



Professional simulation tools

Version 5 User's Guide for:

*Extend
Extend+Manufacturing
Extend+BPR
Extend+Industry
Extend Suite
Industry Suite*

For Windows or Macintosh

A hand-drawn style logo consisting of the words "Imagine That!" in a cursive script. The "I" in "Imagine" and the "T" in "That!" have small registered trademark symbols (®) at their top right ends.

Imagine That!, Inc. • 6830 Via Del Oro, Suite 230 • San Jose, CA 95119 USA
Telephone 408-365-0305 • FAX 408-629-1251
Email: extend@imaginethatinc.com • Web Site: <http://www.imaginethatinc.com>

Copyright © 2000 by Imagine That, Inc. All rights reserved. Printed in the United States of America.

You may not copy, transmit, or translate this manual or any part of this manual in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use without the express written permission of Imagine That, Inc.

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. In addition, please note the following:

- The Extend BPR, Manufacturing, Industry, and Statistics libraries are add-ons to the Extend simulation application. If you or your company want to use the BPR, Manufacturing, Industry, and/or Statistics libraries on more than one machine, you must purchase a separate library license for each machine or obtain a multi-user concurrent or networked license. For example, if you want to use the BPR, Manufacturing, Industry, and/or Statistics libraries on both a Macintosh and a Windows computer you need to obtain separate Extend, BPR, Manufacturing, Industry, and/or Statistics licenses for each computer.
- Extend blocks (including icons, dialogs, and block code) are copyright © by Imagine That, Inc. and may contain proprietary and/or trademark information. If you build blocks, and you use any portion of the Extend, BPR, Manufacturing, Industry, and/or Statistics library blocks in your blocks or you include BPR, Manufacturing, Industry, and/or Statistics library blocks (in whole or in part) in your libraries, your right to sell, give away, or otherwise distribute your libraries is limited. For example, you are not allowed to include blocks from the BPR, Manufacturing, Industry, and/or Statistics library in your libraries, then sell or give your library to someone who does not have a legal copy of Extend and a legal copy of the library you have copied blocks from. For more information, contact Imagine That, Inc.

Imagine That! is a registered trademark and Extend and ModL are trademarks of Imagine That, Inc. Macintosh is a registered trademark of Apple Computer, Inc. Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation. All other product names used in this manual are the trademarks of their respective owners.

Extend was created by Bob Diamond

Extend 5 was designed and written by Bob Diamond, Steve Lamperti, and Dave Krahl

The Extend Discrete Event library architecture was created by Steve Lamperti and Dave Krahl

Extend libraries designed and written by: Dave Krahl, Steve Lamperti, Anthony Nastasi, and Bob Diamond

Contents

Preface	E1
Introduction	E3
Modeling and simulation	E4
Why simulation is important	E4
Extend and spreadsheets	E5
What Extend can do	E5
New features added in version 5	E6
Extend + Manufacturing enhancements	E8
Using Extend	E8
Levels of use	E8
Libraries that come with Extend.....	E8
What this manual doesn't do	E9
How to start	E10
On-line help.....	E10
Technical support.....	E10
Chapter 1: Running a Model	E13
Extend documents.....	E14
Model overview	E15
Starting Extend and opening a model.....	E15
Tutorial	E16
Tutorial, part 1 (all Extend users)	E16
Model basics	E17
Blocks	E17
Libraries.....	E18
Dialogs.....	E19
Connectors and connections	E20
Exploring the Lake Pollution model	E20
Running a simulation.....	E22
Simulation status.....	E22
Plotters.....	E23
Changing assumptions	E24
Adding and removing blocks	E24
Changing dialog parameters.....	E25
Constant block.....	E26
Input Data block.....	E26
Changing a pollution source	E27
Other modifications	E27
Tutorial, part 2 (discrete event users)	E28
Dialogs.....	E29
Exploring the "Bank Line" model.....	E30
Running the simulation	E31
Plotters.....	E31
Changing assumptions	E32
Adding and removing blocks	E32

Contents

Changing dialog parameters	E33
Activity Delay block	E34
Generator block	E35
Queue, FIFO block	E36
Adding a random element	E36
Other modifications	E37
Chapter 2: Building a Model	E39
The Lake Pollution model	E40
Adding blocks	E41
Opening the libraries	E41
Adding a block to the model	E42
Moving blocks	E43
Connecting blocks	E43
Other types of connections	E46
Putting together the model	E46
Random pollution source	E46
Constant pollution source that stops	E48
Combining the sources	E50
Pollution in the lake	E50
Determining the outflow	E51
Taking out the pollutants from outflow	E52
Saving the model	E53
Labeling the blocks and adding comments	E54
Adding the plotter	E54
Running the simulation	E55
Simulation Setup dialog	E55
Adding a second plotter	E58
Additional ways of connecting blocks	E59
Plotting to the Y2 axis	E61
Other modifications	E62
Other examples	E63
Chapter 3: Enhancing your Model.....	E65
Tools and buttons	E67
Files	E67
Editing	E67
Models	E67
Selecting	E68
Patterns and colors	E68
Color palette	E68
Text	E69
Working with the text box	E69
Drag and drop editing	E71
Drawing	E71
Borders	E72
Shuffle front to back	E72
Connection line characteristics	E73
Cloning dialog items onto the model	E76
Cloning dialog items	E77

Using cloned items.....	E78
Unlinked clones	E78
Notebooks.....	E79
Hierarchy	E79
Introduction to hierarchical blocks.....	E79
Viewing a hierarchical block.....	E81
Animation	E81
Built-in animation.....	E82
Animation on the block's icon	E82
Animation between block icons.....	E82
Animation for debugging block code.....	E83
Custom animation	E83
Animation Library blocks.....	E83
Animation functions	E83
Sensitivity analysis	E83
Optimization	E85
Adding optimization to your model	E86
Equation editor	E86
Control blocks	E88
Slider.....	E88
Switch	E90
Meter	E90
Other authoring features	E91
Locking the model	E91
Extend RunTime and Player versions	E91
Sending messages to the user	E92
Stop block.....	E92
Sound block.....	E93
Prompt block	E94
Adding help to the model.....	E94
Additional features if you program	E94
Chapter 4: Fundamental Continuous and Discrete Event Modeling	E97
Choosing the Generic or Discrete Event library.....	E98
Continuous and discrete event modeling defined	E98
Table of continuous and discrete event differences.....	E99
Generic library	E101
Using the Generic library	E101
Blocks by type in the Generic library.....	E101
.....	E102
Arrays.....	E102
Decision blocks.....	E102
Holding blocks.....	E102
I/O blocks.....	E103
Math blocks	E104
Generic library example	E105
Starting with a simple model.....	E105
Statistics blocks	E105
Adding the lynx predator	E106
Making the lynx population variable	E107

Contents

Refining the hare mortality factor.....	E108
Refining the lynx mortality factor.....	E109
Further exploration	E111
Discrete Event library	E112
Using the Discrete Event library.....	E113
Layout of your model.....	E113
Items and informational values.....	E113
Events	E114
Item connectors	E114
Attributes.....	E114
Priorities.....	E116
Item Values	E116
Modifying blocks in the library	E117
Moving items through a simulation.....	E117
Holding and pushing	E117
Pulling and viewing.....	E118
Pitfalls to avoid	E118
Common connectors in the Discrete Event library	E119
Universal input connectors.....	E120
Blocks by function in the Discrete Event library.....	E121
Activities	E121
Attributes.....	E122
Batching.....	E122
Generators	E122
Information.....	E123
Queues.....	E123
Resources	E124
Routing.....	E124
Executive.....	E125
Discrete Event library example	E125
Basic discrete event model.....	E126
Adding multiple servers to a queue.....	E127
Attributes.....	E128
Priorities.....	E131
Values.....	E132
Getting data from files	E133
Further exploration	E135
Beyond this manual	E135
Chapter 5: Using Libraries.....	E137
Extend's libraries	E138
Generic, Discrete Event, and Plotter libraries	E138
Other libraries included with Extend.....	E138
Hierarchical blocks in libraries	E139
Additional library modules.....	E139
Library usage and maintenance	E140
Making a new library	E140
Opening and closing libraries	E140
Changing libraries	E141
Searching for libraries and blocks	E142
Library searches.....	E142

Block searches	E143
Library windows	E144
Maintaining blocks in libraries	E145
Organizing blocks in libraries	E146
Listing blocks by type or alphabetically	E146
Organizing blocks you build yourself	E146
Saving and compiling libraries.....	E147
Protecting the code of library blocks.....	E147
Chapter 6: Input and Output	E149
Plotters.....	E150
Plot and data panes	E151
Plot pane.....	E151
Data pane.....	E152
Plotter tools.....	E152
Trace properties tool	E153
Log tool	E154
Open Dialog tool	E155
Grid density tool	E155
Key on-off tool	E155
Autoscale tools	E156
Zoom in and Zoom out tools	E156
Redraw trace tool	E156
Push plot tool	E156
Plotter dialogs	E157
Types of plotters.....	E157
Copying plotted information	E160
Clearing plotted information.....	E160
Notebooks.....	E160
Printing.....	E162
Selecting what to print	E162
The Print command.....	E162
Worksheet, dialog, or plotter active	E163
Structure or dialog window active	E164
Print and Page Setup hints	E164
Cut, Copy, and Paste with the Clipboard	E165
Copying within Extend.....	E165
Copying from Extend to other applications.....	E165
Copying from other applications to Extend.....	E166
Drag and drop editing.....	E167
Appending models	E167
Importing and exporting with text files	E167
How Extend uses text files.....	E168
Working with text files in Extend	E168
Data sharing using Global Array blocks.....	E169
Global Array Manager.....	E172
Database connectivity with ODBC	E174
Building an Excel Database	E174
Importing Data into an Extend Model Using ODBC	E175
Distributed computing using Mailslots (Windows only)	E175

Contents

The Distributed simulation library.....	E176
Interprocess communication	E177
Hot Links (Windows only)	E178
Linking data from another application into Extend.....	E178
Linking data from Extend into another application.....	E179
Embedding OLE Objects and ActiveX Controls (Windows only)	E179
Embedding an object	E180
Embedding an object on the worksheet.....	E180
Embedding an Object into a Dialog Item.....	E181
Barchart example.....	E181
Paste Links to an embedded excel spreadsheet	E182
Design Mode	E182
Object.....	E182
Publishing and subscribing (Macintosh only)	E183
Publishing data tables and plots.....	E183
Subscribing to data tables	E184
Publisher/Subscriber options	E185
Updating a publisher's edition	E185
Updating a subscriber.....	E186
Locating publishers and subscribers.....	E186
Edition files.....	E186
Model reporting	E187
Steps for reporting.....	E188
Reporting example	E188
Dialogs report	E188
Statistics report.....	E189
Communicating with external devices and instruments.....	E189
Communicating from Macintosh computers.....	E189
Communicating from Windows computers	E190
Accessing code in other languages.....	E190
XCMDs under Macintosh.....	E190
DLLs under Windows.....	E191
Using interactivity as input and output	E191
Chapter 7: More Modeling Techniques.....	E193
General modeling hints	E194
Steps in creating models	E194
Refining models	E195
Low memory problems	E195
Tool tips on the worksheet	E196
Block icons – customizing	E197
Animation – customizing	E197
Copying sections of a model.....	E199
Selecting cells in data tables.....	E200
Stationery (Macintosh only)	E200
Connectors and connections	E200
Mixing Generic and Discrete Event blocks.....	E200
Named connections.....	E201
Show Named Connections command	E202
Unconnected connection lines	E202
Connections to multiple inputs.....	E202

Value connectors	E203
Item connectors	E203
Running models.....	E204
Simulation Setup command	E204
Discrete event tab.....	E205
Continuous tab	E206
Random numbers tab.....	E207
Time units tab.....	E208
Simulation timing.....	E209
Continuous simulation example.....	E210
Discrete event simulation example	E210
Delta times other than 1	E210
Simulation order	E211
Flow order.....	E211
Left to right order	E211
Custom order.....	E212
Units of time.....	E213
Using generic time units.....	E213
Using a global time unit.....	E213
Random numbers	E214
Random seeds	E215
Distribution fitting.....	E215
Speeding up simulations.....	E215
Slowing down simulations.....	E217
Debugging hints.....	E217
Animation.....	E217
Model reporting	E218
Model tracing.....	E218
Tracing commands	E218
Steps for tracing	E219
Tracing example	E219
Blocks for debugging.....	E219
Notebook.....	E220
Show Simulation Order command	E221
Find Block command	E221
Watching the simulation progress.....	E222
Equation editor	E222
Sensitivity analysis.....	E224
Introduction to sensitivity analysis	E224
Steps for using sensitivity analysis.....	E225
Example of sensitivity analysis.....	E226
Specifying the sensitivity method and number of runs	E228
Turning sensitivity on and off.....	E228
Reporting the results	E229
Multi-dimensional scenarios.....	E230
Optimization Tutorials	E231
Introduction to optimization.....	E231
How Optimization Works.....	E232
Steps for using optimization	E232
Optimization tutorials.....	E233
Optimization tutorial, continuous.....	E233

Contents

Setting up the objective function.....	E234
Adding variables to the optimizer	E235
Constraints	E239
Optimization tutorial, discrete event	E241
Setting up the objective function.....	E242
Adding variables to the optimizer	E243
Constraints	E246
Specifying optimization parameters	E249
Variables table	E249
Using Data Table cells in the Variables table	E249
Objective functions	E249
Special objective functions - approaching a constant	E250
Parameters.....	E250
Constraints	E251
Dependency constraints	E251
Global constraints	E252
Interpreting results	E252
Hierarchy	E252
Making a selection a hierarchical block.....	E254
Building a new hierarchical block.....	E255
Building the submodel	E256
Modifying the icon and adding connectors	E257
Saving the block	E259
Connecting the hierarchical block in the model	E259
Saving hierarchical blocks.....	E259
Saving hierarchical blocks in the model	E259
Saving hierarchical blocks in a library.....	E259
Modifying hierarchical blocks	E260
Adding text, drawing objects, or pictures to the layout pane.....	E261
Cloning dialog items to the layout pane	E261
Changing the icon.....	E262
Renaming the block	E262
Adding help	E262
Adding animation	E262
Results of modifying hierarchical blocks.....	E263
Interprocess communication	E265
Discrete event modeling hints	E265
Continuous blocks in discrete event models	E265
Customizing animation pictures.....	E267
Associating animation pictures with attribute values.....	E268
Generating items on demand	E269
Preprocessing.....	E269
Restricting items in a system	E270
Cycle timing.....	E271
Activity-based costing.....	E272
Preserving uniqueness in batching and unbatching.....	E272
Connecting to the “select” connector	E273
Timing irregularities.....	E274
Using scaling for large numbers of items	E274
Using the Status block to track items.....	E274
Continuous modeling hints.....	E275

Using plotters as input to other models or applications	E275
Using a plot line as a reference or standard	E275
Setting dt or the number of steps in continuous models	E276
Integration vs. Summation in the Holding Tank block	E277
Using NoValues to delay calculations	E278
Proof Animation (Windows only)	E278
Moving between Extend and Proof	E280
Building a simple Proof Animation:	E280
Adding paths.....	E282
Creating the Extend model	E282
Linking the simulation and animation	E283
Extend+Manufacturing User's Guide	M1
Manufacturing Introduction.....	M3
Simulating systems and processes	M3
Processes, events, and items	M3
Using simulation in industry	M4
Extend+Manufacturing	M5
The Extend+Manufacturing files	M5
Example and template files.....	M5
The Manufacturing library.....	M5
The Statistics library.....	M5
The QuickBlocks library	M6
How the Extend+Manufacturing manual is organized.....	M6
Installation and requirements	M6
New Extend+Manufacturing features in V5	M6
Further reading	M7
Manufacturing Chapter 1:	
Areas of Application.....	M9
Manufacturing systems.....	M9
Types of manufacturing systems	M10
Important considerations	M10
Performance evaluation	M10
Example models.....	M11
Assembly-Rework model	M11
Machining Operation model	M14
Services	M16
Types of service systems.....	M17
Important considerations	M17
Performance evaluation	M17
Example models	M18
Fast Food model	M18
Ferrari Agency model	M20
Call Center model	M22
About the model	M22
Material handling systems	M25
Types of material handling systems.....	M25
Important considerations	M26

Contents

Performance evaluations.....	M26
Example models.....	M27
Communication systems	M27
Types of communication systems	M28
Important considerations	M28
Performance evaluation	M28
Example models	M29
Overview of computer networks.....	M29
The CSMA_LAN model.....	M30
The CSMA Packet model.....	M33
Manufacturing Chapter 2: Tutorial.....	M35
Description of the problem	M35
Before you build this model	M37
Building the first part of the model	M38
Define a global time unit.....	M38
Lay out the basic model.....	M38
Executive block	M39
Program block.....	M39
Buffer block	M40
Station block.....	M41
Input Random Number block.....	M41
Exit block.....	M42
Running the model	M42
Adding two more operations and two more buffers	M43
Adding a buffer	M43
Activity, Multiple block.....	M44
Adding another buffer.....	M44
Conveyor Belt block.....	M44
Adding a buffer and the nonstandard insertion stations.....	M46
The nonstandard buffer.....	M46
Using attribute values to specify a processing time.....	M46
Station (Attributes) block.....	M47
Combine (5) block.....	M48
Running the model	M49
Completing the model of the existing process.....	M49
Batch (Variable) block.....	M50
Adding a burn-in buffer and station	M50
Buffer block	M50
Station block.....	M51
Unbatch (Variable) block	M51
Plotter, Discrete Event block	M51
Analyzing the model and running the simulation	M52
Model verification	M52
Model validation	M52
Running the model of the “as is” system.....	M52
Experimenting with the model	M53
Adding the new product line.....	M53
Adding another auto inserter.....	M54
Analyzing the results	M55

Adding animation	M56
Calculating production costs	M57
Cost Stats block	M58
Cost By Item block	M58
How confident can you be in the results?.....	M60
Clear Statistics block	M60
Activity Stats block.....	M61
Queue Stats block	M61
Adding another Program block	M61
Final analysis	M62

Manufacturing Chapter 3: Overview of the libraries.....M65

Manufacturing library blocks	M66
Activities	M66
Batching and unbatching	M67
Generators	M67
Queues.....	M68
Resources	M68
Routing.....	M69
Statistics library blocks	M70
The "Stats" blocks	M70
The Cost By Item block	M70
Other Statistics library blocks.....	M71
QuickBlocks library blocks.....	M71

Manufacturing Chapter 4: General Modeling ConceptsM73

Time units.....	M73
The default time unit	M73
Setting time-based parameters using connectors	M74
The length and number of simulation runs	M74
Terminating systems.....	M75
Non-terminating systems	M75
Determining the length and number of runs	M76
Model verification	M77
Model validation	M77

Manufacturing Chapter 5: Items and ValuesM79

Item generation	M80
Generating arrivals at random intervals	M80
Choosing a distribution in the Generator block	M81
Choosing an item's Value.....	M82
Using the Downtime (Unscheduled) block.....	M83
Generating arrivals with custom intervals	M83
Specifying the custom parameters.....	M83
Choosing the correct time units for the columns	M84
Making sure the arrival occurs when expected	M84

Contents

Scheduled arrivals.....	M85
The “start” connector	M86
Attributes	M87
Setting attributes	M87
Changing attribute values.....	M89
Reading attribute values and reporting changes	M89
Managing attribute names.....	M91
Using attributes.....	M91
Assigning properties	M91
Accumulating and tracking information.....	M92
Priorities.....	M92
Manufacturing Chapter 6: Queueing.....	M95
Queueing disciplines	M95
Queue/server systems	M96
Queueing considerations	M97
Choosing a queue.....	M97
Blocking.....	M97
Balking.....	M97
Reneging	M98
Jockeying	M99
Sorting using the Queue Decision.....	M100
Least dynamic slack.....	M101
Minimize setup	M101
Maximize service level	M102
Combined rules.....	M103
Matching queues.....	M104
Priority queues	M105
Holding	M105
Manufacturing Chapter 7: Routing	M107
Items from several sources	M107
Overview of merging, joining, and selecting multiple input streams.....	M108
Balancing multiple input lines.....	M108
Using the Throw and Catch blocks	M109
Items going to several paths.....	M110
Overview of parallel processing, unbatching, and selecting multiple output streams.....	M110
Simple routing	M111
Scrap generation.....	M113
Successive ordering.....	M113
Explicit ordering	M114
Routing decisions based on attributes.....	M115
Attribute routing using the Select DE Output block	M115
Attribute routing using the Throw and Catch blocks.....	M117
Conditional routing	M118
Bringing a system on-line	M118
Balancing multiple output lines.....	M119
Extended routing	M120
Machines that can only process certain types of items	M122

Manufacturing Chapter 8: Processing	M125
Processing in series	M126
Processing in parallel	M126
Parallel processing using a block	M126
Simple parallel connections	M127
Setting the processing time	M127
Fixed processing time	M127
Scheduled processing time.....	M128
Random processing time.....	M129
Custom processing time.....	M130
Implied processing time	M131
Cumulative processing time: time sharing.....	M131
Adding setup time.....	M133
Bringing an activity on-line	M135
Scheduling activities.....	M135
Shift block used to schedule	M137
Controlling the flow of items to an activity	M139
Fixed number of items	M139
Fixed period of time.....	M140
Item preemption and process interruption.....	M141
Preemption	M141
Interrupting or shutting down activities	M143
Scheduled shutdown	M143
Random shutdown.....	M145
Explicit shutdown	M146
Kanban system	M146
Material handling and transportation blocks	M147
Manufacturing Chapter 9:	
Batching and Unbatching.....	M151
Batching.....	M152
Simple batching	M153
Batching identical items	M153
The Batch (Demand) block.....	M153
The Batch (Variable) block.....	M154
Merging attributes.....	M155
Delaying kits.....	M155
The demand connector	M155
Unbatching	M156
Simple unbatching	M157
Variable batching and unbatching	M159
Preserving uniqueness of attributes and priorities	M160
Manufacturing Chapter 10: Resources.....	M161
Closed and open systems.....	M161
Closed systems	M162
Open systems.....	M162
Modeling resources.....	M162
Batching resources with items	M163

Contents

Using the Resource Pool blocks	M164
Same resource used in multiple places	M166
Resources required from different pools	M166
Scheduling resources	M167
Using change connectors	M167
Using the Shift block.....	M169
Simple On-Off Shift Example	M170
Simple Numerical Shift Example.....	M171
Activity Status Shift Control Example.....	M172
Shift Block in Serial Example	M173
Shift controlling Resource Pool	M174
An Important Note About Using Shifts.....	M175
Manufacturing Chapter 11: Activity-based costing	M177
Performing activity-based costing	M178
Item types	M178
Cost accumulators.....	M178
Resources	M179
Defining costs and cost rates	M179
Cost accumulator cost rates.....	M180
Resource cost rates	M181
Activity cost rates	M182
Combining resources with cost accumulators	M182
Batching resources with cost accumulators	M182
Cost accumulators and the resource pool blocks.....	M184
Using the Batch(Variable), Batch(Demand), and Unbatch(Variable) blocks	M184
Working with cost data	M185
Viewing Cost Data.....	M185
Changing Cost Data	M185
Gathering and Analyzing Cost Data.....	M187
How Extend tracks costs	M188
Setting the “_cost” and “_rate” attributes	M188
Combining resources with cost accumulators	M188
Calculating costs	M188
In generator-type blocks.....	M189
In activity-type blocks	M189
In queue-type blocks	M189
In resource-type blocks.....	M190
Combining multiple cost accumulators.....	M190
Manufacturing Chapter 12: Statistics and Model Metrics	M193
Statistical concepts	M193
Constant values versus random variables	M194
Random numbers and probability distributions	M194
Random numbers and seeds	M194
Random distributions	M195
Distribution guide.....	M197
Distribution fitting.....	M199

StatFit distribution fitting example.....	M199
Changing parameters dynamically.....	M201
Confidence intervals.....	M202
Multiple runs and Monte Carlo simulations.....	M202
Measuring and verifying simulation results.....	M203
Measuring performance and debugging models.....	M203
Clearing statistics	M205
Getting information about items.....	M205
Accumulating data using attributes	M206
Accumulating non-processing data.....	M206
Accumulating processing data	M207
Avoiding a conceptual error.....	M208
Timing the flow of items in a portion of the model.....	M210
Using the Timer block.....	M210
Using attributes to record times.....	M211
Using Notebooks for displaying simulation results	M211
Extend+BPR User's Guide.....	B1
BPR Preface	B3
List of BPR library blocks.....	B4
List of Statistics library blocks	B4
BPR Introduction	B5
Background.....	B5
Business process reengineering.....	B6
Process reengineering goals.....	B6
How processes are changed.....	B6
Steps in the reengineering process.....	B7
Business processes, events, and items	B7
Extend+BPR	B8
Using Extend+BPR	B8
The Extend+BPR libraries.....	B9
The BPR library	B9
The Statistics library.....	B9
How this manual is organized	B9
Installation and requirements	B10
Further reading	B10
BPR Chapter 1: Areas of Application	B13
Performance measurement	B13
Why measure?	B13
Support Services models.....	B14
Overview.....	B14
Modeling the process	B14
Showing serial processes	B15
Performance metrics.....	B16
Enhancing the model	B16
Running the models.....	B17
Model highlights	B17

Contents

Cycle time	B18
Cycle time compared to processing time, productivity, and utilization	B18
Invoice Approval model	B19
Model highlights	B21
Technology insertion	B21
Documentation Revision model	B22
Manual process	B22
Automated process	B23
Comparing the “as is” to the “to be” process	B23
Model highlights	B24
Workflow	B24
Routing rules	B24
Order Processing model	B25
Activity-based costing.....	B25
Traditional methods	B26
ABC method.....	B26
Activity Costing model.....	B26
Implementing strategic plans.....	B27
Operational planning	B28
Traditional methods	B28
Planning cycle incorporating modeling	B28
Support Planning model	B29
ISO9000/EN29000	B31
Certification process.....	B31
Testing Process model	B31
Resource capacity analysis	B32
Utilization and constraints on resources	B32
Regional Call Processing model.....	B33
About the model	B33
Model highlights	B34
Causal loops	B34
Roulette model.....	B35
Model highlights	B35
BPR Chapter 2: Tutorial	B37
Before you begin the tutorial	B37
Rightsizing	B37
Description of the problem	B38
Starting the Credit Application model.....	B39
Define a global time unit.....	B39
Add blocks to the model	B39
Executive block	B40
Import block.....	B40
Repository block	B41
Operation block	B42
Export block	B42
Add a second reviewer	B42
Determine which applications pass on the first review	B43
Merge block	B44
Decision (2) block.....	B44

Input Random Number block.....	B45
Add a second review process.....	B45
Stack block.....	B46
Add randomness.....	B46
Import block.....	B46
Input Random Number block.....	B47
Test for reducing the number of reviewers.....	B48
Changing the model.....	B49
Labor Pool block.....	B50
Operation block.....	B50
Transaction block.....	B51
Operation, Reverse block	B52
Activity Stats block.....	B53
Customize animation	B53
Calculate cost of review process.....	B54
Cost Stats block	B55
Cost By Item block	B56
Model highlights	B56
Things to consider.....	B57
BPR Chapter 3: Overview of blocks in the BPR and Statistics libraries	B59
BPR library blocks	B60
Activities	B60
Attributes	B61
Batching.....	B61
Generators	B61
Queues.....	B61
Resources	B62
Routing.....	B62
Statistics library blocks	B63
The "Stats" blocks	B63
The Cost By Item block	B63
Other Statistics library blocks.....	B64
BPR Chapter 4: General Modeling Concepts	B65
Time units.....	B65
The default time unit	B65
Setting time-based parameters using connectors	B66
The length and number of simulation runs	B66
Terminating systems.....	B67
Non-terminating systems	B67
Determining the length and number of runs	B68
Model verification	B69
Model validation	B69
BPR Chapter 5: Items and Informational Values	B71
Item generation	B72
Generating arrivals at random intervals	B72
Choosing an item's Value.....	B73
Generating arrivals with custom intervals	B74

Contents

Scheduled arrivals.....	B76
The “start” connector	B77
Attributes	B77
Setting attributes	B78
Reading attributes	B79
Using attributes	B80
Priorities.....	B81
BPR Chapter 6: Queueing.....	B83
Scheduling algorithms.....	B83
Queueing considerations	B83
Choosing a queue.....	B84
Blocking.....	B84
Balking.....	B84
Reneging.....	B85
Jockeying	B86
Priority queues	B87
BPR Chapter 7: Routing.....	B89
Items from several sources	B89
Overview of merging and joining multiple input streams	B89
Using the Throw and Catch blocks	B90
Items going to several paths.....	B91
Overview of parallel processing, unbatching, and selecting multiple output streams	B91
Simple routing	B92
Explicit parallel processing	B94
Routing decisions based on attributes.....	B95
Parallel processing based on an item attribute or priority.....	B95
Using the decision blocks	B96
Using the Throw and Catch blocks	B97
Conditional routing	B98
Bringing a system on-line	B98
Balancing multiple output lines.....	B99
Extended routing	B100
BPR Chapter 8: Processing.....	B103
Processing in series	B103
Processing in parallel	B104
Parallel processing using the Transaction block	B104
Simple parallel connections	B104
Setting the processing time	B105
Fixed processing time	B105
Scheduled processing time.....	B106
Random processing time	B107
Custom processing time	B107
Bringing an activity on-line	B109
Scheduling activities.....	B109
Shift block used to schedule	B111
Controlling the flow of items to an activity	B112
Fixed number of items	B112

Fixed period of time	B114
Interrupting activities and preempting items	B114
Preemption	B115
Shutting down activities	B116
Scheduled shutdown	B116
Random shutdown.....	B117
BPR Chapter 9: Batching and Unbatching.....	B119
Batching.....	B119
Simple batching	B120
Merging Attributes.....	B121
“Take last” option	B121
Unbatching	B121
BPR Chapter 10: Resources.....	B125
Overview.....	B125
Closed and open systems.....	B126
Closed systems	B127
Open systems.....	B127
Modeling resources.....	B127
Batching resources with items	B128
Assigning attributes to resources.....	B128
Allocating Resources	B129
Using the Resource Pool blocks.....	B131
Same resource used in multiple places	B133
Resources required from different pools	B133
Scheduling resources	B134
Using change connectors.....	B134
Using the Shift block.....	B136
Simple On-Off Shift Example	B137
Simple Numerical Shift Example.....	B138
Activity Status Shift Control Example	B139
Shift Block in Serial Example	B140
Shift controlling Resource Pool	B141
An Important Note About Using Shifts.....	B142
BPR Chapter 11: Activity-based costing.....	B143
Performing activity-based costing	B144
Item types	B144
Cost accumulators.....	B144
Resources	B145
Defining costs and cost rates	B145
Cost accumulator cost rates.....	B146
Resource cost rates	B147
Activity cost rates	B148
Combining resources with cost accumulators	B148
Batching resources with cost accumulators	B148
Cost accumulators and the resource pool blocks.....	B150
Working with cost data	B150
Viewing cost data	B151

Contents

Changing cost data.....	B151
Gathering and analyzing cost data.....	B152
How Extend tracks costs	B153
Setting the “_cost” and “_rate” attributes	B154
Resources combined with cost accumulators	B154
Calculating costs	B154
In generator-type blocks.....	B154
In activity-type blocks	B155
In queue-type blocks	B155
In resource-type blocks.....	B155
Combining multiple cost accumulators.....	B156
BPR Chapter 12: Statistics and Model Metrics.....	B157
Statistical concepts	B157
Constant values versus random variables	B158
Random numbers and probability distributions	B158
Random numbers and seeds.....	B159
Random distributions	B159
Distribution fitting.....	B162
StatFit distribution fitting example.....	B162
Changing parameters dynamically.....	B164
Confidence intervals.....	B165
Multiple runs and Monte Carlo simulations.....	B166
Measuring and verifying simulation results.....	B167
Measuring performance and debugging models.....	B167
Clearing statistics	B168
Getting information about items.....	B169
Avoiding a conceptual error.....	B170
Timing the flow of items in a portion of the model.....	B171
Using the Timer block.....	B172
Using attributes to record times.....	B173
Using Notebooks for displaying simulation results	B173
Extend+Industry User’s Guide	S1
Extend+Industry	S3
SDI Industry includes:	S3
What is the SDI Database?	S3
Benefits	S4
The Embedded Database	S4
Design Criteria for the Embedded Database.....	S4
Importing Data from outside of Extend	S4
Database Manager Block	S5
SDI Database-Aware Blocks	S5
SDI Database Wizards and other Database Tools	S5
Learning to use SDI Industry	S6
Online Help.....	S6
Block Models	S6
SDI Database Tutorial 1	S8
Building a Database from inside of Extend.....	S8

Creating a Database Table	S8
Defining Table Fields	S8
Entering Table Data	S11
Making a Database Cell Random	S11
Setting up the Date and Time	S13
Setting up the Model Worksheet	S14
Setting up the Item Generator Block	S15
Creating Database-Aware Attributes	S15
Setting up the DB Specs block	S17
Setting up a DB Write block	S19
Example: Using the Information (DB) block to verify a Model	S19
SDI Database Tutorial 2	S20
Exporting an SDI Database to an Excel DB Workbook.....	S21
Editing Tables and Randoms in Excel.....	S22
Creating a DB Text File from Excel	S22
Importing the DB Text File into Extend.....	S23
SDI Database Tutorial 3 (Advanced Concepts).....	S23
Creating an Excel DB Workbook	S23
Creating the Routing Tables (on a separate worksheet)	S25
Indexing Tables (Creating Parent-Child relationships).....	S26
Setting up the Model Worksheet	S27
Using the 3D Lookup Blocks	S30
Flow Architecture.....	S31
Why Handle Materials Using Flow Blocks?.....	S32
What is meant by Flow?	S33
How are Items Different?	S33
An Item cannot be in Two Blocks at Once	S34
Item Movement Happens at Steps.....	S34
How is this Behavior Different from Flow?	S35
Comparing Items and Flow Using a Bucket Problem	S35
Model Speed and Accuracy Considerations	S36
Flow Tutorial 1: Simulating the Bucket Problem Using Flow	S36
Observations of Flow Model Behavior.....	S38
How Flow Works	S38
Flow Material wants to get Out and get Done.....	S39
Flow Breakthroughs and Benefits	S39
The Execution Time Breakthrough	S39
The Accuracy Breakthrough	S40
The Naturalness Breakthrough	S40
The Compatibility Breakthrough	S40
Flow Tutorial 2a: Simple Buckets with Blocking.....	S40
Explanation of Flow Tutorial 2a	S41
Flow Tutorial 2b: Additional Constraints	S41
Explanation of Flow Tutorial 2b.....	S42
Flow Tutorial 2c: Adding a Downtime Process	S42
Explanation of Flow Tutorial 2c	S42
Flow Starving & Blocking Behavior	S43
Starving & Blocking for Items- An All or Nothing Proposition	S43
Starving & Blocking for Flow- A Matter of Degree	S43
Starving & Blocking for Store Blocks	S44
Handling Material Additions and Losses	S45

