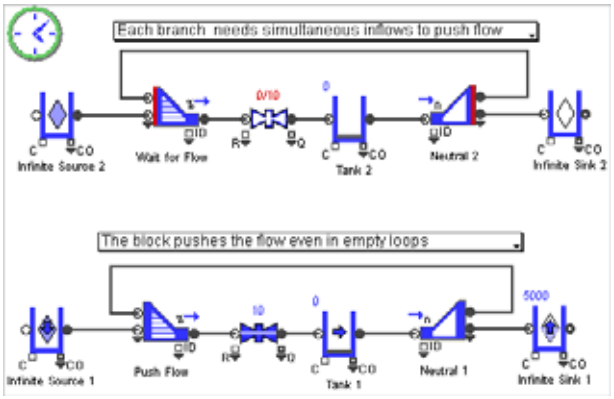
  - Each branch needs simultaneous inflows to push flow. With this choice, flow is stopped at all inflow branches if one or more of them cannot get any flow.

Which of the three empty loop options you choose depends on the behavior of the system you are simulating.

Merge Proportion Setting model
In this model the two flow streams have identical settings except for how the Merge blocks handle empty loops. So that each block can specify its own behavior, the Executive's setting for the section "Merge blocks (Proportional mode and empty loop)" is *Each block defines how it will push flow.*

The Merge block in the top stream (Wait for Flow) specifies that *Each branch needs simultaneous flow* while the Merge block in the bottom stream (Push Flow) is set to *Blocks push flow even in empty loops*. The flow is blocked at the Merge block in the top stream but flows through the Merge block in the bottom stream.

☞ This model is located in the folder \Examples\Discrete Rate\Merge and Diverge.



Merge Proportion Setting model

Common connectors on discrete rate blocks

Most Rate library blocks have inflow and outflow connectors and value input and output connectors. The Interchange block also has item input and output connectors.

Flow connectors pass information about the effective rate from one discrete rate block to another. Item connectors pass discrete items.

For value connectors, many discrete rate blocks use abbreviations or acronyms to indicate the connector's purpose. Some of these abbreviations represent more than one purpose and are context sensitive.

The following connector labels appear for value connectors on Rate library blocks:

Connector	In or Out?	Meaning
AL	Output	Accumulation length - for a Convey Flow block, the distance from its end to the accumulation point.
AQ	Output	Accumulated quantity - the amount of flow that has been accumulated along a Convey Flow block's accumulation length.
B	Input	Bias order number
C	Input	Capacity
CO	Output	Contents
D	Input	Delay
DR	Output	Potential downstream demand rate
factor	Input	Conversion factor in Change Units block

Connector	In or Out?	Meaning
G	Input	Quantity of flow (quantity goal) or duration of time (duration goal) for a Valve. Can also be used to start a new goal, depending on Valve's dialog setting.
G#	Output	Goal number
GD	Output	Goal duration
GS	Output	Goal status: 0 - no goal 1 - starting 2 - in progress 3 - ended 4 - interrupted
GQ	Output	Goal quantity
GO	Input	Activate a status update: 0 or 1 (Sensor and Merge/Diverge)
I	Output	Indicator (Convey Flow, Interchange, or Tank)
IC	Input	Item capacity
ICO	Input	Item contents
ID	Input	Inflow/outflow branch ID for Merge/Diverge in Select mode
IT	Input	Target value to release item
L	Output	Length of item line: 0 or 1
LE	Output	Level of contents
NB	Output	Number blocked: 0 or 1
PE	Input	Preempt item
PT	Output	Process time
Q	Output	Cumulative quantity
R	Input	Maximum rate (Note that the effective rate is reported by the flow input and output connectors.)
S (on Convey Flow)	Output	Status: 0 = empty 1 = intermediate 2 = full
S (on Inter-change or Tank)	Output	Status direction (Interchange): -1 = down 0 = stable 1 = up
S (on Valve)	Output	Status [if Valve only reports limiting or non-limiting]: 0 = limiting 1 = non-limiting

Connector	In or Out?	Meaning
S (on Sensor or Valve)	Output	Status [if Valve reports full status:] 0 = limiting (Valve) or unused (Sensor) 1 = starved 2 = blocked 3 = starved and blocked
S (0-n)	Output	Sensors - numbered from 0 to n (Convey Flow)
SP	Input	Speed parameter
SP	Output	Effective speed
SR	Output	Potential upstream supply rate
start	Input	Start hysteresis
stop	Input	Stop hysteresis
TL	Output	Cumulative time Valve was limiting, effective rate > 0
TU	Output	Cumulative time Valve was not limiting, effective rate > 0
TLO	Output	Cumulative time Valve was limiting, effective rate = 0
TUO	Output	Cumulative time Valve was not limiting, effective rate = 0

Animation

Rate library blocks can be animated during the simulation if Run > Show 2D Animation has been selected before a simulation run. For blocks with animation, the following information explains what each display means.

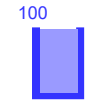
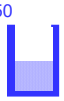
☞ The models illustrated in this section are located in the folder \Examples\Discrete Rate\Miscellaneous.

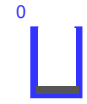
Tank

Tanks animate information about their levels and information about the direction of flow within the Tank.

Level information

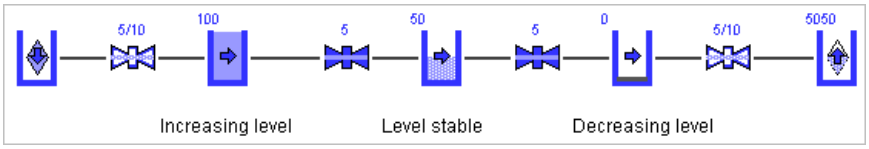
The table below shows animation of the Tank's level under different conditions.

Finite Capacity	Infinite Capacity	Specific Behaviors
 Full (contents 100, capacity 100)	 Full	 Overfull (contents 1,000, capacity 100)
 Some flow (contents 50, capacity 100)	 Some flow (10 units)	 Overfull (contents 1,000, no capacity)

Finite Capacity	Infinite Capacity	Specific Behaviors
		
Empty (contents 0, capacity 100)	Empty	No capacity and empty

Direction information

An arrow is animated on the icon to indicate the direction of change in the Tank's level. When there is no arrow, it means that the Tank has neither an inflow nor an outflow rate.



Interchange

The Interchange block displays the same animation behavior as a Tank, shown above. In addition, a ball appears in the middle of the block's icon when an item is present in the block. The ball is red if the item is ready to leave the block but is blocked downstream. Otherwise, the ball is green. If the option *Tank only exist while Item is in it* is chosen and there is no item in the block, the tank icon animates as white with cross-hatching.

			
Item filling (contents 50, capacity 100)	Item emptying (contents 50, capacity 100)	Item blocked downstream (contents 99, capacity 100)	No item present (Tank only exists while item is in it)

Valve

A Valve reports its maximum and effective rates when animation is turned on. If the option *Valve animates and reports blocking and starving information* is not selected in the Executive block's Discrete Rate tab (the default setting), a Valve will display only its limiting or non-limiting status. If that option is selected, it will also report its blocking and starving status. (See "Valve animates and reports blocking and starving information" on page 515 for full information.)

Displaying limiting and non-limiting status

By default, the Valve only animates information about its limiting or non-limiting status.

Non-Limiting	Limiting
 Effective rate 5, maximum rate 10	 Effective rate 5, maximum rate 5
 No flow, maximum rate 10	 No flow, maximum rate 0

Limiting is when the effective rate equals the maximum rate. In this case, the Valve is what limits the flow. When a Valve is *non-limiting*, the effective rate is less than the Valve's maximum rate. In that case, the Valve has no impact on the flow.

- When the effective rate is not the same as the constraining (maximum) rate, the two numbers are represented as *effective rate/constraining rate* above the icon. If the two rates are the same, only one number appears.
- If the Valve is limiting and the effective rate is greater than 0, the interior of the icon is plain blue. If the Valve is limiting to 0, the interior of the icon is plain red. If the Valve is not limiting, the interior of the icon is white.
- When the effective rate is greater than 0, a blue rectangle appears along the icon, and the rates are written in blue. When there is no flow, the blue rectangle does not appear on the icon and the rates are written in red.

Also displaying blocking and starving status

If the option *Valve animates and reports blocking and starving information* is selected in the Executive block's Discrete Rate tab, Valves will also animate their blocking and starving status information.

Limiting only	Blocked	Starved	Blocked and Starved
 Limiting	 Non-limiting, with flow	 Non-limiting, with flow	 Non-limiting, with flow
 Maximum rate 0	 Non-limiting, no flow	 Non-limiting, no flow	 Non-limiting, no flow
	 Limiting, with flow	 Limiting, with flow	 Limiting, blocked, and starved with flow

Limiting only	Blocked	Starved	Blocked and Starved
	 Limiting, no flow	 Limiting, no flow	 Limiting, blocked, and starved and no flow

If a Valve is blocked, it means that there are one or more blocks downstream which are limiting the flow through the Valve. If a Valve is starved, it means there are one or more upstream blocks that limit the flow to the Valve.

- If there is no flow, there will not be a horizontal line through the Valve. If there is flow, and the Valve is non-limiting, the horizontal line through the Valve is a patterned blue. The horizontal line is a plain blue color if there is flow.
- If a Valve is limiting, the central, vertical part of the Valve is plain blue (if the maximum rate is >0) or red (if the maximum rate = 0).
- If the right side of the Valve's icon is plain blue, the block is partially blocked by downstream blocks. If there is a red line on the right side of the icon, the flow is completely blocked downstream.
- If the left side of the Valve's icon is plain blue, the block is partially starved by upstream blocks. If there is a red line on the left side of the icon, the flow is completely starved upstream.

Goal and hysteresis animation

A Valve that uses a quantity goal to control its flow has a blue line below its icon. If the Valve controls its flow with a duration goal, the line is green. While the goal is On, a progression bar appears along the top of the blue or green goal line.

A Valve that uses hysteresis has a purple line at the bottom of its icon. While hysteresis is active, the purple line is thicker for all or part of its length.

 Quantity goal, about 75% completed

 Duration goal, about 75% completed

 Hysteresis is active

Sensor

The shape and color of the Sensor block's icon indicates if the flow is being blocked, starved, or both., and if there is flow or not.

Flow?	Blocked	Starved	Blocked and Starved
Yes >>>			

Flow?	Blocked	Starved	Blocked and Starved
No >>>			

Convey Flow

The animation of the Convey Flow block shows what mode it is in, the distribution of flow along its length, the position of the accumulation point, and other information.

Mode animation


Accumulate-maximum density


Accumulate-fill empty segments

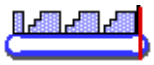

Non-accumulating

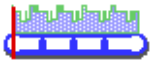
For further exploration, the Compare Convey Flow model compares the behavior of three Convey Flow blocks, each set to one of the possible modes, under different emptying rates. The model is located in the folder \Examples\Discrete Rate\Delaying Flow.

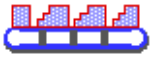
Distribution of flow and other information

A Convey Flow block animates the distribution of flow as follows:

- A blue shape along the top of its icon indicates flow distribution.
- A slowing of flow movement causes the blue shape to have a green border.
- The complete stopping of flow movement causes the blue shape to have a red border.
- If the block has reached its capacity, the shape is a solid blue rather than a dotted blue.
- If the block is empty, a black line will appear at the top of its icon, as seen above.
- The position of the accumulation point is indicated by a red vertical line that moves along the length of the icon. This is shown in the first, second, and fourth screenshots below and is discussed on page 494.


Distribution of flow


Slowing movement


Stopped


Capacity reached

Discrete Rate Modeling

Advanced Topics

Some additional information
for those of you who want to know more

The earlier chapters in the Discrete Rate module discussed important concepts you need to know to build discrete rate models. This chapter provides an overview of the way discrete rate calculations are made and describes the underlying functioning of the Rate library. This information is not necessary to build discrete rate models but will be of interest to advanced users.

It is highly recommended that you read the previous discrete rate chapters before this one, particularly the chapter “Rates, Constraints, and Movement”.

What this chapter covers

- The LP technology that has global oversight over a discrete rate model
  - How the LP area is determined
  - The sequence of events for the LP calculation
  - How bias affects the calculation
  - The types of information provided to the Executive
  - How a relational constraint is calculated
- The potential rates “upstream supply” and “downstream demand”
- Messaging in discrete rate models

LP technology

Linear programming problems involve the optimization of an objective function subject to a set of constraints. The purpose of solving a linear programming problem is to maximize or minimize selected variables in the objective function.

LP technology is the method ExtendSim uses to provide global oversight to maximize the movement of flow throughout a discrete rate system. The discrete rate architecture employs an integrated LP Solver DLL to solve a series of equations to optimize effective rates at each point of the simulation run.

The purpose of the LP calculation is to determine the maximum effective flow rates in the system given the constraints defined by block settings and the structure of the model. After all the rules for storage capacity and movement have been declared in the model, ExtendSim uses the Executive block’s LP calculations to cause as much flow as possible to move through the system. This calculation is handled automatically and internally.

Overview

In a discrete rate model, the Rate library blocks communicate with each other and with the Executive block. In turn, the Executive communicates with an integrated linear program (LP Solver). The Rate library blocks are dependent on each other, have an effect on one another, and are part of a global LP system that evaluates the entire model to calculate its effective flow rates.

- A *rate section* is a network of connected blocks, all possessing the same effective rate. Established at the beginning of the simulation run, rate sections do not change. (See “Rates, rate sections, and the LP area” on page 449 for more information.)
- An *LP area* is made up of one or more rate sections; the actual configuration can change during the simulation. A change in a block’s constraints during the simulation run initiates a propagation of messages through all the rate sections whose effective rates might change. This propagation defines the LP area at that point in time.

- Each rate section within the LP area contributes a part of an *LP equation* for a recalculation. The purpose of the recalculation is to determine the maximum effective flow rates in the system, given the constraints defined by block settings and the structure of the model. The result is the set of effective rates for each section in the LP area at that point in the simulation. The system is optimized such that only the rate sections in the LP area are recalculated; all the other effective rates in the system don't need to recalculate at that moment and won't.
- Among other things, the LP calculations take into consideration each block's:
  - Critical constraints, which place an upper bound on rate sections connected to that block.
  - Relational constraints, which define the way rate sections are related to each other.
  - Bias, which is a block's preference that flow travel one route rather than another.

The LP area

The *LP area* is made up of one or more rate sections linked together by the fact that their effective rates could change at that point in the simulation – a change in the effective rates in one section might impact effective rates in the other sections. The rates, constraints, and biases within the LP area are used by the LP Solver to calculate the optimal set of effective rates for the rate sections contained within that area.

The effective rate of one section can affect the effective rate of another section through relational constraints. When an event occurs that causes a rate section's effective rate to be reevaluated, blocks propagate "rate block flow" messages (see "Block messages" on page 537) throughout the model to determine which other rate sections might be impacted by the new event. The affected rate sections constitute the LP area and rate sections outside of the LP area are not included in the recalculation, reducing redundant computations.

The boundaries of the LP area are determined through the propagation of messages between Rate library blocks. A change in a block's constraints during the simulation run causes the block to notify the Executive and send messages which propagate through all the rate sections whose effective rates might change as a result. Whether the block is the originator of the recalculation request or receives a propagation message:

- The block declares which of its connected rate sections are in the LP area. The propagation process then creates a global list of rate sections to include in the current LP area.
- If the block provides a relational constraint connection between two or more rate sections, the effective rates connected to that block are dependent on one another. The block then continues the message propagation to all the dependent block(s), who propagate the message to their dependent blocks, and so forth.

The change in the originating block will definitely cause a recalculation of the effective rates for all directly connected rate sections, and (depending on relational constraints between the sections) might cause a recalculation of the effective rates for other rate sections.

☞ The boundaries of the LP area will change dynamically during the simulation depending on which effective rates are involved in the recalculation and which relational constraint dependencies occur between rate sections.