Contents

Handling Unit Conversion.....	S46
Flow Rate Conversion	S46
Flow to Item Conversion.....	S47
Knowing What the Material Is	S48
Categories of Flow Blocks	S48
Rules for Using Flow Blocks.....	S48
Flow Connectors and Information Connectors	S49
Flow cannot split or join without a Split or Join Block.....	S50
Start with a Store and end with a Store.....	S51
Do Not Leave Flow Blocks Hanging	S52
Flow Connectors also Report Rate Information.....	S53
Rate Override does not query for Values.....	S53
Flow Block Optimization Techniques.....	S56
First-Order Assumption and Tentative Event Posting	S56
Propagation of Potential Rates Using a Block Connector Message System.....	S57
Deferred Throughput and Contents Calculation.....	S57
Global Override	S57
Appendix A: Menu Command Reference.....	A1
Apple menu (Macintosh only).....	A2
File menu	A3
New Model	A3
New Text File	A3
Open.....	A3
Append Model	A4
Close.....	A4
Revert Model	A4
Save and Save As	A4
Save Selected	A4
Import Data.....	A5
Export Data	A5
Import DXF File (Windows only)	A5
Show Page Breaks.....	A5
Page Setup (Macintosh only).....	A6
Print Setup (Windows only).....	A6
Print.....	A6
Get Info	A7
Three most recent models or text files	A7
Quit or Exit	A7
Edit menu	A8
Undo	A8
Cut	A8
Copy	A8
Paste.....	A8
Clear	A9
Select All	A9
Duplicate	A9
Paste Link (Windows only)	A9
Delete Link (Windows only).....	A9
Show Links (Windows only)	A9
Refresh Links (Windows only)	A9

Insert Object (Windows only).....	A9
Design Mode (Windows only)	A9
Object (Windows only).....	A10
Create Publisher (Macintosh only)	A10
Subscribe To (Macintosh only)	A10
Publisher/Subscriber Options (Macintosh only)	A10
Open Publishers/Subscribers (Macintosh only)	A10
Sensitize Parameter.....	A10
Show Clipboard	A10
Preferences	A10
Model tab	A11
Libraries tab	A12
Programming tab	A13
Miscellaneous tab.....	A14
Library menu	A15
Open Library	A15
Close Library	A16
New Library.....	A16
Tools	A16
Protect Library	A16
Set Library Version.....	A16
Convert to RunTime Library	A16
RunTime Startup Screen Editor	A17
Open All Library Windows.....	A17
Compile Open Library Windows	A17
Compile Selected Blocks	A17
Remove Debug Code in Open Library Windows	A17
Add Debug Code to Open Library Windows	A17
Libraries	A17
Model menu.....	A18
Open Notebook.....	A18
Make Selection Hierarchical.....	A18
New Hierarchical Block	A18
Open Hierarchical Block Structure.....	A19
Rename Hierarchical Block	A19
Connection Lines.....	A19
Show Named Connections.....	A19
Hide All Connections	A20
Controls.....	A20
Reduce	A20
Expand.....	A20
Reduce to Fit.....	A20
Normal Size	A20
Lock Model.....	A20
Use Grid	A21
Find Block	A21
Find Next.....	A21
Show Block Labels	A21
Show Block Numbers.....	A21
Show Simulation Order.....	A21
Set Simulation Order.....	A21

Contents

Open Sensitized Blocks	A21
Spoken Messages (Macintosh only)	A21
Text menu	A22
Define menu	A23
Build New Block.....	A24
Open Block Structure	A24
Rename Block	A24
Open Set Breakpoints Window.....	A24
Compile.....	A24
Generate Debugging Info.....	A25
Continue.....	A25
Step Over.....	A25
Step Into	A25
Step Out	A25
Set Block Type	A25
New Dialog Item.....	A26
Define New Tab	A26
Edit Tab	A26
Move Selected Items to Tab.....	A26
Find	A27
Enter Selection.....	A27
Find Again	A27
Replace	A27
Replace, Find Again	A27
Replace All	A27
Shift Selection Left.....	A28
Shift Selection Right	A28
Go To Line.....	A28
New Include File.....	A28
Open Include File	A28
Delete Include File	A28
Run Menu.....	A28
Run Simulation.....	A29
Simulation Setup.....	A29
Run Optimization.....	A29
Use Sensitivity Analysis	A29
Show Animation	A29
Add Connection Line Animation	A30
Add Named Connection Animation	A30
Show Movies (Macintosh only)	A30
Generate Report.....	A30
Report Type	A30
Add Selected To Report.....	A30
Add All To Report.....	A30
Remove Selected From Report	A30
Remove All From Report	A30
Show Reporting Blocks	A30
Stop	A31
Pause.....	A31
Step.....	A31
Resume	A31

Debugging	A31
Pause At Beginning	A31
Step Each Block	A32
Step Next Animation	A32
Step Entire Model	A32
Show Block Messages	A32
Only Simulate Messages	A32
Scroll To Messages	A32
Generate Trace	A32
Add Selected To Trace	A32
Add All To Trace	A32
Remove Selected From Trace	A33
Remove All From Trace	A33
Show Tracing Blocks	A33
Profile Block Code	A33
Show Debug Messages	A33
Window menu	A33
Help menu	A33
About Balloon Help (Macintosh only)	A33
Show Balloons (Macintosh only)	A33
Extend Help (Macintosh) / Help (Windows)	A33
Imagine That Home Page	A34
Extend Product Updates	A34
Additional Products	A34
Extend User Group	A34
About Extend (Windows only)	A34
Appendix B: Upper Limits	A35
Appendix C: Cross-Platform Considerations	A37
Libraries	A38
Models	A38
Menu and keyboard equivalents	A38
Transferring files between operating systems	A39
File name adjustments	A39
Physically transferring files	A40
File conversion	A40
Model files	A40
Hierarchical blocks in libraries	A41
Libraries	A41
Blocks that use the equation functions	A42
Extensions and drivers	A42
Appendix D: Generic Library Blocks	A45
Table of Generic blocks	A46
Submenus	A46
Arrays - Storing, accessing global data	A47
Decisions - Routing or deciding which value to use	A47
Holding - Accumulating or storing values	A48
Input/Output - Reading and writing files, or generating values	A49

| **Contents**

Math - calculating values.....	A51
Statistics - Calculating Mean, Variance	A54
Appendix E: Discrete Event Library Blocks	A55
Table of Discrete Event blocks.....	A56
Submenus	A56
Executive - needed in every model.....	A57
Activities - Processing items.....	A57
Attributes - Giving items an identity	A57
Batching - Joining and dividing items	A59
Generators - Creating items, scheduling.....	A60
Information - Getting information about items.....	A60
Queues - Holding, sorting, and ranking items.....	A61
Resources -Representing items as resources, shifts.....	A62
Routing - Moving items to the correct place.....	A63
Appendix F: Blocks in the Manufacturing Library	A67
Table of Manufacturing blocks	A67
Submenus	A67
Activities - Processing items.....	A68
Batching - Joining items and dividing items	A70
Generators - Generating items, schedules	A71
Queues - Storing and ranking items	A71
Resources - Representing items as resources.....	A72
Routing - Moving items to the correct place.....	A74
Appendix G: Blocks in the Statistics Library	A75
Table of Statistics blocks.....	A75
Appendix H: Blocks in the BPR Library.....	A77
Table of BPR blocks	A77
Submenus	A77
Activities - Processing items.....	A78
Attributes - Reading attribute values.....	A78
Batching - Joining and dividing items	A79
Generators - Generating items, schedules	A79
Queues - Holding and ranking items	A79
Resources - Representing items as resources.....	A79
Routing - Moving items to the correct place.....	A80
Index.....	IX1

Preface

*Extend'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

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 in 1963, 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 superfast (at the time!) computers by hand, and graphic user interfaces were still decades in the future.

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

Imagine That, Inc. is dedicated to bringing the art of simulation to the desktop, in a form digestible and accessible by everyone. Extend is the first user-extensible simulation package that meets those expectations.

Bob Diamond
President

Introduction

*Where you are introduced to Extend and
learn what you need to begin*

“Begin at the beginning,’ the King said, gravely,
‘and go ‘til you come to the end; then stop.”
— Lewis Carroll

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

Extend

Modeling and simulation

A model is a logical description of how a system performs. Simulations involve designing a model of a system and carrying out experiments on it as it progresses through time. For example, the board game Monopoly is a model of a real system – the hotels and facilities of Atlantic City. When you play Monopoly, you are simulating that system. Simulation with Extend means that instead of interacting with a real system, you create a model which corresponds to it in certain aspects.

You can use a model to describe how a real-world activity will perform. Models also enable you to test hypotheses at a fraction of the cost of actually undertaking the activities which the models simulate. For example, if you are a hardware designer, you can use Extend to simulate the performance of a proposed system before building it.

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 “step-wise refinement” enables you to achieve good approximations of very complex problems surprisingly quickly. As you add refinements, your model becomes more and more accurate.

Why simulation is important

Simulation provides a method for checking your understanding of the world around you and helps you produce better results faster. A simulation program like Extend is an important tool that you can use to:

- Predict the course and results of certain actions
- Understand why observed events occur
- Identify problem areas before implementation
- Explore the effects of modifications
- Confirm that all variables are known
- Evaluate ideas and identify inefficiencies
- Gain insight and stimulate creative thinking
- Communicate the integrity and feasibility of your plans

Extend and spreadsheets

Extend is to simulations what spreadsheet programs have become to numbers. With Extend, you create a block diagram of a process where each block describes one part of the process. In a spreadsheet, you can lay out your numbers in a two-dimensional tabular format; in Extend, you can lay out your process in a two-dimensional drawing environment. While a spreadsheet can be used to represent a snapshot of a system or a process, Extend provides the equivalent of a moving picture. Extend's iterative technique lets you create models of real-world processes that are too complex to be easily represented in a spreadsheet.

You can create a model quickly because Extend comes with all the blocks you need for most simulations. These blocks act like macros, so you can build models without even having to type an equation.

Extend

Since you assemble many blocks into a single model, you can use a series of simple block definitions to describe complex processes. Extend also provides an equation editor (similar to the formula bar in a spreadsheet program) so you can combine the function of several blocks into one. You can even create your own libraries of custom blocks for specialized applications.

What Extend can do

With Extend, you get all the ease-of-use and capability you need to quickly model any system or process:

- A full array of building blocks that allow you to build models rapidly
- Animation of the model for enhanced presentation
- A customizable graphical interface showing the relationships in the system you are modeling
- Unlimited hierarchical decomposition to make even complex systems easy to build and understand
- Dialogs and Notebooks for changing model values, so you can quickly try out assumptions and test your model
- A full-featured authoring environment for simplifying model interaction and enhancing communication
- The ability to adjust settings while the simulation is running
- Evolutionary optimization, Monte Carlo, batch-mode, and sensitivity analysis for optimizing systems
- Customizable reports for presentation and in-depth analysis
- Activity-based costing capabilities for analyzing cost contributors

- Full connectivity and interactivity with other programs and platforms through Copy/Paste, import/export, text (ASCII) files, Publish/Subscribe (Macintosh only), XCMDS and Apple Events (Macintosh only), or DLLs and DDE (dynamic data exchange) and OLE (Windows only).

And Extend's integrated environment adds advanced features to make it the most powerful simulation system available on the desktop:

- *Cross-platform compatibility*—Run models under Windows 3.1x, Windows 95 and 98, Windows NT, PowerMacintosh, or Macintosh systems. Model and library files are cross-platform compatible so you can work collaboratively with colleagues using other platforms.
- *Multi-purpose simulation*—Extend is a multi-domain environment so you can dynamically model continuous, discrete event, linear, non-linear, and mixed-mode systems.
- *Built-in, compiled C-like programming language and dialog editor*—Modify Extend's blocks or build your own for specialized applications; build customized animation into your model.
- *Integrated support for other languages*—Use Extend's built-in APIs to access code created in Delphi, C++ Builder, Visual Basic, Visual C++, and so forth.
- *Library based*—The blocks you build can be saved in libraries and easily reused in other models.
- *Over 500 functions*—Directly access functions for integration, statistics, queueing, animation, IEEE math, matrix, sounds, arrays, FFT, debugging, XCMDS, DLLs, 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.
- *Huge models*—Scalability means model size is limited only by the limits of your system.

New features added in version 5

- OLE (Object linking and embedding) - Bring ActiveX controls and OLE embedded objects into your models. See “Embedding OLE Objects and ActiveX Controls (Windows only)” on page E179.
- Interactive Source Code Debugger - View ModL code as it executes; insert conditional breakpoints; access the values of variables. See “Chapter 5: Using the ModL Source Code Debugger” on page P245.
- ODBC (Open Database Connectivity) - Communicate with ODBC-compliant applications from within your model. See “Database connectivity with ODBC” on page E174.

- Optimization - Reduces experimentation time and guides you to the optimal solution. See “Optimization” on page E85 and “Optimization Tutorials” on page E231.
- QuickBlocks - Solve common modeling problems with a single hierarchical block. See the QuickBlocks section in the Extend + Manufacturing manual.
- New help system and online manuals.
- Dynamic local area network messaging - allow you to send messages and items from machine to machine over a LAN; facilitates distributed simulation. See “Distributed computing using Mailslots (Windows only)” on page E175.
- Improved integration with Proof Animation. See “Proof Animation (Windows only)” on page E278.
- Mouse wheel support - Navigate through models more easily.
- Timer events - Post periodic block messages based on the system clock. See the Breakout.mox model and the StartTimer() function in “Date and time” on page P165.
- Global Array block - Store information in Extend's internal database. See “Data sharing using Global Array blocks” on page E169.
- Improved communication on block icons - Icons change dynamically with input parameters.
- Throw/Catch by using Catch block number.
- Create H-blocks which automatically “connect.”
- Open URL function - Gives you the ability to launch a browser to your own website from within your Extend model. See “Web and Help connectivity” on page P168.
- Call Help - Access your own custom Help system. See “Web and Help connectivity” on page P168.
- Use cursor keys in icon editor for fine control.
- Border control for drawing objects - black border or no border. See “Borders” on page E72.
- Animation enhancements:
 - Animation updates immediately when turned on.
 - Animate H-block built into blocks.
 - Animation of connection line colors. See “Animation” on page P123.

Extend

- Custom simulation order - change the block calculation order for special cases. See “Custom order” on page E212.
- More than a hundred new ModL functions.

Extend + Manufacturing enhancements

- Shift block - Shut down blocks on a schedule.
- Queue, Decision block - Use an equation as a ranking rule.
- Activity Status block - Get details on the status of Machine and Station blocks.
- Random routing in Select DE Output block simplifies models.

Extend

Using Extend

An Extend model is a document that contains components (called “blocks”), usually with connections between the blocks. Each block contains procedural information as well as data that you enter. After you create a model, you can modify it by adding blocks, moving connections, and changing the blocks’ data.

Levels of use

You can use Extend on many levels:

- Run pre-assembled models and explore alternatives by changing data in dialogs, Notebooks, or text files. If you work in a group environment, one user can create models for others to run for experimenting. There are special RunTime and Player versions available which you can use to distribute your pre-assembled models to others.
- Assemble your own models from the blocks that come with Extend. Extend is shipped with libraries of blocks that handle both continuous and discrete event models as well as specialized libraries for electronics and other fields. To assemble a model, pull blocks from menus of libraries and link connectors on the blocks.
- Create new blocks and features using the integrated programming environment. You can also modify the blocks that come with Extend to work with your specific needs.

Extend helps you organize your model by letting you build your own hierarchical blocks containing subsystems. Thus, you don’t have to start over when you need to build a model of a process which has elements in common with a previous model.

As you read this manual, you will see how Extend caters to the needs of users at all levels.

Libraries that come with Extend

Extend libraries hold blocks that you can use to create models in minutes. Extend comes with the Generic library, used to model continuous systems. Extend+Manufacturing, Extend+BPR,

and Extend Suite also contain the Discrete Event, Manufacturing, and/or BPR libraries, providing an extensive set of iconic building blocks for modeling discrete event systems. This allows you to construct models without even typing in an equation. If you create your own blocks, or modify Extend's blocks, you can build new libraries to contain them.

Classical simulation is generally divided into two categories: continuous and discrete event. The Generic library is used for continuous simulations. In continuous simulations, values change when time changes. The Discrete Event library is used for models that use queues, item-specific attributes, and priorities. In a discrete event model, model entities change state based on when events occur. These two libraries are discussed in much greater detail in “Chapter 4: Fundamental Continuous and Discrete Event Modeling”.

Extend

Other libraries that come with Extend include:

- A Plotter library that contains most common types of plotters for graphing model output.
- An Animation library for adding custom animation to models and hierarchical blocks.
- Electronic engineering libraries to simulate system level design of analog, digital, signal processing, and control systems.
- An Interprocess Communication (IPC) library for communicating and multi-tasking with other applications.
- A Utilities library, containing a collection of helpful blocks which count the number of blocks in a model, fit data to a curve, synchronize the model to real time, and so forth.
- Sample libraries from a variety of disciplines that illustrate Extend's scope and features.
- For a complete list of the libraries included in your Extend package, see “Libraries” on page A38.

In addition, other companies develop “Add-On” library modules for Extend. To get the latest information about Add-On libraries for control systems, neural nets, bulk manufacturing and so forth, check the Imagine That, Inc. web page or search the Read-Me file that ships with Extend.

What this manual doesn't do

Extend is a tool for building models across a variety of disciplines. Just as a word processor does not limit you to one kind of writing, Extend does not limit you to one field or modeling technology.

This manual is a general-purpose tutorial and reference for using the Extend program. Every effort has been made to present sample models that can be easily understood by users in any field so you can quickly learn how to use this powerful tool. While you may find that your subject area is not represented in the manual, or that the sample models reflect some disciplines which are unfamiliar

to you, remember that their purpose is to teach you how to use Extend. What you model, and how you model it, are determined mostly by your knowledge and expertise in your subject area.

How to start

This User's Guide gives you a thorough overview of Extend's capabilities as well as many examples of how you might use Extend in your work. There is also a companion manual, the Programmer's Reference, for those who want to create their own Extend blocks or who would like a clearer picture of how Extend blocks work.

The User's Guide is divided into six tabbed sections: Extend, Manufacturing, BPR, Industry, Appendices, and Index. Chapters 1 through 7 in the Extend section show you most of what you need to know about using Extend's libraries and features. The first three chapters are especially important since they provide the basic tutorial for using Extend.

The Manufacturing, BPR, and Industry tabbed sections are for those have purchased those Extend modules. These sections show you how to create discrete event and discrete rate models using the Manufacturing, BPR, and Industry modules.

The Appendices in the User's Guide are reference sections. Appendix A describes the menu commands and Appendix B tells you about some upper limits in Extend. Appendix C discusses moving files and libraries between Windows and Macintosh systems and provides tables of file names and keyboard equivalents when running Extend under those systems. Appendices D-H list and describe the Generic, Discrete Event, Manufacturing, Statistics, and BPR Library blocks, respectively.

On-line help

Extend's on-line help is available any time you are using Extend. Just select the Extend Help command in the Help menu or press F1 on your keyboard.

The block help from any libraries which are open is automatically included with Extend's on-line help topics. This help is also accessed through a button labeled "Help" in each block's dialog.

Technical support

You must be a registered user to receive technical support, so be sure to mail your registration card immediately after purchasing Extend.

We always appreciate hearing from you, and it would help us if you provide us with the following information when you contact us:

- Your Extend serial number, which is located in the front of your manual.
- Your name and phone number (email and FAX number are also helpful), in case we need to return your call.

- The version of Extend you are using:

Apple Macintosh: this number is located in the About Extend command under the Apple menu.

Windows: this number is located in the About Extend command under the Help menu.

- The version of the libraries you are using is located in the Library Window. Open the library's window as described in "Library windows" on page E144. Information on the library version, size, and last modification date is listed at the top.

- The type of computer you are using, such as PowerMacintosh 8100 or Pentium 230.

- The version number of the operating system you are using, such as "Windows 95" or "Macintosh System 8.0".

- Anything else about your machine configuration: hard disk, RAM Cache, video cards, etc.

Telephone:408-365-0305 (8:30 AM to 5:30 PM Pacific Standard Time, Monday through Friday)

Email:support@imaginethatinc.com

FAX:408-629-1251

Mail:6830 Via Del Oro, Suite 230, San Jose, CA 95119 USA

Web Site:<http://www.imaginethatinc.com>

Extend

Extend

Chapter 1: Running a Model

*In which you learn how to run an
Extend model and investigate its components*

*“For the things we have to learn before
we can do them, we learn by doing.”*

— Aristotle

Most of your work with Extend is running models. This may sound a bit strange, since you probably expected that most of your work would be in constructing the models. In fact, you will find that constructing models is so quick that you will spend much more of your time running your models, changing your assumptions about the simulation, and rerunning the model with the new assumptions.

This chapter shows you how to do just that. It covers parts of models, running models, working with blocks, and changing assumptions in your models. When you finish the chapter, you will know most of what you need to run and get results from Extend models.

In business, a common goal is to optimize a system such that it processes the most things using the least amount of resources and time. This type of system is often modeled using Discrete Event queue/server concepts.

- *Queues* are another word for lines, which we encounter in one form or another in almost every aspect of our lives. The items in lines can be customers, parts, data, or any other discrete item.
- *Servers* are the mechanisms by which the members of the line are processed, then sent on their way. Servers can be sales personnel, machines, computers, or any other resource which takes some time to process the members of the line.

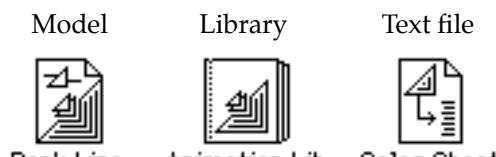
Some common examples of queue/server systems are: bank lines, transportation systems, computer networks, assembly lines, and a multi-tasking computer.

In continuous modeling, a similar analogue would be the accumulator/flow concept.

- *Accumulators* add incoming flows to their contents. This could model the accumulating pollution in a lake.
- *Flows* represent both the incoming and outgoing rate and amount leaving the Accumulator. This might represent streams, both adding pollutants from upstream and draining the water and some of the pollution level downstream from a lake.

Extend documents

There are three types of Extend documents: *models*, *libraries*, and *text files*. Shown below are the Extend documents with Macintosh and Windows icons and their descriptions:



Model, library, and text file icons

- *Macintosh*: file names can be up to 31 characters long. Extend libraries usually end in “Lib”, but this is not required.
- *Windows*: Extend supports long file names for Windows. The extensions are “.MOX” for Extend model names, “.LIX” for library names, and “.TXT” for text file names.

See “Appendix C: Cross-Platform Considerations” on page A37 for more information about differences between the Macintosh and Windows versions of Extend.

Note When talking about model or library names, we will just specify the name without any extensions. For example, the Generic library will have the name Generic Lib on the Macintosh, and Generic.LIX on Windows machines.

Extend

Model overview

Most of your work is done with models. A model is composed of components (called “blocks”) with connections between them. A library is a repository of blocks. When you want to add the function of a block to a model, you copy it from a library. (Blocks and libraries are discussed in detail later in this chapter.) Text files store data in a form that can be read by almost any application. Text files are especially useful for importing large amounts of data into a model, for example from a spreadsheet, or for exporting model data to another application for further analysis or for presentation purposes.

Models are opened and closed in the File menu. When you open a model or create a new model, the model window appears on your screen as a normal document window. All open model files are listed in the Window menu. The three most recently opened files, whether they are still open or not, are listed at the bottom of the File menu. Libraries, on the other hand, are opened and closed in the Library menu. Libraries have their own library windows. Open libraries are also listed at the bottom of the Library menu.

Extend can also produce text files, which are opened and closed in the File menu. These files can be read and edited by Extend and by any word processor and most other applications. To open a text file in another application, use that application’s Open command.

Starting Extend and opening a model

After you have installed Extend, you can start the application. The Extend application icon is the same for Windows or Macintosh:



Start Extend by double-clicking on its icon:

- *Macintosh:* double-click the folder in which you installed Extend, then double-click on the Extend icon.
- *Windows:* Access the Programs sub-menu in the Start menu and select the Extend application.

When Extend starts, you see an empty new model window. Extend does not open any libraries when it starts unless you open an existing model or you specify which libraries should be opened in the Preferences dialog (see “Preferences” on page A10). You are now ready to start using Extend.

Tutorial

This tutorial shows how to open a model, examine it, and make changes. It has two parts:

- **Part 1, for all Extend users:** This section explores the continuous model Pollution.mox that examines the effects on a polluted lake of changing the incoming and outgoing streams.
- **Part 2, for discrete event users:** This section is for discrete event modelers and will only be accessible if you have one of the modules designed for discrete event modeling (such as Extend+Manufacturing). The “Bank Line.” model is a queue/server simulation of bank customers entering a bank and waiting in line for tellers. You will also explore the results of changing the model by decreasing the number of tellers and by having customers arrive randomly.

All modelers should read Part 1 of the Tutorial. If you intend to only build continuous models, you can safely skip Part 2 and proceed directly to “Chapter 2: Building a Model” on page E39.

Tutorial, part 1 (all Extend users)

Because continuous modeling concepts are so important, even to discrete event modelers, our tutorial is split into two sections:

- Part 1 discusses the basics of Extend using a continuous model.
- Part 2 is an abbreviated tutorial that uses a discrete event model, and assumes that the user has completed Part 1.

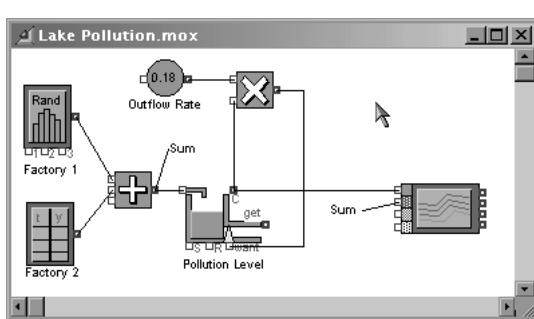
The first model you will open, “Lake Pollution,” uses continuous simulation and is therefore based on the Generic library.

This model is in a folder or directory called “Tutorials”. To open this model:

- ▶ Select Open from the File menu.
- ▶ Find and open the Examples folder (within the Extend5 folder).
- ▶ Find and open the Tutorials folder.
- ▶ Find the “Lake Pollution” model and open it.

Depending on the speed of your computer, as the model opens you may see status messages on your monitor's screen. These indicate that Extend is "Opening" the libraries, and "Reading" and "Initializing" the model. If you have a fast computer, these messages may appear too quickly for you to read.

The model appears on your screen:



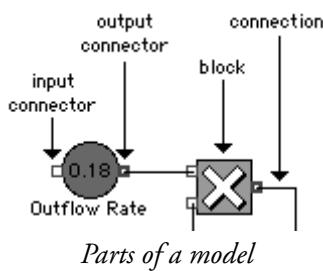
Extend

"Lake Pollution" model (Windows)

This is how a typical model window looks. The window that the model is in is just like other document windows. You can scroll horizontally or vertically using the scroll bars, you can resize and zoom the window, and you can close the window with the close box in the upper left corner. The scroll bars on the model window are restricted to the area you are using plus some extra space on the right and bottom of the model. For example, dragging the scroll box all the way to the right side of the scroll bar will show you an area just a bit further to the right than the active area of the model.

Model basics

You can see from the "Lake Pollution" example that there are many parts in Extend models. The most important parts of a model are the *blocks*, the *libraries* where the blocks are stored, the *dialogs* associated with each block, the *connectors* on each block, and the *connections* between the blocks.



Blocks

Most blocks in Extend are composed of an icon (the picture you see), a dialog for entering and viewing data, and connectors. A block specifies an action or process. Information comes into the

block and is processed by the program that is in the block. The block then transmits information out of the block to the next block in the simulation.

A block in Extend is like a block in a block diagram. It is used to represent a portion of the model. Some blocks may simply represent a source of information that is passed on to other blocks. Other blocks may modify information as it passes through them. Some blocks, called *hierarchical blocks*, contain groups of other blocks. The plotter block at the right side of the Bank Line model is an example of an output block, a block which takes information from the simulation and presents it to you in a visible form.

There is nothing fundamentally different about the structure of these different blocks. Any block may create, modify, or present information, and many blocks perform more than one of these functions.

A block's icon normally represents the function of the block. If you build your own hierarchical blocks or program your own blocks, you can add your own icon drawings to represent what you want the user to see. For instance in the Lake Pollution model, the block labeled "Pollution Level" is a Holding Tank block from the Generic library. Its icon symbolizes a tank with a flow entering it at the top-left, and a flow leaving it at the far right. There are additional connectors on the bottom that allow controlling the block, and another output connector marked *C*, that allow the monitoring of its contents.

There are seven different types of blocks in the "Lake Pollution" model, each unique, but you can have any number of instances of the same block in a model.

Libraries

As mentioned before, libraries are repositories for blocks. The entire *definition* for a block (its program, icon, dialog, and so on) is stored in the library. When you include a block in a model, the block itself is not copied to the model. Instead, a reference to the block information in the library is included. Extend also stores the data that you entered in the block's dialog in the model.

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: storing just a reference to the library saves a great deal of disk and RAM space.

When you save a model on disk, Extend saves the names of the blocks as well as the locations of the libraries that store the blocks. When you open a model, Extend automatically opens the needed libraries. For example, when you opened the "Lake Pollution" model, Extend opened two libraries: Generic, and Plotter. To verify this, pull down the Library menu and note the three libraries at the bottom of the menu.

There are three ways to open a library. As you just saw, Extend automatically opens libraries when you open models. You can also open a library by choosing Open Library from the Library menu.

The third way is have Extend pre-load libraries that you commonly use. In the Preferences command in the Edit menu, you can specify up to seven libraries that will automatically open when Extend is started (see “Preferences” on page A10).

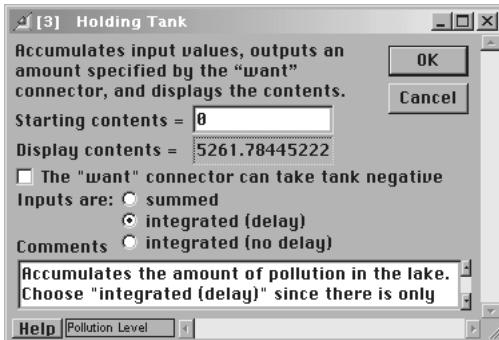
To see the contents of a library, open the library and select the library from the list in the menu. When you do this, a hierarchical menu that contains submenus with all the blocks in the library appears next to the library name. The submenus are organized by block type. Use these menus to add blocks to your model, as described in Chapter 2: Building a Model. For a list of Extend libraries, see “Libraries” on page A38.

Extend

Dialogs

Most blocks have a dialog associated with them. You use dialogs to enter values and settings before you run your simulations and to see results as the simulation runs. Extend’s dialogs act just like dialogs in other programs so it is easy to enter numbers and values. To open a block’s dialog, simply double-click on the block’s icon. (Note that hierarchical blocks, discussed in “Hierarchy” on page E79, have other blocks in them, and you see these blocks when you double-click on the icon.)

For example, double-click on the Pollution Level icon. The dialog for that block opens:



Pollution Level dialog (Windows)

This dialog lets you interact with the Holding Tank by setting how the Holding Tank is to work in the model. For example, to indicate that there is some existing pollution in the lake, you could enter a number into the *Starting Contents* parameter. The “[3]” in the title bar is the block number, indicating that this was the fourth block placed in the worksheet (block numbers start at 0).

You can get more information about the block by clicking on the Help button at the bottom left corner of the dialog. To the right of the Help button is a text entry box for typing in a block label, as discussed in Chapter 2. Click Cancel to close the dialog and discard any changes you made. (In blocks that have plots, such as the Input Data block, changes cannot be canceled.) Clicking OK or the close box in the upper left corner of the dialog indicates that you want to save the changes.

Note that you can resize Extend dialogs just like you can other windows. If you resize the dialogs to be smaller, you can then use the scroll bars to find the items you want.

In dialogs with input fields, you can enter text or numbers. If you are entering text, such as in the Comments tab, Extend automatically wraps your text as you type. You can force Extend to go to the next line of text by pressing Return. If you are entering numbers with decimal points, Extend uses the decimal type of your country (such a comma for most European countries).

You can leave dialogs open while you run your model, although this slows down the simulation a bit. Some dialogs report values from the model, so you can use dialogs to show values that you want to watch during the simulation. You can even change the settings in a dialog as you run the simulation, such as choosing different buttons or typing new values.

Note When you click on 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 stops running until you finish typing and then click the Resume button at the bottom of the screen.

Connectors and connections

Most blocks in Extend have input and output connectors, the small squares attached to each side of the block. As you might expect, information flows into a block at input connectors and out of the block at output connectors. Input and output connectors are usually pre-defined; their specific function is known in advance. A block might have many input and/or output connectors; some blocks have none. Since connectors are important when you build a model (as compared to when you run it), they are discussed in more detail in Chapter 2: Building a Model.

You use connecting lines to hook blocks together. These lines (called *connections*) show the flow of information from block to block through the model. The simulation itself is a series of calculations and actions which proceed along the path of the connections repetitively. Each repetition is called a *step* for continuous models or *event* for discrete event models; your model can run for as long as you want.

Before you run a simulation, you tell Extend how long (in simulation time) you want it to run. Then you start the simulation run. The blocks then calculate in an order determined by Extend as the simulation runs. In the “Lake Pollution” model, you would suspect that the Factory 1 and Factory 2 blocks at the left of the model would be processed first, the block that adds the two inputs is processed second, and so on, up to the Plotter. After that first step, the simulation repeats that calculation order.

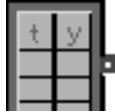
Exploring the Lake Pollution model

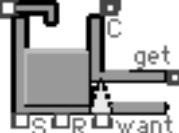
Now that you have seen the basic parts of a model, you are ready to explore the “Lake Pollution” model and see how models operate. After running the model, you will see how easy it is to modify models as you change the assumptions about your simulation.

When you start the modeling process, it is always a good idea to map out what you are modeling and why. The “Lake Pollution” model is a simulation of a lake with pollution sources entering it and a specific flow of clean water that flushes out the pollution from it. The purpose of the model is to see the actual amount of pollution in the lake at any time, with varying amounts of pollution entering it at different times, and a clean stream flowing into it and cleaning it (we ignore the pollution going downstream of the lake, for now).

Like most Extend models, the action in this model runs along the path of the connections. To see a block’s parameters, double-click on the block’s icon; for now, you should not change anything in the block’s dialog.

Extend

Block (Library)	Description
	Generates random integers or real numbers based on the selected distribution. In this model, this block generates a random distributed value (randomly changing) of pollution from a factory.
<i>Input Random Number (Generic)</i>	
	Generates a curve of data over time from a table of values and acts as a lookup table. In this model, this block represents a pollution source that stops after a while so we can observe the lake pollution value decaying.
<i>Input Data (Generic)</i>	
	This block adds the inputs on the left of the block and outputs the total. In this model, we are adding all of the pollution sources so we can input that total value to the “lake.”
<i>Add (Generic)</i>	
	Generates a constant value at each step. You specify the constant value in the dialog (the default constant is 1). In this model, the constant is the percentage of pollution leaving the lake (via an outflow of water) at each time step in the run, or 18% (0.18).
<i>Constant (Generic)</i>	
	This block multiplies the inputs. In this model we use it to multiply the constant of 18% times the actual amount of pollution in the lake.
<i>Multiply (Generic)</i>	

Block (Library)	Description
 <i>Holding Tank (Generic)</i>	Accumulates the total of the input values, allowing you to request an amount to be removed, and outputs that requested amount if it is available. In this model, we use the Holding Tank to hold the current pollution level in the lake, adding more pollution to its input, and drawing out a constant percentage of pollution as water flow cleans the lake.
 <i>Plotter I/O (Plotter)</i>	Gives plots and tables of data for up to four value inputs for continuous models. In this model, we are plotting the level of pollution in the lake at any time, versus the actual value of pollution entering the lake at any time.

Running a simulation

Hopefully, you are by now itching to run a simulation. Well, you can go ahead. Choose Run Simulation from the Run menu and your model runs. That's all there is to it. By default, Extend plays the system default sound at the end of the simulation. You can turn this option off by unchecking the "Sound plays at end of run" option in the Preferences command from the Edit menu.

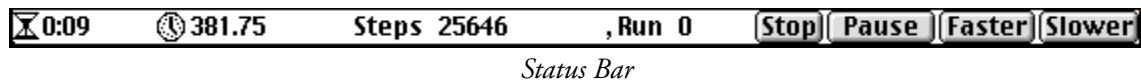
The "Lake Pollution" model runs for a simulated 40 years. While the model runs, Extend displays the results on the plotter. The lines drawn show the pollution input (red) and the level of pollution in the lake (blue). As you can see, when the level of pollution going into the lake changes, its pollution level tends to exponentially approach an equilibrium point. Extend leaves the plotter on the screen when the simulation is finished. This is your primary method for determining what happened during the simulation.

After you have set up your model, there are two major parts to running a simulation: the simulation status and the plotter.

Simulation status

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

Once the simulation run begins, Extend shows a small Status Bar at the bottom of your screen:



The numbers after the hourglass are an estimate of the actual time left in the simulation (expressed as “minutes:seconds”) so you can determine how long it will run. The clock shows the current time of the simulation in simulation time units. *Steps* is the total number of steps or simulate messages in the simulation, and *Run* is the number of the simulation if you are running multiple simulations (simulation runs start at 0). These values are determined by the entries in the Simulation Setup dialog, as described in “Simulation Setup dialog” on page E55.

Note that the time remaining shown in the Status Bar is only an estimate. For continuous simulations, it is usually very accurate. For discrete event simulations, however, it can be very inaccurate, especially at the beginning of a run since it is impossible to know when events are going to happen and thus how long it takes to run a model.

Extend

To stop a simulation in progress, click the *Stop* button in the Status bar or use the keyboard:

- Macintosh: press Command-. (press the  key and the period key at the same time)
- Windows: press Control-. (press the CTRL key and the period key at the same time)

Extend asks if you really want to stop the simulation. If you are running multiple simulations, you can stop just this run or all of them.

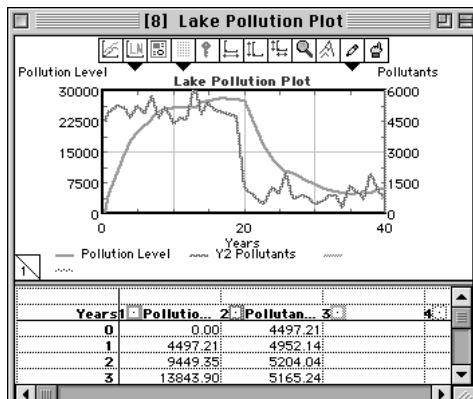
If you just want the simulation to pause, click the *Pause* button instead of the Stop button. That button then becomes the *Resume* button for continuing the run. You can also stop and pause simulations from the Run menu.

If you run a simulation with animation turned on, as discussed in “Animation” on page E81, the animation may occur faster than you want. You can use the *Slower* button to display animation at a slower speed. That button then becomes the *Faster* button which you can use to return to normal speed.

Plotters

Extend comes with flexible plotters that you can use in your models. Plotters show both a graphical representation of the numbers fed to them as well as a table of the numerical values. (If you decide to program in Extend’s ModL language, you can also create your own plotters.) The plotter in the “Lake Pollution” model is the Plotter, I/O block from the Plotter library. Plotters are fully described in Chapter 6: Input and Output.

When you run a simulation, Extend shows the plotter on the screen. The plotter for the “Lake Pollution” model looks like:



“Lake Pollution” plotter I/O(Macintosh)

This is an example of a plot with two lines and two value axes. The blue line and the left axis shows the pollution level in the lake; the red line and the right axis show the pollutant level entering the lake. Note that the scales for the two lines are different. You can observe the values by moving the cursor along the traces in the plotter. The corresponding values are displayed at the top of the plotter’s data table.

The bottom of the plotter window shows the data points which produce the line. You can scroll down this list to see the numerical values for the two lines.

Extend’s plotters remember the pictures of the last four plots. You can see the other plots by clicking on the small turned-up page symbol at the bottom left of the graph part of the plot window.

Changing assumptions

So far, you have run the simulation without changing any of the assumptions that were made when the model was created. One of Extend’s strongest features is the ability to change assumptions on the fly and see the results instantly. Since the plotter remembers the previous four plots, you can easily compare the results after you change assumptions.

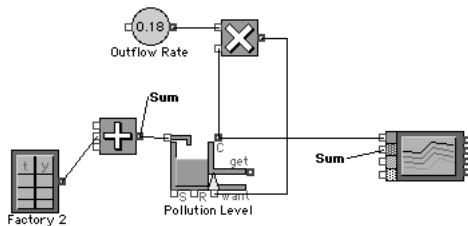
The rest of this chapter is devoted to methods for changing the assumptions that went into the “Lake Pollution” model. You can change model assumptions by adding or removing blocks or by changing parameter values in a block’s dialog.

Adding and removing blocks

If you have some processes running in parallel (such as the two pollution sources in the “Lake Pollution” model), you can easily test the results of adding additional parallel processes or removing

existing ones. Adding or removing a block like this usually takes less than a minute. This section shows you how to remove a block; you will see how to add blocks in Chapter 2: Building a Model.

Removing blocks from a model is easy. Simply click on the block to select it and press the Delete or Backspace key. Like other applications, selecting something and pressing Delete or Backspace removes it from the document. In this case, Extend removes the block and its connections. For example, remove the Input Random Number block (labeled Factory 1) from the model:



"Lake Pollution" with one pollution source

Extend

Since you have one pollution source instead of two, and you have removed the random source, the line will smoothly increase and decrease to follow the pollution change. You can also experiment with changing the pollution amount, as shown below.

Changing dialog parameters

The most direct method for telling Extend what you want in a model is through the blocks' dialogs. To open a dialog, double-click on the dialog's icon. You can then interact with the dialog just like you do with other dialogs.

This section describes the useful dialogs in the "Lake Pollution" model and how they can be changed, then shows you how to change some model assumptions and observe the results. Some dialogs contain less information than others. For example, the Constant (labeled Outflow Rate) block dialog only contains information about the constant value to be used.

Extend remembers any changes you make to the settings in a dialog. If you change a dialog and save the model to disk, the new values are saved with it.

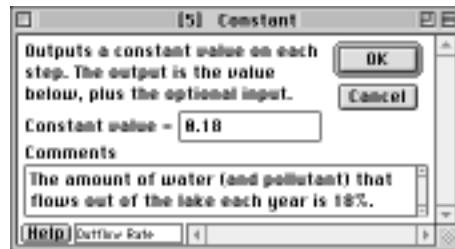
You can change a dialog value by clicking in the parameter field while the model is running. When you do this, Extend pauses the simulation. To continue the simulation, click the Resume button in the status bar or choose Resume from the Run menu.

In "Chapter 3: Enhancing your Model" on page E65, you will see how to vary a dialog value interactively using Control blocks such as Sliders, and how to use sensitivity analysis to explore various scenarios. For simplicity, parameters in the following blocks are entered in the dialog as static values which do not change based on model conditions.

Constant block



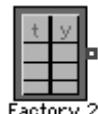
The Constant block outputs a constant value into the model. In this model, it is used to represent the outflow of pollution via the outflow of water. Its dialog lets you specify the value of the constant. The dialog is:



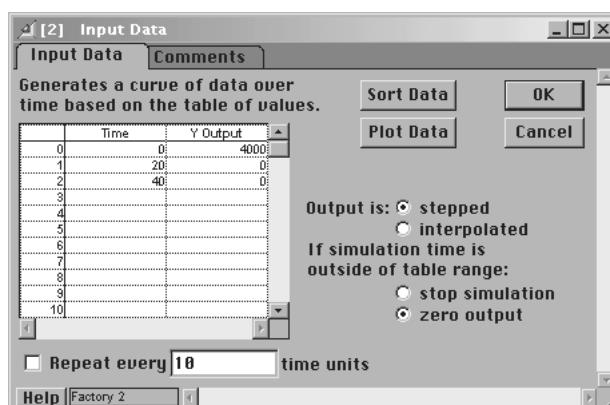
Constant dialog (Macintosh)

Note that the constant is set to 0.18 or 18%. You can see that by increasing this value, the pollution level is affected. If you increase the constant, more pollution will leave the lake. If you decrease it, less pollution will leave the lake. Of course, in real life, pollution outflow cannot speed up or slow down so easily.

Input Data block



The Input data block provides a time-varying value (in this case, a varying factory pollution level) for the simulation. Its dialog lets you set the average time between customers arriving as well as the method with which they are distributed. The dialog is:



Input Data dialog (Windows)

This block looks at the current simulation time, compares it to the time values value in the column labeled *Time*, and outputs the corresponding *Y Output* value. It is similar to a “Lookup table” in that it uses *Time* as an *X* value and outputs a *Y* value.

Notice that this dialog has multiple pages, or *tabs*, containing more information and settings. In this block, the only other tab is the *Comments* tab, useful in documentation of settings.

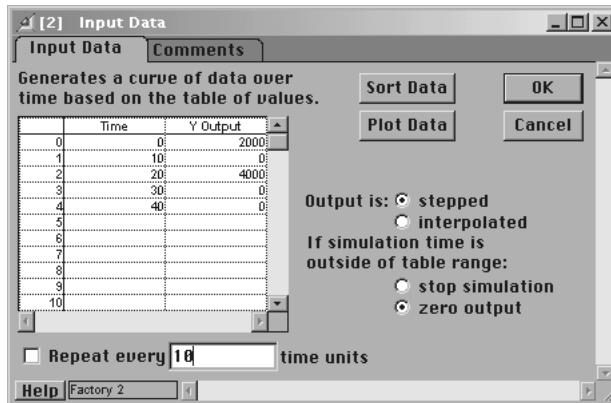
For this model, there are no specific time units, as seen in the Continuous (or Time Units) tab of the Simulation Setup dialog in the Run menu. However, if we want, we can define time in any units we wish (see “Units of time” on page E213 for a discussion about time units).

Extend

Changing a pollution source

Now that you have examined the dialogs, you can try changing some of the parameters to observe the effects. Right now, the Input Data block shows that 4000 units of pollution are fed into the lake at time 0, the beginning of the run. Let’s change the values in that block so that some of the pollution enters the lake at the beginning of the run, and some enters later on.

To do this, change the Time column so that we can have more pollution inputs over the 40 year run. Right now the values are 0, 20, and 40 years in the Time column. Let’s change that to 0, 10, 20, 30, and 40 by selecting each value in the Time column and changing it (you can use the arrow keys to navigate the table). Then, change the values in the Y Output column to reflect our new data: 2000, 0, 4000, 0, 0 so the table looks like:



The Input Data block with changes(Windows)

Now run the model again. Notice the different output.

Other modifications

As you can see from this discussion, Extend gives you many opportunities to change the way a model runs and explore different scenarios. The “Lake Pollution” model can be extended further

in many ways that will not be fully explored here since they involve more advanced techniques. For example:

- You can add more real-life contingencies by adding blocks to the model. For example, you can add more pollution sources, and you can add a monitor that will sound an alarm if the pollution level goes above a certain value (see the Stop block).
- You can open the Holding Tank and enter a value into the *Starting Contents* parameter so the lake has a pollution level when the run starts
- You can change the Constant block's Outflow Rate so that you can see what effect outflow has on the lake. You could also add another Input Data block to control the outflow based on time.
- You can view the status of other blocks by connecting their outputs to the plotter, by leaving their dialogs open while running the simulation, or by taking some of their dialog items and putting them in the model window or Notebook, as described in “Notebooks” on page E79. You can also choose to leave informational blocks open while running the simulation so that you can see changes in values without plotting them.

Extend

Hopefully, you now understand how powerful Extend models are. The next chapter shows you how to build a model from scratch. When you are finished with that chapter, you can see how easy it was to create the “Lake Pollution” model and how you will create your own models.

The next tutorial is for discrete event modelers. If you intend to only build continuous models, you can safely skip the next section and proceed directly to “Chapter 2: Building a Model” on page E39.

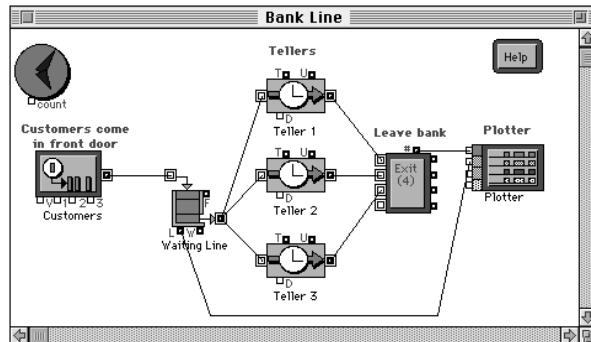
Tutorial, part 2 (discrete event users)

Note The models in this tutorial are only available if you have one of the discrete event modules, such as Extend+Manufacturing. This tutorial also assumes that you have completed the previous section “Tutorial, part 1 (all Extend users)” on page E16. If you have not worked through that tutorial, it is suggested that you start there.

The “Bank Line” model is in a folder or directory called “Tutorial”. To open this model:

- ▶ Select Open from the File menu.
- ▶ Find and open the Examples folder (within the Extend5 folder).
- ▶ Find and open the Tutorials folder.
- ▶ Find the “Bank Line” model and open it.

The model appears on your screen:



Extend

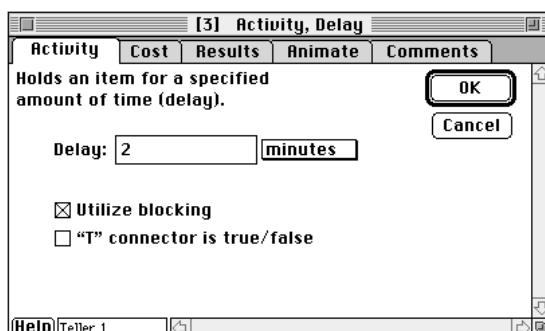
"Bank Line" model (Macintosh)

This model shows a bank, with people entering, waiting on line for an available teller, and then leaving the bank. In this tutorial, we will examine this model.

In the Bank Line model, the block labeled "Customers" generates items representing people arriving at the bank. "Waiting line" is a Queue FIFO block from the Discrete Event library. Its icon symbolizes items going into the queue at the top and coming out in the same order (first-in, first-out) at the bottom. The "Teller" blocks are Activity Delay blocks that hold the customer items for a specified service time. When the customer is done, the Exit (4) block can pull the customer out of the "Teller" and out of the bank.

Dialogs

Double-click on one of the teller icons. The dialog for that block opens:



Teller dialog (Macintosh)

This dialog lets you interact with the tellers by setting how long it takes for a teller to work. For example, to indicate that this teller is faster, you could change the number 2 to a smaller number. The "[3]" in the title bar is the block number, indicating that this was the fourth block placed in

the worksheet (block numbers start at 0). Notice that this dialog has multiple pages, or *tabs*, containing more information and settings.

Exploring the “Bank Line” model

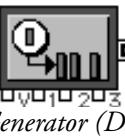
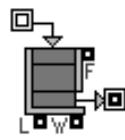
After running the model, you will see how easy it is to modify models as you change the assumptions about your simulation.

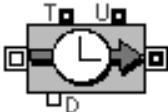
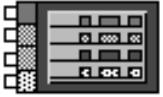
Extend

When you start the modeling process, it is always a good idea to map out what you are modeling and why. The “Bank Line” model is a simulation of customers coming into a bank, standing in line, being served by a teller, and leaving the bank. The purpose of the model is to see the number of customers that pass through in a typical hour under different conditions and how many people are in line at any given time.

In many models, you are interested in the time required for an item (in this case, a customer) to move from one end of the model to the other. Blocks may or may not delay the item. For instance, in the “Bank Line” model, there is a delay at the tellers since the transaction takes time.

Like most Extend models, the action in this model runs along the path of the connections. To see a block’s parameters, double-click on the block’s icon; for now, you should not change anything in the block’s dialog.

Block (Library)	Description
 <i>Executive (DE)</i>	The Executive block from the Discrete Event library is a special block that must be included in all discrete event simulations. See the online help for more information on this block. For now, assume that this block must be here and its left edge must be further left than the left edge of any other block in the model.
 <i>Generator (DE)</i>	The block labeled “Customers come in front door” is a Generator block from the Discrete Event library. The Generator block is used to generate items (in this case, customers coming in the door) at arrival times specified through the dialog. The block is initially set up so that a new customer is generated every 39 seconds.
 <i>Queue FIFO (DE)</i>	When customers enter the bank, the customers wait in line. In a model, this type of line is called a <i>queue</i> . If a teller is free, the customer automatically leaves the queue and goes to the teller. If no teller is free, the customers wait in the queue until a teller is ready. Through its connectors, the block reports the length of the line (<i>L</i>) and the time spent waiting (<i>W</i>). This block is the Queue FIFO block from the Discrete Event library. (“FIFO” stands for “first-in, first-out.”) There is no delay in this block unless there are no free tellers, in which case the delay equals the time spent waiting for a teller.

Block (Library)	Description
	The customer goes to the first available teller, represented by an Activity Delay block from the Discrete Event library. If more than one teller is free, the customer will arbitrarily go to any available teller. The customer is then delayed at the teller by the amount of time assigned in each Activity Delay block (2 minutes).
	After the delay, the customer goes to the Exit (4) block which takes the customer out of the simulation. The Exit block comes from the Discrete Event library. The connector at the top outputs the total number of customers that have come into the block.
	The Discrete Event plotter, from the Plotter library, shows the results of the simulation. Although you can run a simulation without having any plotters in the model, a plotter is the most common method of gathering and presenting simulation data. The plotter is described later in this chapter.

Extend

Running the simulation

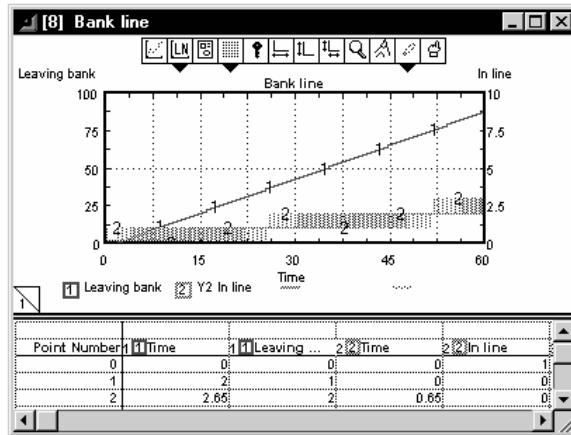
Choose Run Simulation from the Run menu and your model runs. The “Bank Line” model runs for a simulated 60 minutes. While the model runs, Extend displays the results on the plotter. The lines drawn show the queue length (red) and the number of people who leave the bank (blue). As you can see, the length of the waiting line slowly increases. Extend leaves the plotter on the screen when the simulation is finished. This is your primary method for determining what happened during the simulation.

Turn on animation. Go to the Run menu and select Show Animation and then run the model again. If you run a simulation with animation turned on, as discussed in “Animation” on page E81, the animation may occur faster than you want. You can use the *Slower* button to display animation at a slower speed. That button then becomes the *Faster* button which you can use to return to normal speed.

Plotters

Extend comes with flexible plotters that you can use in your models. Plotters show both a graphical representation of the numbers fed to them as well as a table of the numerical values. (If you decide to program in Extend’s ModL language, you can also create your own plotters.) The plotter in the “Bank Line” model is the Plotter, Discrete Event block from the Plotter library. Plotters are fully described in Chapter 6: Input and Output.

When you run a simulation, Extend shows the plotter on the screen. The plotter for the “Bank Line” model looks like:



“Bank Line” plotter (Windows)

This is an example of a plot with two lines and two value axes. The blue line and the left axis show the number of people who have left the bank since the simulated hour began; the red line and the right axis show the length of the line waiting for tellers. Note that the scales for the two lines are different. You can observe the values by moving the cursor along the traces in the plotter. The corresponding values are displayed at the top of the plotter’s data table.

The bottom of the plotter window shows the data points which produce the line. You can scroll down this list to see the numerical values for the two lines.

Extend’s plotters remember the pictures of the last four plots. You can see the other plots by clicking on the small turned-up page symbol at the bottom left of the graph part of the plot window.

Changing assumptions

So far, you have run the simulation without changing any of the assumptions that were made when the model was created. One of Extend’s strongest features is the ability to change assumptions on the fly and see the results instantly. Since the plotter remembers the previous four plots, you can easily compare the results after you change assumptions.

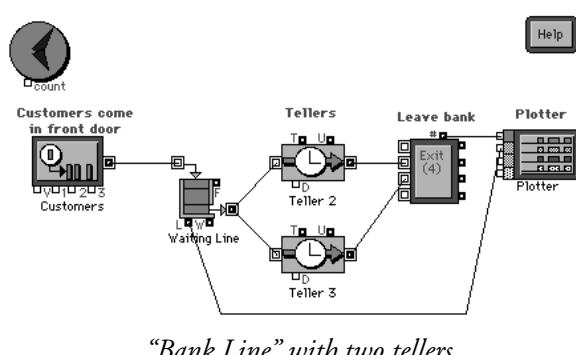
The rest of this chapter is devoted to methods for changing the assumptions that went into the “Bank Line” model. You can change model assumptions by adding or removing blocks or by changing parameter values in a block’s dialog.

Adding and removing blocks

If you have many processes running in parallel (such as the three tellers in the “Bank Line” model), you can easily test the results of adding additional parallel processes or removing existing

ones. Adding or removing a block like this usually takes less than a minute. This section shows you how to remove a block; you will see how to add blocks in Chapter 2: Building a Model.

Removing blocks from a model is easy. Simply click on the block to select it and press the Delete or Backspace key. Like other applications, selecting something and pressing Delete or Backspace removes it from the document. In this case, Extend removes the block and its connections. For example, remove the top Activity Delay block (Teller 1) from the model:



Since you have two tellers instead of three, the line will rapidly increase and the number of people leaving the bank will decrease. You can also experiment with changing the delay times for both of the remaining tellers, as shown below.

Changing dialog parameters

The most direct method for telling Extend what you want in a model is through the blocks' dialogs. To open a dialog, double-click on the dialog's icon. You can then interact with the dialog just like you do with other dialogs.

This section describes the useful dialogs in the "Bank Line" model and how they can be changed, then shows you how to change some model assumptions and observe the results. Some dialogs contain less information than others. For example, the Exit(4) block dialog only contains information about the number of items that have left the model. The Help block contains simple help information for the model.

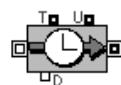
Extend remembers any changes you make to the settings in a dialog. If you change a dialog and save the model to disk, the new values are saved with it.

You can change a dialog value by clicking in the parameter field while the model is running. When you do this, Extend pauses the simulation. To continue the simulation, click the Resume button in the status bar or choose Resume from the Run menu.

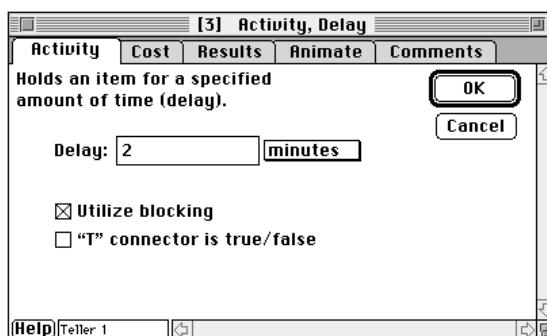
In "Chapter 3: Enhancing your Model" on page E65, you will see how to vary a dialog value interactively using Control blocks such as Sliders, and how to use sensitivity analysis to explore various

scenarios. For simplicity, parameters in the following blocks are entered in the dialog as static values which do not change based on model conditions.

Activity Delay block

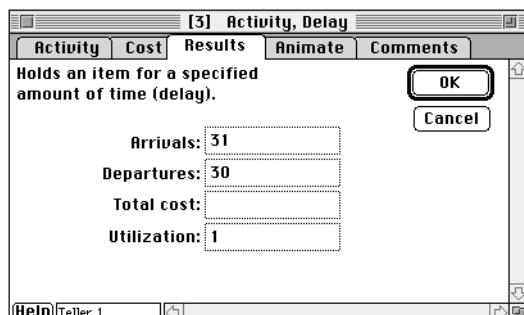


The Activity Delay block “holds” an item for a specified amount of time then releases it. In this model, it is used to represent a teller who waits on one customer at a time. Its dialog lets you specify how long it takes the teller to serve the customer. The dialog is:



Activity Delay dialog (Macintosh)

Note that each teller is set at 2 minutes service time. You can see that changing the speed of one of the tellers can radically alter the length of the queue. If you removed a teller as described above, you could decrease the time it takes the remaining tellers to wait on customers to 1.33 minutes, so that the waiting line would be about the same length as before. Of course, in real life, people cannot speed up or slow down as easily.



Activity Delay dialog -Results tab (Macintosh)

Note that the Arrivals, Departures, Total Cost and Utilization fields are in dotted rectangles. These fields are used to display numbers that the model calculates but which you do not change. Here, they display the number of people who have arrived and departed and the teller's utilization rate (cost is not being calculated in this particular model). Display-only fields automatically report

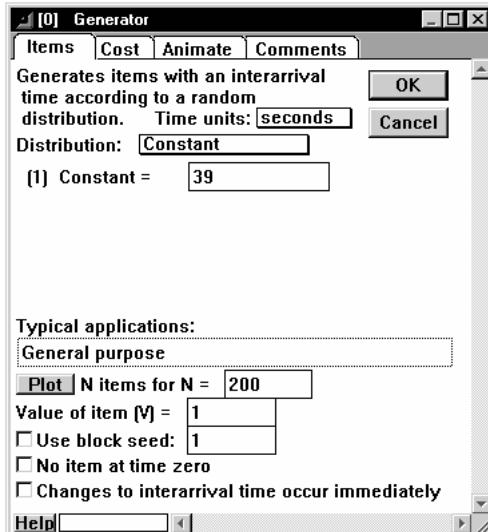
information that may be of interest to you as you analyze the model. You can copy a number in a display-only field by selecting it and choosing the Copy command from the Edit menu.

Generator block



The Generator block provides items (in this case, customers) for the simulation. Its dialog lets you set the average time between customers arriving as well as the method with which they are distributed. The dialog is:

Extend



Generator dialog (Windows)

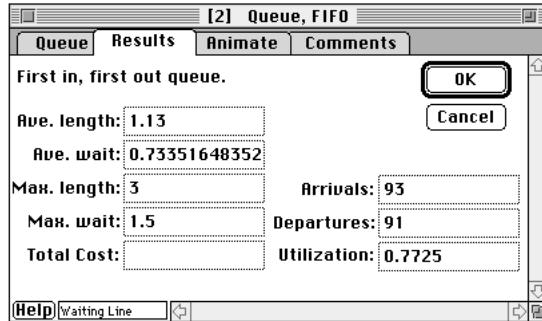
For this model, the global time unit is minutes, as seen in the Time Units tab of the Simulation Setup dialog in the Run menu. However, we can define the interarrival time of the customers in any time unit we wish (see “Units of time” on page E213 for a discussion about time units). Note that the time unit selected in the Generator dialog is seconds, and that this block is originally set up to generate a new customer every 39 seconds. You could change the Constant value and see how this affects the length of the line and the number of customers served. You could also choose a different distribution of arrival rates, as described later in this chapter.

If you select different choices using the popup menu, the input parameters below the popup menu change. For example, if you select “real, uniform” in the popup menu, two boxes appear which allow you to define the range for the real values. The “Value of items (V)” choice should always be 1 in this model. It is an advanced concept described in Chapter 4: Fundamental Continuous and Discrete Event Modeling.

Queue, FIFO block



The Queue, FIFO block holds and releases items on a first-in-first-out basis. Its dialog does not let you change the way that this model runs in any significant way. It does, however, display important information in the Results tab. The dialog is:



Queue, FIFO dialog - Results tab (Macintosh)

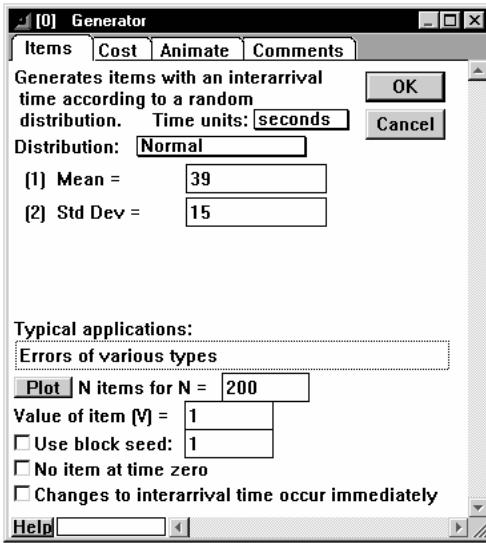
Extend

Adding a random element

Now that you have examined the dialogs, you can try changing some of the parameters to observe the effects. If bank customers always arrived at exact intervals, it would be no problem to schedule just the right number of tellers to always eliminate lines. In the real world, of course, people arrive at the bank at random intervals. Extend makes it easy to simulate random elements so that you can see what might happen in the real environment. Adding randomness to your model for the purpose of exploring possible results is known as *Monte Carlo simulation*.

To make the time at which the clients enter the bank random instead of constant, open the Generator block and change the function to Normal to generate a normal (bell curve) distribution. Enter a mean of 39 and a standard deviation of 15 in the two boxes near the bottom of the dialog. A mean of 39 means that a customer will arrive approximately once every 39 seconds, as before.

The standard deviation of 15 means that the bell curve will be fairly wide around 39, so that most customers will arrive about 39 seconds apart, but some will arrive a few seconds or a minute apart.



Extend

Specifying a random distribution (Windows)

Every run of the model will now be different. To see this randomness, choose Run Simulation again (you do not need to close the plotter first). Notice that the shape of the red line, the number of people standing in line, has changed significantly. If you run the simulation a few more times, you will see many variations on the red line.

Other modifications

As you can see from this discussion, Extend gives you many opportunities to change the way a model runs and explore different scenarios. The “Bank Line” model can be extended further in many ways that will not be fully explored here since they involve more advanced techniques. For example:

- You can add more real-life contingencies by adding blocks to the model. For example, if you walk into a bank wanting to get a small amount of cash but you see a line of 25 people, you might turn around and leave. You can add this contingency to the “Bank Line” model as a block that looks at the value of the *L* connector (the length of the queue) on the queue. If that value is above a certain number, a certain percentage of customers will automatically leave the simulation without waiting in line.
- You can change the assumptions in the model based on the time of day. For instance, if you were simulating the line over a whole day instead of just an hour, you would have to take into account tellers going on break or to lunch. You may also want to change tellers’ speed ratings during the day to simulate their changing efficiency. Of course, in a real bank customers come

in more often around lunch and after work, so you would also need to change that in the model as well.

- You can view the status of other blocks by connecting their outputs to the plotter, by leaving their dialogs open while running the simulation, or by taking some of their dialog items and putting them in the model window or Notebook, as described in “Notebooks” on page E79. You can also choose to leave informational blocks open while running the simulation so that you can see changes in values without plotting them.

Hopefully, you now understand how powerful Extend models are. The next chapter shows you how to build a model from scratch. When you are finished with that chapter, you can see how easy it was to create the “Bank Line” model and how you will create your own models.

Chapter 2: Building a Model

*A description of how to build a model and how
to use Extend's model window features*

*“He builded better than he knew;
The conscious stone to beauty grew.”*
— Ralph Waldo Emerson

The previous chapter described how to run and modify a model. Of course, models must come from somewhere. This chapter describes the steps in creating a model from the libraries that come with Extend. You will see that the process of putting together a model is quite straight-forward.

The Lake Pollution model

The “Bank Line” model in Chapter 1: Running a Model was based on the Discrete Event library. The model in this chapter, “Lake Pollution,” uses continuous simulation and is therefore based on the Generic library.

The reason that this model uses continuous simulation is that there are no “events” in the model. Since each step in the simulation is a year, Extend simply walks through the simulation step by step, calculating values as it follows the connections in the model. In discrete event modeling, Extend waits until an event takes place before any of the blocks take any action; in continuous modeling, action happens every step. Continuous and discrete event modeling are described in much greater detail in Chapter 4: Fundamental Continuous and Discrete Event Modeling.

The “Lake Pollution” model shows the effects of pollution being discharged into a lake over a long period of time. The output is a simple plot. The blocks in this model come from the Generic and Plotter libraries.

Like the “Bank Line” model, the “Lake Pollution” model will evolve as you go through this chapter. The basic concept behind the model is that you want to simulate what happens as pollution is dumped into a lake while the water in the lake is constantly changing. There are two polluters: one who puts a fixed amount of pollution in each year for 20 years and then stops, and a second who puts a smaller amount in for all 40 years. The data for the first polluter comes from a table while the second polluter discharges a random amount. The lake’s volume remains constant because inflow and outflow are equal. While the lake loses about 18% of its volume each year due to outflowing streams, it makes it up in freshwater rain. You can change the outflow rate and the pollution output by the two polluters and see how the new assumptions change the levels of pollution.

The “Lake Pollution” model is an example of a growth and decay model. It shows growth because the amount of the pollutant in the lake increases over time, and shows decay because after one of the polluters stops, the amount of pollution leaving the lake is larger than the amount arriving.

This model is a simple Extend model, however it would be difficult to set up in a spreadsheet like Excel. You will also see that changing assumptions in Extend is much easier than changing assumptions in a spreadsheet.

For your reference, the final version of the “Lake Pollution” model is located in the Tutorial folder or directory. However, you will learn more about Extend and modeling if you build the model yourself.

To start building a new model, choose New Model from the File menu. Extend opens a new model window. (If you just started Extend, you don't need to use the New Model command because Extend starts with an untitled model.)