The sequence of events

A change in a block's constraints during the simulation run initiates a reevaluation of all the effective rates for the LP area at that point in time. The block's constraint change will definitely cause a recalculation of the effective rates for all directly connected rate sections, and (depending on relational constraints between the sections) might cause a recalculation of the effective rates for other rate sections. The sequence is:

1) A block's constraint changes.
   - If a block's status changes it can affect its effective inflow and/or outflow rates. For example, the effective inflow rate for a finite Tank block that is filling up is greater than its effective outflow rate. When the Tank becomes full, it creates an event because its effective inflow rate can no longer exceed its effective outflow rate.
   - When a block reacts to new parameters, the effective inflow and/or outflow rates must be reevaluated. For example, a Merge block in Select mode might choose its top inflow branch at the start of the simulation. If the block subsequently receives an order to select its bottom inflow branch, its effective rates have to be recalculated.
2) The block posts a zero-time event requesting a reevaluation of effective rates.
3) The LP area is determined based on the propagation of block messages.
   - Starting with the originating block. messages are propagated through the model to all the blocks that might be affected by the change. This propagation of messages defines the boundaries of the LP area.
   - The LP area encompasses all the rate sections with effective rates that might change during the calculation.
   - For a complete description of the LP area and how it is determined, see page 527.
4) As the LP area is determined, the blocks update their status.
   - Each block that is part of the LP area updates the amount of flow which has passed through it since the last update.
   - If the block is a residence block (Convey Flow, Interchange, or Tank), the amount of flow the block is holding is also updated.
   - Note: The frequency of status updates outside of an LP calculation is set in the Executive. See "Update flow status" on page 515 for information.
5) Blocks report to the Executive how they impact effective rates.
   - Each block in the LP area declares the flow rules (critical and relational constraints) that it applies to the rate section it is connected to.
   - The block's bias order is stored in a list. Each block declares its bias order (if any) and provides coefficient information to the Executive. The coefficients allow the Executive to build an objective function that considers the effect of the bias.
   - See "Types of information provided to the Executive" on page 529.
6) The Executive determines an objective function.
   - The objective is to maximize the flow rate for each rate section subject to the constraints defined by the blocks' flow rules.

- Each decision variable in the objective function is the effective rate of a section in the LP area. For instance, the objective could be "Maximum effective rate = 1*ER1+ 1*ER2 + 1*ER3…", where ERn is the effective rate for a rate section in the LP area.
- If there is no bias, or all the intermediate calculations related to bias order have been completed, the coefficient of each variable will be 1. If there is bias, the coefficient could be other than 1.

7) The Executive communicates with the LP Solver.
   - The Executive gives the LP Solver the objective function and the information from the blocks and rate sections within the LP area.
   - Critical constraints put an upper bound on some of the effective rates (the decision variables). Relational constraints provide a link between the effective rates of different rate sections and must be taken into consideration during the calculation. For instance, if effective rate X is less than or equal to effective rate Y, the objective function must conform to that information.
8) The Solver performs a calculation.
   - The Solver calculates an optimized set of effective rates for the LP area.
   - The result is an intermediate LP calculation (if there is bias) or the final LP calculation (if there is no bias).
   - For full details, see "The LP calculation" on page 532.
9) Steps 6-8 are repeated, if necessary.
   - If the blocks in the LP area have bias, the Executive must determine intermediate objective functions and the Solver must perform intermediate calculations. The number of recalculations is equal to the number of blocks with bias, plus 1.
   - For each recalculation, the list of critical and relational constraints is changed to include the constraints caused by the particular bias order being considered.
   - See "Bias information" on page 530.
10) The rate sections are notified.
   - The Executive sends "Executive block flow" messages to the head of the rate sections to update to the new effective rates.
   - The blocks receiving this information update according to their dialogs and value connectors. They then post new events if necessary.

☞ Since the purpose of the recalculation is to maximize the entire set of effective rates, some rates will change while others might not.

Types of information provided to the Executive

When effective rates need to be recalculated, Rate library blocks provide critical and relational constraint and bias information to the Executive.

To learn which blocks contribute which types of information, see "Table summarizing constraint and bias information" on page 531.

Flow rules

Flow rules consist of critical and relational constraints.

- If a rate section has a critical constraint, it places an upper limit to the effective rate within that section. Since effective rates are decision variables in the LP's objective function calculation, critical constraints place an upper bound on some of the equation's variables.
- Relational constraints describe the dependencies between different rate sections. In some blocks, relational constraints can vary depending on the state sensitivity of the block; in others they are permanently active.
    - An example of a *state sensitive* relational constraint can be found in a Tank block. (Tanks are always within two rate sections; the input side and the output side define the sections.) As long as a Tank is empty, its relational constraint is defined as "effective outflow rate is less than or equal to effective inflow rate". Once the Tank becomes "not empty", the relational constraint doesn't apply.
    - An example of a *permanent* relational constraint can be found in a Change Units block where a conversion factor defines the relationship of the inflow effective rate to the outflow effective rate. If the factor varies over time, the relational constraint may also vary. But the dependency between the inflow and outflow effective rates is active for each calculation which includes the two rate sections.

Bias information

When the LP area is created, the Executive ranks all Bias blocks and any Merge or Diverge blocks with a bias order in a list. The Executive considers the top bias from that list as part of the objective function and instructs the Solver to perform an intermediate LP calculation using that function and the current critical and relational constraints. Then the Executive takes the next bias order into consideration, and so forth.

Since bias affects critical and relational constraints, each succeeding bias order means a new objective function will be determined and a new set of critical and relational constraints will be added to the previous ones. The results from the previous LP are used as inputs to the next LP. This results in multiple intermediate LP calculations – one for each bias order in the list – until the final result.

Most blocks with bias order supply:

- Flow rules (critical and relational constraints) which apply to all the LP calculations. (This information is not supplied by a Bias block.)
- A set of coefficients that the Executive will use to build the objective function corresponding to this bias order. Depending on the bias information received from the blocks in the LP area, the intermediate objective function can include coefficients that are other than 1. (A coefficient of 0, for instance, indicates that a particular effective rate does not need to be maximized; it causes that effective rate to not be directly affected by the maximization.)
- Flow rules for that bias order which use the results of the intermediate calculation. (After the intermediate calculation, the results of the calculation are used to add new flow rules to the succeeding calculations.)

Some additional situations that enter into the calculation include:

- If a Bias block has a bias order that is Blank or is less than or equal to 0, the block does not express any preference.

- If multiple Bias blocks have identical bias orders, the effective rates for these blocks cannot be predicted. The model will use one of the possible solutions.
- Merge and Diverge blocks with bias order are always lower on the bias list than any Bias block.

Table summarizing constraint and bias information

The following table summarizes the types of bias and constraint information (if any) Rate library blocks supply to the Executive for the calculation of effective rates:

Block	Mode	Critical Constraint	Relational: Permanent	Relational: State Sensitive	Bias Order
Bias		No	Yes	No	Yes
Catch Flow		No	No	No	No
Change Units		No	Yes	No	No
Convey Flow		Yes	No	Yes	No
Diverge	Neutral, Proportional, Select, Unbatch	Depends	Yes	No	No
Diverge	Distributional, Priority, Supply Sensing	Depends	Yes	No	Yes
Interchange		Yes	No	Yes	No
Merge	Batch, Neutral, Proportional, Select	Depends	Yes	No	No
Merge	Distributional, Demand Sensing, Priority	Depends	Yes	No	Yes
Sensor		No	No	No	No
Tank		Yes	No	Yes	No
Throw Flow		No	No	No	No
Valve		Yes	No	No	No

☞ Diverge and Merge blocks can imply a critical constraint depending on the branch parameter and the mode.

The relational constraint calculation

The table that follows describes how the relational constraint is calculated for blocks that provide either permanent or state sensitive relational constraints.

For the purposes of this table:

- Xin is the effective inflow rate of a block and Xout is its effective outflow rate.
- Index i describes one of the inflow branches (for a Merge) and one of the outflow branches (for a Diverge).
- The number of branches is n.

Block	Calculation
Change Units	The permanent boundary is Xin = factor*Xout

Block	Calculation
Convey Flow	If the block is in an accumulating situation (outflow rate is blocked or partially blocked downstream) and doesn't have any more capacity for accumulation, a state sensitive boundary applies: Xin<=Xout
Diverge	Select. A permanent relational constraint applies between the inflow effective rate and the selected outflow effective rate: Xin=Xout selected Proportional. A set of permanent relational constraints applies to insure the proportions: Xout_i = factor_i*Xin (i: 0=>n-1) Batch/Unbatch. Permanent relational constraints: Xout_i = Xin (i: 0=>n-1) Neutral, Priority, Distributional and Supply Sensing. Permanent relational constraint Xin = Xout_1+…… Xout_n (Other calculations are beyond the scope of this document.)
Interchange	As long as the tank is full, Xin<=Xout. As long as the tank is empty, Xout<=Xin (State sensitive relational constraint)
Merge	Same as the Diverge with Xout and Xin reversed
Tank	See Interchange

The LP calculation

The Executive block maximizes an objective function composed of the set of effective rates for the LP area. If there is no block with bias order involved in the LP area, the Executive block maximizes the sum of all the effective rates in the LP area – in this case, only one LP calculation is necessary.

If the blocks in the LP area have bias, the Executive must determine multiple intermediate objective functions and the Solver must perform multiple intermediate calculations. The number of recalculations is equal to the number of blocks with bias plus 1.

When the LP area involves blocks with bias order, the calculations are made in cascading order. For each bias order, an intermediate calculation is made with an objective function depending on the type of block that is being evaluated:

- Bias block. The function to maximize is the sum of the effective rates attached to the Bias blocks with an identical bias order. When the calculation is made, the function is used as a new rule for the succeeding intermediate calculations. The calculation is: sum effective rates within the bias order>= result of the maximized function.
- Merge/Diverge in Priority mode. The function to be maximized contains the effective rates from the variable inflow and/or outflow branches of the block. The lower the priority of the branch, the higher the coefficient associated with the effective rate. The calculation is: sum p*Xp (for p: 1=>n with 1 top priority and n lowest priority). When the calculation is finished, the objective function is used as a new rule for the next intermediate calculations (sum p*Xp >= result of the maximized function).
- Merge/Diverge in Batch/Unbatch, Distributional, Neutral, Proportional, Select, or Sensing mode. This calculation is beyond the scope of this document.

When all the intermediate LP calculations have been made, the last LP calculation maximizes the sum of all the effective rates for the LP area.

Upstream supply and downstream demand

The effective rate is not the only result a rate-based model can provide. The *potential upstream supply rate* and the *potential downstream demand rate* can also be useful in special situations – the rates determine the branch proportions for Merge and Diverge blocks in Sensing mode (as discussed in "Sensing mode" on page 471) and the Sensor block can be used to report the potential rates for making model decisions.

☞ This is an advanced topic because the concept is complex and there is an elevated potential for error, as discussed in the section "Cautions when using potential rates", below. Careful model verification and validation, and an advanced knowledge of the ExtendSim LP technology, are required to avoid unexpected results.

Definition

The *potential upstream supply rate* is the theoretical rate at which an upstream source could provide flow to the beginning of a rate section if there weren't any downstream limitations on flow movement (downstream capacity is infinite). For instance, for a not-full Tank at the beginning of a rate section, the upstream supply rate would equal the Tank's effective inflow rate. A not-full Tank does not limit the inflow rate it can receive. In this case, the effective inflow rate is also the potential upstream supply rate.

The *potential downstream demand rate* is the theoretical rate at which a downstream section of the model could receive flow from the end of an upstream rate section if there were an unending upstream supply of flow (upstream source is infinite). For instance, for a not-empty Tank at the end of a rate section, the potential downstream demand rate would be equal to the Tank's effective outflow rate. A not-empty Tank does not limit the outflow rate it can provide. In this case, the effective outflow rate is also the potential downstream demand rate.

Upstream supply and downstream demand could potentially be infinite. For example, the upstream supply rate right after a Tank (if the tank doesn't declare any constraint on its outflow rate) is infinite.

☞ When flow movement is from left to right, the information to calculate the upstream supply rate is propagated from left to right while downstream demand information is propagated from right to left.

Requirements for the supply/demand calculation

For a model's Executive block to calculate a potential supply or demand rate, at least one of the following blocks must be part of the LP area:

- Sensor block. The only purpose of the Sensor block is to display the potential rates wherever it is located in the model. The Sensor block only provides information; it has no direct impact on the calculation of effective rates.
- Diverge block in Demand Sensing mode. Proportions for the outflow branches are calculated as a function of the potential downstream demand. For example, the downstream demand placed on an outflow branch becomes the proportion for that outflow branch.
- Merge block in Supply Sensing mode. The block uses the available upstream supply rate to define the Supply Sensing proportions for each inflow branch.

Cautions when using potential rates

We strongly recommend exercising caution when using the Merge/Diverge blocks in Sensing mode, or when relying on a potential rate reported by the Sensor block, because:

- Getting information about these rates slows down the simulation. The Executive has to make a lot more LP calculations to extract the potential rate information.
- In some cases the potential rates as reported by the Sensor block are not accurate. Depending on how flow is diverged and merged, the potential supply or demand rates could be aggregated even if they should not be. This is illustrated in the "Supply & Demand Warning model" described below.
- Merge/Diverge blocks in Distributional, Neutral, and Priority modes are not always compatible with Merge/Diverge blocks in Sensing mode. Consequently, models with blocks that mix Sensing mode with Distributional, Neutral, or Priority modes are prone to error.
- A Merge or Diverge block in Sensing mode has a bias order. Depending on the block's location in the Executive's list of bias orders:
  - The effective rates might be different.
  - The block might not follow the proportions specified by the Sensing mode.

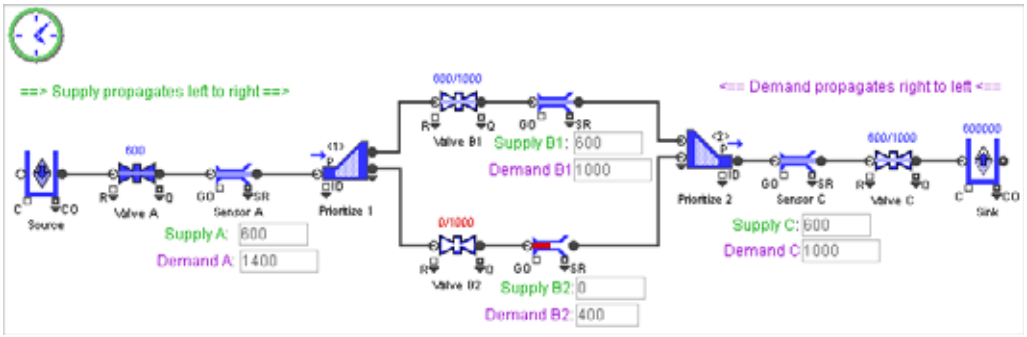
  For instance, the effective rates are different in the "Combine Priority Sensing model" and the "Combine Sensing Priority model", and neither of those models follow the Sensing rule.

Issues when relying on the Sensor to report potential rates

Based on how a model is configured, and the mode selected in a Merge or Diverge block, a Sensor block could report erroneous information.

Supply & Demand Warning model

In this model, the Sensor blocks correctly report the upstream supply. But Sensor A reports an invalid downstream demand of 1400 gallons/minute. The actual downstream demand is 1000 gallons/minute, reported by Sensor C. This is determined by verifying the demand results, compared to what was expected, from right to left.



Supply & Demand Warning model

Evaluating the model from right to left (for the downstream demand rate):

- Sensor C reports a demand (Demand C) of 1000 gallons/minute because Valve C limits flow to 1000 gallons/minute. The Sink at the end of the line doesn't limit the inflow rate at all.
- Demand B1 is 1000 gallons/minute because the Prioritize 2 block has its top priority at branch B1. (Verification: If a Tank with flow is present at branch B1, Prioritize 2 and Valve C would accept 1000 gallons/minute coming from branch B1.)

- Demand B2 is 400 gallons/minute because the effective rate on branch B1 is 600 and Demand C is 1000: Demand B2=Demand C-effective rate B1. (Verification: If a Tank with flow is present at branch B2, it could provide 400 gallons/minute because the total Prioritize 2 can accept is 1000 and the block already takes 600 from the top priority branch.)
- Sensor A reports a demand of 1400 gallons/minute because Demand B1 is 1000 and Demand B2 is 400. However, this is incorrect. (Verification: If a Tank with flow is placed right before Prioritize 1, the outflow effective rate would be 1000 gallons/minute and not 1400 gallons/minutes. The erroneous result comes from the fact that the association of "correct" local rules doesn't guarantee a "correct" global result.

☞ By understanding flow rules and how they can affect the global result, you can avoid this problem. The most important step is verifying this model from left to right for the supply and from right to left for the demand. Two solutions are 1) having the Prioritize 2 block be in Neutral mode, and 2) placing a Valve with a maximum rate of 1000 between Sensor A and Prioritize 1.

Mixing Merge/Diverge block modes

Situations where Merge or Diverge blocks in Sensing mode are mixed with Diverge or Merge blocks in Distributional, Neutral, or Priority modes within an LP area should be avoided as they are prone to give inaccurate results.

In these types of mixed-mode situations, the LP area is going to be recalculated multiple times to provide the effective rates and the upstream supply and downstream demand rates. All the blocks in the LP area must provide one set of constraints to the Executive so it can solve the effective rates and a second set of constraints so the Executive can solve the supply and demand rates. This causes repeated intermediate LP calculations, the results of which are affected by constraints which are applied for the supply and demand rates. This may yield inaccurate results.