Adding blocks

As described in detail below, the four steps to adding blocks to a model are:

- ▶ Open the library, if necessary
- ▶ Add the block to the model
- ▶ Move it to the desired position
- ▶ Connect it to blocks before and after it in the model

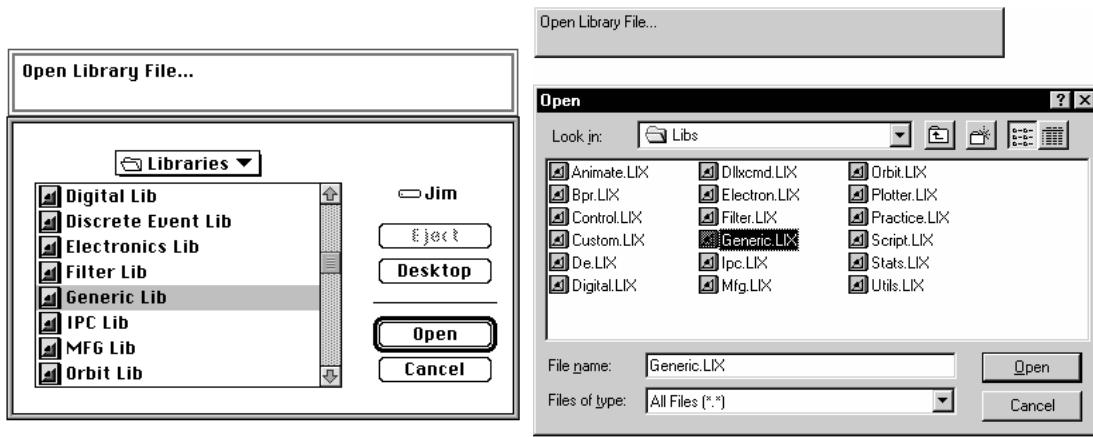
Extend

Opening the libraries

In order to copy a block into a model, the library in which that block resides must be open. Libraries and library maintenance are discussed in “Chapter 5: Using Libraries” on page E137. For now, all you need is to have the Generic and Plotter libraries open so that you can get blocks from them.

To open the Generic library:

- ▶ Choose Open Library from the Library menu.
- ▶ In the dialog, locate and select the Generic library. Unless you moved it, it is in the Libraries folder in the same folder or directory as Extend:



Macintosh

Windows

Opening a library

- ▶ Click Open to open the library. Once they are opened, the libraries are listed at the bottom of the library menu.

Use the same steps to open the Plotter library, which is in the same folder or directory as the Generic library.

Adding a block to the model

Extend

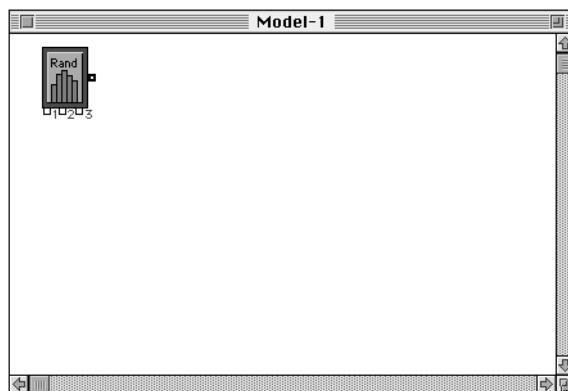
As stated earlier, your models do not actually contain blocks. Instead, your model holds references to blocks that are in libraries. In your model, you see the block's icon and, when you double-click on the icon, you see the block's dialog.

To add a block:

- ▶ Click in the Library menu.
- ▶ Drag down to the name of the library that holds the desired block. When the library is selected, you will see a hierarchical menu of the different types of blocks in the library. When a type is selected, the names of all the blocks of that type in the library appear to the right (see “Block types” on page P56 for a discussion on block types).
- ▶ Select a type, drag to the right and then down the list to the name of the block you want.
- ▶ Let go of the mouse button.

This puts a copy of the block in the upper left corner of the window and selects it.

To start constructing the “Lake Pollution” model, add an Input Random Number block from the Inputs/Outputs submenu of the Generic library. The block is selected when you add it to a model; to deselect the block, click anywhere else in the window. The model looks like:



Input Random Number block added (Macintosh)

Moving blocks

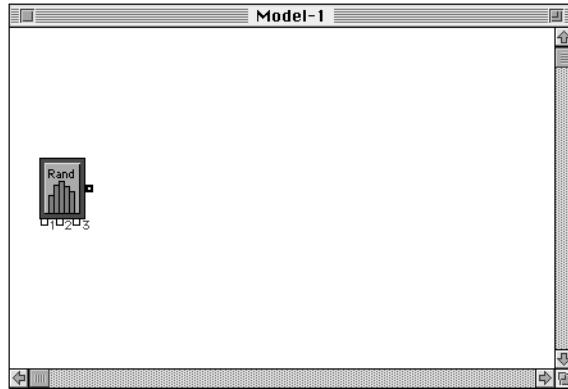
To move a block, click on the block and drag it with the mouse. When the cursor is over a block, it turns to a drag hand:



Click on the block, drag it to the desired position, and let go. You can also move selected blocks one pixel at a time using the arrow keys.

Move the Input Random Number block that you just added down a bit in the window:

Extend

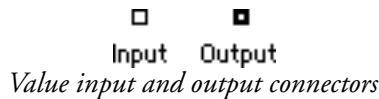


Input Random Number block moved (Macintosh)

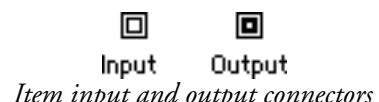
Connecting blocks

Connectors are used to hook blocks in your model together. Connectors are linked by connections, the lines that you see between the blocks in the model window.

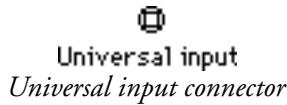
There are several types of connectors in Extend. Many blocks use value input and output connectors to pass values:



Discrete event blocks can also use value connectors to pass values, but their main connectors are item input and output connectors which pass entities (called items):



Universal input connectors can be connected to either value or item connectors:



In the “Bank Line” model from the previous chapter, the blocks use item input and output connectors to pass items and use value connectors to pass values (these are described in more detail in Chapter 4: Fundamental Continuous and Discrete Event Modeling). Extend makes sure that you connect the right types of input with the right types of outputs. For example, if you try to connect an item output to a value input, Extend will stop you. Since the “Lake Pollution” model is a continuous model, the blocks in this chapter only use the value input and output connectors.

Output connectors can be connected to more than one input connector. In the “Bank Line” model from the previous chapter, notice that the output of the “Waiting line” block is connected to three different tellers. The effects of using multiple connectors is different between discrete event and continuous models, as described in “Chapter 7: More Modeling Techniques” on page E193.

You cannot, however, connect more than one output to a single input. This difference makes sense because the information flowing out of an output connector can be useful in many places, but each input connector can only have one source of information. Blocks that need to have many sources of input have a separate input connector for each piece of information.

In Extend, the behavior of most connectors is predefined. This makes model building easy since you can connect blocks and run simulations without having to define an equation or formula beforehand.

The most common method for a block to pass information to other blocks is by being connected to their connectors. Connecting connectors in Extend is easy.

To connect an output connector on block A to the input connector on block B:

- ▶ Move the cursor to the output connector of block A. The cursor changes from an arrow to a technical drawing pen:



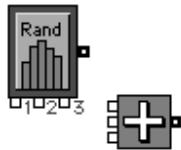
- ▶ Click in the output connector, then drag the line to the input connector on block B. You can tell when you are over the connector because the line you draw becomes thicker.
- ▶ Let go of the mouse button.

Be sure not to release the mouse button until you see the line thicken. If you accidentally let go and the line is not connecting the two connectors, you can easily get rid of it. Click on the line so that it thickens (indicating that you have selected it) then press the Delete or Backspace key to get

rid of it, or choose Clear Connection from the Edit menu. You can then make the correct connection again.

To test out your connection skills, you will add a second block to the “Lake Pollution” model and connect it to the Input Random Number block. Since Extend always puts new blocks at the last place you clicked in the model window, you should click near where you want the block before you add each block. Click on the worksheet to the right and slightly below the block. Add an Add block from the Math menu of the Generic library and move it so it looks like:

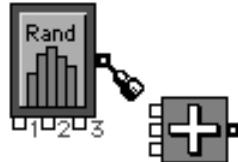
Extend



Position of the first two blocks

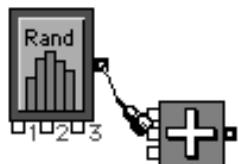
Now connect the output of the Input Random Number block to the top input connector on the Add block.

- Move the cursor to the output connector of the Input Random Number block:



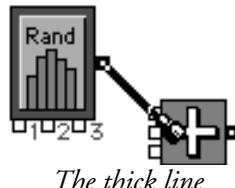
Pen over output connector

- Click in the connector, then drag the line towards the Add block:



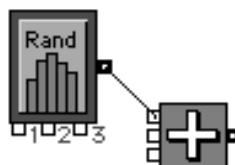
Creating the connection

- Move the cursor to the top input connector on the Add block. The line becomes thicker.



Extend

- Let go of the mouse button. The line becomes thin and the connection is made.



The connection made

If the connection was not made correctly, the line will show as a dotted line instead of the solid line you see above. In that case, click on the line so that it thickens (indicating that you have selected it) then press the Delete or Backspace key to get rid of it. Then redo the connection.

Other types of connections

Usually, connections are made by drawing a straight line between two connectors as you did in the previous section. Extend also lets you connect through *multi-segment connections* and through *named connections*. These two methods allow you to make your model more attractive and easy to follow. Multi-segment connections involve drawing a connection using one or more *anchor points* at each bend in the connection line. Named connections are text labels that are used to represent one output at many locations in your model. Both multi-segment connection and named connections are described later in this chapter.

Putting together the model

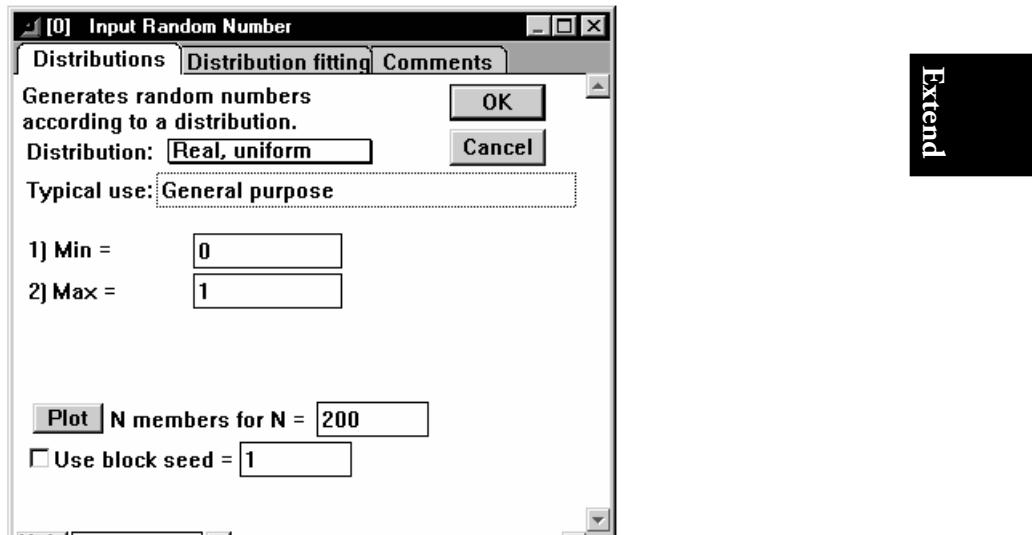
Now that you know how to add blocks to a model and how to connect them, you can put together your model. As stated above, your data for this model comes from a table of values and from a random sample. The model will only require a few more blocks and, of course, a plotter for the output.

Random pollution source

If you measure pollution per year over many years from a factory that is not growing, you will find that the values are fairly consistent from year to year. This can be expressed as a normal curve where there is little deviation from the average or mean. In this model, the Input Random Number block is used to specify a random distribution of pollution from one of the two sources. (As

you saw in Chapter 1: Running a Model, adding randomness to your model is known as Monte Carlo simulation.)

When you open the block, its dialog looks like:



Input Random Number dialog (Windows)

The popup menu specifies the type of random distribution you want. Below the popup menu, the number of input parameters and their labels change according to the type of distribution you choose.

- ▶ Select the LogNormal distribution in the popup menu. This gives a normal distribution where all the numbers are positive. The labels on the entry boxes change to:

1) Mean =	<input type="text" value="0"/>
2) Std Dev =	<input type="text" value="1"/>

Entry boxes for LogNormal distribution

Assume that the factory puts out about 1000 cubic meters of the type of pollutant you are measuring into the lake per year. At least two thirds of the time (that is, about one standard deviation from 1000), the amount you actually put in will be between 500 and 1500 cubic meters.

- ▶ Enter the two numbers:

1) Mean =	1000
2) Std Dev =	500

Mean and standard deviation entered

Extend

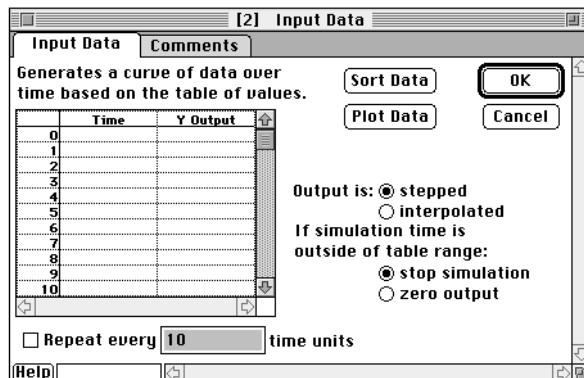
You can click on the Plot N Members button to verify that the majority of numbers fall between 500 and 1500.

- ▶ Click OK to close this dialog.

Constant pollution source that stops

A second type of polluter is one who puts out a fixed amount of pollution each year. In this model, the second polluter dumps a larger amount (4000 cubic meters) into the lake each year, but then stops at 20 years. Since the model runs for 40 years, you will be able to see the effect on the lake once this polluter stops. The Input Data block is used to specify a certain output over time in a table format.

- ⇒ Add an Input Data block from the Inputs/Outputs menu of the Generic library to the model and drag it below the Input Random Number block.
- ⇒ Double-click on its icon. You see:



Input Data dialog (Macintosh)

In the table, the left column specifies the time and the right column specifies the output desired. At each step, Extend checks the table for the time that is less than or equal to the current simulation time and outputs the corresponding number to its right.

- ⇒ Indicate that the output should be stepped by clicking on the “stepped” choice:

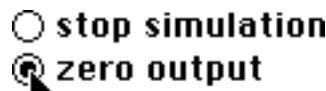


Choosing stepped values

This means that Extend will use the exact values you enter in the table, not an interpolated amount.

Extend

- ⇒ Click on the “zero output” choice to indicate that the simulation will output 0 when the simulation time exceeds the range specified in the table.



Choosing zeroed output

This is useful because you may want to run the model for longer than 40 years. Because this polluter stops polluting at 20 years (outputs 0), having the block output 0 will have no effect on the results, but will allow you to run the simulation for longer than the table values specify.

- ⇒ Enter the following table of numbers for the amount of pollution from this factory:

Row	Time	Y Output	Up
0	0	4000	
1	20	0	
2	40	0	

Table for pollution amounts

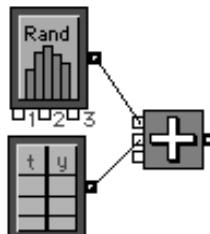
This means that from time 0 (the first step) until time 20, exclusive, the block will output the number 4000, meaning that the pollution rate is 4000 cubic meters for the first 20 steps (years). At time 20 until the end of the simulation run (time 40) the block will output 0.

- ⇒ Click OK.

Extend

Combining the sources

The Add block will be used to combine the values for the two factories. Simply connect the output from the Input Data block to the second connector on the Add block:

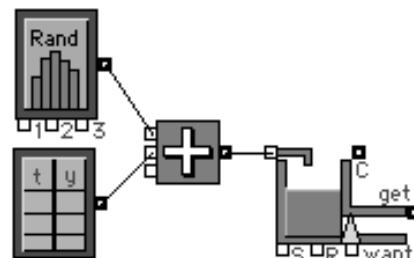


Two blocks connected to the Add block

Pollution in the lake

The Holding Tank block is used to model the level of the pollution in the lake and the amount of pollutant that flows out with the lake water. The Holding Tank has many connectors, but the three you will use in this model are the top left input connector, the *C* connector, and the *want* connector. A Holding Tank block takes in amounts at the top left input connector and accumulates them. The contents are reduced by whatever you ask for at the *want* connector. You can check the current level in the block by looking at the *C* (contents) connector.

- ▶ Add a Holding Tank block from the Holding submenu of the Generic library to the model and drag it to the right of the Add block.
- ▶ Since the inputs are in rates per year, select the “integrated (delay)” choice in the Holding Tank block’s dialog. For more information about integration, see “Integration vs. Summation in the Holding Tank block” on page E277.
- ▶ Connect the output of the Add block to the top left input connector on the Holding Tank block.



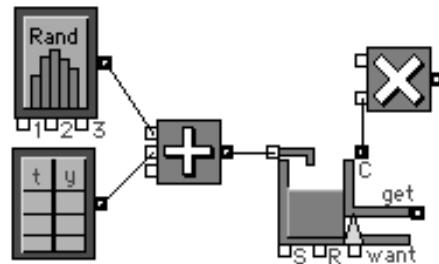
Holding Tank block added

You may have noticed that the connection in the example is nice and perpendicular while the one that you made may be slightly diagonal. You can move your blocks at any time, even after they have been connected to other blocks. To make the connection lines neater after you make them, move the blocks by clicking and dragging.

Determining the outflow

Now add a Multiply block to determine the amount of pollutants that are flowing out of the lake each year due to seepage and outflow. The result of multiplying the contents of the Holding Tank block (that is, the current pollution amount) by the outflow rate determines the amount of pollution that leaves the lake that year.

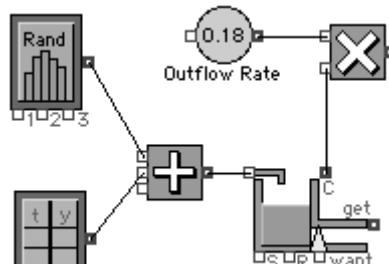
- ▶ Add a Multiply block from the Math menu of the Generic library to the model and move it above the Holding Tank block as shown below.
- ▶ Connect the *C* (Contents) connector on the Holding Tank block to the bottom input connector on the Multiply block.



Multiply block added

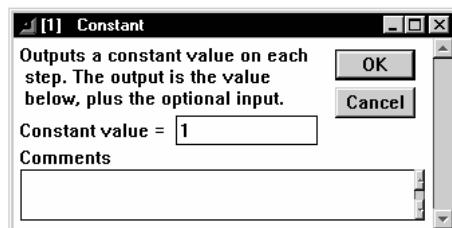
Next, indicate the outflow rate. In this model that rate is a constant.

- ▶ Add a Constant block from the Inputs/Outputs menu of the Generic library to the left of the Multiply block.
- ▶ Connect the output of the Constant block to the input of the Multiply block.



Constant block connected

- ▶ Double-click on the Constant block to show its dialog.



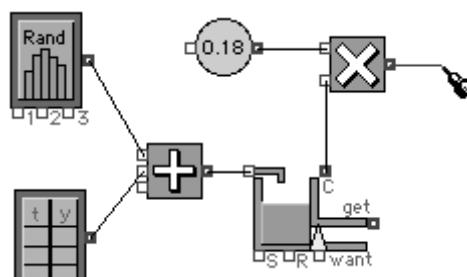
Constant dialog (Windows)

- ▶ Change the value to **0 . 18**. This indicates that the multiplier of the lake contents will be 0.18 or 18%. Click OK to close the dialog.

Taking out the pollutants from outflow

To remove the pollution, you now need to connect the output of the Multiply block to the *want* input of the Holding Tank block. This will determine the amount of pollutant that flows out with the lake water, based on the amount of pollutant in the lake and the outflow percentage (the constant 18%). If you made this connection directly, it would go through the icon of the Holding Tank block and would probably be fairly unsightly. This is a perfect use for anchor points. You will draw a line out from the output connector on the Multiply block, down the model, then back up to the Holding Tank block, pinning each bend with an anchor point, creating a *multi-segment* connection.

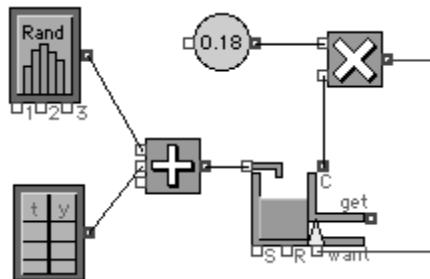
- ▶ Click in the output connector of the Multiply block and drag to the right of the Holding Tank block.



Drawing the first segment of the connection

- ▶ Release the mouse button. This creates the first segment. The cursor remains a technical pen because you are pointing at an anchor point. Immediately click again and drag down.

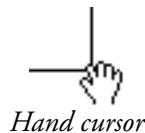
- Release the mouse button. Click again and drag to the left, to the *want* input connector on the Holding Tank block. When the line thickens, release the mouse button; you will have connected the two connectors through two anchor points.



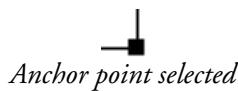
The connection completed

Anchor points are like blocks in that they can be moved. If your connections did not come out exactly where you want them, you can move the anchor points.

- Move the cursor over one of the anchor points. It becomes a hand.



- Click the hand to select the anchor point. If you move the cursor away, you can see that there is a handle on the point:



- Click on the selected anchor point and drag it to the desired location.
► Deselect the anchor point by clicking somewhere else in the model window.

To delete one segment of a multi-segmented connection, click on the line so that it thickens, then press the Delete or Backspace key or choose Clear Connection from the Edit menu. To delete all segments of a multi-segment connection, double-click on one segment, then press the Delete or Backspace key or choose the Clear Connection command.

Saving the model

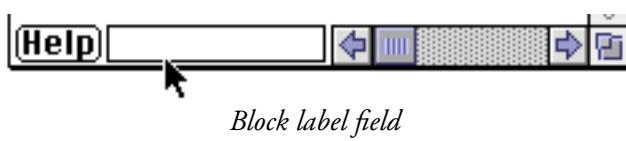
This is a good time to save the model so far. Choose Save Model As from the File menu and give the file a name: *My Lake Pollution*.

Labeling the blocks and adding comments

Your model now has six blocks. Although you can try to remember each block's function, Extend lets you add *labels* to your blocks to document their purpose in the model.

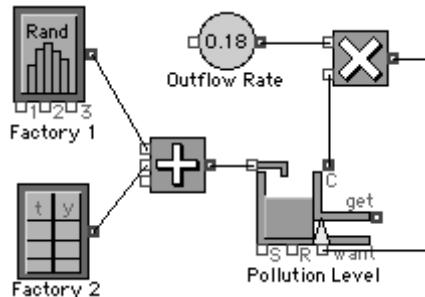
To add a block label:

- ▶ Open the block's dialog.
- ▶ Click in the block label field near the bottom scroll bar:



- ▶ Type your desired text. You can enter up to 15 characters, including spaces.
- ▶ When you are finished typing, close the dialog. The block label will appear at the bottom of the block.

For instance, add the following labels to your blocks:



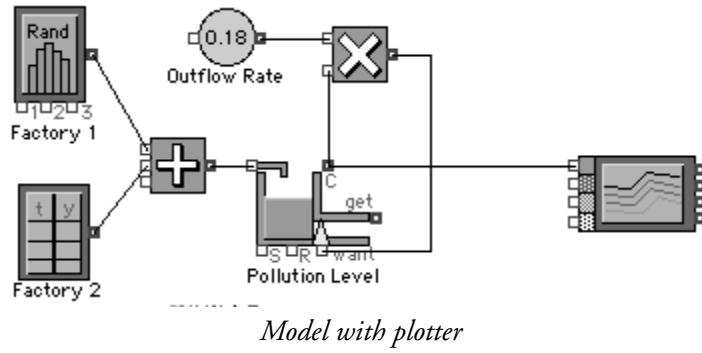
Model with block labels added

Most block dialogs also have a *comments* field or tab you can use to make notes about the block's usage. Comments are limited to 255 characters.

Adding the plotter

- ▶ Add a Plotter I/O block to the right side of the model. This block is in the Plotter library.

- ▶ Connect the C (contents, or amount of pollution in the lake) output connector of the Holding Tank block to the top input of the plotter.

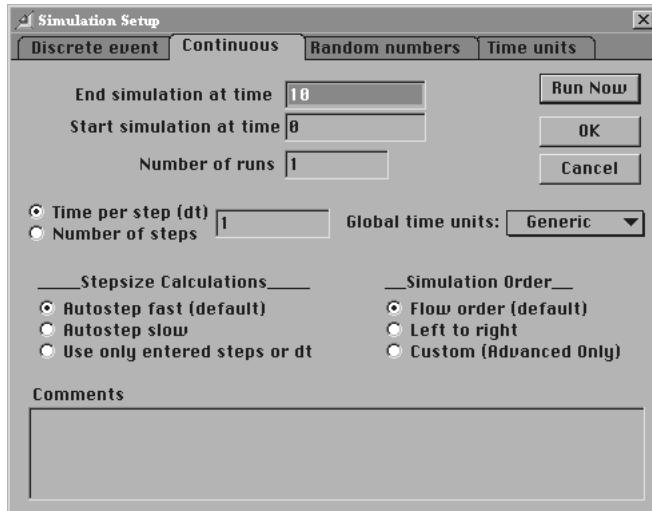


The model is now completed. You should save this model again since you have added the plotter and block labels.

Running the simulation

Simulation Setup dialog

The Simulation Setup command from the Run menu lets you specify how the simulation will run and for how long. The dialog has tabs for discrete event and continuous models as well as tabs dealing with random numbers and time units. The tab for continuous models looks like:

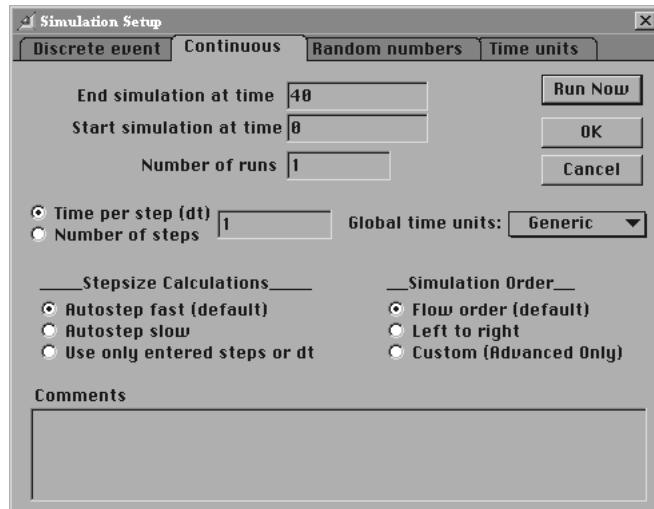


Continuous tab of Simulation Setup dialog (Windows)

Each time you run a simulation, Extend uses the same dialog values entered in the Simulation Setup dialog. Thus, you will usually only use the dialog once per model.

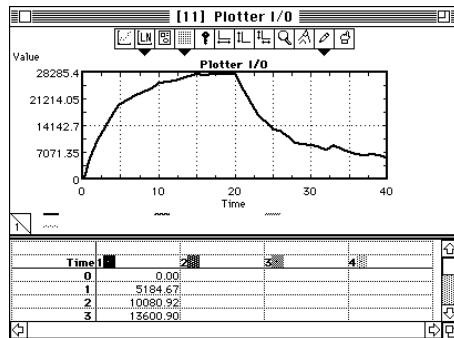
Generally, the only setting you will need to change in the dialog is the “End simulation at time” parameter located on the Continuous and Discrete event tabs. For most purposes, you want the simulation to start at the beginning, so you would use the default start time of 0. Also, most continuous simulations use a time per step (dt) of 1. You would only change the “Number of runs” option if you want to repeat the simulation to perform sensitivity analysis and look at how results change over many runs. The “Comments” field is a convenient place to store information such as the time units used in the model, or why a particular dt was used. The Simulation Setup command is discussed fully in “Simulation Setup command” on page E204.

For the “My Lake Pollution” model, set the end time to 40 and use the default start time of 0 and default dt of 1. This means that the model will step from time 0 to time 40 inclusive. Each step will be 1 time unit long, or 1 year. When you run the simulation, you will see the level of pollution each year for 40 years.



Continuous tab set for 40 years (Macintosh)

Click the Run Now button. Notice that the choices in the Status Bar at the bottom of the screen are based on the settings in the Simulation Status dialog. The simulation runs and you see the following output for the plotter:



Scaled plotter output (Macintosh)

Extend

At the end of the simulation, the plotter will automatically rescale its axes. Notice that the upper and lower limits on the Y axis may be slightly different in your plotter since the amounts of pollution from one of the two factories is random. The lower limit will be around 5000 and the upper limit around 30,000 cubic meters. Plotters are described in much more detail in Chapter 6: Input and Output.

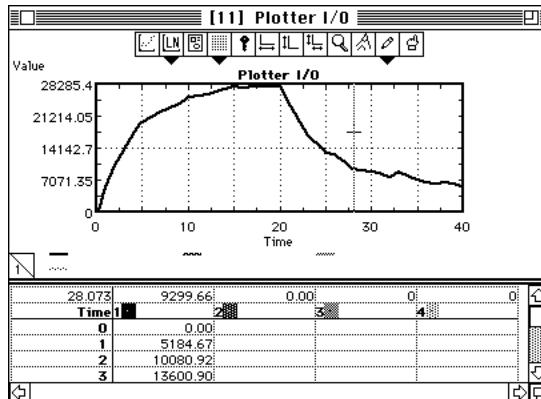
Note See “Plotter dialogs” on page E157 for information on how to cause plotters to automatically scale their axes. Also see “Autoscale tools” on page E156 to see how to manually autoscale plotter axes.

You can now see what the plotter tells you about your model. Note that the pollution begins to reach an equilibrium near 16 years as the line flattens out. This indicates that the lake is losing as much of the pollutant as it is gaining. The decay is also asymptotic, indicated by the flattening out after ten years. The level of pollution in the lake decreases sharply for the first 10 years after the larger factory stops polluting, then decreases more gradually.

You can scroll through the data table at the bottom of the window to see the values that correspond to the line in the plot. You can also scan the plotter simply by moving the cursor over the

Extend

plot. As you do, you see a vertical line that connects to the X axis. Above the table, you see the coordinates of where the vertical line crosses the trace. For example, near time 28, you see:



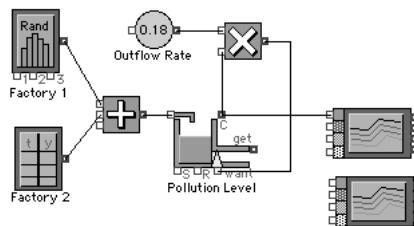
Plot near time 28 (Macintosh)

Adding a second plotter

You can have as many plotters as you want in a model. You may want to add a second plotter if you have used all four lines of the first plotter or if you simply want to see the various lines on different plots.

For example, you may want to plot just the amount of pollution that is being added each year. This would use a different scale than total pollution since the amount would be so much lower than the highest amount of total pollution.

- Add a second Plotter I/O just below the first one. There are three ways to do this:
 - You can pull another Plotter I/O from the Library menu.
 - You can select the plotter on the model window, choose Copy, click on the model window just below it, and choose Paste.
 - You can select the plotter, choose Duplicate from the Edit menu, and move the duplicate plotter to the desired location.

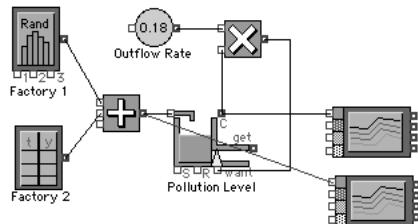


Second plotter added

Additional ways of connecting blocks

There are four ways to connect this plotter to the output of the leftmost block:

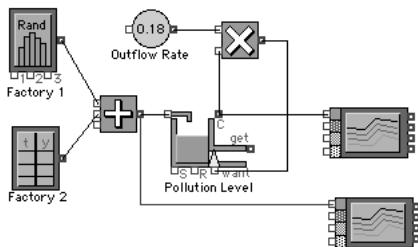
- Connect the blocks directly with a straight connector:



Direct connection

This has the disadvantage of having a connection run directly over other blocks. Delete this connection by clicking on it and pressing the Delete or Backspace key.

- Connect the block with a right-angle connection. This lets you change the way the line lies and is described in “Connection line characteristics” on page E73.
- Use an multi-segment line and anchor point as shown earlier. The result will look something like:



Connection through an anchor point

Although this works fine, the model is starting to get lots of lines in it which may be distracting. Delete this segmented connection by double-clicking on it and pressing the Delete or Backspace key.

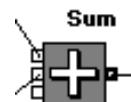
- Use a *named connection*. Named connections are text labels that are used to represent one output at many locations in your model. If you have two text labels with the exact same text, you can use these to have the flow of data jump from one part of the model to another. As you drag a line between a connector and a piece of text, the line thickens, indicating when you are connected to the text.

Note In named connections, you must use identical spelling in the text names (including spaces and returns), but named connections are not case sensitive.

To add a text label for the named connection:

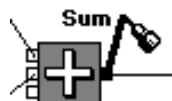
- ▶ Double-click on the model window where you want the text. This opens a *text box* for entering text. (Working with text is described in more detail in “Text” on page E69.)
- ▶ Type your desired text.
- ▶ When you are finished typing, click anywhere else on the model window or press the Enter key.

For example, add the label “Sum” above the Add block:



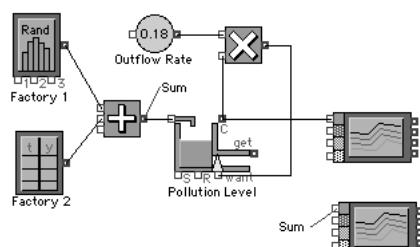
Sum label added

You can connect the output of the block to the text that is directly above it. Drag between the output connector and the text until the line thickens:



Line connected to text label

Near the lower plotter, add the text **Sum**. You do this by clicking once on the “Sum” text, choosing Duplicate from the Edit menu, then moving the text to the desired location. Make a connection between this text and the top input connector on the plotter using the same technique you used to connect between two connectors:



Named connection completed

In any of the four examples, the second plotter now plots the amount of pollution being discharged into the lake each year. When you run the simulation, the second plotter will open on top of the first one (you will have to scale the Y axis for this plotter also).

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. You can see the connection by choosing Show Named Connections from the Model menu.

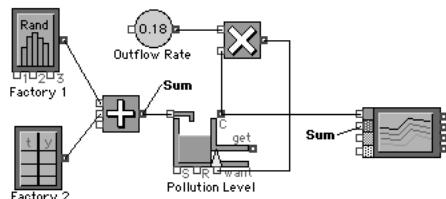
Note As discussed in “Chapter 7: More Modeling Techniques” on page E193, named connections only work in *one* level of the model at a time. If you have a named connection inside a hierarchical block, it will not transmit data to the named connection outside of the block.

Extend

Plotting to the Y2 axis

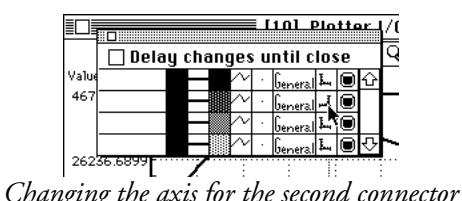
The “My Lake Pollution” model now has two plotters, one with a scale that goes up to almost 30,000 and another with a scale that only goes up to about 6,000. Some people prefer to see these two traces separately, but you might also want to see them on a single plot. Extend lets you have two scales in a single plot so that you can plot traces which are measured in different units.

To see this, you need to rearrange the “My Lake Pollution” model a bit. Delete the second plotter that you added by selecting it and pressing the Delete or Backspace key. Notice that this also removes the connection from the “Sum” label to the now-missing block. Connect the second input connector on the remaining plotter to that “Sum” label:



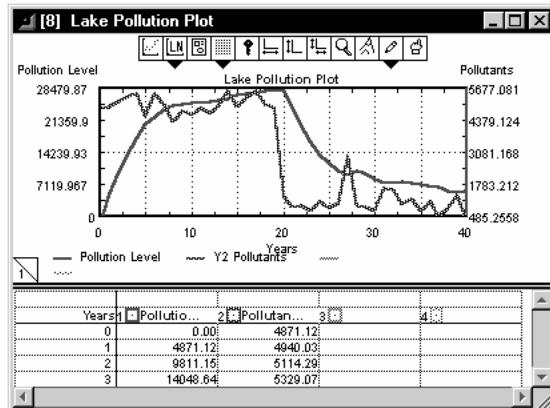
New configuration

You may now change the plotter to display two vertical axes: one for 30,000 and one for 6,000. Double-click on the plotter block’s icon to open the plotter, then click the icon, the first tool at the top of the plotter. In the dialog that comes up, click on the L-shaped icon on the second line:



Changing the axis for the second connector

This indicates that you want to change the Y axis for the second connector's input so that it uses the Y2, or right axis. The icon becomes , indicating that data from this input will be plotted on the right side. Click the close button in this dialog. Run the simulation again, then click on the  icon near the top of the window to scale the Y axes. The resulting plot looks like:



Plot with two lines and two axes (Windows)

Note that, as expected, the yearly amount jumps all around, yet the total amount in the lake moves in a very even curve.

Working with Plotters is described in detail in “Plotters” on page E150. Since you are finished with this model, you can close the model and save your changes.

Other modifications

As you read the chapter, you probably thought of many ways that you could extend the “My Lake Pollution” model. You will get such thoughts almost every time you make a model in Extend since it is so easy to add features to models. Some ideas for expanding this model include:

- Add blocks to represent more accurate measurements of pollution or additional factories.
- Have the Holding Tank block integrate its inputs and set delta time to a value less than 1, as is done in the “Green Lake Pollution” model. This calculates output values between the steps, so you can see finer resolution of the model results.
- Try many different values for the expected outflow rate, or make it random.
- Make the outflow rate a function of time instead of a constant.
- Add animation to represent pollution levels. For example, the Cedar Bog Lake model in the Science and Engineering folder.

You have now seen how to run models and how to build them. The next chapter shows you some examples of Extend's presentation and authoring features. As you look at these features, you will probably continue to think of modifications you can make to this model.

Other examples

Chapter 4: Fundamental Continuous and Discrete Event Modeling contains more examples of building models using the Generic and Discrete Event libraries. In addition, on the disk there are numerous examples of using Extend in various disciplines.

Extend

Extend

Chapter 3: Enhancing your Model

*Methods and techniques that present
your models in the best light*

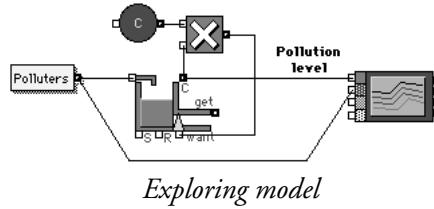
“Beauty is nothing other than the promise of happiness.”
— Stendhal

Now that you know how to run a model and how to build one, you are probably wondering what else there is to learn. Extend is more than a simulation program. Extend also allows you to organize and present your models in a form that is aesthetically pleasing and comprehensible.

As you may have guessed, some models can be complex. The features shown in this chapter help simplify building, running, and presenting models. When you create models, Extend's authoring features allow you to transform your models into easy-to-use, multi-media environments.

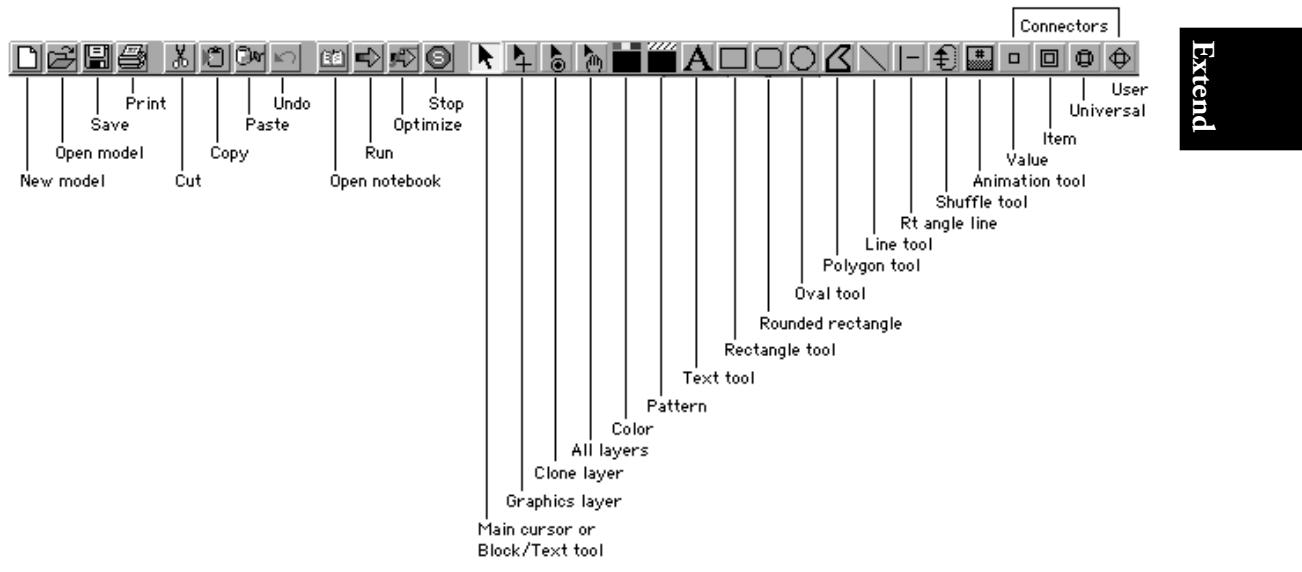
In this chapter, you will explore using the tools in the toolbar, model animation, hierarchy, cloned dialog items, Notebooks, Control blocks, typing in equations, sensitivity analysis, and user-interaction. After reading this chapter, you will know most of what you need to modify the model worksheet so your models express your ideas as well as perform the modeling actions.

For this chapter, you will be using the “Exploring” model in the Tutorials folder within the Examples folder. The model is an extended version of the “Lake Pollution” model from Chapter 2: Building a Model. With Extend running, choose Open Model from the File menu to open the “Exploring” model. The model looks like:



Tools and buttons

Use the tools and buttons in the *toolbar* at the top of the screen to edit files, to add text and drawing objects to your models, or to select objects in the model. Some of the following tools are available when a worksheet is active:



Tools in the toolbar

(The Animation and connector tools are available when you are building or modifying a block. These are described in “Chapter 1: Parts of a Block” on page P3 and are of no concern to you now.)

Files

The first group is a standard set of buttons that deal with files. You open a new model worksheet by clicking on the *New model worksheet* button. The *Open* button will open existing worksheets or text files. The *Save* and *Print* buttons respectively save and print the active worksheet or text file.

Editing

The second group is a standard set of buttons for editing. The *Cut* and *Copy* buttons respectively cut and copy the current selection. The *Paste* button will paste the contents of the clipboard into the active window. Use the *Undo* button to reverse the previous action. Clicking the *Undo* button a second time will redo the action.

Models

The next two buttons deal with the model in the active worksheet. The *Open notebook* tool is used to open the model’s notebook. The simulation can be run by clicking the *Run simulation* tool.

Selecting

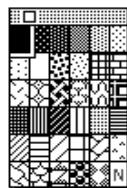
Although the model worksheet appears to be flat, it is actually composed of several layers which hold objects such as blocks, text, pictures, and so forth. The first three tools in this group (Main cursor, Draw layer, and Clone layer) let you select model or Notebook items (blocks, drawing objects, or cloned dialog items) in specific layers. The *All layer* tool will select all items on the worksheet or Notebook. Since you will probably use the *Main cursor or Block/text layer* tool most of the time, it is the default tool.

The specific selection tools are especially useful for selecting an item if both types of items are near each other in your model. For example, if you have a drawing item behind a block and you want to just move the drawing item, use the Draw layer tool. The tools are also helpful when you want to choose what to copy. For instance, to copy the entire model worksheet, including drawing objects and cloned dialog items, use the *All layers* tool. To copy only specific objects, use the tool for that type of object. Tool usage is more fully described later in this chapter, along with the types of objects they manipulate.

When windows other than the model window are active, the *Main cursor or Block/Text layer tool* is still the default tool, but changes its behavior to accommodate the most typical use. For example, when a Notebook is open, the default tool allows you to select drawing items as well as text.

Patterns and colors

Every drawing object has both a pattern and a color, chosen from the menu in the toolbar (text can have color but not a pattern). The default is solid pattern and black color. Click on the Pattern or Color icon to open the window. For example, the patterns are:



Pattern palette window

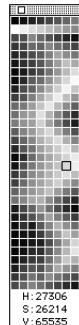
In these choices, black gives a solid pattern, white gives an opaque white pattern, and the "N" signifies no pattern, meaning transparent.

The Pattern palette window can be positioned anywhere on your screen by clicking on the top of the window and dragging it to the desired location. To close the window, click on the close box in the upper left corner.

Color palette

The color tools work similarly to the pattern tools. When you click on the Color icon, the Color palette window opens. You can type text in color or add color to text by selecting the text and

choosing a color from the color tools. As you will see with formatting text, if you change the color *before* you start a text box, all new text will be in that color.



Color palette



Notice that when you select a color the hue, saturation, and brightness (value), or HSV, settings for the color are listed at the bottom of the color palette window. This is helpful when you need to know the HSV settings for a given color (for example, see “Initializing animation objects” on page P185).

Text

The next tool is the *Text* tool. Use the *Text* tool to add text to your model, such as descriptive labels or names for named connections.

Working with the text box

To add text to your model, select the Text tool, click on your model where you want to add text (or just double-click in the model window with any one of the layer tools) to start a *text box*, then type in the text you want. To stop entering text, press the Enter key (located on the numeric keypad) or click anywhere else in the model window. You can access the text box after you have stopped typing either by selecting the Text tool and clicking once on the text you have already typed, or by selecting the Block/Text or All layers tool and double-clicking on the text.

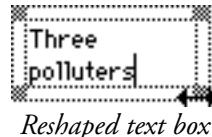
As you type, the text box is a gray box with four *handles*. For example, in the “Exploring” model, select the Text tool, then click above the block on the left and type **Three polluters**.



Text box

Before you press Enter on the numeric keypad or click elsewhere, click and drag on a handle to change the size and shape of the text box. For example, to make the box narrower and taller, put

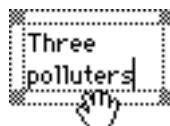
the cursor over a handle. When the cursor changes shape to a two-headed arrow, drag it to the left and down.



Reshaped text box

Extend

Moving text around the model is easy. Before you press Enter to finish entering text, you can move text by moving the mouse pointer to the border of the text box away from the handle so that it turns into a hand. Then click and drag the text box to the desired location:



Moving text before pressing Enter

If you have already finished entering text, simply move the cursor over the text so that it becomes a hand and drag the text to the desired location.

To copy the entire contents of a text box to a model window, click once on the text box to select it and choose Copy from the toolbar or give the Copy command from the Edit menu. Then click on the model window where you want to place the text and give the Paste command. To copy all or a portion of the text in a text box and paste it into another text box or onto the model worksheet, double-click the text box (or click once using the Text tool) and select the text to copy. Then double-click to create or open a text box and paste the text.

Note When you paste text onto the model worksheet or into another text box, it remains editable text. However, if you select and copy an entire text box and paste it to the Notebook, it becomes a picture. To paste editable text into a Notebook, copy the *contents* of the text box, not the text box itself.

You can delete text by selecting it with the cursor and pressing the Delete key, choosing Clear from the Edit menu, or choosing Cut from the tool bar.

You can also add formatting to text. To format existing text, access the text box, select the text, and then choose a command from the Text menu, such as Bold or Align Center. You can also change the color of text, as described earlier.

Note If you want to type new text with a particular format, select the desired format *before* you start the text box. Extend will remember that format every time you start new text. However, if you change the format of text within an existing text box, Extend won't use that format on the next new text you enter.

Drag and drop editing

Extend supports a text editing feature called drag and drop. Drag and drop is the easiest way to move a selection of text a short distance or between documents. This feature is an alternative to the Cut, Copy, and Paste commands.

To move text using drag and drop editing:

- ▶ Select the text you want to move.
- ▶ Point to the selected text and hold down the mouse button. When you drag the selection, an insertion point will appear to the left of the cursor. Drag the insertion point to the desired location and release the mouse button.

Extend

To copy text using drag and drop editing:

- ▶ Select the text you want to copy.
- ▶ Hold down the Option (Macintosh) or Alt (Windows) key, point to the selected text, and then hold down the mouse button while you drag the insertion point to the new location. Release the mouse button.

Drag and drop can be used within a text box on a worksheet, a text file, a block's structure window (see “Working with the structure and dialog windows” on page P7) and between any combination of the above with one exception: drag and drop 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 on the model worksheet, you can have only one text box open for editing at a time. However, you would be able drag the text to a text file window or a block's structure window.

Drag and drop can also be used to create *text clippings* (Macintosh only) as discussed on page E167. By dragging a piece of text onto the Finder (i.e. the desktop or a folder), you can create a text clipping that can be dragged into any other document of any application that supports drag and drop text.

Drawing

The next seven tools let you add drawing objects to the model and arrange them in layers. Use these tools to make your models easier to read or to make them more aesthetically pleasing.

The *Rectangle*, *Rounded Rectangle*, *Oval*, and *Polygon* tools add those shapes to your worksheet or Notebook. For example, to add a rectangle, select the *Rectangle* tool, click in your model where you want one of the rectangle's corners, and drag to the diagonally opposite corner. Similarly, you can use the *Line* tools to draw lines on the model. The *Line* tool will draw a line at any angle and the *Draw right angle line* tool restricts the lines to horizontal and vertical lines. You can also add colors and patterns to the shapes and lines you draw, as described earlier.

If you hold down the Shift key as you reshape the rectangle, rounded rectangle, or oval shapes, the shape will become a square or a circle. If you resize a square or circle while holding down the Shift key, it will maintain its proportional measurements.

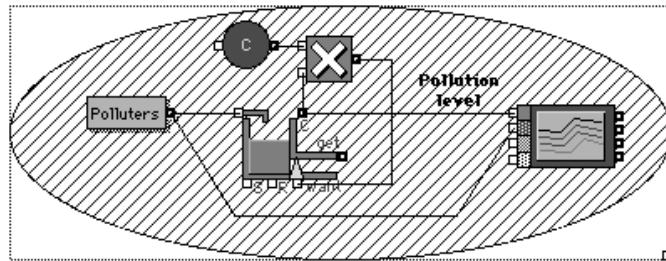
Borders

The Border command in the text menu becomes active when you select a drawn object. You can activate and deactivate the black border around objects with this command, just like you can with text.

Shuffle front to back

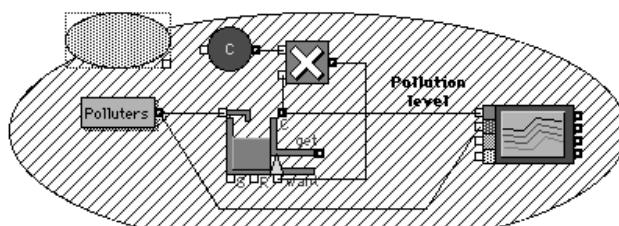
The *Shuffle graphics* tool lets you arrange drawing objects (not text) that are on top of each other. Click on this tool, then click its crosshairs on the object in the window. If the object is in front, it will be sent behind. If it is not in front, it will be brought to the front. Blocks, text, and cloned dialog items (described below) are always in front of any drawing item and therefore cannot be shuffled.

To see how the color and drawing tools work, select the *Oval* tool and draw an oval behind your model. Note that it comes out black and the text and icons come out on top of the oval automatically. Choose a new pattern for the oval from the Pattern tools, such as the striped pattern. Try changing the oval's color as well.



Oval drawn

Now draw a second oval near the top left of the first:



Second oval drawn

Experiment by changing the pattern of one or both to a different pattern and use the Shuffle graphics tool to change their order. Remove the ovals by choosing the Draw layer tool, selecting them, then choosing Clear.

Try selecting an oval and then using the Text menu *Border* command to add and remove the black border.

Connection line characteristics

The lines that make up connections can be formatted to show the connections better. The Connection Lines command from the Model menu lets you specify the style of the connections. To use the command, choose an option from the hierarchical menu and draw a new line:

✓	Ctrl-Shift-A
✓	Ctrl-Shift-D
→	Ctrl-Shift-E
←	Ctrl-Shift-F
✓ View Using Defaults	Ctrl-Shift-G
✓ —	Ctrl-Shift-H
—	Ctrl-Shift-J
—	Ctrl-Shift-K
—	Ctrl-Shift-M
✓ —	Ctrl-Shift-N
---	Ctrl-Shift-O
✓ Black Connections	Ctrl-Shift-Q
Color Connections	Ctrl-Shift-T

Macintosh

✓	Ctrl-Shift-A
✓	Ctrl-Shift-D
→	Ctrl-Shift-E
←	Ctrl-Shift-F
✓ View Using Defaults	Ctrl-Shift-G
✓ —	Ctrl-Shift-H
—	Ctrl-Shift-J
—	Ctrl-Shift-K
—	Ctrl-Shift-M
✓ —	Ctrl-Shift-N
---	Ctrl-Shift-O
✓ Black Connections	Ctrl-Shift-Q
Color Connections	Ctrl-Shift-T

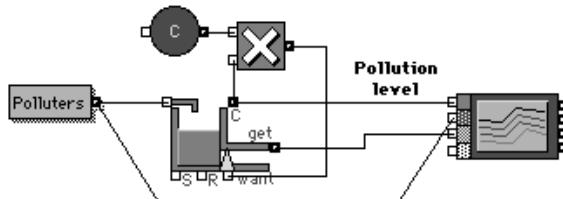
Windows

Connection Lines command

The top two choices tell whether the connection can be diagonal or must be made of right angles. If you choose right angles, Extend will make the selected connection into a three-part connection with anchor points. You can specify which type of line you want by default in the Preferences command from the Edit menu.

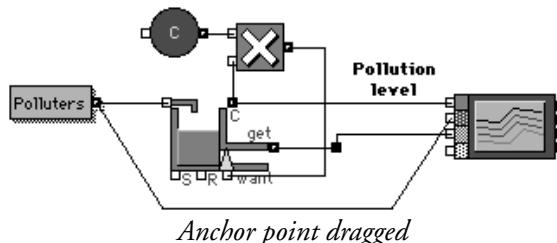
Extend

You can easily see how right-angle connections differ from diagonal ones. Select the right-angle choice from the Connection Lines command and draw a line from the *get* connector on the Holding Tank to the third input on the plotter:



Right-angle line drawn

Select the line you just drew and click on one of the anchor points and drag it to the left. Notice that Extend keeps the connections at right angles no matter where you drag it:



The second set of connection line choices specifies whether or not you want arrows on your right-angle connections (arrows can *only* be used with right-angle connections). The direction of the top arrow head follows the direction you draw the line when you make the connection; the bottom arrow head follows the opposite direction.

The third set of choices in the Connection Lines command specifies the look of the lines. When you connect blocks in Extend, you can choose to use Extend's default connection type (discussed below) or choose connection line styles and widths that add additional meaning to your model. You can choose to make connections thin, medium, thick, or hollow.

Extend also lets you automatically differentiate connection lines based on the type of connector. These default connection line types are:

Type of connector	Line type
Value	
Item	
Diamond	

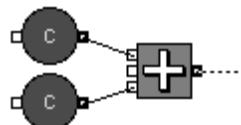
For example, most discrete event models track flows of items as well as calculate and show values. You can visually differentiate the flow of items from the processing of values by using the default line types. When looking at such a model, it is much clearer which flows are those of items and which are of values. For example, the “Bank Line” model in Chapter 1: Running a Model uses the default types to specify item and value connection lines.

To specify the default connection type, choose the View Using Defaults option. When you use this command, any connection line style choices such as dotted or thickness are ignored. You can leave the View Using Defaults command on while you build a model, or only turn it on when you are viewing the model. Note that this is only a view option, not a line style. You can choose in the Preferences command whether or not to use the default connection types.

Extend

The fourth set of choices in the Connection Lines command lets you make your right-angle connections appear as dashed lines instead of solid lines. This choice is not available for diagonal lines.

Note Connection lines and named connections that are unconnected at one end are shown as dotted lines. This gives you a quick way of seeing which lines are not connected in your model. For example, the output of the Add block here is unconnected:

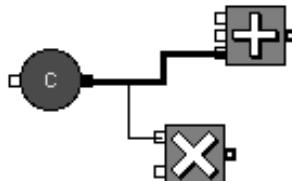


Unconnected line

The final set of choices 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. There is no limit to the number of different colors you can use for connection lines in a single model. Selecting Black Connections in the Connection Lines command causes all new connection lines to be drawn black regardless of the current color selected in the color palette. Choosing Color Connections will cause all new connections to be drawn using the color currently selected in the color palette. To change the color of an existing connection line, choose Color Connections, select the line, and choose a color from the color palette window.

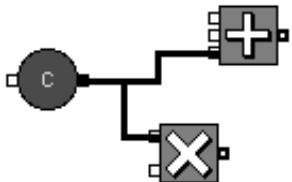
You can select all segments of a connection line between connected blocks using the mouse or the Select All Segments command in the Edit menu. This makes it easier to delete the entire chain of segments, or to change line properties, such as thickness, at once. This works with both regular connection lines and named connections.

By double-clicking on a connection line segment you can select all segments between two blocks. Any other line segments that may be connected to additional blocks will not be selected, as shown below.



Selecting all line segments between two blocks

To select all the connection line segments that connect from a single output (shown below), you double-click on a line segment while holding down the Option or Alt key. Alternately, you can click on one segment of the connection, then choose the Select All Segments command from the Edit menu.



Selecting all line segments

Cloning dialog items onto the model

In most programs, you see buttons and dialog parameters only in dialogs. The advantage of this is you always know where to find them. In very large models, having all your choices in dialogs can be a disadvantage. For instance, you may want easy access to parameters in several blocks that are scattered throughout the model. Extend overcomes this problem by giving you freedom to *clone* dialog items and place them in a more convenient location.

By default, dialog items appear in dialogs. If you want, however, you can drag a clone (a copy) of a dialog item to the model window, to a Notebook, or to the window of a hierarchical block. Notebooks are a natural place for cloned items; they are described later in this chapter and in Chapter 6: Input and Output.

You can clone the same dialog item to more than one location. You can clone from the dialog again, or you can make a clone of the clone. For example, you might want a clone in two parts of the model window. Every clone acts exactly like the original: if you change the original or any clone, all instances are updated immediately.

You can imagine how using cloned dialog items can help put you in direct control of your models. You can use the dialog items as the simulation runs, such as clicking buttons or entering values in

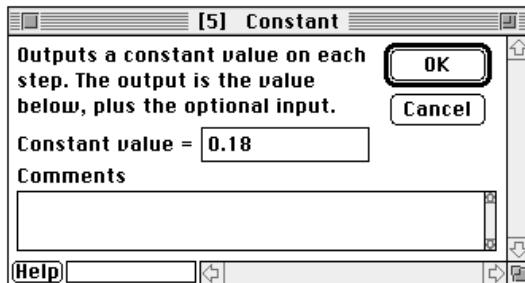
Extend

text entry boxes. For instance, you may have a model with blocks that have controls in them that you can vary as the simulation is running. Instead of having those controls in a dialog, you can put them out on the model window. Or, you might want to have an area of the model that shows many displays in a numeric form instead of the graphical form of the plotter.

Cloning dialog items

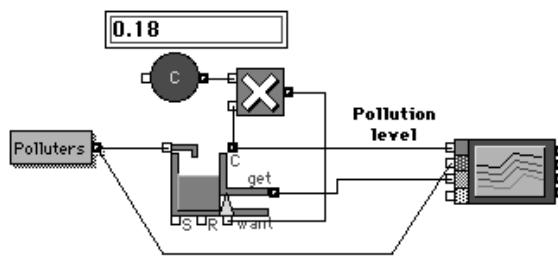
To do this, simply open the desired dialog and select the Clone layer tool, ; note that all the dialog items are outlined. Click and drag the desired item using the Clone layer tool. For multiple items, select by dragging a frame or holding down the Shift key as you select more items. Then close the dialog by clicking the close box or choosing the Close command from the File menu.

For example, assume that you want to be able to change the outflow rate as the “Exploring” model is running. You could leave the Constant block open as you run the model, but it covers a lot of area on the window for just that one number. Instead, you can clone the constant value parameter. To do this, choose the Clone tool, and double-click on the Constant block:



Constant block open (Macintosh)

Click on the dialog item that contains the number “0.18”, and drag it out to your model near the top. Close the dialog (since you don’t need it anymore). The result is:



Item dragged to the model

To add text near the clone, click on the Text tool and add a label to the number:

Outflow:
Label added

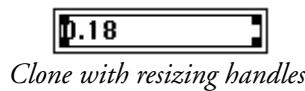
Now run the simulation again. To change the outflow at a point in the simulation, click in the number and type a new value. Extend stops so that you can type a value. Then, click the Resume button in the Status Bar at the bottom of the screen, and the simulation continues with the new value.

Notice that the plotter in this model is configured not to open until the simulation ends. You can choose to have a plotter open during the simulation, at the end of the simulation, or only when double-clicked. These choices are described in “Plotter dialogs” on page E157.

Using cloned items

Once you have a cloned item on your worksheet, you can drag it anywhere you want. Many people prefer to have all the items together at one end of the model, for example. (You will see in the next section how Notebooks are also useful for cloned items.) To move a cloned item, choose the Clone tool from the toolbar, click on the item, and drag it.

You can resize a cloned dialog item to any size. To do this, choose the Clone tool and click once near the center of the cloned dialog item so that the resizing handles appear:



Then click and drag a handle to change the size and shape of the clone.

To remove cloned items from your model, simply select them (using the Clone tool) and press the Delete key or choose Clear from the Edit menu. If you delete the block from which you cloned a dialog item, the cloned item is automatically deleted.

You can find and open the dialog from which an item was cloned by choosing the Clone tool and double-clicking on the item.

Note that the graphs in plotters and data tables (such as the table in the Input Data block) can also be cloned just like buttons and text entry boxes.

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.



Choosing the clone tool and double-clicking the unlinked clone will cause an alert message to appear. After which, 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.

Notebooks

Each model has a *Notebook* which can be used for controlling the model parameters, reporting simulation results, and documenting the model. This feature gives you even more flexibility with cloned dialog items. If you do not want dialog items on your model but still want easy access to them, you can move them into the model's Notebook. The Notebook acts like a normal window; you can resize it and scroll its contents. The Notebook is a good place for your cloned items if you do not want to see them all the time but still want to collect items from many dialogs.

Extend

Use Notebooks for both input and output values. For example, you may want to clone buttons in the Notebook and change them as the simulation progresses. Or, you might want to use the Notebook mostly for looking at the various outputs of your model. You can even clone plots into the Notebook. The Notebook is also handy for combining all of your output into one spot so you can print it or make a copy of it for a report.

You can create and view the Notebook for a model by choosing Open Notebook from the Model menu. For example, open the Notebook for the “Exploring” model. Note that it has a clone of the contents from the Holding Tank and its label, as well as a clone of the plotter. Notebooks are described in much more detail in “Notebooks” on page E160.

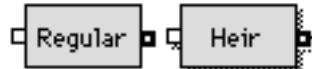
Hierarchy

Up to now, each block that you have seen performs an action that is one part of the model. The action the block performs is represented by the block's icon. Most blocks in Extend work this way. There is a second type of block you create, called a *hierarchical block*, that works differently. A hierarchical block contains other blocks that are connected like they would be in a model. When you open a hierarchical block, you see a group of blocks nested inside of it. These blocks represent a portion of the model, or a *subsystem*. The hierarchical block allows you to nest subsystems for top-down or bottom-up modeling. This chapter gives you general information about hierarchical blocks; “Hierarchy” on page E252 shows you how to build your own hierarchical blocks.

Introduction to hierarchical blocks

You can imagine a hierarchical block as a container that holds other blocks. A hierarchical block can contain simple blocks, other hierarchical blocks, or both. The blocks in a hierarchical block are connected just like other blocks in a model, and hierarchical blocks can have input and output

connectors like regular blocks. A hierarchical block can be set to look slightly different if the user selects *Hierarchical blocks have drop shadows* using the Edit:Preferences command:



Standard and hierarchical blocks

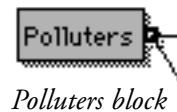
Extend

There are many uses for hierarchical blocks. Although you do not have to use them as you build models, you will find them very handy.

- If you have a complex model with thousands of blocks, you can use hierarchical blocks to simplify the model by grouping areas of the model. These grouped blocks can then be reused in other models without having to reproduce all of the connections.
- Instead of showing all the detail, you can present your model as a few simple steps. To reveal the subsystems within a step, just double-click on the hierarchical block.
- As you develop a new model, you can go from the simplest assumptions to more complex ones by creating more and more levels of hierarchy. This helps you structure your thinking and makes your models easier to follow as they become more complex.

When you open a hierarchical block by double-clicking on it, you see the blocks that make up the higher block. They are laid out like a model, with connections, labels, and so on. You may also see the dialog items that control the blocks in the hierarchical block.

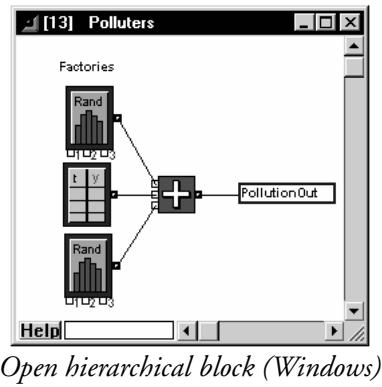
In the “Exploring” model, the polluters block looks like:



Polluters block

Viewing a hierarchical block

When you double-click on the polluters block you see:



Extend

Hierarchical blocks' connections with the rest of the model are shown as connection text boxes (named connections with borders around them). In this example, "PollutionOut" is the block's connector, as you can tell from the text at the right of the hierarchical window. (The number in the title bar is the global number for the hierarchical block. It is used for locating a block by number as discussed in "Find Block" on page A21.)

If you want to change the settings in one of the blocks in the hierarchical block, simply open the hierarchical block and double-click on the desired icon. For example, you might want to change one of the dialogs for the polluters. You can make any changes in these blocks just as you would in blocks on the model worksheet.

You can also clone from a dialog within a hierarchical block to the hierarchical block's window. To do this, open the hierarchical block's window, open the dialog (such as the Input Random Number block), choose the Clone tool, click on the desired item (for example, the value of the mean), and drag it out to the hierarchical block's window. You can now change the mean without having to open the Input Random Number block's dialog.

"Hierarchy" on page E252 describes more about hierarchical blocks, including customizing hierarchical icons. For now, you can see how they can help you in your work by reducing clutter and grouping related blocks into single blocks.

Animation

When you look at the output of a plotter after you run a model, you can see how the plotted values change over time. You can also leave the plotter open while the simulation is running to see the values as the simulation progresses. However, you may want additional feedback as your model runs. For this, you would rely on Extend's animation capabilities. You can animate a model using built-in animation, your own custom animation, or an add-on animation package such as Wolverine Software's Proof Animation, which can be purchased from Imagine That, Inc. Please

see “Proof Animation (Windows only)” on page E278 for more details on adding Proof animation to your models.

To see built-in or custom animation, choose Show Animation from the Run menu so that the command is checked, then run the simulation. The reason animation is not shown all the time is that it slows the simulation. You may want to leave animation on while you are changing, debugging, or presenting your model and then turn it off when you are running the model for quick results. Selecting “Bitmap blocks” in the Preferences command speeds up animation.

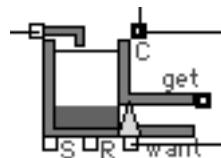
Built-in animation

Many of the blocks in the Generic and Discrete Event libraries have animation built into them. The animation in these blocks is described in “Appendix D: Generic Library Blocks” on page A45 and “Appendix E: Discrete Event Library Blocks” on page A55. Also, the help text for these blocks tells whether they are animated and, if so, which aspects are animated.

Animation on the block’s icon

A good example of an animated block icon is the Holding Tank block in the “Exploring” model. You have already seen this block in Chapter 2: Building a Model without animation. So that you can see the animation without interference from the plotter, the plotter in this model is configured to remain closed until the end of the simulation. (Settings in the plotter’s dialog specify when the plotter opens. Using dialogs to change how plotters function is fully described in “Plotter dialogs” on page E157.)

To see the Holding Tank block in action, choose the Show Animation command so that it is checked, then run the model. You will see the level go up and down as it fills and empties. On the first simulation run, the level will go from 0 to an estimate of the maximum value; later runs will use an average of the preceding run’s maximum as the level’s maximum value.



Holding Tank block with animation

If you are using animation and the animation goes by too quickly, you can slow down your model with the Slower button in the status bar. The Slower button only slows down animated models. Click on the Slower or Faster buttons to change the speed of the simulation.

Animation between block icons

As discussed in “Continuous and discrete event modeling defined” on page E98, discrete event models track the flow of individual items through a process. The blocks in the Discrete Event library are capable of representing the flow of these items with animation pictures or icons that travel along the connection lines between the blocks.

When Show Animation from the Run menu is checked, animation icons representing the individual items in the model will be shown as the items are passed from block to block in the model. Initially, all items in the model will be represented by the default animation picture (a green circle). However, you can choose to use any of the animation pictures included with Extend, existing clip art, or pictures that you have created in an external drawing package. See “Customizing animation pictures” on page E267 for a discussion on how to define and customize animation pictures in discrete event models.

Extend

Animation for debugging block code

The Show Block Messages command from the Run menu is a type of animation used by programmers when they debug blocks that they build. It is discussed further in “Chapter 4: ModL Programming Techniques” on page P173.

Custom animation

You can add custom animation to your models by using blocks from the Animation library or by using the animation functions in blocks that you create yourself.