Unexpected effects of bias order

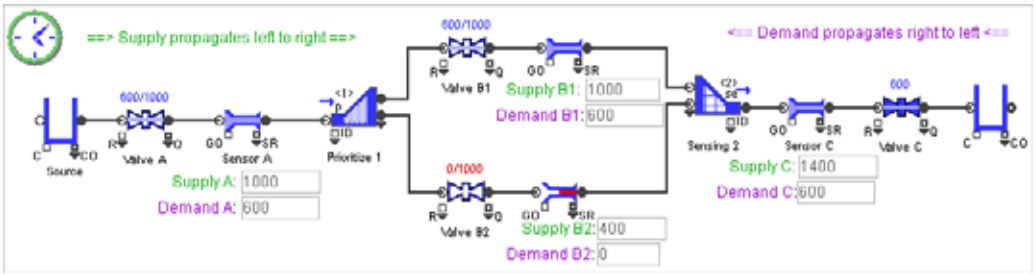
As discussed on page 512, the Distributional, Priority, and Sensing modes require bias order to be defined. This affects their effective rates. However, the only mode whose proportions are influenced by the bias ranking is the Sensing mode.

When effective rates within an LP area have to be recalculated, the Executive makes a list of blocks ranked from the top bias order to the bottom. Depending on a Merge or Diverge block's ranking in the Executive's list, the effective rates might be different. If the block is in Sensing mode, the proportions will also be influenced. The two models that follow illustrate these issues.

Combine Priority Sensing model

In this model, a Diverge block (Prioritize 1) is set to Priority mode and a Merge block (Sensing 2) is set to Supply Sensing mode. The blocks' Model Settings tabs are set to *Each block*

defines its own bias order. The table in the tabs indicates Prioritize 1 has the top bias order and Sensing 2 has the lower bias order.



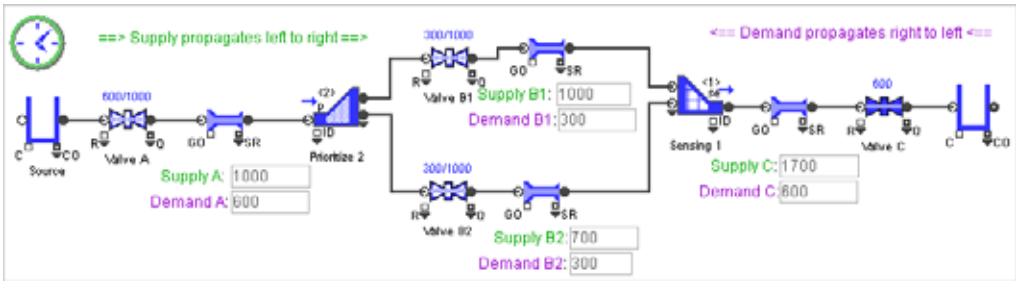
Combine Priority Sensing model

The supply in branch B1 is 1,000 while the supply in branch B2 is 400. However, the Sensing 2 block doesn't distribute flow following the expected supply proportion.

- Because the Prioritize 1 block has the top bias order, the Executive gives priority to that block in choosing how to distribute the flow between the branches B1 and B2.
- At the first intermediate result of the LP calculation, branch B1 gets an effective rate of 600 gallons/minute and branch B2 gets 0 gallons/minute.
- Because this decision has been made for branches B1 and B2, the Sensing 2 block with its lower bias order cannot control the proportion of flow it will get through branches B1 and B2. For example, if the supply from B1 is 1000 gallons/minute and the supply from B2 is 400 gallons/minute, the proportion between the effective rates for the Sensing 2 block's inflow is not 71% B1 (1000/1400) and 29% B2 (400/1400) but rather 100% B1 and 0% B2.

Combine Sensing Priority model

Like the preceding model, the Diverge block (Prioritize 2) in this model is set to Priority mode and the Merge block (Sensing 1) is set to Supply Sensing mode and the Model Settings tabs for the blocks are set to *Each block defines its own bias order*. However, in this model the table indicates that Prioritize 2 has a lower bias order than Sensing 1.



Combine Sensing Priority model

- The Sensing 1 block has the top bias order, so the Executive gives priority to that block to choose how the flow it receives should be distributed between branches B1 and B2.
- At this point of the LP calculation, the Prioritize 2 block gets a supply of 1000 gallons/minute and has not decided how to distribute the flow. With that limited information, the poten-

tial upstream rate is calculated as 1000 gallons/minute for Supply B1 and 1000 gallons/minute for Supply B2. Therefore, Sensing 1 decides to get 50% of the flow from inflow branch B1 and the other 50% of the flow from branch B2.

- Because the decision has been made for branches B1 and B2, Prioritize 2 with its lower bias order can no longer control the distribution of flow between branches B1 and B2.

Messaging in discrete rate models

As discussed in "How ExtendSim passes messages in models" on page 101, the ExtendSim architecture allows application messages to be sent from ExtendSim to a model's blocks and block messages to be passed between blocks.

Discrete rate models use the same application messages as do continuous and discrete event models. The block messages sent between Rate library blocks are discussed below.

Block messages

Discrete rate blocks have a sophisticated and complex messaging structure for communicating with each other and with blocks from the Value and Item libraries. They can be categorized as:

- Event
- Value connector
- Item connector
- Flow connector
- Rate block flow
- Executive block flow

When a block initiates a recalculation of the set of effective rates, a succession of messages and calculations are also initiated. Understanding how block messages work can prevent redundant messages from being created. At the very least redundant messages will cause runtimes to be longer than need be. At the very worst, redundant rate calculations could introduce bugs into a model when effective rates are temporarily calculated using one or more out-of-date parameter values. For instance, see "Limiting the number of recalculations" on page 475.

☞ By limiting the number of times the set of rates have to be recalculated, the operational efficiency of the model is maximized. For example, see "Limiting the number of recalculations" on page 475.

Event messages

Event messages communicate between the Executive block and Rate library blocks. In a discrete rate model, the simulation clock advances from one event to another. Each time the clock advances, the Executive block sends event messages to the blocks that have associated themselves with that event. There are two types of events: future and current.

- A *future* event message occurs when the simulation clock reaches a time posted by a block. For instance, when the level in a Tank increases, it posts a future event to the Executive corresponding to the Tank's "full time". Once the simulation clock has advanced to this future event, the Executive sends an event message to the Tank, alerting it that it is full.
- A *current* event message occurs when a block wants to be activated before the simulation clock advances, but after it has completed its response to another message. For instance, instead of a Valve immediately calculating a new effective rate when its constraining rate changes, it will post a current event to the Executive letting it know that it will have to recal-

culate at a certain time. This gives all the other blocks in the model the opportunity to update before the recalculation occurs.

☞ In discrete rate models, blocks from the Value library typically neither post events to the Executive nor receive event messages from the Executive. This has important ramifications on the behavior of continuous blocks in discrete rate models.

Value connector messages

Blocks in a discrete rate model send value connector messages either because a new number is needed by an input connector or because the value of an output connector has changed. These messages request updated information for the input connectors or notify connected blocks that the output value has changed. For example, if a Valve block's *R* value input connector is connected and the Valve receives a new value at *R*, the constraining rate on the flow changes. This will cause a recalculation of the set of parameters in the model.

These messages work the same in discrete rate models as in discrete event models. They are discussed fully at "Value input and output connector messages" on page 419.

Item connector messages

Discrete rate models often have portions that are item-based, using blocks from the Item library. The Rate library's Interchange block also has item connectors; it provides a mechanism for interacting with item-based blocks in a discrete rate model.

Item connector messages (primarily *wants*, *needs*, and *rejects*) use a conversation of messages to propel items from one item-based block to another through the model. The item connector messages work the same in discrete rate models as in discrete event models. They are discussed fully at "Item connector messages" on page 421.

Flow connector messages

Flow connectors provide the value of the effective inflow and outflow rates. Flow connector messages cause the effective inflow/outflow rate to be updated for all connected blocks each time the LP calculation determines that the effective rates have changed.

Rate block flow messages

As soon as a block from the Rate library receives a message, if it determines that the effective inflow and outflow rates might change, it initiates a propagation of these messages. This propagation of messages is used to define the LP area – the area of the model which could be impacted by the originating block's change.

Executive block flow messages

When a calculation of a set of effective rates for the LP area has been initiated, the Executive block calculates the new set of effective rates and sends messages to the blocks in order to propagate the results. These message update all the blocks in an affected area with a new effective rate.

Appendix

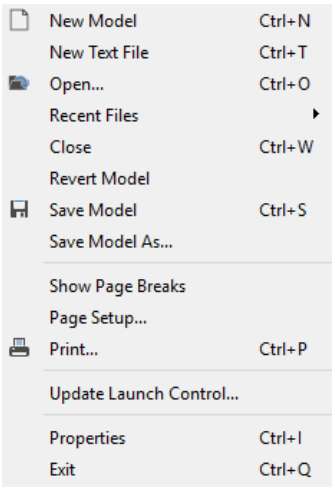
Menu Commands and Toolbars

A reference section describing
each menu item and tool

*"I wish it, I command it. Let my will
take the place of a reason."*
— Juvenal

This chapter explains all the commands that appear in the menus and the circumstances in which you might use them. At the end of this chapter is a description of the buttons in the application toolbar and ExtendSim database.

File menu



New Model

Creates an untitled model worksheet.

New Text File

Creates an untitled text file. You can use this to create text files for the Read and Write blocks, as input to sensitized blocks, or for any other ExtendSim feature that uses a text file as input. See “Text files” on page 240 for more information.

Open...

⚠ Libraries are opened from the Library menu, shown on page 550. Databases open when the model opens, as described at “Database menu” on page 557.

Opens an existing model (or text file) along with any libraries and databases that the model uses. (To open a library file not used by the model, use the Open Library command described on page 551.)

You can also open a model by double-clicking the model file; if ExtendSim is not already open, this action will launch ExtendSim before opening the model. However, this action only opens the most recently installed release of ExtendSim, which may not be what you want. See “Update Launch Control (Windows only)” on page 542 to control which application launches.

You can have any number of models open at the same time; open models are listed at the bottom of the Window command.

☞ If ExtendSim can’t find a library used by a model, the message **Searching for library...** will appear as described in “Searching for libraries and blocks” on page 57. If ExtendSim cannot find all of the blocks used in the model, the missing ones will be replaced with text blocks. When this occurs, ExtendSim will open the model as **Model-x** to prevent accidentally saving over the old model.

Recent Files

Lists the 20 most recently opened files; new files will only be listed after they have been saved.

Close

Closes the active window—model worksheet, dialog, or text file. If there are any files with unsaved changes, first asks whether you want to save them.

Revert Model

Becomes active after you make a change to a saved model. Reverts the model to the last version saved, discarding any changes made since then. ExtendSim warns you before it completes this command.

Save and Save As

Saves the active window. Choose Save to save the file under the current name or Save As to give a name to a new file or to save an existing file under a different name. The exact wording will be different, depending on which active window is being saved: Save Model As, Save Block to Library As, or Save HBlock to Library As.

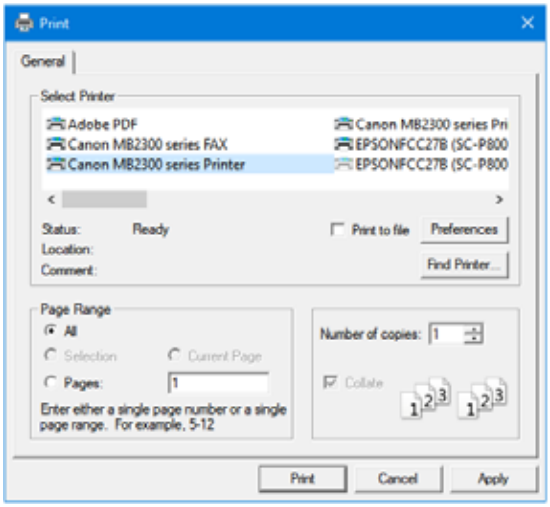
☞ If a crash occurs during the save process, your original file could get corrupted. To protect against file corruption when saving, ExtendSim first renames the *previously saved version* of the model as **ModelName.BAK**. To recover a model from a backup file, rename the backup file as **ModelName.MOX**.

Show Page Breaks

Causes ExtendSim to draw a set of page boundaries on the active window (worksheet, notebook, etc.) and place page numbers in the upper left hand corner of each page. The page boundaries show where page breaks will occur if you print the window. The command applies to each active window separately and the location of the page breaks is dependent on choices in the Print and Page Setup commands.

Print...

Opens a dialog so you can print the active window—model worksheet, notebook, tab in a block dialog, etc.



The Print dialog provides standard options such as page range and number of copies. Click the Preferences button to specify other parameters such as paper size, border, and printing on both sides.

☞ The parameters displayed in the Printing Preferences dialog, as well as whether you can print to paper, file, PDF, or fax, depend on the selected printer and which drivers have been installed.

To save your settings after making changes, click the Print dialog's Apply button. If you don't want to print after selecting Apply, click Cancel.

To print to PDF, choose a PDF writer such as Adobe PDF or Microsoft Print to PDF. All the parameters and settings are ignored and the document will be saved at full size as long as it is under 200" x200". Documents bigger than that will automatically be scaled to fit that maximum size.

Choosing Model > Show Object IDs or Model > Show Simulation Order before choosing Print, will cause the blocks on the model worksheet to print with that information on them.

☞ In some situations, such as when trying to print a library window, use the right-click menu instead of the menu command.

Page Setup
Opens a window for selecting paper size, source, orientation and margins. These parameters are dependent on the printer selected in the Print dialog.

☞ For the most options, use the Preferences button in the Print dialog.

Update Launch Control (Windows only)
If you typically launch ExtendSim by double-clicking a model file, the ExtendSim application that was most recently installed will open. This may not be what you want.

Selecting this command causes the currently active ExtendSim application to be the application that opens when a model (.mox) or library (.lix or .lbr) file is double-clicked. This is only applicable if you have more than one instance of the ExtendSim application installed and you want to cause a specific instance of the application to open when you double-click files. For example, give this command in ExtendSim 9 if you have both v9 and v10 installed, and you want model files to automatically launch v9 when they are double-clicked.

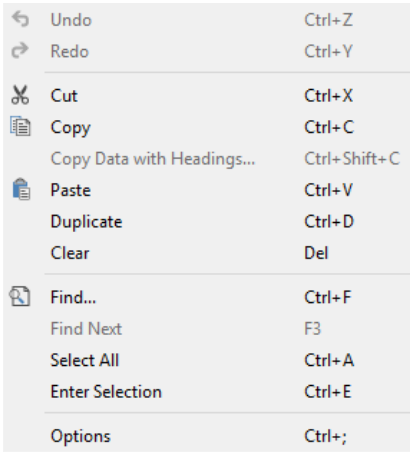
☞ Depending on your privileges, you may need to right-click the ExtendSim application and select Run as Administrator before launching and giving this command.

The application in which this command was last given will control the launching of the model and library files. To switch applications that will launch those files, just give the command in the File menu of whichever application you want to be in control of launching.

Properties
Opens a window that gives information about the selected graphic object, block, etc.

Exit/Quit
Leaves ExtendSim. If there are any files with unsaved changes, first asks whether you want to save them.

Edit menu



Undo
Reverses actions in a last-done, first undo order.

Redo
Reverses the Undo actions in a last-undone, first reverse order.

Cut
Removes the selected item (such as a block, some text, or numeric data from a data table) and places it on the Clipboard.

Copy
Copies the selected object to the Clipboard. Copying is useful for duplicating parts of a model as well as for exporting to other applications. You can copy a single block, a piece of text, a group of blocks and text, graphical objects, or numeric values from a data table. You can also copy sections of a notebook or dialog box as a picture.

⚠ The objects copied (blocks and text, graphics, etc.) depend on the cursor used to make the selection. To copy multiple objects of different types, such as text and a graphic, use the All Objects cursor.

Copying data and pictures is discussed in "Copy/Paste" on page 223.

Copy Data With Headings

Copies the table headings as well as the data from the selected data table.

Paste

Copies the contents of the Clipboard to the model. If the Clipboard contains text, a block, or a graphic item, the copied item is placed at the insertion point. For example, if you copy a block, click on the model worksheet, and then use the Paste command, the block will be pasted where you clicked the mouse. If there is no insertion point, the item is placed in the upper left corner of the worksheet.

Duplicate

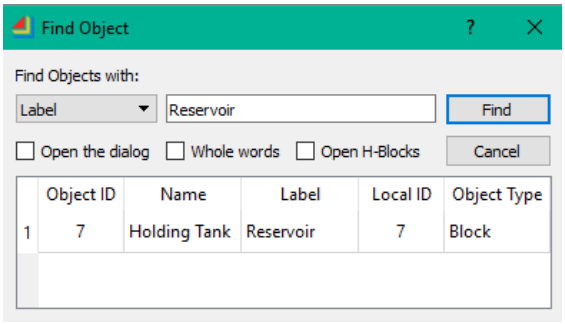
Makes a copy of the selected item and puts it near the original item. This is faster and often more convenient than first copying and then pasting an item.

Clear

Removes the selected item. The menu changes to indicate what is selected, such as "Clear Data" or "Clear Blocks". A cleared object is not held in memory and is not in the Clipboard.

Find...