Animation Library blocks

The Animate Value, Animate Item, and Animate Attribute blocks, all from the Animation library, show customized animation in response to model conditions. Based on choices in their dialogs, these blocks can display the value that is input, show a level that changes between specified minimum and maximums, display text or animate a shape when the input value exceeds a critical value, or change an item’s animation icon based on an attribute value. For more information, see “Animation – customizing” on page E197.

Animation functions

- If you build your own blocks, Extend offers a suite of functions which allow you to design how your block will animate. For example, the Planet Dance model, in the Custom Block Models folder in the Science and Engineering folder, shows three planets orbiting in space. Their orbits are determined by numbers entered in the block’s dialogs. See “Animation” on page P123 for a description of the animation functions. Also, see “Animation” on page P184 for an example of how to add animation to a block.

Sensitivity analysis

Sensitivity analysis allows you to conduct experiments and investigate the effects of changes in a structured, controlled manner. You do this by running simulations many times, changing the value of a variable or numeric parameter each time the simulation is run. When you repeatedly run a simulation that has parameters that vary with each run, you can see the range of the results and look for trends or anomalies. For example, if you run the same simulation a hundred times, you can look for extremes in results and look for averages of critical values.

Extend’s sensitivity analysis feature gives you the ability to explicitly specify individual parameters to change and provides several methods for changing them. For instance, in the “Bank Line”

model from Chapter 1: Running a Model, you might want to vary the time it takes for a teller to serve a customer and see if and how that affects the length of time the customers must wait. Another example would be in the attributes variation of the “Car Wash” model on page E114, varying the length of time it takes to wash a car and looking at the resulting throughput.

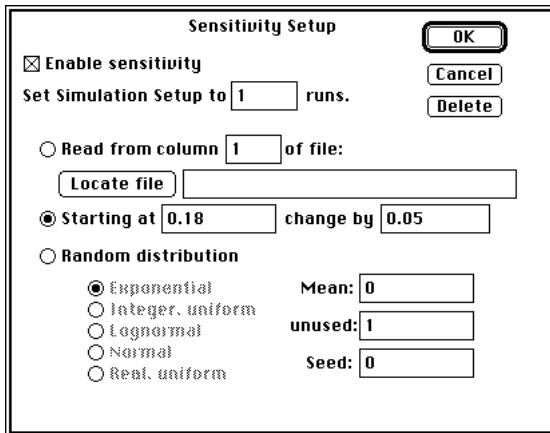
Sensitivity analysis differs from using random numbers as inputs in your model. When you run a model and use a random input, such as the Input Random Number block from the Generic library, a random number will be input at each step or event. For example, the time it takes to wait on a customer could vary randomly during the simulation run. When you use sensitivity analysis, the randomness occurs from run to run, not between steps in the run. For instance, the customer service time could increase in succeeding simulation runs. This is particularly useful when you want to investigate how a change in one parameter impacts the pattern of behavior for the entire model.

To use sensitivity analysis in a model:

- Choose a parameter value to investigate
- Choose how the parameter will change
- Select the number of runs for the analysis
- Run the simulation with Use Sensitivity Analysis selected in the Run menu

For example, the “Exploring” model already has the constant outflow rate (represented by the Constant block) set to increase by 0.05 for each run of the simulation. To see this, double-click on the Constant block to access its dialog. Then hold down the Command (Macintosh: ⌘) or Control (Windows: CTRL) key as you click once on the constant value (the parameter “0.18”)

or, optionally, click once in the parameter field and use the Sensitize Parameter command in the Edit menu. This opens the Sensitivity Setup dialog:



Extend

Sensitivity Setup dialog

In the dialog, change the “Set Simulation Setup to...” to 4 runs, but do not change any other settings. Then run the simulation.

As mentioned earlier in this chapter, the plotter is set to open at the end of each simulation. When the fourth simulation run has finished, you can flip the pages of the plotter to see the effect varying the outflow rate has on the results. To see the changes in order, flip from page 4 to page 1.

Sensitivity analysis is a very powerful feature that you can use in your models. For complete details, see the section “Sensitivity analysis” on page E224.

Optimization

Optimization is a very powerful feature that can determine ideal values for parameters in your model in an automated way. It accomplishes this by running the model many times with different parameters to search the solution space until it is satisfied that it has found an acceptable solution. The optimizer supplied with Extend uses an evolutionary algorithm to reduce the number of times it has to run the model before a solution is found.

Most models have parameters, such as an activity time or a flow rate, that are specified by the modeler and shouldn't vary. Some parameters, possibly the number of machines doing a task, could vary and change the efficiency of the model. To find the most cost effective number of machines, you could manually run the model and change the number of machines for each run until you were happy with the results. This could take several runs, or hundreds of runs...but you could use optimization to find the best answer automatically.

For example, using more machines can lower a backlog of unfinished work. But those machines cost money to use, and maybe the cost of backlog isn't as much as the cost of using more

machines. Knowing these costs, we can create a simple cost equation and run the model, plugging in different numbers for each run until the cost is minimized. Optimization automates this task, freeing the modeler from manually running the model countless times. In addition, cost equations can also have constraints; values of some variables sometimes need to be limited by the values of others, and this can be handled via optimization.

Extend

Adding optimization to your model

For details showing how to use optimization in your models, see the section “Optimization Tutorials” on page E231.

Equation editor

Extend’s libraries are toolkits of blocks that allow you to build models quickly. However, this may not be sufficient for all your needs. For example, there may not be a block that provides the function or equation that you want. Or, you may want to combine the function of several blocks into one. Some possible solutions are:

- Select several blocks and make them into a hierarchical block, as discussed in this chapter and in “Chapter 7: More Modeling Techniques” on page E193.
- Add features to Extend’s blocks by modifying the structure of a block (its dialog and code) as discussed in Chapters 9 through 12.
- Use the Equation block (Generic library, Math) or the DE Equation block (Discrete Event library, Attributes) to directly combine functions.

The Equation and DE Equation blocks allow you to type an equation directly into that block’s dialog. The equation must be of the form *output = formula*; (note that the semicolon at the end is required). The equation is automatically compiled when you click OK.

The Equation and DE Equation blocks are similar in many ways 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 value connectors that you identify by name, and the blocks output the results to the value connectors of other blocks.

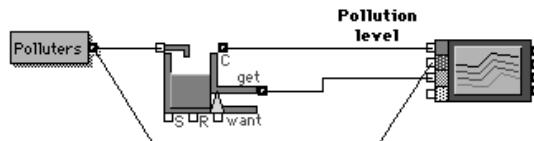
Both equation blocks have five value inputs and one value output. The DE Equation also has one item input and one item output. This block has the additional ability to read in attribute values from items that pass through the block, then use those values within the equation. Within the dialog of both equation blocks, you can type in an equation that uses any of Extend’s built-in functions and operators in combination with the values from the connectors. Your equation can be as simple as performing mathematical operations on an input value, or it could be as complex as a full programming segment.

To access the values from the inputs and to output the results of the equation, you can enter names that have more relevance to your model. You do not have to use all the names you assign to the connectors in your equation. However, if an input is connected, you have to use its name in your equation (Extend assumes that if you have an input connected, you want to use it in your equation). When you run the simulation, Extend will warn you if you have connected inputs that are not used in the equation. Extend will also warn you if your equation uses a connector that is not named or is not connected.

For example, you could use the Equation block to replace the Multiply and Constant blocks in the Exploring model:

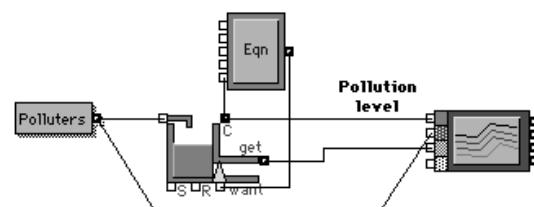
Extend

- ▶ Delete those two blocks from the model. Notice that when you do this, the cloned dialog item from the Constant block is automatically deleted. Also remove the clone's label. The result is:



Two blocks deleted

- ▶ Add an Equation block from the Math submenu of the Generic library. Connect from the *C* output of the Holding Tank block to the bottom input of the Equation block, and connect the Equation block's output back to the *want* connector of the Holding Tank:

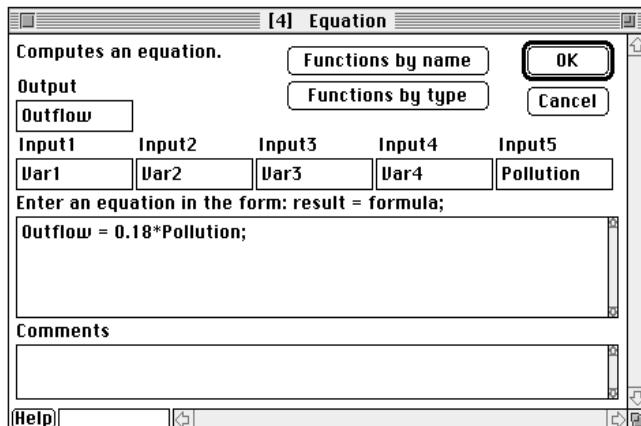


Equation block added

- ▶ Open the Equation block's dialog. Change the Output connector's name from "Result" to "Outflow", and change the bottom input connector's name from "Var5" to "Pollution".
- ▶ Type the following equation in the text entry box:

```
Outflow = 0.18*Pollution;
```

⇒ The dialog looks like:



Equation dialog for outflow (Macintosh)

⇒ Close the Equation block to compile the equation. When you run the simulation, you get the same results as you did before. The Equation block substituted for the two other blocks.

You can see how easy it is to change the equation. For example, if you want the outflow to be 5% of the pollution plus 100, you would not have to add an Add block and a Constant block with 100. Instead, you would change the equation to:

```
Outflow = 0.05*Pollution + 100;
```

“Equation editor” on page E222 has more information about the Equation block, including its limitations and rules of usage.

Control blocks

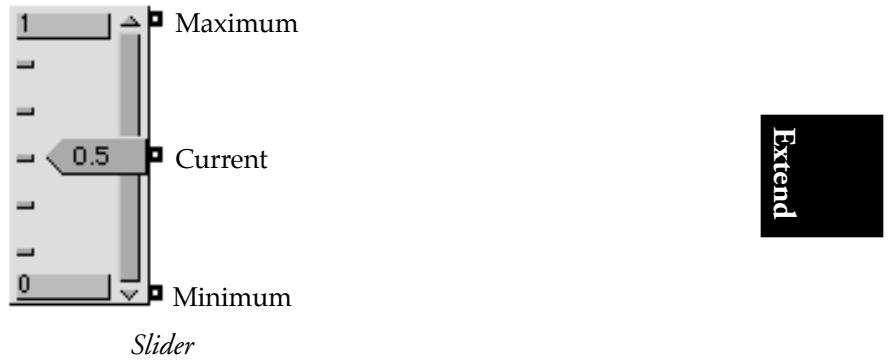
Extend has three special blocks that you choose with the Controls command from the Model menu. These blocks are used to add interactive control directly to your model. They are used to control other blocks and show values directly as the simulation runs.

The controls are the *Slider*, *Switch*, and *Meter*. The Slider and the Switch are used to set values in your models. The Meter is used to see values as the model is running. The Meter also has a dialog that you can access by double-clicking on it.

Slider

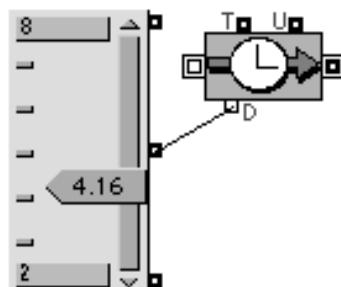
A Slider resembles a slider on a stereo. You can drag the knob to output a number. You set the maximum and minimum values by selecting the numbers at the top and bottom of the Slider and typing in the desired value. Since the Slider calculates its output based on the specified minimum and maximum, it is important that you enter the maximum value first, and that its value be greater than the value entered for the minimum. If you enter a value at the bottom of the Slider

which is higher than the maximum, or if you set the value at the top of the Slider to be less than the minimum, Extend will warn you.



You change the output from the Slider by dragging the level indicator up and down. As you change the indicator, the value shows on it and is output through the middle connector. You can also output the values set for the maximum and minimum by connecting to the output connectors on the right. The top connector tells the maximum and the bottom connector tells the minimum.

The Slider is useful for when you want to output a number in a range but the number does not need to be exact. For example, if you have a delay 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, you might put in a Slider with 8 as the maximum and 2 as a minimum. Then, as the simulation is running, simply drag the Slider down and up to indicate fast and slow:



Example of using a Slider

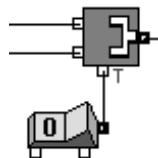
Switch

The Switch control has two inputs and one output and looks like a standard light switch:



The Switch outputs either a 0 (zero) or a 1 (one) depending on which side is down. When you click on the side that is not down, the Switch changes to the other value and makes a small clicking sound. The number you see on the switch is the number that is output.

The Switch control is very valuable in controlling blocks that have true-false inputs. Because Extend sees 0 as false and 1 as true, you can think of the Switch as a true-false switch. For example, you might attach a Switch to the *T* connector of a Select Input block:



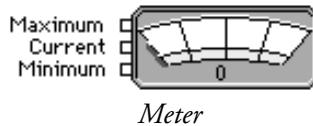
Example of using a Switch

The input connectors are used to change the state of the Switch by setting their side of the Switch to true from within your model. For example, you might want to set the Switch to off when a model starts, then change it to on after some period of time. The next time you run the model, the Switch would be automatically set to off again when the model begins.

When either input gets a true value (0.5 or greater), it sets its side of the Switch to true (unless it is already true, in which case no change is made). If the other input later gets a true value, the Switch is set to true for that side. That is, when either side gets a true value after having a false value, the Switch will shift to that side.

Meter

You can use a Meter to show values that vary between a known maximum and minimum. Set the maximum and minimum values through the Meter's dialog or by connecting other blocks (such as Constant blocks) to the top and bottom connectors. A Meter looks like:



For example, if you have a system where part of it always varies between known minimum and maximums, and you want to see the values but you do not need to save them in a plotter, use a Meter.

Other authoring features

You can probably see how the features described above would come in handy if you build models for others to use. For instance, the Notebook, coupled with the ability to clone dialog items, serves as a central location for controlling the model and for getting simulation results. Control blocks allow users to directly and easily interact with the model and see the results of that interaction. And hierarchy, again coupled with cloned dialog items, provides a familiar “front end” to the model, shielding unsophisticated users from the detail of complicated models.

Extend

In addition to the features already discussed, Extend provides several more methods you can use to help others interact with the models you build. You can prompt the user for inputs, warn when monitored values change, and even use other languages, such as Visual Basic, to add special features to your models.

Locking the model

The Lock Model command in the Model menu prevents any modification of a model other than changing dialog values. This command also hides most of the tools in the tool bar so that the user cannot add or change connection lines, drawing elements, and so on. Locking models is especially useful if you are giving the model to others who are unfamiliar with Extend’s features and may accidentally move or delete blocks. To unlock the model, simply choose the Lock Model command again.

Extend RunTime and Player versions

The model locking feature discussed above works with the full version of Extend. However, your intended user may not have a need for the full version. For instance, you might want to distribute a model as part of the response to an RFP (request for proposal). Or you may want to provide models to your company’s sales staff so they can show customers the outcomes of various equipment options. The Extend RunTime and Player versions provide low-cost, convenient methods for distributing the models you build (and your libraries, if you program).

Both the RunTime and Player versions allow users who do not have the full version of Extend to:

- Run simulations built with the full version
- Change parameter values in a model and obtain graphical and tabular output
- See animation in the model, if the model includes animating blocks
- Import and/or export files and interface through the serial ports, provided the model has these features.

- Save changes to the model (**RunTime version only**)
- Print models, dialogs, etc. (**RunTime version only**)

The limitations of the RunTime and Player versions are:

- You cannot build your own models. Models must be developed in the full version of Extend. Blocks (the components of the model) cannot be added or removed from the RunTime/Player models nor can connections between the blocks be altered.
- You cannot build your own blocks or access the structure and dialog editor of an existing block. Therefore, you cannot modify icons, change a block's programmed behavior, add or delete dialog items, etc.
- The RunTime and Player applications can only read RunTime libraries. Extend libraries must be converted to RunTime format by the RunTime/Player model developer using the full version of Extend (see "Convert to RunTime Library" on page A16). This conversion must be done on the platform the RunTime or Player users will be using. For example, the Macintosh RunTime version can only read libraries that have been converted to RunTime in the Macintosh version.

For ordering information on the RunTime version, contact Imagine That, Inc. or your local distributor. The Player version is available for free on the Imagine That, Inc. website (<http://www.imaginethatinc.com>).

Sending messages to the user

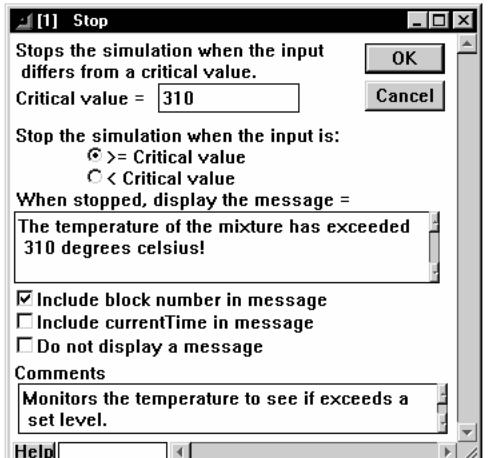
Some Extend blocks provide a convenient method for monitoring conditions in the model and reporting changes to the user. In addition, if you build your own blocks you can add customized alerts and prompts to display results and prompt for input data.

Stop block

The Stop block, from the Input/Output submenu of the Generic library, is used to stop the simulation and alert the user if a monitored parameter reaches a critical value. The block notifies the user if the parameter goes either above or below the critical value, which is set in the dialog. You can have the block display any warning message you want, up to 255 characters. You can also set

Extend

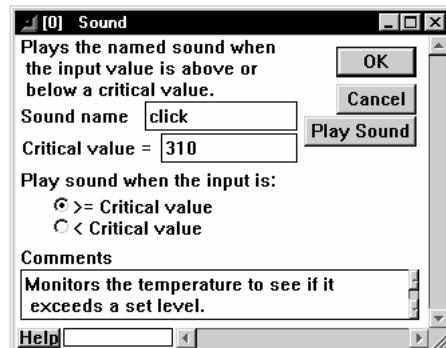
the block to display the number of the block containing the watched parameter and the exact time that the critical value was reached. The block's dialog looks like:



Dialog of Stop block showing message (Windows)

Sound block

Like the Stop block discussed above, the Sound block (also from the Input/Output submenu of the Generic library) is used to monitor a model parameter. However, rather than stopping the simulation and displaying a message, the Sound block plays a sound when the monitored value is above or below the critical value. You enter the critical value and sound name in the dialog of the block:



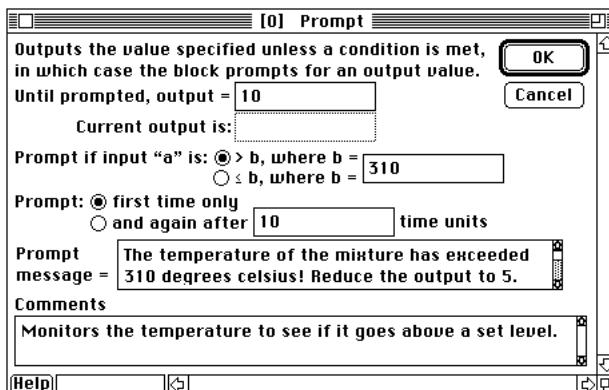
Dialog of Sound block with Click sound chosen (Windows)

- *Macintosh:* You can enter the name for a sound from the System Folder, or for any sound in Extend's "Extensions" folder. To see the System sound names, access the Sound device in the Control Panel under the Apple menu. To hear the sound that is entered in the Sound block's dialog, click the Play Sound button.

- Windows: You can enter the name for a sound from the Window's system, or for any sound in Extend's "Extensions" directory. Sound files can only be in .wav format. To see the system sound names, access Window's Sound Control Panel. To use a sound, enter the sound name in the Sound block's dialog. To hear that sound, click the Play Sound button.

Prompt block

Like the Stop and Sound blocks, the Prompt block (also from the Input/Output submenu of the Generic library) alerts the user if a monitored parameter reaches a critical value. However, the Prompt block outputs a value during the simulation and is mainly used to prompt the user to change that value once the message is given. In addition, the critical value can be set either in the dialog or dynamically based on conditions in the model. Its dialog is:



Dialog of Prompt block showing message and new output value (Macintosh)

You would 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 you click the Cancel button in the Prompt's dialog, the simulation is stopped.

Adding help to the model

Adding instructions or explanation to the user of a model is easily done by creating a new empty hierarchical block and then adding explanatory text to explain what is happening, or why a particular modeling approach was used.

Additional features if you program

If you program, Extend provides even more capability for delivering messages and interacting with users:

- The list of functions in the section titled "Alerts and prompts" on page P121 includes functions for displaying a message, prompting the user to input a value, making a sound, or speaking a message if the Speech Manager is present (*Macintosh only*). These functions are also useful for debugging Extend's ModL code, although using the Source Code Debugger (see the Extend

Programmer's Reference "Chapter 5: Using the ModL Source Code Debugger" on page P245) is the preferred approach.

- DLLs ( Windows: dynamic-link libraries) and XCMDS ( Macintosh: external commands) are especially handy where you want to add a feature or functionality that Extend's language (ModL) does not support. For example, you would use an XCMD or DLL 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. XCMDS and DLLs are segments of code written in any language, such as Visual Basic or C++. Extend's XCMD and DLL functions allow you to call these code segment resources from within a block's ModL code and perform operations. XCMDS are discussed in "XCMDS and XFCNs" on page P237 and DLLs are discussed in "DLLs" on page P238.
- As discussed in Chapter 4: ModL Programming Techniques, you can change what is shown in a dialog depending on what occurs in the model or what is done 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.

Extend

Extend

Chapter 4: Fundamental Continuous and Discrete Event Modeling

*In which you will learn how to use these
two extensive modeling techniques*

*“Things don't turn up in this world
until somebody turns them up.”*
— James A. Garfield

The first three chapters introduced you to the Generic and Discrete Event libraries. This chapter goes into much more detail about the blocks in those libraries and how to use them. The first section of this chapter discusses how to choose which library to use; the second section is devoted to the Generic library, and the third section is devoted to the Discrete Event library. These last two sections describe the blocks in the libraries and show you how to build an increasingly complex model using those blocks.

Extend

Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

For a description of the Generic and Discrete Event libraries, see “Appendix D: Generic Library Blocks” on page A45 and “Appendix E: Discrete Event Library Blocks” on page A55.

Choosing the Generic or Discrete Event library

The most common libraries to use are the Generic and Discrete Event libraries. Before you begin building your own models, it is important to decide which library you should use. As you saw in Chapter 1: Running a Model, models built with Discrete Event blocks can also contain Generic blocks, such as the Help block in the “Bank Line” model. However, *any* model that contains Discrete Event blocks automatically becomes a discrete event model. If you want your model to show only continuous flows, you would use the Generic library; you would not use any blocks from the Discrete Event library.

If you are familiar with simulation, you will already know whether to build your models using the Generic or the Discrete Event library. You can proceed to “Generic library” on page E101 or “Discrete Event library” on page E112.

If you are not familiar with simulation, you are probably wondering which library you should use. The answer is that it depends on the type of model you want to build. Before you proceed, you should become familiar with the terms *continuous* and *discrete event* modeling.

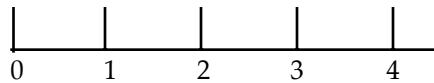
Continuous and discrete event modeling defined

A system is the object of study or interest. When you simulate, you model the performance of a system over time. Models of systems are classified as either *discrete event* or *continuous*, although in the real world there is no such distinction and it is possible to model some real-world systems either discretely or continuously.

Before you build a model, you need to decide whether to model the system as discrete event or continuous. It is important 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 accuracy of the information you want to gather.

Continuous modeling (sometimes known as process modeling) is used to describe a smooth flow of homogenous values; discrete event models track individual and unique entities, known as items. In both types of simulations, what is of concern is changes in the state of the model:

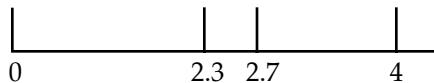
- Continuous simulations are analogous to a constant stream of fluid passing through a pipe. The volume may increase or decrease, but the flow is continuous. In continuous models, values change based directly on changes in time, and time changes in equal increments. These values reflect the state of 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 changes in state (such as position or velocity) change continuously with respect to time. The time line for a continuous model looks like:



Time line for continuous model

Extend

- Using the pipe analogy for discrete event simulations, the pipe could be empty or have any number of separate buckets of water traveling through it. Rather than a continuous flow, buckets of water would come out of the pipe at random intervals. In discrete event models, discrete items change state as events occur in the simulation. The state of the model changes only when those events occur: the mere passing of time has no direct effect. For a discrete event 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). The time line for a discrete event model could look like:



Time line for discrete event simulation

Note In engineering, it is common to use the term “discrete” to describe a system with periodic or constant time steps. Discrete, when it refers to time steps, indicates a continuous model; it does not have the same meaning as “discrete event”. Continuous models in Extend are stepped using constant time intervals; discrete event models are not.

Table of continuous and discrete event differences

Although not definitive, the following table will help you determine how to model your system:

Extend

Factor	Continuous Modeling	Discrete Event Modeling
What is modeled	Flows (“stuff”)	Discrete items (“things”)
Characteristics	Random number “simulates” characteristics of flows and must be repeated for each query or junction.	Characteristics are assigned to items by attributes which can then be tracked throughout the model.
Time steps	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.
Ordering	Flows are in FIFO order.	Items can flow in FIFO, LIFO, Priority, or customized order.
Routing	Flows need to be explicitly routed by being turned off at one branch and turned on at the other (flows can go to multiple places at the same time).	Items are automatically routed to the first available branch (items can only be in one place at a time).
Statistical detail	Only general statistics about the system: amount, efficiency, transit time.	In addition to general statistics, each item can be individually tracked: count, utilization, cycle time.
Common Uses	Scientific (biology, chemistry, physics), Engineering (electronics, control systems), Bulk processes, Systems thinking, Economics, Systems dynamics	Manufacturing, Service industries, Business processes, Strategic thinking, Networks (computer, telephone), Systems engineering

Some systems, especially when a portion of the flow has a delay or wait time, can be modeled as either discrete event or continuous. 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 a system than continuous models. Continuous models, on the other hand, run faster than discrete event models.

The Generic library lets you solve almost any continuous modeling problem in Extend with just a few basic blocks. The Generic library contains blocks that perform such basic functions as math, decision handling and input/output. These blocks lend themselves to modeling in the areas of finance, economics, electronics, demographics, biology, physiology, chemistry, physical systems, and so on.

The Discrete Event library has all the basic tools for creating models that use queues, servers, item-specific attributes, and priorities. Discrete event blocks are useful for simulating computer and communications networks, business and service industry processes, queuing theory, paper production, and so on.

Remember that you may combine blocks from different libraries within the same model. It is quite likely that you will use blocks from the Generic library when you create a non-continuous model with the Discrete Event blocks. However, the blocks in the Discrete Event library can only be used in event-driven (non-continuous) models. If you use any discrete event blocks in your model, the timing will change and the model will be discrete, not continuous. You should only use discrete event blocks if you want to have a discrete event model.

Generic library

Extend

The blocks in the Generic library are useful for both quickly building continuous models, and for performing specialized tasks when connected to value connectors in discrete event models. Using the pre-programmed blocks from the Generic library lets you avoid the need to type in equations or program in ModL (the underlying language of Extend blocks) for most tasks. These blocks have error-checking built in and often allow you to perform complex modeling tasks with the click of a button. However, if your models end up being cluttered with blocks, you may want to combine the functions of many blocks into a single block:

- Combine a group of blocks into a hierarchical block
- Use the Equation block from the Generic library
- Program your own blocks that combine the functionality of many blocks

Using the Generic library

All of the blocks in the Generic library have dialogs associated with them. You use the dialog to enter values and to look at the current values of the inputs. If you have connected an input and also enter a value for that input in the dialog, the connector's value always overrides the value in the dialog. The dialog for each block has a help button in the lower left corner that gives an explanation of the block, its connectors, its parameters, and its options.

The icons for the blocks are color-coded. Inputs have green borders, outputs have red borders, and computational blocks have blue borders.

Blocks by type in the Generic library

The blocks are grouped here by their type: decisions, holding, inputs/outputs, math, and statistics (see “Block types” on page P56 for a discussion on block types). Each line in this brief list tells the block’s name, use, and examples of how you might use it in a model. A more detailed alphabetical list is shown in “Appendix D: Generic Library Blocks” on page A45.

Extend

Arrays	Use	Example
Global Array	Allows access to global arrays of data. Array values can be set, reset, or modified.	Holding sales, shipping, and size data
Global Array Manager	Creates, resizes, deletes and views Extend's global arrays.	
Decision blocks	Use	Example
Decision	Makes a decision based on the inputs and internal logic you define.	Contingency planning, pass/fail tests, routing
Select Input	Selects its output to be either of two inputs according to the value of the <i>T</i> connector.	Physical characteristics, alternating sources of information
Select Input (5)	Selects its output to be one of five inputs according to the value of the <i>T</i> connector. The top input is selected if the value is 1, and the bottom if it is 5.	
Select Output	Passes the input data to one of two outputs according to the value of the <i>T</i> connector.	Routes, categories, binning
Select Output (5)	Passes the input data to one of five outputs according to the value of the <i>T</i> connector. The top output is selected if the value is 1, and the bottom if it is 5.	
Holding blocks	Use	Example
Accumulate	Adds up the numbers at its input from each step and shows the total contents at the <i>C</i> connector.	Profits in retained earnings, billable hours, customers served
Holding Tank	Accumulates the total of the input values and outputs a requested amount, if it is available.	Reservoir, waiting line, bank account, warehouse
Holding Tank (Indexed)	Represents up to 100 Holding Tank blocks. Accumulates the total of the input values and outputs a requested amount, if it is available.	Reservoirs, waiting lines, bank accounts, warehouses
Retain	Outputs and retains the value of its input under certain conditions.	Bulletin board with reminder notes

Holding blocks	Use	Example
Wait Time	Holds its inputs for a certain amount of simulation time before passing them to the output.	Delays, build time, lead time
I/O blocks	Use	Example
Constant	Generates a constant value at each step.	Steady flow of fluid or values, offset
File Input	Reads data from a text file.	Control parameters for many blocks in a model or many models
File Output	Writes data to a text file.	Export to spreadsheet or database
Input Data	Generates a curve of data from a table of values, based on the time.	Lookup tables for time-based sales, production, traffic
Input Function	Generates a function over time.	Functions for trigonometry, math, pulse
Input Random Number	Generates random integers or real numbers.	Inputs that need a statistical distribution
Prompt	Prompts for an output value if the input is above or below a critical value.	Pause the simulation and request information
ReadOut	Displays the value of the input connector at each simulation step.	
Sound	Plays a sound when the input is above or below a threshold.	Warnings, announcements
Stop	Stops the simulation when its input goes above or below a critical value that is set in the dialog.	Warnings, failure conditions
System Variable	Allows you to regulate some aspects of a model based on the status of the simulation.	Current time to measure when events occur
TimeOut	Outputs a “1” after a specified delay, then repeats.	Alarm clock, timed gates

Extend

Extend

Math blocks	Use	Example
Add	Adds the three inputs on the left of the block.	
Conversion Function	Modifies the input by a mathematical function.	Money into parts, sunlight into energy, transfer function
Conversion Table	Contains a table of values that calculates an output value for a given input value.	Lookup table for price/unit or customers/server, steam pressure
Divide	Divides the top input by the bottom one.	
Equation	Allows you to enter equations	Equation customizing
Exponent	Raises the bottom input to the power of the top input.	
Financials	Calculates the rate, period, payment, present value, or future value based on the other four values.	Loan calculations
Integrate	Integrates the input over time.	Differential equations
Limits	Allows you to limit the output to a given maximum and minimum value.	Age range, salary draw, temperature tolerances
Logical AND	Perform logical AND operation on the inputs.	Do something if both A and B happened.
Logical NOT	Perform logical NOT operation on the inputs.	Do something if A did not happen.
Logical OR	Perform logical OR operation on the inputs.	Do something if either A or B happened.
Max & Min	Determines the maximum and minimum value of the input connectors.	Choosing highest priority, routing
Multiply	Multiplies one input by the other.	Multiplicative operations
Subtract	Subtracts the bottom input from the top input.	Subtractive operations
Threshold	Has an output of 0 until the input exceeds the threshold value; when the input exceeds the threshold, the output is the difference between the input and the threshold.	Metal fatigue, loan covenants
Time Unit	Converts values from one time unit to another.	Hours converted to minutes

Statistics blocks	Use	Example
Mean & Variance	Calculates the mean, variance, and standard deviation of the input.	Deviation calculations

Generic library example

The Generic library lets you set up models that involve the basics of continuous modeling. You can mathematically combine the output of one block with the output of other blocks. You can also feed the output of blocks into the inputs of other blocks.

Extend

This example shows the steps involved in building a model of predator-prey interaction. The model starts out quite simple and becomes more complex as simple assumptions are changed into more complex time-based activities. Each step of the model is shown in a separate file in the Predator/Prey folder within the Tutorials folder.

The final model will show a small ecosystem composed of hare and lynx. Each population has a direct effect on the other: lynx feed on hare, the hare population declines; the diminishing food supply causes a decrease in the number of lynx, the hare population increases, and so forth. The model particulars and assumptions are:

- The blocks come from the Generic and Plotter libraries
- The 100 hectare ecosystem initially contains 6000 hare and 125 lynx
- On the average, hare produce 1.25 offspring each and lynx produce 0.25 offspring each, per year
- Hare always have sufficient food supply; their only cause of death is being eaten by a lynx
- The lynx hunting range is 1 hectare and their only food source is the hare
- The number of hare killed depends on their density in the ecosystem and the number of lynx who hunt them
- The lynx mortality rate depends on how many hare they consume in a year
- There are no outside factors (such as the time of year) which affect the rate at which the populations change
- The final model runs for 24 years and calculations are ten per year (dt is .1)

Starting with a simple model

It is common when building a model to start with a simple model and add details until you have captured the essence of the system. In general, you should build the simplest model that will answer your questions.

The first step in building the “Predator/Prey” model is to model the dynamics of the hare population. To do this, use a Constant block to represent the hare birth rate (as stated above, this is 1.25) and a Holding Tank block to represent the hare population. Connect from the “C” (contents) output of the Holding Tank block back to the top input of the Multiply block; this feeds the current population amount to the Multiply block. In the Holding Tank block, enter 6000 as the initial number of hares and choose the “integrated (no delay)” choice. (Because the model will be showing interdependencies, this is the correct choice. To understand why, see “Integration vs. Summation in the Holding Tank block” on page E277.)

When you multiply the current hare population by the birth rate, you get the number of hares born each year. As the simulation runs and calculates values from year to year, the number of new hares is added to the original population through the input at the left of the Holding Tank block. This revised total is then fed back to the Multiply block. The model is:

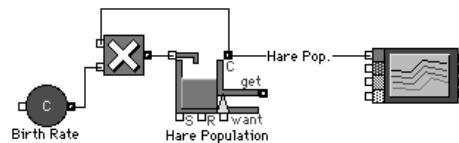


Figure 1: Model of hare population

As seen in the plot below, the population has grown to 7,035,624 hares after only 6 years. Since there is no predator, the population growth is unconstrained.

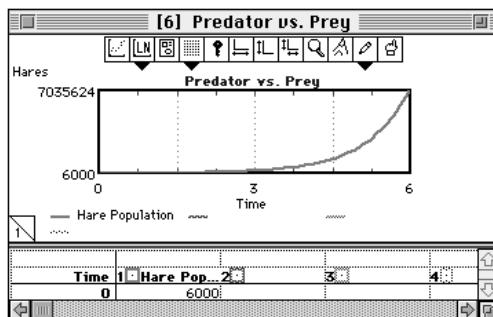
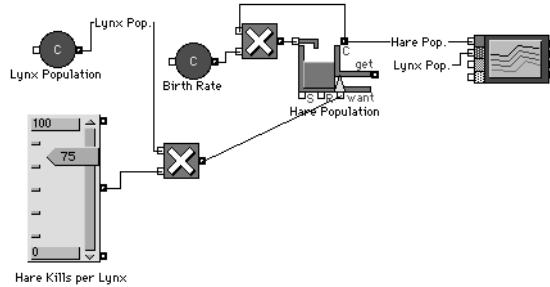


Figure 2: Unconstrained growth of hares (Macintosh)

Adding the lynx predator

In a real ecosystem other factors, such as food supply and predators, would limit the hare population. The next step is to build a model showing a predator: in this case, the lynx. This model uses a Constant block to represent the lynx population and a Slider control to set a constant number of hare killed per lynx. A Multiply block is the obvious choice to multiply the number of hares killed per lynx by the current lynx population; its output is the total number of hares killed each year.

The “want” output on the Holding Tank block removes that amount from the hare population at each step of the simulation. The modified model now looks like:



Extend

Figure 3: Hare population affected by lynx

This is a good place to point out that the Holding Tank block is set to not allow its contents to become negative. This means that the population of hares cannot be reduced below the available amount, no matter what is requested through the “want” connector. For instance, assume the number of hares killed per lynx is 75 and the lynx population is 125. With those settings, 9375 hares would be requested at the “want” connector on the Holding Tank block at the first step. However, since the initial hare population is 6000, only 6000 hares would actually be removed.

In the model, the Slider control allows you to interactively experiment with the number of hares killed per lynx and see the effect on the hare population. For example, try running the simulation with the Slider set to 50. Then run it again with the setting at 60, and finally 53. (Of course, instead of the Slider you could use a Constant block to represent the number of kills per lynx and vary the constant parameter using sensitivity analysis, but the Slider is more fun.) With the Slider set at 60, the model shows that the hare population declines rapidly as seen below:

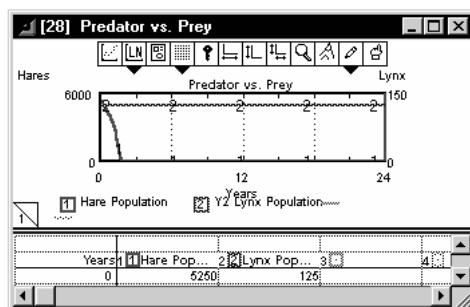


Figure 4: Effect on hare population when the lynx population is constant (Windows)

Making the lynx population variable

You may have noticed in the previous model that the lynx population is a constant. Now you want to refine the model to have the lynx population be more realistic.

To do this, change the model to reflect the initial population of 125 lynx and their birth rate of 0.25 offspring each per year, as stated earlier. The easiest way to do this is to duplicate the section of the model that represents the hare population and change the parameters to reflect lynx statistics. Then delete the Constant block that represented the lynx population, and use a named connection to feed the actual lynx population to the Multiply block, as seen below:

Extend

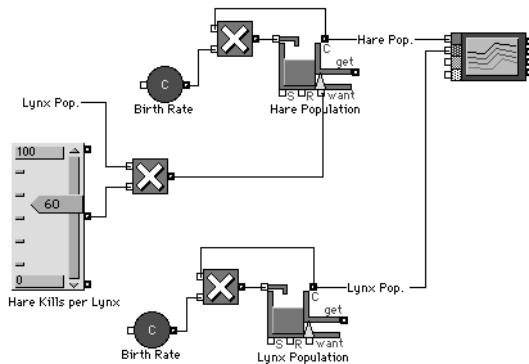


Figure 5: Variable lynx population

As seen in the plot below, the lynx population is now variable. However, the model shows the lynx continuing to proliferate even though they have completely eliminated their food source in the first year. To correct this, you first need to improve how the hare mortality factor is modeled; later you will refine the interaction between the lynx and their food source.

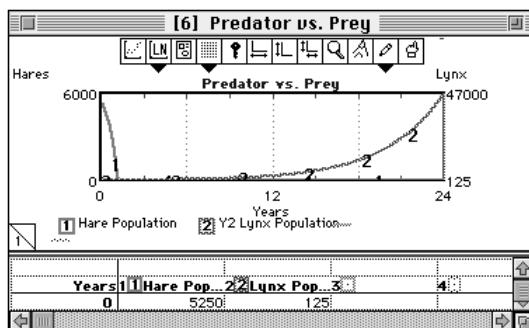
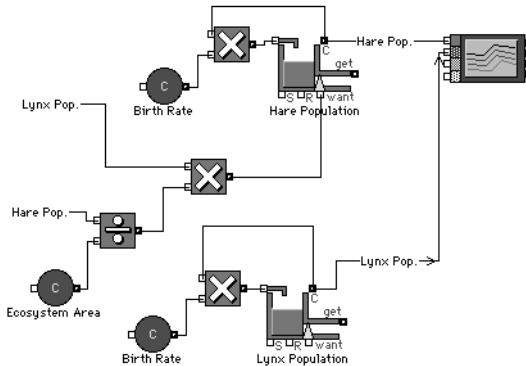


Figure 6: Plot showing variable lynx population (Macintosh)

Refining the hare mortality factor

The model built so far uses a Slider to provide a constant mortality rate for the hare. However, as stated originally, the number of hare killed is dependent on their density in the ecosystem and the number of lynx available to hunt them. The assumptions also state that the lynx have a territorial range of one hectare and can consume any hare in their territory.

The next step is to change the hare mortality factor to be variable, based on their density in the ecosystem. To model this, remove the Slider and substitute two blocks for it: a Constant block (representing the area of the ecosystem), and a Divide block (which divides the hare population by the ecosystem area):



Extend

Figure 7: Model with variable hare mortality rate

The model now indicates that the hare mortality rate is dependent on the density of hares in the ecosystem and that the lynx will consume any hares that are available.

The plot shows a slight change in the hare population (line 1), but the change in the lynx population (line 2) is still independent of the change in the hare population:

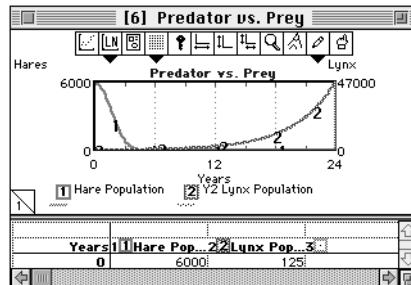


Figure 8: Plot for Figure 7 model (Macintosh)

Refining the lynx mortality factor

The final step in building this model is to include the interdependency of the lynx and the hare. Lynx need hare to survive and the number of hare consumed directly affects the mortality rate of lynx. The relationship of the number of hare consumed to the lynx mortality rate is given in the

Extend

following table, where the first column is the density of hare and the second column is the corresponding lynx mortality rate:

Row	x in	y out
0	0	0.5
1	10	0.45
2	20	0.4
3	30	0.35
4	40	0.3
5	50	0.25
6	60	0.2
7	70	0.15
8	80	0.1
9	90	0.05
10	100	0.005
11	100000	0.005

Figure 9: Relationship between number of hares consumed by a lynx and its mortality rate (Macintosh)

To include this table in the model, add a Conversion Table block and enter the table numbers in its dialog. Connect from the Divide block's output to the input of the Conversion Table block, so the Conversion Table block can see the current density of hare. Then add a Multiply block to multiply the output of the Conversion Table block (the lynx mortality rate) by the current lynx population, as seen below:

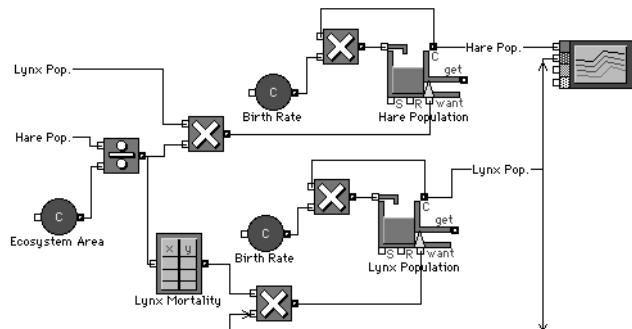
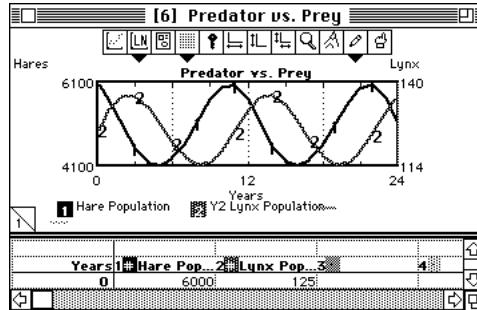


Figure 10: Completed model with interdependencies between hare and lynx

At each step of the simulation the Conversion Table block compares the hare density at its input to the values in the first column of the table, then outputs the corresponding mortality rate from the second column. In the Conversion Table dialog, output values are set to “interpolated”. For example, an input value of 65 (which is half way between the “x in” values of 60 and 70) will cause the block to output 0.175 (which is half way between the “y out” values 0.2 and 0.15).

When you run the simulation, you see the direct relationship between the hare and the lynx as seen in the following plot:



Extend

Figure 11: Plot showing interdependencies (Macintosh)

Further exploration

To make the model more attractive and easier for others to understand, you could use Extend's hierarchical block feature to encapsulate the blocks. For instance, the hierarchical block's icon for the lynx might be a picture:



Figure 12: Hierarchical block's icon

If you clone critical parameters to the Notebook, you can experiment with the model and view the results of the experiments in one location:

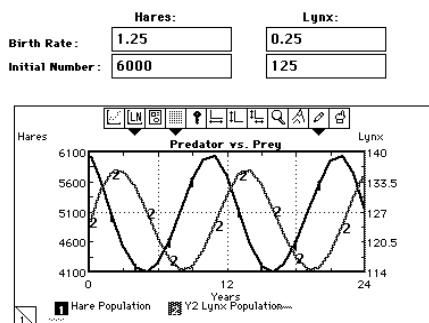


Figure 13: Notebook from Predator/Prey model

The “Predator/Prey” model was built using standard blocks from Extend’s Generic library. For some users, models built using standard blocks may not be the correct choice. For example, you may want one block that represents all aspects of the hare population: the natality and mortality rates, the initial number in the system, and so forth. If you program, you can easily build blocks with custom dialog boxes and robust behavior. For example, the dialog of the Fish block in the Custom Blocks library is:

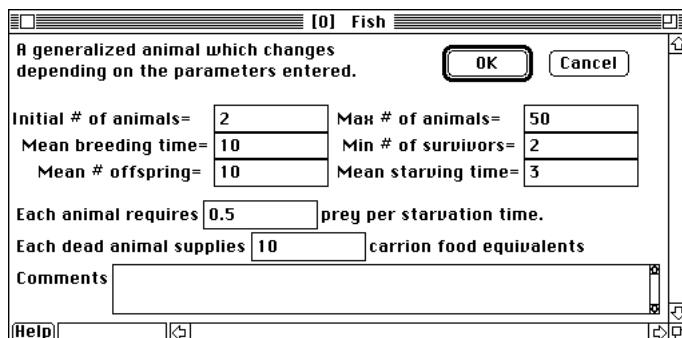


Figure 14: Fish block from Custom Blocks library (Macintosh)

If you examine the Predator/Prey model and review the initial assumptions, you can probably see several enhancements that could be made:

- You could have the birth rates for both the hares and the lynx vary based on model conditions or on outside factors. For instance, the birth rate could be dependent on the health of the parents, the level of crowding, or the amount of pollution in the ecosystem.
- You might add a predator for the lynx, or add an additional food source.
- The model assumes an unlimited food supply for the hares. To a food source, the hares would be considered the predator, so modeling a food source would follow the same logic as adding the lynx predators.
- You could factor in outside conditions, such as the time of year and expected weather conditions, and examine the effect on the mortality rates.

Discrete Event library

As described at the beginning of this chapter, blocks in the Discrete Event library correspond to typical activities, operations, and resources in many environments. These blocks are connected in an activity or data flow diagram that represents a system. The complexities of generating and posting the events are handled within the blocks, alleviating the need to do any programming in the ModL language.

Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

Using the Discrete Event library

As discussed below, discrete event models pass entities (called *items*) from block to block during the simulation. The items in the simulation are usually generated by the Generator block as a random distribution within specific parameters, or by the Program block as a list of when events occur. These items often have priorities and attributes which help them correspond more closely to parts, customers, and jobs in real life.

In discrete event blocks, an item connector passes an item and all the information associated with it to the next item connector. Value connectors and dialog parameters provide specific information about the item and its properties (attributes, timing, and so on) and information about the effects that the item has in the model (such as queue length and wait). Note that it is this value information which is plotted and displayed in a discrete event model, not the items themselves. Value and item connectors were introduced in “Connecting blocks” on page E43; item connectors are discussed in more detail below.

Extend

The object of the simulation generally is to determine where there are bottlenecks in the process and to see which parts of the process might be improved. Each branch of the flow diagram should either feed into another block or end in an Exit block. You can examine the output of the Exit block to find when items are leaving the model.

All of the blocks in the Discrete Event library have dialogs associated with them. You use the dialog to enter values and to look at the current values of the inputs. If you have connected an input and then enter a value for that input in the dialog, the connector’s value always overrides the value in the dialog. The dialog for each block has a help button in the lower left corner that gives an explanation of the block, its connectors, its parameters, and its options.

The icons for the blocks are color-coded. Inputs have green borders, outputs have red borders, and computational blocks have blue borders.

A model can combine continuous blocks, such as the ones in the Generic library, with discrete event blocks. If you use any discrete event blocks in a model, the model will become discrete event and will require the Executive block.

Layout of your model

You can place the blocks in your model anywhere you want, remembering that Extend evaluates discrete event blocks along the path of the connections. The only exception to this generality is that the Executive block (which is required for all discrete event simulations) must be to the left of all other blocks.

Items and informational values

The basic units that are passed between discrete event blocks are *items*. Items are individual entities that can have unique properties as specified by their attributes, priorities, and values. In manufacturing models, items may be parts on an assembly line; in network models, an item would be a packet of information; in business models, items may be invoices or people. Items are passed

from block to block through item connectors. Items are generated by the Generator and Program blocks, and are provided by the Resource block. The Generator block can generate items with a random distribution or at a constant rate of arrival. The Program block generates items at a fixed schedule or on demand. The Resource block provides a finite pool of items.

Values provide information about items and about model conditions. Values tell you the number of customers in queue, how many parts have been shipped, and how frequently telephone calls occur. Values also report processing time, utilization, and cycle time. Informational values are passed through value connectors. When you use a plotter in a discrete event model you are plotting information about items, not the items themselves. For example, when you connect the # output of an Exit block to a plotter, you are displaying the time that each item left the model and how many items have left.

Events

Extend moves items in your model only when an *event* occurs. Events are occurrences such as receipt of an order, a telephone call, or a customer arriving. Events are controlled by the Executive block and only occur when particular blocks specify that they should. When you first start a simulation, a single event is generated. Blocks that depend on time cause events to happen at the appropriate time. For instance, the Activity Delay block holds an item until a particular time. When the time is reached, the block causes an event to occur.

Blocks that do not generate events allow the blocks after them to pull items during a single event. Thus, an item can pass through many blocks after a single event if those blocks do not stop them. For instance, the Set Attribute block sets the item's attribute and passes the item to the next block in the same event.

Item connectors

Most of the blocks in the Discrete Event library pass a set of information through item connectors at each event. This is different than the blocks in the Generic library which only pass values. This set of information carries data about discrete items, their attributes, priorities, quantities, and so on. Item connectors are different than value input and output connectors in Extend. The item input connector is  and the output connector is 

When you are combining blocks from other libraries with Discrete Event blocks, you will only be able to connect compatible connectors. Value connectors can only connect with value or universal connectors. Value connectors cannot connect with the item input or output connectors.

Attributes

Attributes are a very important part of a discrete event simulation. Attributes are characteristics of an item that stay with the item as it moves through the simulation. Each attribute consists of a name and a numeric value (not to be confused with an *Item Value*, discussed below). The attribute name is used to identify the attribute and the attribute value is used to specify some aspect associ-

ated with the name. For example, an item's attribute name might be "ColorValue" or "Process-Time", and its value might be "14652" or "0.5". Attributes can be used for routing instructions, operation times, or part quality in statistical process control.

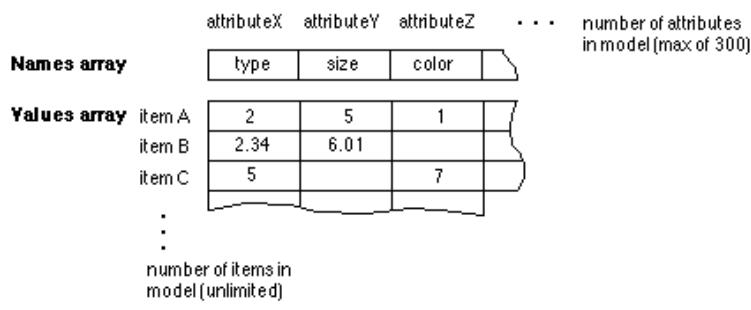
You can set attribute values for the items in a discrete event simulation. See the Set Attribute, Get Attribute, and Activity Delay (Attributes) blocks for information on manipulating item attributes. Use these blocks to create and check attributes in your simulation.

There can be up to 300 different attribute names within a model and up to 300 attributes for each item. The attribute names and values are stored in a pair of dynamic, global arrays:



- The one-dimensional Names array stores the name of each attribute. Attribute names can be up to 15 characters long, but they are stored in a string with a 2550 character limit. Thus your models are limited to 300 attribute names with an average length of 8.5 characters. 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:



The attribute arrays

As different 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 300. 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 you can see in the example above, 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 attribute to be used, you must assign a value to it using one of the attribute-handling blocks (such as the Set Attribute block). If there is no value assigned, the attribute is not used by that item. This is shown in the figure.

ure 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”.

Note If the total length of the string of attribute names exceeds 2550 characters in your model, you will not be able to add new attributes. However, unused attribute names are automatically removed when you run the model, freeing up any extra space.

Extend

Priorities

Priorities allow you to specify the importance of an item. For instance, you might have a step in a manufacturing process where a worker looks at all the pending job orders and chooses the one that is most urgent. Each item can only have one priority. The top priority has the lowest value, including negative values (that is, an item with a priority of “-1” has a higher priority than one with a priority of “2”). The Set Priority block assigns a priority to an item. The Queue Priority block passes items out highest priority first. You can view an item’s priority with the Get Priority, Status, or Information blocks.

Item Values

Each item can be a single entity or a group of duplicates. If the *Value* of an item is 1, it represents one item; if it is other than 1, it represents a group. Items originally have a Value of 1 unless the Value is changed by the Set Value, Generator, Get Attribute, or Program blocks.

For most purposes you would not want to change the Value of an item from 1. However, you may want to model a change of shift consisting of five workers going on duty at the same time. Or you might want to simulate the delivery of a box of 300 pieces of mail to a mail room. In those cases you could set the Value of an item to be greater than 1.

Items with Values other than 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 block, or are sent into a universal connector (such as change, demand, select, or start).

- When an item with a Value other than 1 reaches an exit, a queue, a batch, or a resource block, it is decomposed into separate identical items. For example, when it enters a queue, an item with a Value of 10 will become 10 distinct items, each with a Value of 1 and each with the same properties (attributes, priorities, and so on) of the original item.
- A *change* connector, such as on the Resource block, works the same as its regular item input connector, except it is also capable of accepting items with negative Values. An item with a negative Value at a block’s *change* connector will decrease the number of items available in the block by that Value. The block will also keep track of a negative demand if there were not enough items to delete at the time the negative Value came in. If the item has Value smaller than -1, the item will be decomposed into separate items, as discussed above.

- When items with Values other than 1 are sent into a universal connector (such as *demand*, *select*, or *start*), they are treated as one item, but the Value of the item may be used by the block as control information. See below for more information on how universal connectors deal with the values of incoming items.
- All the other blocks deal with items that have Values greater than 1 as a single item, ignoring the Value associated with it. Note that Activity Delay, as well as other delay blocks, accept items with a Value of greater than 1 as a single item, and process them as one unit. If you want the items to be processed separately, you must precede the activity block with a queue block, since the queue blocks decompose items with Values greater than 1.

Extend

Modifying blocks in the library

The Discrete Event library uses special data structures and programming methods for the blocks in the library. If you are a programmer and want to build a discrete event block of your own, you should use the blocks in the Discrete Event library as a base. Either use a copy of a block similar to the one you want to build, or use the “Make Your Own” block from the Make Your Own DE library as a template. The ModL code of the Make Your Own block is commented to help you understand how certain features are implemented. Be sure to read Chapters 9 through 12 before modifying the blocks so you have a better understanding of how blocks work internally (Chapter 4: ModL Programming Techniques has detailed information on modifying Discrete Event blocks). You may also wish to look at the ModL code of the Make Your Own block to get a better understanding of what the blocks in this library do.

Moving items through a simulation

In general, item input connectors on Discrete Event library 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, activity blocks such as Activity Delay pull items from preceding blocks, process those items, and hold them to be picked up by another block. 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.

Holding and pushing

Discrete Event library blocks treat their output items in one of two ways:

- The item is held in the block and leaves only when another block pulls in the item (this is the default method for most blocks).
- The item is pushed from the block when a new item comes in, regardless of whether it will be picked up by another block.

The Generator, Program, Schedule (Manufacturing), and Import (BPR) blocks have to push items out, because their items are “arrival time” related. (Usually you will want to follow these blocks with a queue to collect the items and hold them, so that they are available for the rest of the model.) All other discrete event blocks always hold items, expecting their outputs to be pulled.

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 *sensor* connectors on the Timer and Gate blocks
- the item input connector on the Status block when “View items” is selected in the dialog

A more detailed discussion on the movement of items through a model can be found in “How discrete event blocks and models work” on page P194.

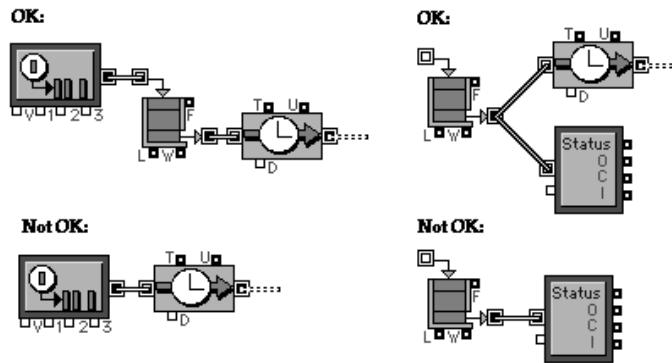
Pitfalls to avoid

When a block that pushes items out at specific arrival times, such as the Generator, 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. For the most part, you should avoid this situation.

When a block holds an item and it is *only* viewed by a status block, the item does not move through the simulation. This is never desired. For example, if the items in block A are being viewed by Status block B, the output of block A must also be connected to the input of block C. Block C will then pull items in, process them, and send them on their way. Otherwise, block A would only get the first item, which would then have nowhere to go.

Although you will not usually encounter these problems, you should be careful about using the blocks that always push or that view when provided items. You should probably connect blocks that always push (the Generator and Program blocks) to queues so that items do not get lost.

Always connect blocks that view (such as the Status block) in parallel with other blocks that will pull the items as the block views them.



Extend

Using blocks that push and view safely

Common connectors in the Discrete Event library

The following connectors appear in many blocks in the Discrete Event library:

Connector	Meaning
#	Number of items
A	Attribute for the item passing through the block
a, b, c	Nondifferentiated inputs and outputs for blocks that combine, split up, or select between many inputs and outputs
D	Amount of time to delay the item in the block
demand	Causes the block to accumulate a demand for multiple items if an item input is connected. It is also used to trigger the operation of the Activity Service block.
Δ	Indicates a change in value
F	Full: outputs 1 when the block is full, otherwise outputs 0
L	Length of queue: the number of items in the queue
P	Priority for the item passing through the block
sensor	Watches for the presence of an item (views) but does not receive the item

Extend

Connector	Meaning
select	Selects the input or output based on the choices in the dialog.
T	Time in use: outputs a non-zero value while the block is in use. This value represents the amount of time since the last event, at each event when the block is active. You can connect it to an Accumulate block to find the total time in use during the simulation.
U	Utilization; the percentage of time the block was in use over the course of the simulation run. The total time that items were in the block is used in the calculation, including delay times and times while items were blocked (when items were finished processing but remained in the block because the next block was not ready to pull them).
V	Value for the item passing through the block (this is usually 1)
W	Wait time for items leaving the queue

In a queue block's dialog, selecting "histogram" causes the *L* and *W* connectors to output noValue if between departures (this is only important for the Histogram block in the Plotter library which counts the absolute number of times a value appears). If "continuous" is selected, the values are repeated between departures.

You can set the maximum number of items that can be held in the queue blocks and the Activity Multiple block in those blocks' dialogs. When this number is reached during the simulation, the *F* connector changes from 0 to 1; at that point, the block will not receive any more items until an item has been removed from the block. For example, you can connect the *F* connector to the *demand* connector on an Activity Service block to activate another server when a queue becomes full.

Universal input connectors

The universal input connector, \oplus , is a special-purpose connector on discrete event blocks. Universal connectors accept inputs from either value output connectors or item output connectors. The presence of an input at a universal connector is used to trigger or control some aspect of the block, as specified in the block's help.

- If a *value* output connector is connected to the universal input, it is used as a true/false indicator, triggering some action by the block. The actual value from the value connector is ignored; what is considered is whether or not it exceeds a limit. Value connectors outputting a value less than or equal to 0.5 are considered false, while a value above 0.5 is considered true. For example, a value connector sending the number 4 to the *start* connector of a Program block will cause the block to start its program. If the value had been 0.3, the program would not have started.
- If an *item* connector is connected to the universal input connector, any item it receives is considered the same as a true value and causes the block to be activated. For example, an item received

at the *start* connector of the Program block will cause the block to start its program. Until an item is received, the program will not start. Note that all universal input connectors absorb the items they receive, removing them from the simulation.

The common universal input connectors (*select*, *start*, and *demand*) all work as discussed above. The *demand* connectors, such as on the Batch and Activity Service blocks, have an additional feature when connected to item connectors: they take into consideration the value of the item at demand. For example, if an item connector sends an item with a value of 4 to the *demand* connector of the Activity Service block, the demand connector will cause the block to pull in 4 items for processing before it stops pulling in items. If required items are not immediately available, the block will remember the demand amount and decrement it only as items become available and are pulled in for processing. In addition, the Activity Service block accumulates demand: two items at the *demand* connector with values of 4 and 3 will cause the block to accumulate a demand for 7 items at its item input connector. Note that this demand feature only applies to items which have values; as mentioned above, it does not apply to the values coming from value connectors.

Extend

Each of the universal input connectors has its own characteristics, as explained in the help of the block. You should look there for information about what aspect of the block is triggered, what happens when another input is received before the block is finished processing, what the value of an item means to a particular demand connector, and so on.

Blocks by function in the Discrete Event library

The blocks are grouped here by their type: activities, attributes, batching, generators, information, queues, resources, routing and executive (see “Block types” on page P56 for a discussion on block types). Each line in this brief list tells the block’s name, use, and examples of how you might use it in a model. A more detailed alphabetical list is shown in “Appendix E: Discrete Event Library Blocks” on page A55.

Activities	Use	Example
Activity, Delay	Holds an item for a specified amount of time, then releases it.	Red lights in traffic, teller’s service time, multitasking CPU time
Activity, Delay (Attributes)	Activity Delay that can interact with item attributes.	Customers with specific problems that take specific lengths of time, preparation periods
Activity, Multiple	Holds many items and passes them out based on the delay and arrival time for each item.	Ceramics kiln, pizza oven, supermarket, cafeteria table

Extend

Attributes	Use	Example
Change Attribute	Combines the features of the Get Attribute and Set Attribute blocks.	Marking up, annotating
DE Equation	Calculates an equation when an item passes through the block.	Adding attribute values
Get Attribute	Displays and/or removes attributes on items.	Reading a parts list, a movie rating, or an employee record
Get Priority	Displays the priority of items.	Police dispatcher, network scheduler
Get Value	Returns the value of the item.	Checking number of customers
Set Attribute	Sets the attributes of items passing through it.	Adding to a parts list, writing part of a movie rating, writing in an employee record
Set Attribute (5)	Sets up to five attributes of items passing through the block.	Applying many tags at once
Set Priority	Assigns a priority to items that pass through.	Job priority, first class passenger
Set Value	Assigns a value to items.	Number of customers, number of parts

Batching	Use	Example
Batch	Allows items from several sources to be joined into a single item.	Parts kitting, assemble parts into a whole, combine labor resource with item during assembly process
Unbatch	Generates several items from a single input item as specified in the dialog.	Route messages or invoice copies, duplicate a message packet, release a resource in use, generate a signal

Generators	Use	Example
Generator	Introduces items into a discrete event simulation at specified arrival times.	Customers, factory parts, computer programs, network messages

Generators	Use	Example
Program	Schedules many items.	Coffee breaks, work shifts, computer programs, resource allocation
Shift	Generates a schedule over time which can be used to change the capacity of other blocks in the model.	Employee shifts.

Extend

Information	Use	Example
Count Items	Counts the number of items that pass through the block.	Customers, factory parts, computer programs, network messages
Information	Displays information about the items that pass through it.	
Show Times	Displays when the next event will occur for each block that posts events in the model.	
Status	Views and displays information about an item or values going between blocks.	
Timer	Displays the time that it takes an item to pass between two parts of the model.	Cycle time, production time, transit time

Queues	Use	Example
Queue, Attributes	Provides a queue where items with one or more specified attributes have a higher priority than other items.	Sort on color, failure rates, or quality
Queue, FIFO	Provides a first-in-first-out (FIFO) queue.	First-come-first-served lines, inventory turnover
Queue, LIFO	Provides a last-in-first-out (LIFO) queue.	Unrotated stock, vertical stack of books at the bookstore
Queue, Matching	Provides a queue where items are released only if they have the specified attribute and that attribute's value matches the value at the ID connector.	Tagged item retrieval

Extend

Queues	Use	Example
Queue, Priority	Provides a queue that releases the highest priority item first.	Priority mail, project management, triage
Queue, Resource Pool	Provides a queue where items are released only after specified resource requirement are met. Works in conjunction with Resource Pool block.	Storage area for parts waiting for labor to become available

Resources	Use	Example
Release Resource Pool	Releases resource by incrementing the number available in the appropriate Resource Pool block.	Return of labor to labor pool after completing work assignment
Resource	Similar to a queue, but is set up to be used as a resource pool of items, with an initial amount specified in the dialog.	Labor pool, parts bins, finite memory, disk size
Resource Pool	Provides an inventory of resources to be used with the Queue, Resource Pool block	Labor pool, parts bins, finite memory, disk size

Routing	Use	Example
Activity, Service	Passes an item only when the <i>demand</i> connector is connected and either <i>demand</i> connector's value is true (greater than 0.5) or the <i>demand</i> connector pulls in an item.	A new checkout line has opened, a new machine is available
Catch	"Catches" items sent by Throw blocks even though the blocks aren't connected by connection lines. Use these blocks instead of several Combine blocks.	Merging traffic, customers coming from many entrances to form one line
Combine	Combines the items from two different sources into a single stream.	Merging traffic, customers coming from many entrances to form one line
Exit	Passes items out of the simulation.	Scrap, finished goods, completed projects
Exit (4)	Passes items from many streams out of the simulation.	

Routing	Use	Example
Gate	Passes a new item into a section of the model only when an item leaves that section.	Flow restricter into a designated section of the model
Prioritizer	Prioritizes the outputs, allowing items to be sent into parallel process activities based on a priority you assign.	Watching for available slots
Select DE Input	Selects inputs based on a decision.	Traffic signals at an intersection, candidate selection, CPU interrupt access
Select DE Output	Selects outputs based on a decision.	Choice of shipping method, routing priority
Throw	“Throws” items to a Catch block without using an output connector or connection lines.	Merging traffic, customers coming from many entrances to form one line

Extend

Executive	Use	Example
Executive	Controls the timing and passing of events in a discrete event model. This block is the heart of each discrete event model and must be placed to the left of all other blocks in the model.	

Discrete Event library example

The Discrete Event library lets you create models that are based on events occurring exactly when they are needed. The most common discrete event model involves the handling of one or more waiting lines or queues, such as those found in supermarkets or factories. The key to discrete event modeling is the construction of a flow diagram using the available blocks to represent operations and resources in your problem.

The following example, a car wash model, shows the basic applications of many of the blocks in the Discrete Event library. It also shows how blocks from the Generic library are used in discrete event models. The model assumptions for this entire section are:

- The blocks come from the Discrete Event and Plotter libraries
- Cars arrive approximately every 4 minutes
- There is only one line into the car wash
- It takes 6 minutes to wash a car; it takes 8 minutes to wash and wax a car

- Approximately 25% of the cars want to be waxed
- Some drivers tip to get their cars moved to the front of the line

Each step of the model is shown in a separate file in the Car Wash folder, located within the Tutorials folder.

Extend

Basic discrete event model

As was true for the continuous model built earlier in this chapter, it is important to start with a simple model, then add complexity as needed until all important factors are considered and the model approximates the system you are modeling.

The first step for this model is to model the car wash with one bay which just washes the cars. The model shows dirty cars arriving in a random fashion averaging one every four minutes. These cars form a line waiting to enter the wash. Every car must wait for the car in front of it to move through the car wash before it can enter. The model for this car wash is shown in Figure 1:

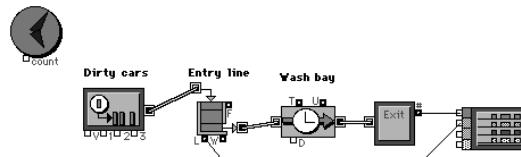


Figure 1: Car wash model

The Executive block is at the far left and sets the simulation clock to the time of the next event (it is not shown in the other models in this chapter, but is to the left of all other blocks). As described in the Discrete Event library reference, this block must always be to the left of all other blocks in the model. The Executive block converts Extend's engine from continuous to discrete event timing.