This command opens a different window depending on where the search is initiated: the worksheet, the block's structure, or a database window.

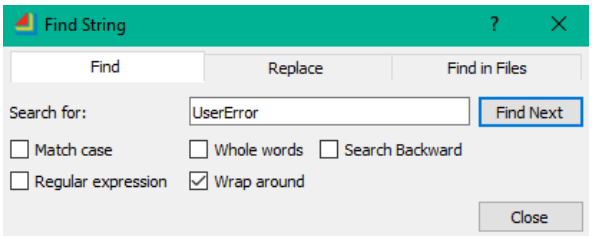


1) If invoked when the cursor is on the model worksheet, the command presents the *Find Object* dialog. In this dialog, search for a given object ID, label, or name. The *Object ID* is a unique, permanent identifier for all objects in a model—blocks, text, graphics, connection lines, animation objects, and so forth. *Name* means the name of the object and *label* is a secondary identifier assigned to the object. For example, a block named "Equation" (Value library) could be labeled as "Calculate Overflow" and have an Object ID of 4 on the model worksheet. Note that the Find Object dialog also reports the Local ID (which may different from the Object ID if the object is nested within a hierarchical block) as well as the Object Type (block, text, oval, etc.)

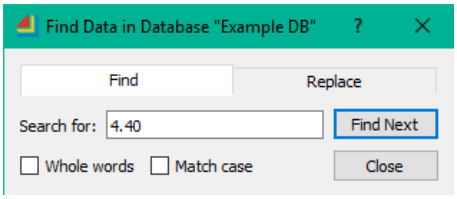
☞ Blocks have a given name and can be assigned a label through their dialogs. Blocks and all other objects can be assigned names and labels through their properties.

☞ See the Find and Replace block (Utilities library) for additional block search capabilities, including the ability to find a dialog value and replace it with another value.

2) If invoked when the cursor is on the Script or Help tab of a block's structure, the command presents the *Find String* dialog. Enter the text you want to search for in the Find box. ExtendSim searches for text starting at the current selection and going to the end of the text. If the specified string is found, the string is selected. If it is not found, the insertion point is not moved. For more information about how this dialog is used, including the use of the *Regular expression* option, see the Technical Reference.



3) When searching in an ExtendSim database, this command provides two dialogs that help locate a number or string. The *Find Data in Database* command is invoked in Schema view when the cursor is on the name of a table or is in the table pane. The *Find Data in Table* command is invoked in Data view when the cursor is on the name of a table, or in the tables pane, or in the blank space of the Viewer window. The search process, as well as the Whole words and Match case options, work the same as for the Find String dialog, above.



☞ To find a table, sort the tables by name in the database window.

Find Next

Finds the next instance of the string entered in the Find String dialog, discussed above. Is only enabled if the Find String dialog is active.

Select All

Selects all the objects, such as all the blocks in a model or all the text in a field.

⚠ The objects that get selected (blocks and text, graphics, and so on) depend on the cursor chosen. To select multiple objects of different types, such as text and a graphic object, use the All Objects cursor.

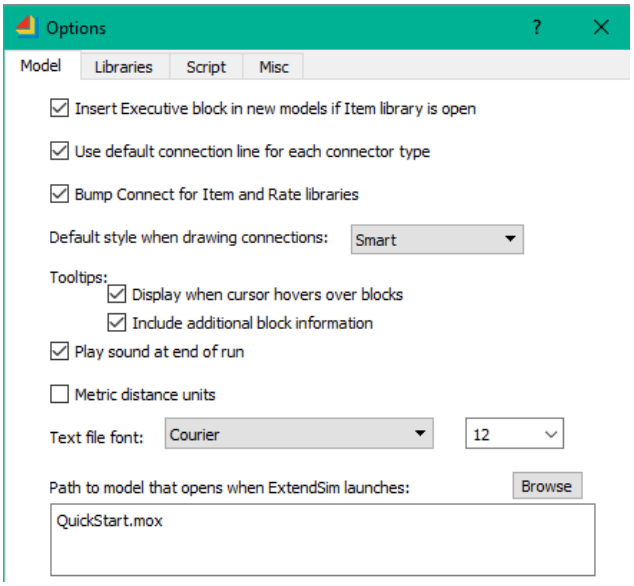
Enter Selection

Causes the selected text to be the search string in the Find String dialog and the Find Next command; both are discussed above. Especially useful for finding all instances of text that is already in the code of a block.

Options

Lets you specify how you want ExtendSim to behave in general, controlling actions for all models. Note that this is different from the Simulation Setup command which only affects the running of a specific model.

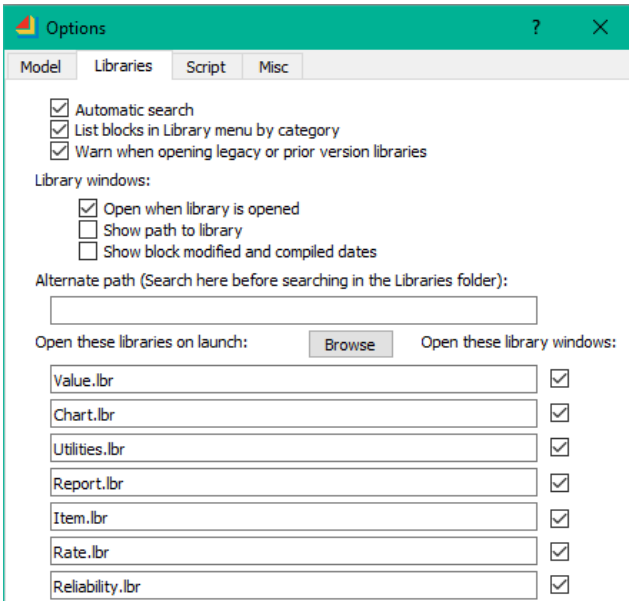
Model tab of Options command



Option	Description
Insert Executive block in new models if Item library is open	If your ExtendSim product installs the Item library, the assumption is that you want to build discrete event or discrete rate models, which require an Executive block. If instead you want to build continuous process models, uncheck this box or delete the Executive from your models.
Use default connection line for each connector type	Specifies that blocks in the Value, Item, and Rate libraries will use the default connection line for each **type** of connector. See “Connection line appearance” on page 116.
Bump Connect for Item and Rate	Use *Bump Connect* to add an Item or Rate library block to a block in a model. Click on the desired block in the library menu or library window, then place the block in the model by bumping its itemIn or inFlow connector to the corresponding output of a block on the worksheet. This adds and connects the new block to the existing block using whichever connection line choices you’ve made. This option only apply to Item and Rate connectors.
Default style when drawing connections	Specifies which **style** (Smart, Right Angle, Straight, or Free Form) is the default when creating new connection lines. The selected style will be indicated in the Model > Connection Line Style menu.
Tooltips: Display when cursor hovers over blocks	Causes the block name, number, and library to be displayed when the cursor is hovered over a block on the model worksheet. If you hover the cursor over a connector, the connector name and value are displayed. Note that Help captions for the toolbar stay on even if this choice isn’t selected.

Option	Description
Tooltips: Include additional block information	In addition to the block name, number and library, when a cursor is hovered over a block this command will cause additional information, such as a description of the block, to be displayed.
Play sound at end of run	Causes ExtendSim to play the default system sound at the end of every simulation run, if sound is on. (Note: whether sound is on or off the status bar shows “Simulation finished at timexx” at the end of each run.)
Metric distance units	Specifies that default distance or length units in the Convey Item and Transport blocks (Item library) is meters instead of feet. In the Convey Flow block (Rate library), distance and length units are specified directly in the block’s dialog and this setting is ignored.
Text file font	Sets the default font for text files, such as when you give the command New Text File or when you open a text file. **This option does NOT set the default font and size in the application toolbar nor does it affect the font used when creating a text box or working in a block’s Script tab.** Note that there is a separate option for setting the default font when working in a block’s Script tab; see “Script tab of Options command” on page 549.
Path to model that opens when ExtendSim launches	If a model pathname is entered here, that model will open when ExtendSim is launched. (By default, when ExtendSim is installed Getting Started.mox is the default model.) If no path is specified, ExtendSim will look for the model in the application directory or folder.

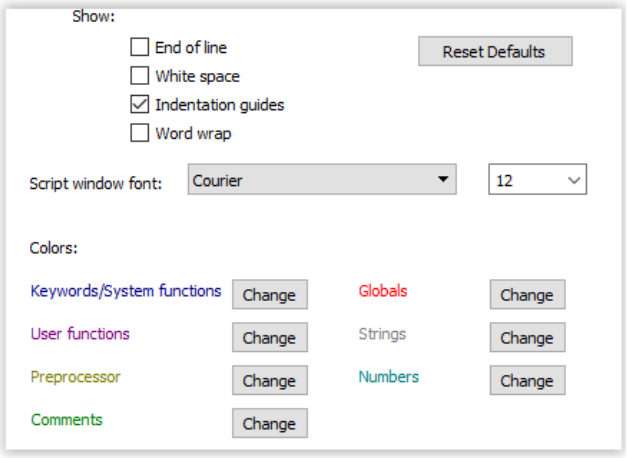
Libraries tab of Options command



Option	Description
Automatic search	If selected, causes ExtendSim to automatically find and open the libraries used in a model. If this is not selected, ExtendSim will prompt for the location of each library that the model uses. See also “Substituting one library for another” on page 68.
List blocks in Library menu by category	Causes blocks in each library to be listed in hierarchical menus by category. Unselect this choice if you want all of the blocks listed alphabetically.
Warn when opening legacy or prior version libraries	The warning appears if you use the Library > Open Library command to open a specific type of library: either a legacy (obsolete) library or an unconverted library developed in a prior ExtendSim release. NOTE: Legacy libraries are included so that models created using those obsolete libraries can still run. Legacy libraries should not be used to create new models. Whether checked or not, the warning is not displayed if the library automatically opens due to the model being opened.
Library windows: Open when library is opened	Causes the corresponding library window to open whenever a library opens. This occurs whether the library is opened directly from the Library menu or because a model that uses that library is opened.
Library windows: Show path to library	Causes the library window to display the path to the library below the library’s name. The same information is supplied if you hover over the library’s name in the library window. (If the path is just “Libraries”, the library is in the Libraries folder of the current application.)

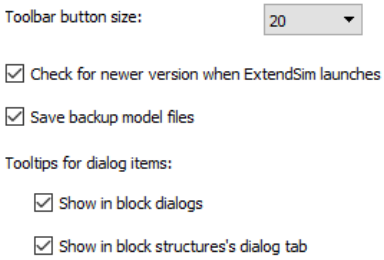
Option	Description
Library windows: Show block modified and compiled dates	Displays the modified and compiled dates for each block in the library windows.
Alternate path	By default when opening a model, ExtendSim first searches for the needed libraries in the folder that contains the model. If not found, it then searches in the Documents/ExtendSim/Libraries folder. The alternate path specifies a location that ExtendSim will search after the model folder but before searching in the Libraries folder. Click in the Alternate path field and use the Browse button to specify the alternate path. See also page 57.
Open these libraries on launch	Enter names of libraries you want automatically opened when ExtendSim starts. Type in the name or use the Browse button to locate a library and cause its name to be entered in the selected field. Note that ExtendSim will still ask for the location of any libraries located outside of the active application's Libraries folder. To cause a library to not preload, delete its name from the field. ☞ Depending on the particular product, the major libraries are listed on this tab and will preload by default. For example, launching ExtendSim CP causes the Chart, Report, Utilities, and Value libraries to also open.
Open these library windows	When ExtendSim is launched, opens the window of the library listed directly to the left of the checkbox. By default library windows stack on the right side of the application window and each library window has its own tab on the right side. ☞ The order of the tabs depends on the order in which the libraries are listed in the Edit > Options > Libraries tab, or the order libraries are opened if they are not listed in that table. And by default the library window displayed in front will be the first library in the list.

Script tab of Options command



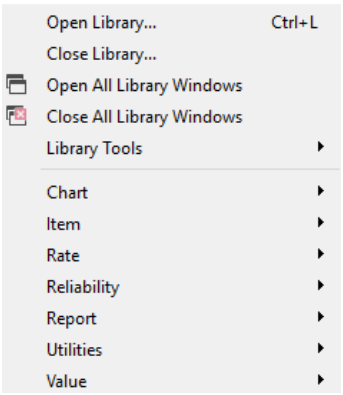
Option	Description
Show: End of line	Shows the end of line characters as visible characters. The script editor places invisible characters at the end of each line; this option makes those characters visible.
Show: White space	Shows both tab and space characters as visible characters. The script editor places invisible characters wherever there is a tab or space; this option makes those characters visible.
Show: Indentation guides	Shows vertical lines to indicate the tab indentation of the code. Useful for seeing if you've indented the code correctly, especially for long indented sections of code.
Show: Word wrap	Otherwise known as line wrap. When the line is full, this option automatically moves subsequent words to the beginning of the next line, so that the text stays within the viewable window.
Script window font	Sets the default font and size for text in the Script tab of a block's structure window.
Colors:	Shows the default colors for keywords, comments, and so forth, and provides Change buttons for selecting different colors.

Miscellaneous tab



Option	Description
Toolbar button size	Allows you to change the size of the buttons in the toolbar
Check for newer version when ExtendSim launches	Allows ExtendSim to do an Internet search for new versions when it launches, determining if the installed version is the most current version. The search occurs each time ExtendSim is launched. If the installed version is current, nothing happens. If the installed version is not current, or if ExtendSim is unable to connect to the Internet, a message is placed on the About ExtendSim startup screen.
Save backup model files	If a crash occurs during the save process, your original file could get corrupted. To protect against file corruption when saving, ExtendSim first renames the *previously saved* version of the model as "ModelName.BAK". To recover a model from a backup file, rename the backup file as ModelName.MOX. This option is checked by default for your safety Rob Brownie!
Tooltips for block dialog items: Show on block dialogs	Displays the variable/message dialog names when the cursor is hovered over fields in a block's dialog.
Tooltips for block dialog items: Show in block structure's Dialog tab	Displays the variable/message dialog names when the cursor is hovered over fields in the Dialog tab of a block's structure.

Library menu



When you open a model, ExtendSim opens the required libraries automatically. And depending on settings in the Edit > Options > Libraries tab, specific libraries will open whenever ExtendSim is launched. To open or close a library manually, use the Library menu. For more information about libraries and their usage, see “Opening, closing, and searching for libraries” on page 56.

Open Library...

Opens a window for selecting which library file(s) to open. If a default path to a folder has been entered in the Edit > Options > Libraries tab, this command will open that folder. Otherwise, it opens the ExtendSim/Libraries folder.

Once opened, the library’s name will appear at the bottom of the Library menu in alphabetical order. To view the blocks in the library, go to the Library menu and then to the name of the library. When the library name is selected, blocks will be listed to the right of the menu either alphabetically or by category, depending on the option selected in Edit > Options >Libraries tab. The first choice in each list is Open Library Window which opens a window listing the blocks in the library, as discussed at “Library windows” on page 58.

Close Library...

Opens a window listing all the open libraries. Select the library or libraries to be closed from the list and click Close. Libraries that are in use cannot be closed. Closing a library also closes its library window.

Open All Library Windows

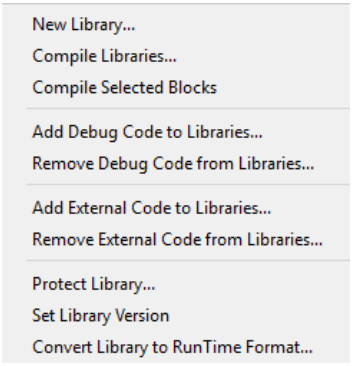
Opens the library windows for every library that is open and causes the library windows to dock on the right side of the application window. Depending on settings in the Edit > Options > Libraries tab, library windows might also open automatically whenever ExtendSim is launched. Note: see also the two buttons at the top of the library window docking area for opening and closing all library windows.

Close All Library Windows

Closes all the library windows that are currently open. Note: see also the two buttons at the top of the library window docking area for opening and closing all library windows.

☞ Does not close the library; see the Close Library command for that.

Library Tools



☞ The following commands are of interest mainly to block developers.

New Library

Creates a new library and opens a dialog for entering its name.

Compile Libraries

Opens a dialog for selecting which libraries to compile. Libraries must be open to be included in the list. To compile libraries with debugging code, give the Add Debug Code to Libraries command before giving this command. To remove debugging code, give the Remove Debug Code from Libraries command, then give the Compile Libraries command.

Compile Selected Blocks

Causes the selected blocks to be recompiled. To activate the command, select blocks in the library window. Uncompiled blocks show in the library window with their names in red italics.

Add Debug Code to Libraries

Opens a dialog for selecting which libraries will be recompiled with debugging code. Give this command, then give the Compile Libraries command. Blocks with debugging code enabled show in the library window and on the model worksheet with a red border around their icons.

Remove Debug Code from Libraries

Opens a dialog for selecting which libraries will be recompiled without debugging code. Give this command, then give the Compile Libraries command.

Add External Code to Libraries

Opens a dialog for selecting libraries. For each block in a selected library, moves its source code into a separate text file. This is useful for source code control. For more information, see the Technical Reference. Blocks with external code enabled show in the library window with a green CM on their icons.

Remove External Code from Libraries

Opens a dialog for selecting libraries. Moves the external source code back into each selected library's blocks. For more information, see the Technical Reference.

Protect Library

After two warning messages, opens a dialog for selecting which library to protect. Protection removes the ModL source code from all the blocks in the selected library so users cannot access block code. For more information, see the Technical Reference.

⚠ You would only use this command to protect libraries of blocks you build yourself. Do not use this command with the libraries that are included with ExtendSim.

Set Library Version

Allows you to set the long and short version strings for libraries you have developed. This is useful for version control if you are programming your own libraries.

Convert Library to RunTime Format

Changes a copy of a selected library to Runtime format. This removes the ModL code, as discussed in the Protect Library command, above and ensures that Model Developer editions of ExtendSim cannot open or use the protected library.

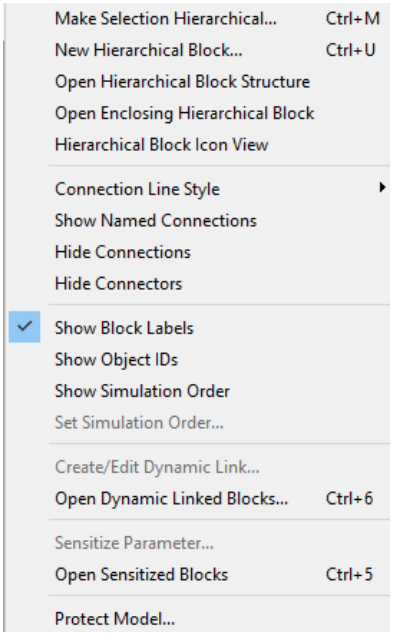
Converting to Runtime format provides an easy method for providing libraries to others for evaluation or model running, while preventing those libraries from being used to build models.

☞ If creating Run Time libraries on a different computer platform than the eventual target, you must compile the library on the target platform before converting it to a Run Time library.

List of libraries

As libraries are opened, they are listed at the bottom of the Library menu in alphabetical order.

Model menu



Make Selection Hierarchical

Encapsulates selected blocks into a single hierarchical block and replaces those blocks on the model worksheet with the hierarchical block. This is described in “Hierarchy” on page 120.

New Hierarchical Block...

Opens a window for starting a new hierarchical block. This prompts you for the name of the new hierarchical block, then opens a blank hierarchical block structure for building the encapsulated model. This is described in “Creating a new hierarchical block” on page 124.

Open Hierarchical Block Structure

Opens the structure of a hierarchical block so you can edit its icon or Help. Note that the hierarchical block must be selected on the model worksheet. This command is equivalent to holding down the Alt (Windows) or Option (Mac OS) key while double clicking a hierarchical block on the model worksheet. See “Modifying hierarchical blocks” on page 128.

Open Enclosing Hierarchical Block

Finds the hierarchical block that contains the selected block. If the enclosing hierarchical block is at the top level of the model worksheet, the hierarchical block will be selected (shows as highlighted) on the worksheet. If the enclosing hierarchical block is below the top level of the worksheet, the command opens the hierarchical block’s worksheet, showing the submodel that contains the selected block.

Hierarchical Block Icon View

Places a miniaturized picture of its submodel on the selected hierarchical block’s icon. This enables a live view of the worksheet and contents of the HBlock, including updates of the animation inside the HBlock.

Connection Line Style

Sets the format of the connection lines. For details, see “Connection line appearance” on page 116.

Show Named Connections

Inserts connection lines between named connections. This is useful to show data flow in complex models with many named connections. Named connections are discussed at “Named connections” on page 118.

Hide Connections

Hides the connection lines between blocks. This is a cosmetic change that is mostly used to enhance presentations. Select the command again to show the connecting lines.

☞ This command works at each level of hierarchy separately and takes effect in whichever level is the active window. Thus if you want only the top level of the model to not show connections, but the hierarchical levels to show connections, choose the command when no hierarchical levels are open.

Hide Connectors

Hides the connectors visible on blocks. This is a cosmetic change that is mostly used to enhance presentations. Select the command again to show the connectors.

☞ This command works at each level of hierarchy separately and takes effect in whichever level is the active window. Thus if you want only the top level of the model to not show connectors, but the hierarchical levels to show connectors, choose the command when no hierarchical levels are open.

Show Block Labels

Shows the block labels below the blocks in the model.

Show Object IDs

Puts the Object ID in square brackets on each block in the model. Object IDs are unique, permanent identifiers. The blocks inside a hierarchical block show two numbers—the first is the block’s global Object ID and the second is the block’s local number within the hierarchical block.

☞ While each object in ExtendSim—block, text, graphic object, and so forth—has a global Object ID, this command only shows the IDs for blocks. To see the Object ID for a non-block object, right-click the object and look in its Properties dialog.

Show Simulation Order

Puts the number of the block’s order of execution on the blocks in the model. Hierarchical block internals have their own simulation order relative to the parent block. Simulation order only relates to continuous process models; it is discussed in the Continuous Modeling Quick Start guide.

Because blocks can override the system’s simulation order, this display may be inaccurate for discrete event (Item library) and discrete rate (Rate library) blocks that send block-to-block messages.

Set Simulation Order...

Opens a window for setting the selected block's order of execution in the model. Simulation order only relates to continuous process models; it is discussed in the Continuous Process Modeling Quick Start guide.

Create/Edit Dynamic Link...

Opens the Link dialog so a selected block's parameter field or data table can be linked to an *internal* data structure (ExtendSim database or global array). This action creates a two-way dynamic link between the data structure and the dialog item. Dialog parameters can be linked to a cell in a global array or database table; dialog and graph data tables can be linked to a global array or database table. Although you can link data tables to a database table, you cannot link an individual cell in a data table to a cell in a database.

☞ The Create/Edit Dynamic Link command is only used for linking parameters and tables to *internal* data structures.

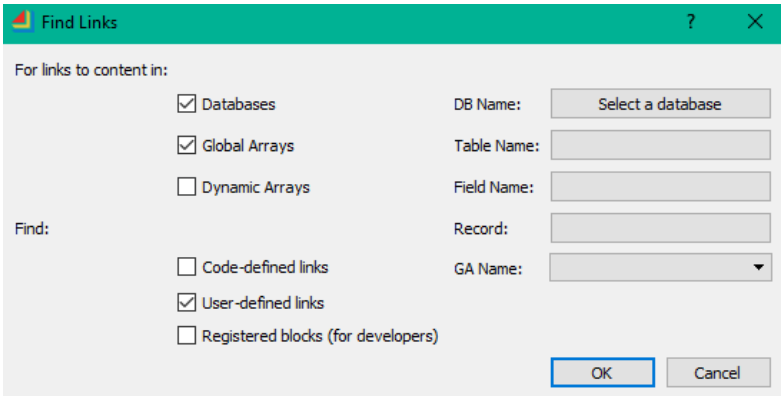
When you give the Create/Edit Dynamic Link command, a window appears with tabs for selecting the data structure. Depending on the structure selected, other options are presented. To delete a dynamic link, click the linked parameter or select the linked data table and give the Create/Edit Dynamic Link command. In the dialog, choose Delete Link.

The command is equivalent to right-clicking the dialog item and choosing Create/Edit Dynamic Link, or (for a data table) clicking its Link button. Dynamically linked parameters are outlined in light blue. Dynamically linked data tables display the words DB (database) or GA (global array) in their upper left corner. For more information, see "Dynamic linking to internal data structures" on page 229.

☞ Another type of internal data source is a *dynamic array*, which is implemented through a block's code. You cannot use the Create/Edit Dynamic Link command to link to a dynamic array without programming. However, if a data table is already programmatically linked to a dynamic array, its upper left corner will be blue.

Open Dynamic Linked Blocks...

Opens the Find Links dialog, shown on the right.



The Find Links dialog is useful for locating and examining linked dialog items or registered blocks.

The options in the dialog allow you to choose where you want to search (the contents of databases, global arrays, and/or dynamic arrays) and what you want to find (code-defined links, user-defined links, or registered blocks). *User-defined links* were created by the modeler through the user interface; *code-defined links* were defined by block developers through ModL code. *Registered blocks* do not have any specific dialog items associated

with a link. Instead, the block just requests that it be informed when the linked data source changes.

☞ Since registered blocks are mainly for developers, they are discussed in the Technical Reference.

The DB selectors and the GA popup menu on the right of the dialog specify which database or global array structure will be searched for.

☞ Leaving the DB name or GA name popup menus to the default (empty) choices as seen above will find any linked database or global array.

When the OK button is clicked, ExtendSim opens the dialogs of all the blocks with the specified types of links or registration.

☞ If the model is large and you specify all types of links this command can open many block dialogs. It is usually advisable to search for specific types of links, so as not to have more dialogs open than needed.

Sensitize Parameter...

If a dialog parameter is selected, this command opens the Sensitivity Setup dialog, which lets you set values for sensitivity analysis. An alternate method of opening the Sensitivity dialog is to click on the dialog parameter once while holding down the Control (Windows) or Command (Mac OS) key.

A parameter that has sensitivity settings has a frame inside of it. If sensitivity analysis is active for a parameter (that is, if the **Enable sensitivity** choice is checked in the Sensitivity Setup dialog), the frame is green. If the sensitivity analysis is inactive for the parameter or if it is turned off for the model as a whole, the frame is red.

Sensitivity analysis is discussed in “Sensitivity analysis” on page 138.

Open Sensitized Blocks

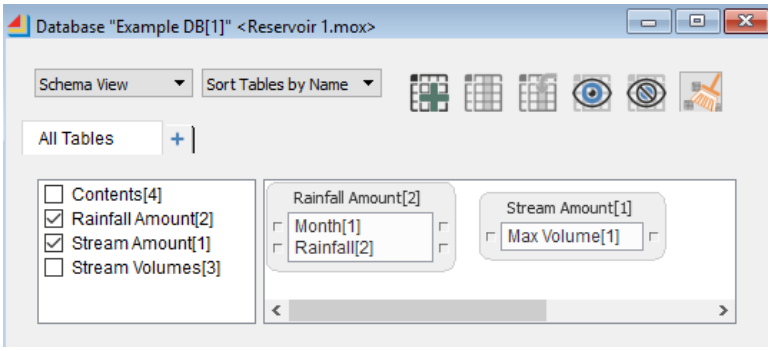
Opens the dialogs of all blocks that have sensitized parameters. This is useful for finding which blocks are used in the scenario. The command opens dialogs with sensitized parameters even if sensitivity has been disabled for a parameter or is not active for the entire model. For more information, see “Sensitivity analysis” on page 138.

Protect Model...

Opens a window for protecting the model from any modifications other than changing dialog values. Can also be used to lock a hierarchical block. Protecting models and hierarchical blocks is useful if you are giving the model to others who are unfamiliar with ExtendSim features and may accidentally move or delete blocks. For more information, see “Protecting the model” on page 255.


Always make unlocked copies of models before you lock them. Once you have locked a model with a password, you must have that password to unlock the model.

Database menu



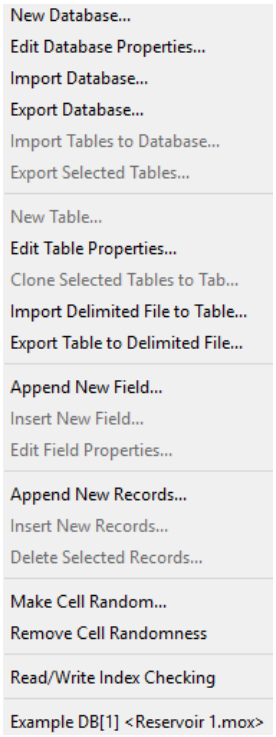
The database window is for viewing and managing the *schema* and *data* for an ExtendSim database. The schema is the skeleton structure that represents the logical view of the entire database—tables, fields, and parent/child relationships. The data view is where records are created and data is entered.

To open a database window for an existing database, do one of the following:

- Click the name of the database at the bottom of the Database menu.
- Give the Window > Database List command and double-click a database in the list.

ExtendSim databases are stored with the model. They open when the model opens and are saved or closed when the model is saved or closed. For more information about creating and using ExtendSim databases, see “ExtendSim databases for internal data storage” on page 232 and the eBook ExtendSim Database located at Documents/ExtendSim/Documentation.

☞ Database windows have their own sets of buttons, as shown on page 573.



New Database...

Opens a dialog for naming and creating a new ExtendSim database for the model. The dialog also shows a list of the model's current databases so you can avoid using a duplicate name—ExtendSim will warn you if you try to use a duplicate name.

☞ A model window must be opened for this menu to be enabled.

☞ The database's name may not *start* with an underscore (_), may not be used by another database in that model, and cannot exceed 63 characters. Database names are not case sensitive and they can contain spaces.

Edit Database Properties...

Opens a dialog for selecting a database to rename.

Import Database...

Opens a window for creating a new ExtendSim database by importing a database text file that has been exported from an entire ExtendSim or SDI Industry database. (See the "Export Database..." command for how the database text file was created.)

In the window, select the database text file to import and give it a name. This command imports all the tables, fields, records, data, and relations from the exported database and creates a new database. If you choose the name of an existing database instead of a new name, a dialog gives options for how like-named and other tables should be handled.

To instead just import tables to an existing database, see the "Import Tables to Database..." command.

Export Database...

Exports the entire selected ExtendSim database—all tables, fields, records, and so forth—into a special database text file. Do this to import the database into another model, send the database to another user, or prepare a database text file for use by the ExtendSim DB Add-In for Excel. Exported databases can only be imported to ExtendSim or to the ExtendSim DB Add-In for Excel.

To instead export only specific tables from a database, see the "Export Selected Tables" command.

Import Tables to Database...

With an existing ExtendSim database window active, opens a window for selecting a database text file that was created using the "Export Selected Tables" command and adds the imported tables to the existing database. This command imports all the tables from the exported file and places them below the existing tables in the Tables pane. If you end up with unneeded tables, you can delete them.

To instead create an entirely new database using imported files, see the "Import Database..." command.

Export Selected Tables

Creates an ExtendSim database text file containing only the selected tables. Use this command instead of the "Export Database..." command, when you want an exported file that contains only a portion of the database. To enable the command, select one or more tables in a database window.

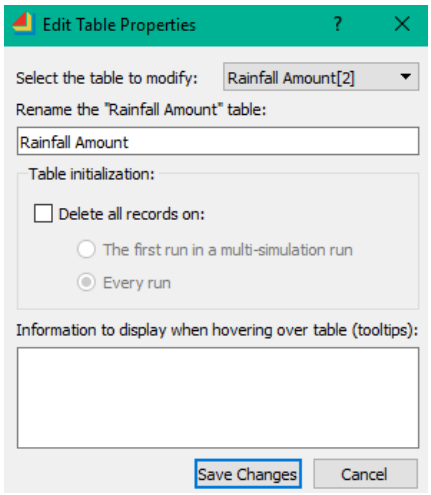
New Table...

Opens a window for creating a new table for an ExtendSim database and adds it to the list of tables in the database's All Tables tab. The dialog that appears displays a list of the database's existing tables. After naming the table, click OK. To enable this command, bring a database window to the front.

☞ To delete a table from a database, select the table and use the Delete or Backspace key or go to the Edit > Clear Selected Tables command. A dialog displays the delete options and, if there is data dynamically linked to the table, warns you before deleting it.

Edit Table Properties...

Opens a dialog for editing table properties such as the table name, whether it should be initialized, and its tooltips. This dialog is discussed in the separate document titled ExtendSim Database Tutorial and Reference. The command works the same as right-clicking a database table and selecting *Edit Table Properties*.



Clone Selected Tables to Tab...

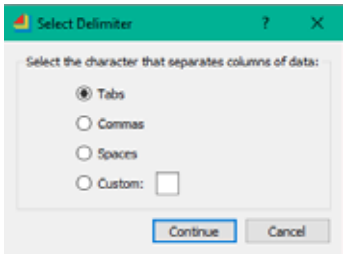
Clones the table or tables to another tab in the database window. To enable this command, bring a database window to the front, as described under Database menu, above. Bring the source tab to the front. Select the tables to clone using the Block/Text cursor and give the command. (Or right-click a selected table and choose "Clone Selected Tables to Tab".) In the dialog, select the tab to copy the table to.

☞ Cloning is different than duplicating or copying and pasting a table. Tables that are copied are neither linked to each other nor to the original table. However, a cloned table will behave exactly like the original table and will change when the original table changes. Likewise, changes made to the cloned table will be reflected in the original table.

Import Delimited File To Table...

Use this command to import into a table a file that was created in Excel or some other application. See note below about exported delimited text files.

Copies data from a text file into the selected table. After choosing the file to be imported, the Select Delimiter dialog appears.



Rows in a text file are automatically separated by returns; you must specify how the columns are delimited (separated). Most exported text files have columns delimited by tabs but check the format of the text file before choosing this command. For more information, refer to “Importing and exporting data” on page 223.

☞ A table from a database must be selected before this command can be used.

Export Table to Delimited File...

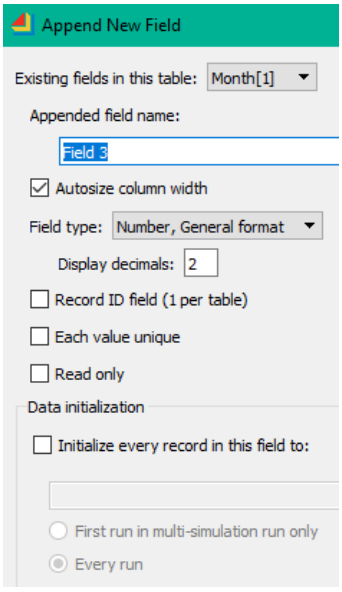
⚠ The main purpose of exporting a table as a delimited text file is so the file can be read by Excel or some other application. And the main purpose of importing a delimited file is to use a file created in Excel or some other application. However, delimited files are not automatically formatted as an ExtendSim database table as would be generated using the Export Selected Tables or Import Tables to Database commands. If you want to work on ExtendSim databases in Excel, a better method would be to use the ExtendSim DB Add-In.

Use this command to create a text file that can be read by Excel or some other application.

Copies data from a selected database table to a text file. It works the opposite of the “Import Delimited File To Table...” command. After you give the file name, ExtendSim puts up the column separator dialog (as shown in the previous section), so you can specify which type of separator to use in the text file. For more information, refer to “Importing and exporting data” on page 223.

Append New Field...

Creates a new field for the selected table and puts the field below any other fields in the list. Works the same as right-clicking a database table and selecting *Append New Field*.



As seen above, the dialog gives options for setting the field’s name, type, initialization, and other properties. To enable this dialog, select a table in the Schema view and either right-click for the command or give the command from the menu.

☞ To delete a field, use the Delete or Backspace key, right-click and choose Clear Selected Fields, or go to Edit > Clear Fields. A dialog displays the delete options and, if there is data dynamically linked to the field, warns you before deleting it.

Insert New Field...

Works like the Append New Field command, except inserts the new field above the selected field in the table.

Edit Field Properties...

Opens a dialog for editing field properties; the dialog is similar to the Append New Field dialog shown above. This command works the same as right-clicking a database field and selecting *Edit Field Properties*.

Append New Records...

Creates new records for the selected field and puts the records below any other records in the field. A dialog requests the number of records to add. To enable the command, choose the Data view in the database window or double-click the table to open the Table window, then right-click on a field.

☞ To delete a record, use the Delete or Backspace key or go to Edit > Clear Record. A dialog displays the delete options and, if there is data dynamically linked to the record, warns you before deleting it.

Insert New Records...

Works like the Append New Records command, except inserts the new record above the selected record.

Delete Selected Records

Removes the selected records from the currently active ExtendSim database table.

Make Cell Random...

Makes the selected cell random. Choosing this option opens a dialog for selecting the distribution (including a named distribution) and its parameters.

Remove Cell Randomness...

Returns a randomized cell to being a constant.

Read/Write Index Checking

"For database read/write functions, this command enables error messages if a read/write function call has illegal indexes. This is a good tool for finding illegal indexes and enabling this option does not impact the speed of a simulation run. Leave it unchecked if it is preventing a legacy model from running. New models have this checked by default.

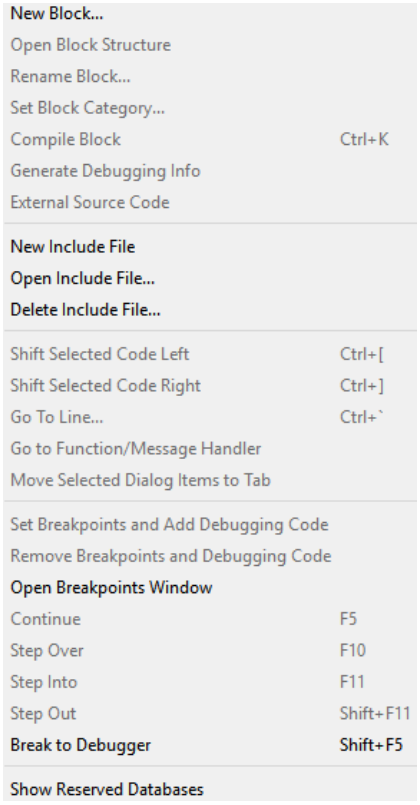
List of databases

At the bottom of the Database menu is a list of open databases. The list can also be seen by giving the command Window > Database List.

☞ Some databases are reserved and will only be listed if the command Show Reserved Databases is given in the Develop menu. The names of reserved databases, such as RightClickConnect shown here, are preceded by an underscore (_) character and in general should not be modified unless you really really really know what you're doing.

Develop menu

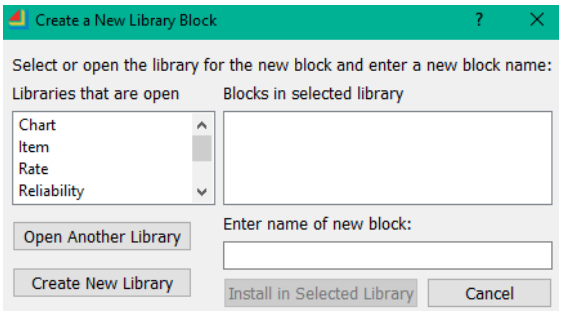
The Develop menu is used when creating or modifying blocks and storing them in libraries. See the ExtendSim Technical Reference to learn how to create your own libraries of blocks and for additional information on using these commands.



New Block...

Opens a dialog for creating a new block. In the dialog, select the library where the block will reside, then specify the block's name.

Currently open libraries are listed on the left. Use the Open Another Library button to open a library that is not yet open, or the Create New Library button to create a library for the new block.



After the block has been named and installed in the selected library, the block's default structure window appears with tabs for the icon, dialog, script, and help. For a quick overview on building a new block, see the Technical Reference.

Open Block Structure

Opens the structure of the selected block, with tabs for the block's icon, dialog, script, and help. You can also right-click a block's icon and choose this command.

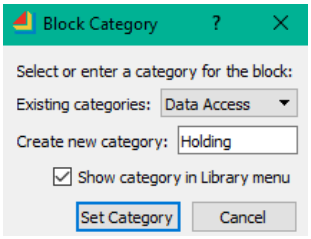
☞ This command is equivalent to holding down the Alt (Windows) or Option (Mac OS) key while double-clicking a block's icon in the library window or on the model worksheet.

Rename Block...

Opens a dialog for renaming a block. This command is only enabled when a block's structure is the active window.

Set Block Category...

Opens a dialog for setting a block's category. This serves two functions:



- To organize blocks within the Library menu by functionality.
- To organize blocks within statistical reports by functionality.

This command is only enabled when the block's structure is the active window. If Show category in Library menu is not checked in the dialog, the block name will be listed in alphabetical order directly under the library's name in the Library menu. Note that you cannot change the categories of ExtendSim blocks.

Compile Block

Compiles the ModL code for the block. This is useful for checking the syntax of the code that you are working on without having to close the block's structure. This command is only available when the block's structure is the active window. You can also choose to compile the block with debugging information and with external source code, as discussed below.

Generate Debugging Info

If this command is selected when the Compile Block command is given, ExtendSim generates debugging information for the block. This allows you to debug block source code using breakpoints, watch points, and so forth. If compiled with debugging code, block icons will show on the model worksheet and in the library window with a red border around them.

When used in models, blocks with debugging code run slower. To remove debugging code from a block, open the block's structure and uncheck the Generate Debugging Info command, or right-click the block on a model worksheet and select *Remove Breakpoints and Debugging Code*. To remove debugging code for an entire library, see "Remove Debug Code from Libraries" on page 552.

External Source Code

If this command is selected when the Compile Block command is given, ExtendSim generates a text file containing the block's source code. This is used for version control. If a block has been compiled with external source code, it will be listed in the library window with the designation CM (code management) on the right side of its icon.

New Include File

Creates an untitled Include file.

Open Include File...

Opens a window for browsing to an existing Include file.

Delete Include File...

Opens a window for browsing to an existing include file so it can be deleted.

Shift Selected Code Left

Moves the selected lines of the ModL code one tab stop to the left. This command is only enabled when a block's structure is open and the Script tab is the active window.

Shift Selected Code Right

Moves the selected lines of the ModL code one tab stop to the right. This command is only enabled when a block's structure is open and the Script tab is the active window.

Go To Line...

Allows you to quickly move to a specific line in the block code. This command is only enabled when a block's structure is open and the Script tab is the active window.

Go To Function/Message Handler

Jumps to where the function or message handler is defined in the code. *Select the function or message handler name where it is used in the code, then give the command*. This is equivalent to holding down the Alt (Windows) or Option (Mac OS) key while double clicking the function or message handler name, or right-clicking on the function or message handler name. This command is only enabled when a block's structure is the active window and the cursor is in the structure's Script tab.

Move Selected Dialog Items to Tab

Allows you to move the selected dialog items from one tab to another. Only enabled when the block's structure window is open, the Dialog tab is active, and one or more dialog items is selected.

Set Breakpoints and Add Debugging Code

Recompiles the block in debugging mode (if not already done) and opens two windows: the Breakpoints window for the model and the Set Breakpoints window for the block. As discussed below, the Breakpoints window shows the breakpoints for all the blocks in the model. The Set Breakpoints window shows the source code for the block and has a breakpoint margin at the left. This command is only enabled when a block is selected on the model worksheet.

Remove Breakpoints and Debugging Code

Removes the breakpoints from the selected block and closes the Set Breakpoints window. This command is only enabled when a block is selected on the model worksheet.

Open Breakpoints Window

Opens the global Breakpoints window, showing all breakpoints from all blocks in the model. This command is only enabled when a model worksheet is open.

Continue

Continues execution from a breakpoint. This command is only enabled when the source debugger is the active window and you are stepping through the code.

Step Over

Steps over a function call when stepping after a breakpoint. This command is only enabled when the source debugger is the active window and you are stepping through the code.

Step Into

Steps into a function call when stepping after a breakpoint. This command is only enabled when the source debugger is the active window and you are stepping through the code.

Step Out

Steps out of a called function to return to the caller when stepping after a breakpoint. This command is only enabled when the source debugger is the active window and you are stepping through the code.

Break to Debugger

Immediately opens the debugging window if libraries are compiled with debugging code and you are running a model or starting some operation in a block's dialog. This will show exactly where and what block code is executing. This is very useful in finding problems and loops in block code.

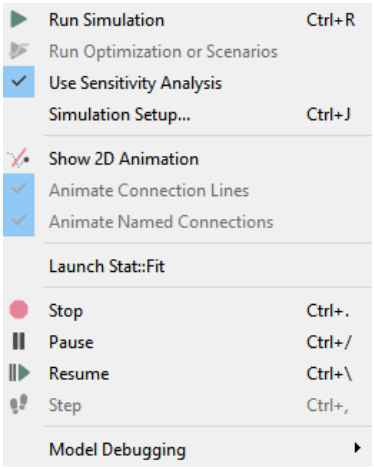
Show Reserved Databases

Causes the model's reserved databases, if any, to be listed at the bottom of the Database menu and in the Database List. Reserved databases allow programmers to use database capabilities without the model-user being aware of the database. Since reserved databases can only be created and modified through the ExtendSim database API, they are discussed in the ExtendSim Technical Reference.

⚠ There's a really good reason why a database is reserved. So unless you are the developer who created the database, you probably should not be giving this command.

Run menu

The Run menu lets you modify when and how your simulation runs, show animation, and generate model reports. A hierarchical menu at the bottom provides commands for debugging models. For more information about running models, see "Model Execution" on page 83.



Run Simulation

Starts a simulation. This is the same as clicking the Run ▶ or ⏩ button in the Model toolbar. The command first checks every block in the model to see that it has been compiled.

☞ The icon of the Run Simulation button changes depending on which Run Mode has been selected in the Model toolbar. If the model has been set to run in the fastest (rather than multi-threaded) mode, the Run button will be ▶. For more information, see "Run modes" on page 87.

Note that during the simulation run, the Run button in the toolbar changes to a Pause button while the simulation is running and to a Resume button while the simulation is paused. See "Model toolbar" on page 569 for more information.

Run Optimization or Scenarios

Runs an optimization or scenarios. This command is disabled unless there is an Optimizer or Scenario Manager block (Value library) in the model. See "Optimization" on page 158, or see "Scenario analysis" on page 143. Note that you cannot run optimization and scenarios at the same time. ExtendSim will warn if you give this command, or use the Run Optimization or Scenarios button, with both an Optimizer and a Scenario Manager block in the model.

Use Sensitivity Analysis
Causes ExtendSim to use sensitivity analysis settings when you run the simulation. Only enabled if a dialog parameter value has sensitivity settings. For more information, see “Sensitivity analysis” on page 138.

Simulation Setup...
Opens a dialog for setting the start time, end time, and other settings for a simulation run as well as options for setting random number seeds. The dialog is described in detail in “Simulation setup” on page 84.

Show 2D Animation
Causes blocks in the model that have animation to animate when the simulation is run. This is discussed in“Animation” on page 110. Note that some blocks can show some animation, such as text on the icon reporting final values, even if Show Animation is not selected. You can also choose to animate along connection lines or between named connections, as discussed below.

Animate Connection Lines
This option controls whether or not 2D animation pictures will be displayed along connection lines in discrete event models. If on, blocks from discrete event libraries (such as the Item library), will display their item animations; this is discussed in “Animating the movement of items between blocks (discrete event modeling only)” on page 111. If off, only animations on block icons will be displayed. This command requires that Show 2D animation be enabled and is only available for discrete event models.

Animate Named Connections
For named connections (where text labels, rather than connection lines, indicate the path of items), this option will cause the 2D animation picture to travel in a straight line between the two text labels. If this option is turned off, the animation picture will disappear when it has reached a text label and reappear at the matching text label. This command requires that Show 2D animation be enabled and is only available for discrete event models.

Launch StatFit (Windows only)
Opens the Stat::Fit application. This command is only enabled if Stat::Fit is installed. Stat::Fit is a Windows only product that is included with certain ExtendSim products; it can be purchased separately for use with other ExtendSim products. For more information about Stat::Fit, see “Stat::Fit (Windows only)” on page 172.

Stop
Stops the simulation. This is the same as the Stop button  in the toolbar. As an alternative, hold down the Ctrl (Windows) or Command (Mac OS) key while pressing the period (.) key. In any case this will cause a window to appear where you can choose to continue the run or end it.

Pause
Halts the simulation temporarily. This is the same as the Pause button  in the toolbar. Note that the Pause button replaces the Run button in the toolbar once the simulation starts running. Once the simulation is paused, the Resume button  appears until the simulation is resumed. To restart the simulation, give the Resume command, below, or click the Resume button.

Resume

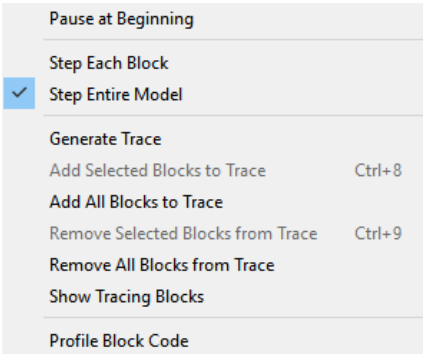
Restarts a paused simulation. This is the same as the Resume button in the toolbar. Once the simulation has resumed, the Pause button appears in the toolbar until either the simulation gets paused or the run finishes.

Step

Steps the simulation run depending on which option (Step Each Block or Step Entire Model) is selected in the Run > Model Debugging menu. This is the same as the Step button in the toolbar. See the Step commands in the Model Debugging menu below, as well as “Stepping through a model” on page 89.

Model Debugging

This hierarchical menu shown to the right facilitates finding a modeling problem. For additional model debugging help, see “Debugging Tools” on page 205.



Pause at Beginning

If enabled, pauses the simulation after it starts so that you can step through the run from step zero. The first pause occurs before the first step but after the initial model processing (initialization, error checking, etc.) While the simulation is paused, the word **Paused** appears in the model’s status bar.

Step Each Block

Controls the behavior of the Step command or button so that you can step through a simulation block by block. Only active when the Pause button or command has been activated.

Step Entire Model

Controls the behavior of the Step command or button so that you can step through an entire cycle of all the blocks in the model. Each Step command starts at the selected block and continues the simulation run until the execution order returns to the original block. Only active when the Pause button or command has been activated. This is a good way to examine what happens in the intervals between when a block is called.

Generate Trace

The Trace commands are used to generate a Trace file of the values for each selected block in the active model worksheet at each step of the simulation. ExtendSim will prompt for a name for the new trace file when the model starts running. The content of the trace file depends on which blocks have been selected to be included. For more information, see the following commands and “Model tracing” on page 216.

Add Selected Blocks to Trace

Causes blocks that have been selected in the active model worksheet to be included in the trace file. Trace files are generated if Generate Trace is checked when the model is run.

Add All Blocks to Trace

Causes all blocks in the active model worksheet to be included in the trace file. Trace files are generated if Generate Trace is checked when the model is run.

Remove Selected Blocks from Trace
Causes blocks that have been selected in the active model worksheet to be removed from the trace file.

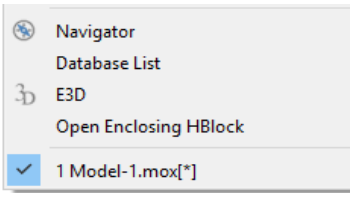
Remove All Blocks from Trace
Causes all blocks in the active model worksheet to be removed from the trace file. Use this before starting a new type of trace.

Show Tracing Blocks
Causes blocks that have been included in a trace to show the word **Trace** on them in the model worksheet.

Profile Block Code
Generates a text file showing the percentage of time that each block spent executing during the simulation run. See “Look for inefficient settings or code” on page 99 for more information about profiling a model.

Window menu

For the Windows operating system, the Window menu list the standard *Tile*, *Cascade*, *Next*, and *Previous* commands.



The menu also lists open ExtendSim windows—models, dialogs, and library windows—at the bottom. To bring a window to the top of your workspace, select it from this menu.

Navigator

Opens or brings forward the Navigator for the active model. This is the same as choosing the Navigator button in the toolbar. The Navigator is an explorer-type interface for a model worksheet, showing all blocks and hierarchical layers. For more information about the Navigator, see “Navigator” on page 110.

Database List

Opens or brings forward a window showing the databases (if any) used in the model. This is the same list of databases that is shown at the bottom of the Database menu. Databases and their usage are discussed at “ExtendSim databases for internal data storage” on page 232.

Open Enclosing Hierarchical Block

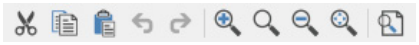
Finds the hierarchical block that contains the selected block. If the enclosing hierarchical block is at the top level of the model worksheet, the hierarchical block will be selected (shows as highlighted) on the worksheet. If the enclosing hierarchical block is below the top level of the worksheet, the command opens the hierarchical block’s worksheet, showing the submodel that contains the selected block.

Tools menu

File toolbar

The standard buttons for opening a new model, opening an existing model, saving, and printing.

Edit toolbar

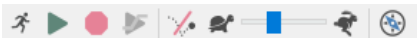


The standard Cut, Copy, Paste, Undo, Redo, and Find buttons. Between the Undo and Find buttons are buttons for zooming in and out, returning to normal size, and zoom to fit.

Hint: on the model worksheet, hold down Ctrl while moving the mouse's scroll wheel to zoom in and out.

☞ What happens when you use Cut, Copy, and Paste depends on the cursor that is used. For example, using the Block/Text cursor to frame-select and copy disparate items (blocks, text, graphic objects, pictures, etc.) will only copy the blocks and the text. Using the All Objects cursor would copy all the items.

Model toolbar



Controls simulation and animation behavior and opens the Navigator.

- The first button is the Run Mode button. As discussed in "Run modes" on page 87, toggle to fastest run mode (shown above) when running a single model one or more times and use the multi-threaded run mode when running multiple models at the same time.
- The Run Simulation button:
  - Is either a single green arrow if the Run Mode (discussed above) is set to fastest run or multiple arrows if the Run Mode is set to multi-threaded. Use the fastest run for single models, even if running them multiple times. Use multi-threaded if you are running multiple models at the same time.
  - Changes to a Pause button while the simulation is running and to a Resume button if the simulation is paused.
- The Run Optimization/Scenarios button is only enabled if the active model contains the Scenario Manager or the Optimizer block; both are from the Value library. Furthermore, you cannot run optimization and scenarios at the same time and ExtendSim will warn if you use this button with both an Optimizer and a Scenario Manager in the model.
- The Animation Slider and the Faster and Slower Animation icons apply only to 2D animation.
- The last button on the right, the Navigator, presents an explorer-type representation of the model; it is discussed on page 110.

Text toolbar

For setting the font and its style (such as Bold or Underline) and size.

- For model-wide use, set the font and format before creating any text boxes

- Create a text box by double-clicking on the model worksheet or by selecting the Text Box button in the Shapes tool.
- The Align Text button, shown at right, cycles through options for the selected text: left adjusted, centered, or right adjusted. 
- By default text is drawn with a transparent background. Toggle the Transparent Background button, shown to the right of the Align Text button above, to cause the background of the selected text to be an opaque white.

☞ For text files, which by nature don't retain font information, set the default font and size in the Edit > Options > Model tab. That option does not apply to the text boxes discussed above.

Shapes toolbar

For drawing and coloring shapes, lines, and text and for adding borders to shapes and text. For more information, see "Graphic shapes, tools, and commands" on page 107.

☞ To get a square or circle, right-click a Rectangle or Oval shape to access its Properties. Then set the object's width and height to be the same.

☞ By default, text and shapes other than lines do not have a border. Before selecting a border color for text or a shape, choose a border width using the Line/Border Thickness button.

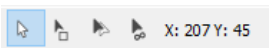
Alignment toolbar

Use these tools to manipulate graphic objects such as shapes, text, and lines. The buttons allow you to align selected text and graphics, cause an overlapping object to be sent to the back or brought to the front, flip a selection of multiple objects, and rotate a selected object 90 degrees from where it is or (Free Rotate) a specified number of degrees from zero.

☞ Depending on which cursor is used, different behaviors will occur. For example, using the Block/Text cursor to frame-select and align disparate items (blocks, text, graphic objects, pictures...) will only align the blocks and the text. Using the All Objects cursor would align all the items.

⚠ The Flip buttons only flip the relative positions of multiple objects that have been selected together. They do not result in the displaying of the reverse image of a graphic. Use an application such as Photoshop to create reverse images of graphics.

Cursors toolbar



Displays the Block/Text, Graphics, Clone, and All Objects cursors. Also shows the position of the cursor on the model worksheet and on the Icon and Dialog tabs of a block's structure.

☞ Depending on which cursor is used, different behaviors will occur. For example, using the Block/Text cursor to frame-select and copy disparate items (blocks, text, graphic objects, pic-

tures, etc.) will only copy the blocks and the text. Using the All Objects cursor would copy all the items.

Library Windows toolbar

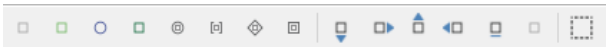
These two tools are located by default at the far right of the toolbar, over the library window docking area. Use them to open or close the library windows for all open libraries.

☞ Opening or closing a library window does not open or close the associated library. See the Library menu for opening and closing a library.

Dialog Items toolbar

Inserts dialog items (buttons, checkboxes, popup menus, controls, etc.) used when creating a block's dialog. Since this set of buttons is only used when building or modifying blocks, the tool is unchecked by default. See the *Technical Reference* for information about each of these dialog items and how they are called from the block's code.

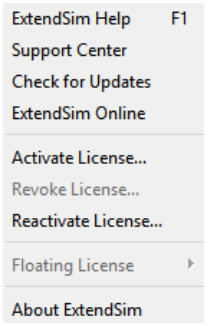
Icon toolbar



Inserts connectors and animation objects used when creating a block's icon. Since this set of buttons is only used when building standard or hierarchical blocks, the tool is unchecked by default. The connectors are described on page 61.

Help menu

This menu has commands for accessing ExtendSim Help and useful web sites as well as for activating your ExtendSim license.



ExtendSim Help

Opens a window with a list of available Help topics.

Support Center

Uses your current browser to open an ExtendSim web page with support resources—example models, FAQs, manuals, and more.

☞ Use the form on this web page to initiate a support ticket. Your query will be dealt with by an experienced Tech Support representative and answered as quickly as possible.

Check for Updates

Uses your current browser to open an ExtendSim web page with information about the latest release of ExtendSim.

☞ Your product's current release number is located on the startup window. Select the **About ExtendSim** command in the Help menu.

ExtendSim Online

Uses your default browser to open the home page at www.ExtendSim.com.

Activate License

Opens the Activation Dialog so you can enter an Activation Key for an Individual license of ExtendSim. The Activation Key represents your license for ExtendSim as well as which ExtendSim product you purchased.

- If your device has Internet access and its security system doesn't block the process, activation can occur automatically.
- If automatic activation is blocked, ExtendSim will display an alternate method so you can activate the license manually.
- License activation is tied to the computer upon which activation is performed; it is granted and handled by our activation server and database.

⚠ In most cases Individual licenses of ExtendSim may be installed on only one device and activation on a second device will fail unless the license has been revoked from the original device.

Revoke License

Use this command to remove your license from your current device.

- Revoke the license from the current device before selling or decommissioning it
- Then activate ExtendSim on a different device using the same Activation Key

⚠ Revoking the license does not uninstall ExtendSim. Use the normal operating system procedures to uninstall ExtendSim from your device.

Reactivate License

⚠ Do not use this command if you are updating ExtendSim from one minor release to another, such as from 10.x to 10.y.

Only use this command for an Individual license and only if there has been a major change to the actual license, such as:

- A change in the expiration date for the annual Maintenance and Support Plan
- A change in the number of Reliability Event Cycles

The command tries to automatically revoke the current license over the internet without exiting ExtendSim, then immediately tries to reactivate it using the same Activation Key as before. If revoking and reactivation cannot be accomplished automatically, ExtendSim will display an alternate method so you can perform the process manually.

⚠ Do not use this command if you are changing the number of concurrent users for a Floating license or to increase the number of concurrent instances for a Cloud license.

Floating License

This command is only enabled if you have a Floating license, which is controlled by License Manager. As opposed to an Individual license, a Floating license allows multiple installations of ExtendSim and multiple concurrent users.

Check Out a Floating License
Allows you to check out a computer's Floating license, temporarily disconnecting that ExtendSim license from the License Manager. This is most often done for a laptop that is to be used offsite. Checking out a license reduces the maximum number of concurrent users the License Manager allows. That number will be restored when the license is properly checked back in.

Check In a Floating License
After reconnecting the client computer to the server that hosts the License Manager, use this command to return a license that had been temporarily checked out from the network. This also restores the number of concurrent users the server allows.

About ExtendSim (Windows only)
Opens the startup window which lists information about your ExtendSim product and its license. To close the startup window, click it.

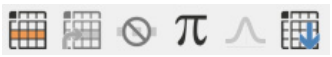
☞ The **About ExtendSim** window also lists the ExtendSim product's release number.

ExtendSim database window toolbars and buttons

Database window - Schema view

Buttons for adding a new table, adding or inserting fields in a table, showing and hiding all tables, or cleaning up the display of visible tables so none overlap.

Database window - Data view



Buttons for appending, inserting, or deleting records, making a cell random or removing randomness, and sorting a table's data.

☞ If you double-click a table, these buttons also appear in the Table Viewer except the Viewer has an additional button to open or bring forward the database window.

Appendix

Value Library Blocks

A detailed description of the
building blocks in the Value library

This chapter provides tables of the blocks in the Value library, separated by category. Each Value library block has an icon that represents its function, predefined input and output connectors for quick model building, and a dialog for entering parameters and viewing results.

The tables have brief descriptions and are useful to get an idea of a block's functionality in your model. For more details about the usage of a block:

- Click the Help button in the lower left of the block's dialog
- Look in the index of ExtendSim's online Help for the block's name

Submenus

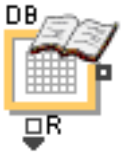
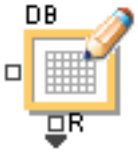
The blocks are listed by the block categories displayed in hierarchical submenus of the Value library menu:

- Data Access: Accessing global data
- Holding: Accumulating or storing values
- Inputs: Generating values
- Math: Calculating values
- Model Analysis: Finding the best solution
- Outputs: Writing data to files or display
- Routing: Routing or deciding which value to use
- Statistics: Calculating mean, variance

Data Access

The blocks in this category are used to access and store data in your models.

Block	Function
Data Import Export 	Enables the direct exchange of data between internal ExtendSim files (ExtendSim databases or global arrays) and external files (Excel, text files, ADO and ODBC compatible databases, and files from the Internet via FTP.)
Data Init 	Defines any values needed to initialize multiple database tables and global arrays before a simulation run starts.
Data Source Create 	Creates, resizes, deletes, and views global arrays and text files. Global arrays are general purpose arrays available anywhere in the model. Text files are ascii files containing text data, used as a data source, that can be saved on the computer's local hard drive or on the network. Note: If you are interested in creating, editing, or viewing database tables, use the commands in the database menu.

Block	Function
Data Specs 	Outputs selected specifications on data sources. Data sources can be either ExtendSim databases or global arrays. Each row in the table defines a specification whose value will be output on the corresponding variable output connector on the block.
Read 	Reads data from a data source to be used in a model. The data sources supported are the ExtendSim database, global arrays, Excel workbooks, Text Files, and local tables. You can specify whether you want to read a single number or a row or column of data and you can specify when the data should be read.
Write 	Writes data from a model to a data destination. The data destinations supported are: ExtendSim databases, global arrays, Excel Workbooks, Text Files, and Local Tables. You can specify whether you want to write a single number or a row or column of data, and when the data should be written.

Holding

The blocks in this category are used to accumulate or store contents.

Block	Function
Holding Tank 	Accumulates the total of the input values, and allows you to request an amount to be removed if it is available. You can also choose to allow a request that would make the contents go negative (such as an overdraft). You can specify that the inputs are summed or integrated.
Wait Time 	Holds its inputs for a specified amount of simulation time (the delay) before passing them to the output. This block works like a conveyor with slots: values come into a slot, advance position based on an advance in simulation time, then exit when their slot reaches the end of the conveyor. Note: This block should not be used in a discrete event model.

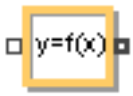
Inputs

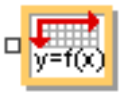
The blocks in this category generate values to be used as inputs for other blocks.

Block	Function
Constant 	Generates a constant value at each step. You specify a constant value in the dialog (the default constant is 1.0). This block is typically used for setting the value for inputs to other blocks. For example, you can use it for a steady flow of fluid, cash, or a delay time value. If the ValueIn input on the left is connected, the input value is added to the constant in the dialog and the sum of those two numbers is output.
Pulse 	Outputs a true value (1) at specified times, and a false value (0) at all other times. In the dialog, you specify the time between outputting true values (the delay or time out); the dialog value is overridden by the D connector. The R connector resets the block back to the beginning of the delay period.
Random Number 	Generates random integers or real numbers based on the selected distribution. You can use the dialog or the three inputs, 1, 2, and 3 to specify arguments for the distributions. You can select the type of distribution or use an Empirical Table. The Empirical distribution uses a table to generate a discrete, stepped, or interpolated distribution.
Simulation Variable 	Outputs the value of a simulation variable. It is usually used in conjunction with a decision-type block, for example, to halt a process after current time reaches a certain value. The variables you can use are: current run number, current step, current time, end time, number of runs, number of steps, start time, time step, and random seed.

Math

The blocks in this category are used to perform mathematical calculations and functions.

Block	Function
Decision 	Makes a decision and outputs TRUE or FALSE values based on the inputs and defined logic. The dialog lets you perform the following tests comparing A to B: greater than, greater than or equal to, equal to, less than, less than or equal to, and not equal. You can also test for A being an invalid number (noValue). The block can be set to use hysteresis.
Equation 	Outputs the results of the equations entered in the dialog. You can use ExtendSim's built-in operators, functions, and some or all of the input values as part of the equation. The equations can have any number of inputs and any number of outputs.

Block	Function
Integrate 	Integrates the input values over time using either Euler or Trapezoidal integration methods. If present, an initial value is added to the outputs.
Lookup Table 	Acts as a lookup table (x in and y out) that are used to calculate what the output value would be for the given input. Input values can come from an input connector (the default) or can be simulation time. The output can be discrete, interpolated or stepped.
Math 	Performs a selected mathematical operation on its inputs and outputs a result.
Max & Min 	Determines the maximum and minimum values from among the values input. The dialog shows the maximum and minimum values and the input connectors they came from. The block outputs the maximum or minimum values and the respective connector number.
Query Equation 	Intelligently searches a database and reports the best-ranked record and its field values. Outputs to database or connectors.
Time Unit 	Converts the value at the input connector from one time unit to another.

Model Analysis

The blocks in this category are used to find optimum parameter and input values for your simulation.

Block	Function
Optimizer 	Searches for the best set of model parameters that maximizes profit or minimizes cost, given parameter limits and any entered constraints. Uses evolutionary strategies that are similar to genetic algorithms.

Block	Function
Scenario Manager 	Evaluate and compare different model configurations with results.

Outputs

The blocks in this category output data to files or to display.

Block	Function
Command 	Sends Excel macro or other commands to a spreadsheet application when triggered by the “Send” connector.
Display Value 	Displays the value at the input connector on each simulation step. This is useful for debugging models and scripts because you can display the value of a block's value output connector at any time.
Notify 	Notifies the user when an event occurs. The notification can take the form of playing a sound, stopping the simulation, or querying the user for a new input value.

Routing

The blocks in this category route values or decide which values to use.

Block	Function
Catch Value 	Outputs values thrown from a Throw Value block. You specify in the dialog of the block which throw block(s) this block is connected to. This block is typically used for passing a value from one point in a model to another without using connectors, in conjunction with Throw Value blocks.
Select Value In 	Selects its output value to be one of its inputs according to the value of the select connector. Up to 253 inputs can be used.
Select Value Out 	Sends its input value to one of its outputs according to the value of the select connector. Up to 253 outputs can be used.

Block	Function
Throw Value 	Throws a value to one or more catch blocks in the model. You specify in the dialog of the block which catch block(s) this block is connected to. This block is typically used for passing a value from one point in a model to another without using connectors, in conjunction with Catch Value blocks.

Statistics

The blocks in this category report and clear statistics on various blocks.

Block	Function
Clear Statistics 	Clears the statistics in various blocks in a model at a specific time or event. Useful in reducing the effects of warm-up in a model.
Mean & Variance 	Calculates the mean, variance, and standard deviation of the values input during the simulation.

Appendix

Item Library Blocks

A detailed description of the
building blocks in the Item library

This chapter provides tables of the blocks in the Item library, separated by category. Each Item library block has an icon that represents its function, predefined input and output connectors for quick model building, and a dialog for entering parameters and viewing results.

The tables have brief descriptions and are useful to get an idea of a block's functionality in your model. For more details about the usage of a block:

- Click the Help button in the lower left of the block's dialog
- Look in the index of ExtendSim's online Help for the block's name

Submenus

The blocks are listed by the block categories displayed in hierarchical submenus of the Item library menu:

- Activities: Processing items
- Batching: Joining and dividing items
- Data Access: Accessing global data
- Information: Getting information about items
- Properties: Assigns and displays properties for items
- Queues: Holding, sorting, and ranking items
- Resources: Representing items as resources
- Routing: Moving items to the correct place
- Executive: Needed in every discrete event and discrete rate model to handle events

Activity

The blocks in this category are used to process items in the model.

Block	Function
Activity 	Holds one or more items and passes them out based on the process time and arrival time for each item.
Convey Item 	Behaves as a conveyor (accumulating or non-accumulating) that moves items from one location to another.
Transport 	Transports item from one point to another based on distance and speed information.

Block	Function
Workstation 	Behaves as a workstation that has both processing and queueing aspects.

Batching

The blocks in this category are used to join and divide items.

Block	Function
Batch 	Allows items from several sources to be joined as a single item. Useful for synchronizing resources and combining various parts of a job (“kitting”).
Unbatch 	Produces multiple items from a single input item. This block can be used to disassemble a kit, break a message packet into component messages, route the same message to several places, or distribute copies of invoices.

Data access

The blocks in this category are used to access and store data in your models.

Block	Function
Read(I) 	Reads data from a database when an item arrives. You can define an indefinite number of reads to be made by the block when an item passes through.
Write(I) 	Writes data to a database when an item arrives. You can define an indefinite number of writes to be made by the block when an item passes through.

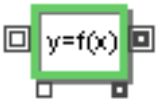
Information

The blocks in this category provide information about the blocks in your model.

Block	Function
Cost By Item 	Views and displays the cost of the cost accumulators that pass through it. By using a sorting attribute or the row connector, the throughput, average cost, and total cost can be calculated for different item types.
History 	Views and displays information about the items that pass through it. You specify which properties will be displayed. Properties can be attributes on the item, priority values, or other more obscure values that are available on the item.
Information 	Reports statistics about the items that pass through it, such as cycle time and TBI (Time Between Items).

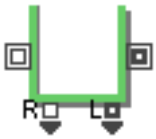
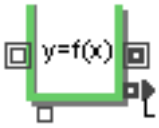
Properties

The blocks in this category assign and display item properties.

Block	Function
Equation(I) 	Calculates equations when an item passes through. The equations can use multiple inputs and properties of the item as variables, and the result(s) of the equations can be assigned to multiple outputs and properties of the item.
Get 	Displays and outputs properties from items that are passing through. The property value is shown in the dialog and output at the value output connector. You can specify multiple properties and multiple output connectors.
Query Equation(I) 	Intelligently searches a database and reports the best-ranked record and its field values when an item arrives. Outputs to database, properties, or connectors.
Set 	Sets the properties of items passing through the block from input connectors, values in the dialog, or databases.

Queues

The blocks in this category hold, sort, and rank items.

Block	Function
Queue 	Queues items and releases them based on a user selected queuing algorithm, such as Resource pool queue, Attribute value, First in first out, Last in first out, and Priority. Options include reneging and setting wait time. If you need more advanced control over the queueing algorithm, consider using the Queue Equation block, below.
Queue Equation 	Queues items and releases them based on the results of user entered equations. The result(s) of the equations can optionally be assigned to properties of the item
Queue Matching 	This block is useful for matching one type of item with another, such as in reassembling parts in the correct order or to insure that subassemblies are correctly matched with each other. Has a specified number of internal queues for holding items in separate groups. Releases a group when there is downstream capacity and the group requirements have been met.

Resources

The blocks in this category represent resources.

Block	Function
Resource Item 	This block holds and provides items (cars, workers, orders, and so forth) to be used in a simulation. It can be used as part of an open or closed system.
Resource Manager 	Develop sophisticated resource structures and requirements.
Resource Pool 	This block holds resource pool units to be used in a simulation. These units limit the capacity of a section of a model. For example, this could be used to represent a limited number of tables at a restaurant. The Resource Pool block works with the Queue block in Resource Pool mode and the Resource Pool Release blocks.

Block	Function
Resource Pool Release 	Releases a resource back to its resource pool as an item passes through.
Shift 	Generates a schedule over time which can be used to change the capacity of other blocks in the model. Multiple Shift blocks can be connected together to create complex shift patterns. For example the typical 40 hour work week can be built with two connected Shift blocks, the first containing the work days, the second contains the work hours.
Shutdown 	Generates shutdown information for activities in the Item library and valves in the Rate library according to inputs or distributions specifying time between failures and time to repair.

Routing

The blocks in this category move items to the correct place.

Block	Function
Catch Item 	This block catches items sent by Throw Item blocks without using connection lines. Any number of Throw Item blocks can send items to a Catch Item block.
Create 	Provides items or values for a discrete event simulation at specified interarrival times. Choose either a distribution or a schedule for the arrival of items or values into the model.
Exit 	Passes items out of the simulation. The total number of items absorbed by this block is reported in its dialog and at the value output connectors.
Gate 	Limits the passing of items through a portion of the model, either by demand or by using a sensor connector to monitor how many items are in a section of the model.

Block	Function
Select Item In 	Selects items from one input to be output based on a decision.
Select Item Out 	Selects which output gets items from the input, based on a decision
Throw Item 	This block throws items to a Catch block without using connection lines. Any number of Throw blocks can send items to a single Catch block. You could use the Throw and Catch blocks instead of using Combine blocks, even from inside one hierarchical block to inside another one.

Executive

The block in this category is needed in every discrete event and discrete rate model to handle events.

Block	Function
Executive 	This block must be placed to the left of all other blocks in discrete event and discrete rate models. It does event scheduling and provides for simulation control, item allocation, attribute management, and other discrete event and discrete rate model settings.

Appendix

Rate Library Blocks

A detailed description of the
building blocks in the Rate library

This chapter provides a brief description of the blocks in the Rate library. There are no block categories used in this library. For more details about the usage of a block:

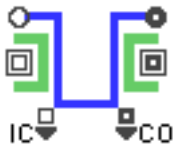
- Click the Help button in the lower left of the block's dialog
- Look in the index of ExtendSim's online Help for the block's name

Block descriptions

The following table is useful to get an idea of a block's functionality in your model. More details about usage of a block can be obtained in two ways:

- Look in the index of ExtendSim's online Help for the block's name
- Click the Help button in the lower left of the block's dialog

Block	Function
Bias 	Prioritizes the flow going through it, and thereby allows you to specify a preference for where the model's flow should be directed. The bias from a Bias block is by definition stronger than the bias from any Merge or Diverge block.
Catch Flow 	Receives flow from non-connected Throw blocks. The Throw Flow and Catch Flow blocks (and the Merge and Diverge blocks in certain modes as well) can be used to move flow without the use of connection lines.
Change Units 	Changes the flow unit of measurement. Blocks that are connected together through flow connections and share the same flow unit are called a unit group. The Change Units block uses the conversion factor to create a new unit group. This causes the blocks downstream of the Change Units block to be in a unit group different from its upstream blocks.
Convey Flow 	Add a delay to the flow available at the output of the block. The Convey Flow is FIFO and can be accumulating or non-accumulating.
Diverge 	Distributes the input flow to two or more outputs. Systems modeled using discrete rate technology frequently have one flow stream that needs to be split (or diverged) into multiple streams (referred to as branches). It has seven different rule-based options that define how the inflow will be distributed across the outputs.
History(R) 	Views and displays information about the flow that passes through it. The block has several filter, display, and alert options as well as an option to store the history in a database at the end of the run.

Block	Function
Interchange 	Acts as a Tank interfacing between Flow and Items. The Tank acts as source, intermediate storage, or sink.
Merge 	Merges flows from multiple inputs into one output. Systems modeled using discrete rate technology frequently have multiple flow streams (referred to as branches) that need to be merged into one stream. It has seven different rule-based options that define how the inflow will be merged from all inputs.
Sensor 	Displays potential upstream supply and potential downstream demand in a Flow branch.
Tank 	Acts as source, intermediate storage, or sink. As a residence type block the Tank has the capacity to hold defined amounts of flow as time advances. If a Tank has no outflow connection, by definition it is being used as a sink. Conversely, if a tank has no inflow connection, by definition it is being used as a source.
Throw Flow 	Sends flow to non-connected Catch blocks. The Throw Flow and Catch Flow blocks (and the Merge and Diverge blocks in certain modes as well) can be used to move flow without the use of connection lines.
Valve 	Controls, monitors, and transfers flow. This block places an upper bound on the rate at which flow is allowed to pass through. Goals can be set up to control flow.

Appendix

Utilities Library Blocks

A detailed description of the
building blocks in the Utilities library

This chapter provides tables of the blocks in the Utilities library, separated by category. Each Utilities library block has an icon that represents its function, predefined input and output connectors for quick model building, and a dialog for entering parameters and viewing results.

The tables have brief descriptions and are useful to get an idea of a block's functionality in your model. For more details about the usage of a block:

- Click the Help button in the lower left of the block's dialog
- Look in the index of ExtendSim's online Help for the block's name

Submenus

The blocks are listed by the block categories displayed in hierarchical submenus of the Utilities library menu:

- Developer Tools: Provides information about OLE objects
- Discrete Event Tools: Managing and reporting on discrete event models
- Information: Getting information about the model
- Math: Performing mathematical calculations
- Model Control: Controlling how the model runs
- Time: Working with time functions

Developer Tools

The block in this category provides information about OLE objects.

Block	Function
Object Mapper 	Gets information about IDispatch properties and methods exported by activeX controls or objects that have been automated. It is useful for those needing to navigate the object model of an outside application via Extend's OLE programming capabilities.

Discrete Event Tools

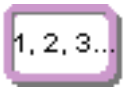
The blocks in this category are used to manage and report on discrete event models.

Block	Function
Event Monitor 	Displays the event calendar in discrete event models, revealing the event times for all event posting blocks.
Item Messages 	Displays detailed information about the messages used to pass items. This block is "invisible" to the items and messages and passes each message it gets on to the next block. Also see the Record Message block, below.

Block	Function
Link Alert 	Registers a section of a database so that the block receives a message when data in the linked database changes.
Queue Tools 	Views and optionally initializes the contents of a queue via tables. Connect the input connector to the length (L) connector on a queue (this is the only type of connector that can be connected to this block).
Record Message 	Records all the messages that pass between two value connections in a discrete event model. Also see the Item message block, above.
Stop Message 	Transfers the value from the input to the output without relaying any discrete event messages. The output connector is set equal to the input connector, but the message is not sent out through the output connector.

Information

The block in this category provides general model information to the user.

Block	Function
Model Compare 	Used with the mini-app Compare Results. Reports a summary of any differences between models run in different releases of ExtendSim.
Block Info 	Reports information (name, number, position, etc.) about the block connected to its input connector.
Count Blocks 	Calculates the number of blocks of each type in the model.
Find and Replace 	Drag a clone of a dialog item onto this icon to search for similar dialog items. You can also manually enter search criteria into the dialog.

Block	Function
Memory Usage 	Calculates the amount of memory required for each block, global array, or database table in the model. It is helpful in locating where memory is being used in large models.

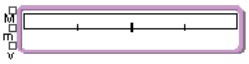
Math

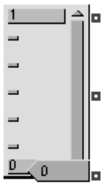
The blocks in this category perform mathematical functions.

Block	Function
Data Fitter 	Uses matrix techniques to obtain a least mean square curve fit to a set of data. Both the data and the fitted curve are output during a simulation through the output connectors (D and F, respectively).

Model Control

The blocks in this category provide various means of controlling the simulation run.

Block	Function
Buttons 	Creates a pushbutton interface for a model. An equation is executed whenever the button is pushed. The button itself can be cloned to the model worksheet, notebook, or hierarchical block to create a user interface for the model.
Feedback 	Helps resolve feedback calculation order issues in continuous models. This is only for advanced use and only to be used in situations where you know that you have a calculation problem that is based on the simulation order of a feedback loop.
Meter 	Displays the input relative to minimum and maximum values set in its dialog.The color and pattern can change when the current value reaches specified values. Useful as a progress bar or to animate hierarchical blocks.
Model Interface 	Creates an interface for controlling a model.
Pause Sim 	Causes the simulation to pause when certain conditions are met.
Popups 	Allows you to define a custom popup menu that you can clone to the worksheet and also use as a numeric input to your model. Outputs the value of the popup defined in the dialog, which is usually cloned to the worksheet.

Block	Function
Run Model 	Primarily used to clone out the Run Simulation Now button onto the worksheet or notebook. This can also be done with the Buttons block. Runs the simulation when the “Run Simulation Now” button is pressed.
Slider 	Provides a flexible slider for models. It can be cloned to the worksheet or Notebook, be linked to a database, and more.
Switch 	Defines a switch for use on a model worksheet. Differs from the use of the switch control defined in the model menu in that it sends out a message if it is in a discrete event model.This will have the effect of making the other blocks in the model react immediately to the switch click.

Time

The blocks in this category work with time functions in the model.

Block	Function
RealTimer 	Reports information about simulation run times, including duration of the run and profiles of the execution time and memory allocation for blocks in the model. It should be placed at the far right side of the model worksheet.
TimeSync 	Synchronizes the model to run in real time. It does this by pausing on each simulation step until the amount of simulation time that has passed equals the amount of real time that has passed. This is only effective if the model is running faster than real time.

Appendix

Upper Limits

A list of the maximum numbers of things
that you can do at one time

Like every program, ExtendSim has its limits. It is unlikely you will find them in your normal work, but it is good to know what they are. Note: Some limits depend on available memory.

Steps in a simulation run	2 billion
Total model data size	2.8E14 bytes
Number of simulations in a multiple run	2 billion
Block name or label length, characters	31
Blocks in a model	2 billion
Blocks in a library	200
Libraries open at one time	40
Text files open at one time	200
Databases per model	2 billion
Tables per database/fields per table/records per table	2 billion/1,000/2 billion
Output connectors in a model (nodes)	2 billion
Connectors per block	255
Length of a block's ModL code (characters)	10 megabytes
Dialog items in a block	1024
2D animation objects	255/icon; unlimited through function
Dynamic arrays (each array)	2 gigabytes
Number of array dimensions	5
Maximum index for array dimensions	2 billion elements total
Dynamic arrays per block	255
Columns in a table	255 (data table); 255 (text table)
Total table size (cells) per block for static data tables	3260 (data table); 2030 (text table)
Total table size (cells) each, for dynamic data tables	2 billion
Variable name length: dialog item msg, connector names	63
Variable name length: ModL local, static variables	127
Static and local variable declarations	32,767 bytes
Maximum popup menu length	5100 characters or 20 strings
User defined function arguments	127
Nested loops	32
Maximum, minimum of real numbers	±1E±308

Maximum, minimum of integer numbers	±2,147,483,647
Significant figures in real calculations	16 (double)
Number of attributes for discrete event item	500

Index

Symbols

Numerics

A-C

D-F

G-I

J-L

M-N

O-P

Q-S

T-V

W-Z