The arrival of dirty cars is modeled by a Generator block. The Generator block is used to produce items (such as customers for a store or items of inventory for a warehouse) in discrete event models. In this case, you use a Generator with an exponential distribution and a mean arrival time of 4 minutes. The exponential distribution causes more arrivals close together than far apart. This corresponds to the familiar complaint that either everything is happening at once or nothing at all is happening. Because the Generator block pushes items as discussed on page E118, it should be followed by a queue or Resource block, so items do not disappear from the model.

Use a Queue FIFO block (first in, first out) for the line of waiting cars since the first car into line will be the first out of the queue. The delay that the cars experience in the wash is represented by an Activity Delay block with a constant service (delay) time of 6 minutes set in the block's dialog.

Cars that have passed through the system end in the Exit block. The Exit block allows items to be counted as they pass out of the simulation. All sections of a discrete event model should either be

connected to another section or end in an Exit block. Otherwise, items will back up in that part of the model.

A Plotter Discrete Event block from the Plotter library is used to plot both the queue length and the number of cars that exit the system. You cannot attach an item output connector (such as the one coming from the Activity Delay block) directly to a plotter; you must first go through an Exit block. You can, however, connect the L connector directly to the plotter because these are both value connectors and therefore compatible. Set the simulation end time to 480 minutes in the Discrete event tab of the Simulation Setup command (Run menu), an average 8-hour day.

Extend

There are several ways to monitor what happens in this simulation. The queue length and number of cars exiting the wash are plotted with a Plotter Discrete Event block, as shown in Figure 2. The Exit block keeps track of how many customers pass through the system; you can also see the number by opening its dialog. The Queue FIFO block gathers statistics on the average queue length, average wait time, and utilization; all of these are shown in the block's dialog.

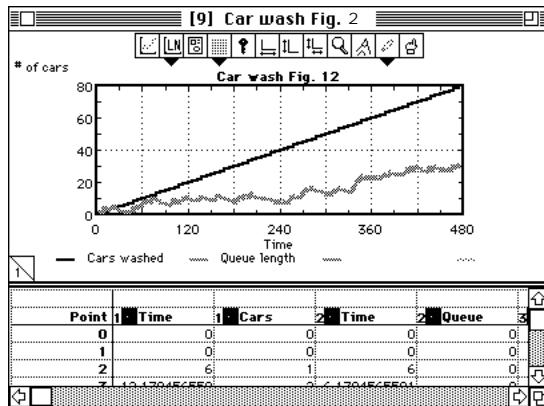


Figure 2: Car wash output (Macintosh)

Adding multiple servers to a queue

When you run the simulation, you see the entry line length (the green line in the plot) increasing over time because, on the average, the cars are arriving faster than they are being washed. Since long lines deter customers, it would be better to keep the entry line short. One way of doing this would be to add a second washing bay to the car wash. This is an example of *parallel processing* of the cars (parallel processing is described in more detail in “Connections to multiple inputs” on page E202). You can do this in Extend by simply selecting the Activity Delay block, copying, and pasting it just above the existing Activity Delay. This provides two servers (wash bays) to the line.

Delete the Exit block and replace it with an Exit (4) block so both lines exit. The new two bay car wash is shown in Figure 3:

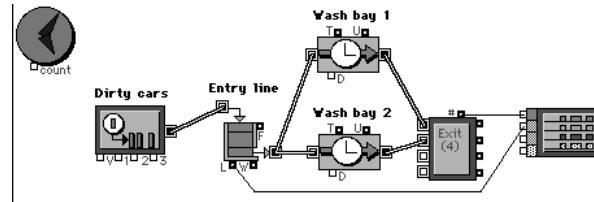


Figure 3: Car wash with two bays

Extend

When you run this simulation, the entry line length stays near 0 most of the time, as shown in Figure 4. If you look at the dialog of the Exit (4) block, you will probably see that one number is larger than the other. Since you have not specified any rules concerning how the cars are routed to a wash bay, a car will go to the first available bay. If both bays are free, the car will arbitrarily go to either wash bay. This could result in cars using one bay more frequently than the other.

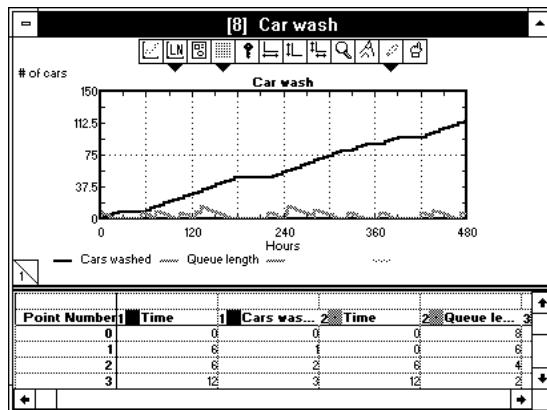


Figure 4: Output of car wash with two bays (Windows)

Attributes

Most car washes let you decide whether or not you want wax applied after your wash. You can use attributes in your model to indicate that specific cars should or should not be waxed. You do this by adding an attribute to the cars coming from the Generator block, then checking for the value of that attribute as the car gets washed. A value of 0 means no wax, a value of 1 means wax.

To add an attribute to an item, use the Set Attribute block from the Attributes submenu and name the first attribute “wax”. Notice that the value at the *A* connector, if connected, determines the attribute value. In this case, make the input to the *A* connector an Input Random Number block from the Inputs/Outputs submenu of the Generic library that reflects the assumption that 75% of the cars are no wax (0) and 25% are waxed (1). Do this in the dialog of the Input Random Number block by selecting an empirical distribution with Discrete option and a general table of

values like that shown in Figure 5. Figure 6 shows how you might include these two blocks in the preceding model.

Row	Value	Probability
0	0	0.75
1	1	0.25
2		

Figure 5: Empirical distribution table for attribute value

Extend

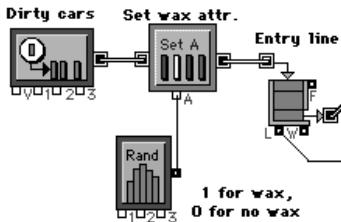


Figure 6: Set Attribute and Input Random Number blocks added to model

When you do this, the Set Attribute block attaches attributes to the items (cars) provided by the Generator. The attribute name is the name, “Wax”, specified in the Set Attribute dialog. The attribute values are either 0 (approximately 75% of the time) or 1 (approximately 25% of the time), as specified by the Input Random Number block.

Assume that your top car wash bay is used for the cars that do not want wax and the bottom one is for cars that will be waxed. Also assume that adding wax increases the amount of time to wash the car by two minutes. Change the delay time for the bottom line from 6 to 8 and change the labels on the delays for clarity. To determine which line each car should go in, add a Get Attribute (Attributes submenu) and a Select DE Output block (Routing submenu) between the queue and the lines as shown in Figure 7. By default, the Get Attribute block looks for the first attribute on each item (in this case, the “Wax” attribute) and reports the value of the attribute at the *A* connector when the item passes out of the block.

The Select block is used to route the cars based on the attribute found. In the Select DE Output dialog, indicate that each Select input chooses a connector and that if the Select input is 0, the top

output is used. This will route the “no wax” cars to the top activity block and the “wax” cars to the bottom.

Extend

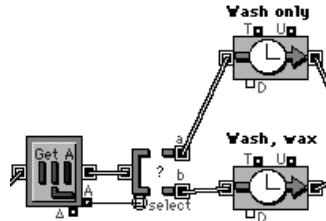


Figure 7: Get Attribute and Select DE Output blocks added to model

Figure 8 shows the whole model and Figure 9 shows output from this model. Note that fewer cars pass through this car wash than the example without attributes. This is due to the problem of a car with a particular attribute following another car with the same attribute. In this case, there is only one entrance to the wash bays, so the second car must wait for the first one to finish even if the other bay is free. If you look at the numbers in the Exit (4) block, you will see that they are roughly the same as the distribution specified in the Input Random Number block.

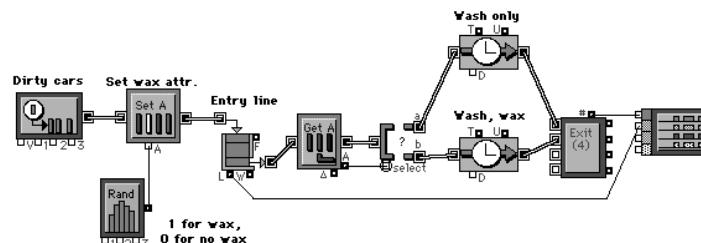


Figure 8: Attribute model

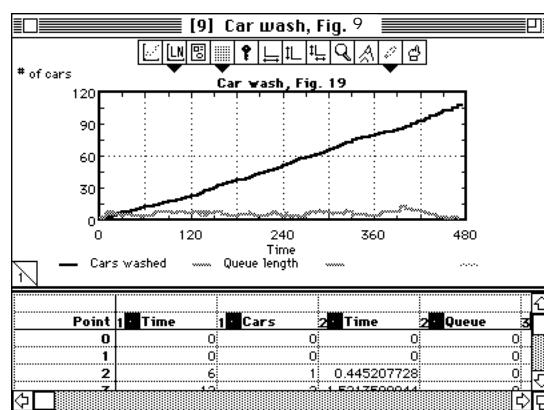


Figure 9: Output from attributes example (Macintosh)

Priorities

Using priorities is similar to using attributes except that there is only one value to set. Instead of the wax/no wax attribute, you might let your customers give an extra tip to get a higher priority. Of course, not many people would do this (and many would be offended at the suggestion), so it is likely that you will have many more low-priority cars than high-priority ones. Figure 11 shows the two-bay car wash with priorities added. After the Generator block, a Set Priority block (Attributes submenu) is added to give each item a priority. The queue in the model should now be a Queue Priority block (Queues submenu). In this case, you set the priority using an Input Random Number block from the Inputs/Outputs submenu of the Generic library with an general distribution with 10% getting a priority of 1 (biggest tip), 20% a priority of 2 (small tip), and 70% getting a priority of 3 (no tip), as shown in Figure 10.

Extend

Row	Value	Probability
0	1	0.1
1	2	0.2
2	3	0.7
3		

Figure 10: Distribution of priorities

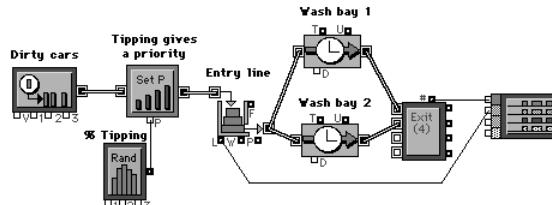


Figure 11: Car wash with priorities

When you run this model, you see that the length of the queue is not changed from what you saw in the original two-bay car wash because cars are simply being reordered with no effect on the delays. The average wait in the queue is also unaffected since low-priority cars are waiting more but high-priority cars are waiting less.

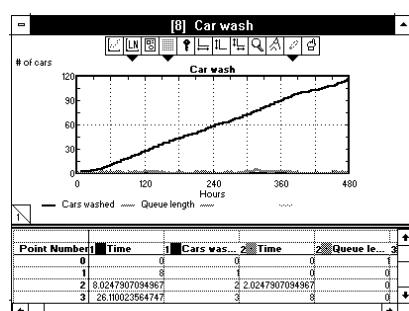


Figure 12: Priorities output (Windows)

Values

There are some times when you want to indicate that more than one item is passing through at a time. For this, you change the Value on the item to be greater than 1.

For example, assume you want to model a commercial car wash that washes a group of cars delivered periodically by a car delivery truck. Using the two-bay car wash in Figure 3 as a base, you must change the parameters in the Generator block. You can plot the number of cars being delivered as well as the number waiting in the queue.

Open the Generator block and change the mean arrival times and Value of item (V), as shown in Figure 13.

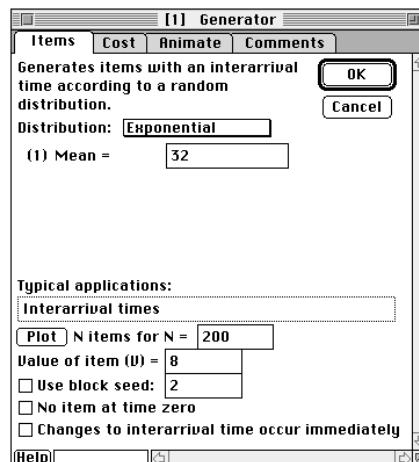


Figure 13: New arrival times and value (Macintosh)

This indicates that, compared to the two-car wash in Figure 3, there are 8 times as many cars per event, but they arrive one eighth as often. As discussed in the section on Values earlier in this chapter, queue blocks separate items with a Value other than 1 into clones of the original item. Although the Generator block provides a truck-load of cars, the truck-load is separated into eight cars after it enters the queue. Each of the eight cars is then processed individually. The model looks like Figure 14.

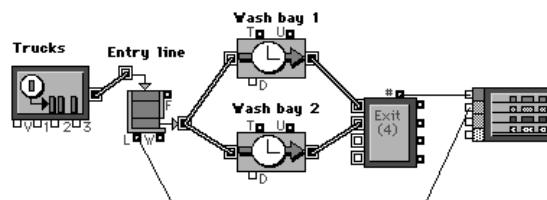
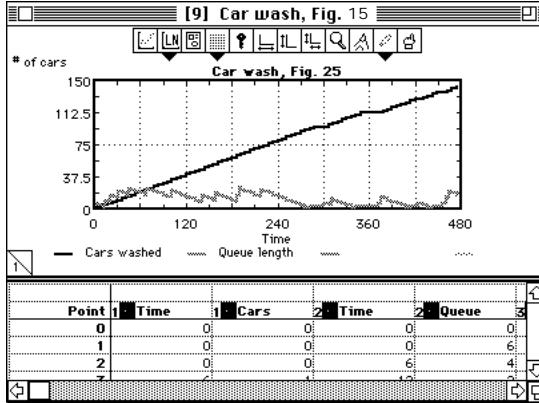


Figure 14: New model

The plotter in Figure 15 shows that the length of the waiting line now goes up and down more regularly, reflecting the arrival of 8 cars at once, with longer periods in between arrivals.



Extend

Figure 15: Plot of truckloads of cars being washed (Macintosh)

Getting data from files

So far, all of the characteristics of the cars have come from statistical distributions. You may have real-world data that you want to use instead of just a distribution. For instance, if you owned the car wash you were modeling in the previous sections, you may have kept data from typical days for each car: did it want wax, did it tip for a priority, did it come from a truck, and so on. If you keep this data in a file, you can hook that file into your model with the File Input block and use it instead of a statistical distribution.

The File Input block from the Inputs/Outputs submenu of the Generic library reads in a text file created in Extend or in a word processing or spreadsheet program. The text file should have one row for each car that comes to the car wash, with three columns (separated by tab characters) representing the attribute, priority, and Value of the car. The first column will contain a 0 for no wax or a 1 for wax; the second column specifies the priority (1, 2, or 3, with 1 being the highest priority), and the third column is the number of identical cars being delivered at one time (this is usually just 1, meaning a single car). For this example, the data is included in a text file (Macintosh: "Car Wash data"; Windows: "data.txt"). The first few lines of the file look like:

0	0	0
0	3	1
1	3	1
1	1	4

Figure 16: Car wash data

The first row is all zeros, because you want the relevant data to start at row 1, as discussed below. The next line corresponds to row 1 and the first car. The first car wanted no wax and gave no tip,

and the second car wanted wax and gave no tip. The third line indicates a group of four identical cars that want wax and the truck driver left a big tip.

To model this data, you need to know how often the cars arrived at your car wash. Assume that your records indicate arrival times approximately 4 minutes apart, normally distributed.

Figure 17 shows the entire model. The Generator block sends out items whose arrival times correspond to a normal distribution with a mean of 4 and a standard deviation of 1. In that dialog, the Value for “value of items (V)” is the default of 1. The Status block *views* the items as they are generated and gathers information about them. The *Status connector* of the Status block outputs the number of items that have been generated so far in the simulation. This value is put into the *row* connector of the File Input block so that as each item is generated, the File Input block will output the data for the row corresponding to the number of that item. For example, when item number 2 is produced by the Generator, the File Input block will put out the data for row two.

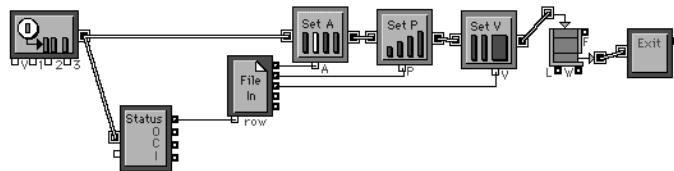


Figure 17: Getting values from a file

The three output connectors of the File Input block then give those values to the Set Attribute, Set Priority, and Set Value blocks all from the Attributes submenu of the Discrete Event library. After these three blocks, the item has all the desired properties. You could then decide how to process the cars after the Queue FIFO block based on these properties.

There are several things to notice about the File Input block in this model. By default the block will read in a text file at the beginning of a simulation as this model does, or you can choose *Read* in the dialog to read in the file instantly. If you choose *Read*, you may want to deselect *Read at beginning of simulation* so that the block does not read the file in again once the simulation starts. The block will read in the text file you specify in the dialog. You must type in the complete path name. Alternately, you can leave the file name blank and Extend will prompt you for the location of the file later; this is usually easier than typing in the path name yourself.

As you can see in the dialog of the File Input block, the first line is row number 0. Because the Status block counts items starting with 1, the text file starts with a row of zeros. This will mean that the data in the first row is ignored and all relevant data will start with row number 1 (car number 1).

You must have enough data in the text file to supply numbers for the entire simulation run. If you do not have enough data, Extend will warn you that there was not enough data for the items generated, and will stop the simulation.

When you run this simulation, if the file name was left blank in the File Input block, Extend will prompt you to enter a file name. Find and open the Car wash text file (Macintosh: “Car Wash data”; Windows: “data.txt”) in the Tutorials folder. The model will then read in the text file, and use it in the simulation.

Further exploration

You can probably think of other ways to modify the models shown in this section. For instance:

- You could have the car wash or wax processing times be dynamic rather than static. For instance, you could connect an Input Random Number block (from the Generic library) to the “D” input of the Activity Delay blocks and have the time be random. Or you might connect a Program block to the “D” input and have the amount of time it takes to process a car be dependent on the time of day.
- You could assign attributes to the cars that would give the expected time to process the car, then process cars using an Activity, Delay (Attributes) block. Assigning the processing time using attributes is especially useful in a manufacturing environment where you have several types of products that require different process times.
- If you have several wash or wax bays, you could use the Prioritizer block to give preference to specific bays rather than just letting cars randomly go to any available bay.
- You can also model other aspects of the car wash, such as the limited capacity of a parking lot that holds the cars before and after the wash process. An excellent method for doing this would be to use the resource pool blocks in the Discrete Event library. These blocks are used in situations where there are capacity restraints in some portion of the model. For the car wash model, the Resource Pool block would represent the total number of parking spaces available. The Queue, Resource Pool block would hold cars waiting for a parking space and the Release Resource Pool block would release parking spaces as the cars pass through it. For more information about this advanced discrete event concept, see the documentation for the Manufacturing or BPR modules.

Extend

Beyond this manual

The concepts and examples discussed above are meant to give you an overview of some of the capabilities of the Discrete Event library. If you are building more detailed models, you should probably be using one of the auxiliary packages such as Extend+Manufacturing or Extend+BPR (Business Process Reengineering). The BPR and Manufacturing libraries include many blocks with more advanced features and capabilities. In addition, these packages contain information about such modeling concepts as merging and routing streams of items, activity-based costing, batching and unbatching, blocking/balking/renegeing, statistical analysis, methods for setting the processing time, and so forth. For more information, contact Imagine That, Inc. or your local distributor.

Extend

Chapter 5: Using Libraries

*A description of libraries 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*

As you have seen in previous chapters, the basic documents in Extend are models. You have had very little exposure regarding how to use Extend libraries in general. The first part of this chapter discusses the major libraries that come with Extend, and the second part shows you how to use and maintain libraries in general.

Extend's libraries

Extend

The basic Extend package includes libraries of blocks for modeling continuous systems. There are also additional library packages you can add which make it easier to model discrete event systems (for example, the BPR and Manufacturing modules described later in this chapter).

Generic, Discrete Event, and Plotter libraries

The two most-used libraries that come with the basic Extend package are the Generic and Plotter libraries. Parts of these libraries were shown in Chapter 1: Running a Model and Chapter 2: Building a Model; they were also discussed in detail in Chapter 4: Fundamental Continuous and Discrete Event Modeling.

As you saw in Chapter 4, the *Generic* library is used for continuous modeling and the *Discrete Event* library is used for discrete event modeling. That chapter describes these two types of modeling and guides you in choosing which library to use.

Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

The *Plotter* library holds all the common types of plotters you will use in your models. Some of these plotters are specific to continuous or discrete event models, while others can be used with either. In addition to the simple plotters you have already seen, Extend's plotters can show scatter plots, moving strip charts, histograms, and so on. All of the plotters are described in detail in “*Plotters*” on page E150.

Other libraries included with Extend

Extend also comes with several other libraries. These can be used by themselves or as a base for expansion. These other libraries include:

- The Animation library for adding custom animation to models and hierarchical block icons. It is discussed in the section titled “*Animation – customizing*” on page E197.
- Electronic engineering libraries to simulate system level design of analog, digital, signal processing, and control systems.
- The IPC library, which facilitates interprocess (client/server) communication between other applications and Extend while the simulation is running.

- A Utilities library, containing a collection of helpful blocks for counting the number of blocks in a model, fitting data to a curve, synchronizing the model to real time, timing the duration of a simulation, adding active buttons to models, and so forth.
- Sample libraries that illustrate Extend's scope and features, such as the ModL Tips (corresponds to the techniques described in Chapter 4: ModL Programming Techniques), Custom Blocks, Make Your Own - DE, and Orbit libraries.

Hierarchical blocks in libraries

As discussed in “Chapter 3: Enhancing your Model” on page E65, you can create hierarchical blocks that nest portions of your models within them. If you use a particular hierarchical block frequently, you may want to save it in a library that you create yourself. This process is described in the section titled “Saving hierarchical blocks” on page E259.

Extend

Additional library modules

In addition to the libraries included in the basic Extend package, other modules are available which can be added to Extend. For example, Imagine That, Inc. offers the following:

- The *Manufacturing module* is useful for modeling discrete manufacturing, industrial, and commercial operations. The manual and examples included in this package explore such discrete event modeling concepts as activity-based costing, merging and routing streams of items, batching and unbatching, scheduling, parallel and serial operations, blocking, and so forth.
- The *BPR module* allows you to apply systems analysis techniques to business process reengineering (BPR) and process improvement efforts. While similar in capabilities to the Manufacturing library, the BPR library uses process-flow icons and has a business-process orientation. The BPR package is useful for analyzing new processes for technology insertion, providing metrics for strategic planning, and for modeling organizational change.
- The *Industry module* from Simulation Dynamics, Inc. makes it easy to manage data and model high-volume operations. Industry contains a powerful, embedded database to manage large amounts of data. Industry also provides components for modeling high-speed/high-volume operations such as packaging lines.
- Extend Suite is a bundled package that adds discrete event modeling, professional animation and distribution fitting to Extend. The package includes the Manufacturing module, the BPR module, Proof Animation® from Wolverine Software, and Stat::Fit® distribution fitting software from Geer Mountain Software.
- Industry Suite includes everything in the Extend Suite package plus the Industry module from Simulation Dynamics, Inc.

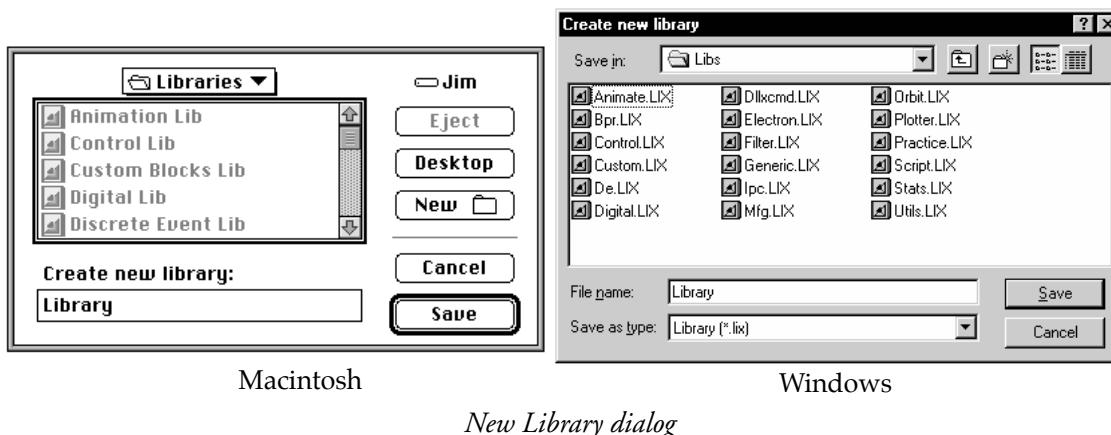
Other companies also develop “Add-On” library modules for Extend. To get the latest information about Add-On libraries for control systems, neural nets, bulk manufacturing and so forth, check the Imagine That, Inc. web page at <<http://www.imaginethatinc.com>>.

Library usage and maintenance

Extend libraries hold blocks. The purpose of a library is to be an easy-to-use repository for the definitions of the blocks that you use in your models. As discussed in previous chapters, each instance of a block in a model is linked to its library. Changes that are made to the underlying structure of a block (its icon, help, dialog structure and ModL code) cause all instances of that block to change. However, changing parameters or text in block dialogs only affects the local occurrence of that block in the model.

Making a new library

You usually do not need to start a new library unless you build your own blocks or want to save hierarchical blocks in a library. To start a new library, choose New Library from the Library menu. A dialog appears where you can select a location for the new library and give it a name:



Note If you build your own blocks or modify Extend’s blocks, you should put them in your own library. Do not put them in the libraries that come with Extend.

Also see “Saving and compiling libraries” on page E147.

Opening and closing libraries

Extend opens the libraries that are used in a model automatically when you open the model. You can also instruct Extend to automatically open up to seven libraries when Extend is started. These libraries must be listed in the Libraries tab of the Preferences command in the Edit menu (see “Preferences” on page A10). To open a library manually, choose Open Library from the Library menu.

When you open a library, Extend adds the library name to the Library menu in alphabetical order. To view the blocks in the library, click the Library menu and drag down to the name of the library. When the library name is selected, the types of all the blocks included in the library are listed to the right of the menu. Select a type to see a submenu of all the blocks of that type included in the library. Or if you would prefer to view all of the blocks in the library alphabetically in the menu rather than by type, unselect the “List blocks by type” option in the Libraries tab of the Preferences command in the Edit menu.

You can close libraries by choosing Close Library from the Library menu and selecting the library or libraries you want to close:

Extend



Close Library dialog with one library selected for closure (Macintosh)

It is unlikely that you will need to do this often since open libraries do not take up much memory and it is usually convenient to leave all your commonly-used libraries open. You can select multiple libraries to close in the Close dialog. Extend will warn you and will not let you close a library that is being used by an open model.

Changing libraries

By default, Extend automatically opens the libraries that a model uses. However, you may want to use blocks from a different library when you open a model. For example, you would do this when you have developed a new version of a library. Normally, Extend opens the library too fast for you to stop it.

In order to bypass this process, you can choose to not have Extend automatically find and open libraries. You do this by unselecting the “Automatic library search” option in the Libraries tab of the Preferences command (Edit menu). If you do this before opening a model, Extend will stop and request the location of each library used in the model. 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 the model is the block names rather than the library name.

Once all the required libraries are open, you can save the model. The next time you open the model, Extend will automatically locate and open the new libraries (as long as you remembered to reselect the “Automatic library search” option in the Preferences command before reopening the model).

Searching for libraries and blocks

Usually the process by which a model opens the libraries it uses works fine. However, in some cases, Extend may not be able to find the library or blocks used in a model. Keeping all your libraries in the Libraries folder will make this search process easier.

Library searches

In the following situations, Extend may not be able to find a library it needs:

- You have moved the model since the model was last saved.
- You have renamed, deleted, or moved the library since the model was last saved.

If this occurs, you will see the message “Searching for library xxx...”:

Extend will search in the default library directory specified in the Preferences dialog. If the desired library is not found there, Extend will search the *Libraries* subdirectory within the Extend directory. If the library is not found, Extend will search first from the model directory down and then from the application directory down. If the library is still not found, Extend will give up the search.

You can stop this search process at any time by using the keyboard:

- **Macintosh:** press Command-. (press the ⌘ key and the period key at the same time)
- **Windows:** press Control-. (press the CTRL key and the period key at the same time)

In the following situations, Extend will ask “Where is the library ‘xxx lib’?”

- You have manually stopped the library search process.
- Extend has finished the search process but cannot find the library.
- Automatic library search (in the Preferences command of the Edit menu) is unchecked.

Extend then puts up a file selection dialog for directly finding the library or substituting a different library. Your choices are:

- “Cancel” the library search operation. Extend will then try to find the blocks in the library, as described in “Block searches” below.

- Find and “Open” the library. Since Extend 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.

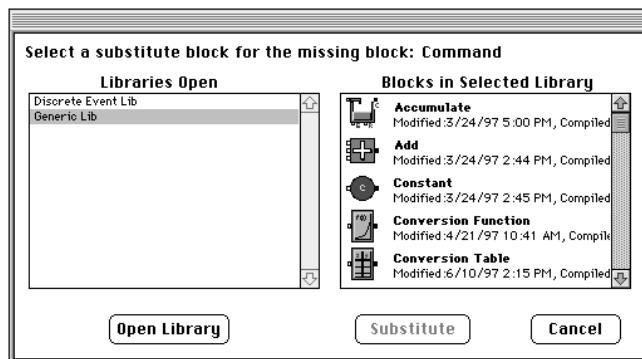
When the model file is subsequently saved, any new library name and/or location will be saved as well, so searching will not be necessary the next time the model file is loaded. (If “Automatic library search” is unchecked, Extend will still ask you to locate the libraries.)

Block searches

Once Extend finds all the libraries, it tries to find all the blocks named in the model in their respective libraries. If a block is not found (for example, if you removed the block from the library or renamed it), you are notified that Extend cannot find the block. The dialog offers two choices:

- Choose “Open” to locate and open another library where the block resides. If Extend is unable to locate the block in this library, the Substitute Block dialog will open which will allow you to “substitute” an alternative block from any open library:

Extend

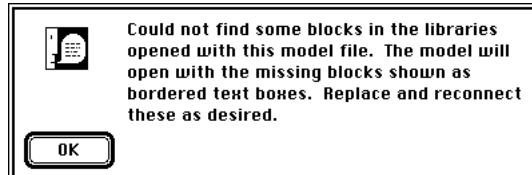


Substitute Block dialog (Macintosh)

Open libraries are listed on the left side of this dialog; the blocks in the selected open library are listed on the right. Use the dialog to locate and select the block you want to substitute. You can select a block from an open library, “Open” a library and select a block, or “Cancel” the search operation entirely.

Once you have selected a block, choose “Substitute” and Extend will try to use that block in the model. (Any substituted block must be substantially the same as the searched-for block, or Extend will report error messages.) If you cancel the search, Extend will notify you that it will put text in the place of the block, as mentioned below.

- Select “Cancel” if the block no longer exists, or if you can’t substitute another block as discussed below. In this case, Extend notifies you that it will put text in the place of the block:



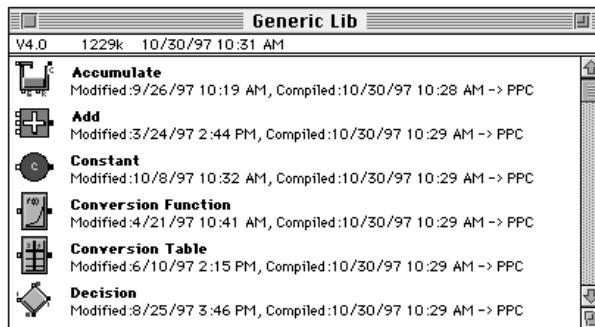
Warning message when block is not found

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.

Note If an open library contains a block with the same name as the searched-for block, Extend will attempt to use that block rather than search for the original library. If you build your own blocks, it is strongly suggested that you do not use duplicate names for blocks.

Library windows

The first choice in the blocks list for each library in the Library menu is Open Library Window. When you select this command, Extend opens a library window for that library. For example, the library window for the Generic library looks like:



Library window (Macintosh)

The top of the library window gives information about the library version, size, and date last modified. The blocks in the library are listed with pictures of their icons and their last modified date. You can scroll through the list. You can also move and resize the window like you would other windows.

Your primary method for adding blocks to a model is by using the hierarchical menus in the Library menu as described in a previous section. The library windows give you another method for adding blocks to models and help you maintain your libraries, as discussed below. If you pro-

gram blocks, you can also use the library window to access a block's structure, as described in Chapter 1: Parts of a Block.

To add a block to a model using the library window, first position the windows so that the right edge of the library window is to the left or right of the edge of the model window. Then select the block in the library window and drag it to the model. You can select many blocks at once by holding down the Shift key as you select them, then drag them all at once to the model.

To close the library window, click on the window's close box or choose the Close command from the File menu. Note that this closes the library window but not the library.

Extend

If you select "Open library window" in the Libraries tab of the Preferences command, the library window for the selected library will automatically open when Extend is launched. For more information, see "Libraries tab" on page A12.

Maintaining blocks in libraries

Extend makes it easy to handle blocks using the library windows. For example, to copy a block from one library to another, simply drag that block from its library window to the destination library window.

To make a copy of a block in the same library, select the block in the library window and choose Duplicate from the Edit menu. The Duplicate command copies the block into the current library and renames it with the block name followed by the word "Copy". This is common when you want to use the ModL code in one block as the template for a different block or when you want two blocks with the same actions but different icons.

To change the name of a block, select the block in the library window and choose Rename Block from the Define menu. Then type in 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. Extend will then ask you where the block is located and present a dialog box for searching, as discussed in "Block searches" on page E143. You can use this search procedure to locate and substitute the renamed block. When you save your model, Extend will save the new name so it will not prompt for the block when you open the model again.

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 choose Clear from the Edit menu or press the Delete or Backspace key.

Extend will not let you remove a block that is in use in an open model window. If you remove a block that is used by a model and later open that model, Extend will warn you and put a place holder in the model window.

It is a very rare occurrence, but you could get a message as you open a library which indicates that a block has been corrupted or is bad. The corrupted block will appear in the library listing 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 your backup copy of the block into this new library (this is another good reason to always back up your work!).

Organizing blocks in libraries

You can cause the blocks within a library to be listed by their functional types. If you build your own blocks, you can also decide which type of block should go into which libraries.

Extend

Listing blocks by type or alphabetically

Under the library menu, blocks can be organized within each library by *type* or alphabetically. You choose how you want the blocks listed by selecting or unselecting the “List blocks by type” option in the Libraries tab of the Preferences command (Edit menu). If you choose this option, blocks of the same type will appear within submenus under the library name.

If you program your own blocks, you can assign a type to them. See “Block types” on page P56 for a discussion on setting block types.

Organizing blocks you build yourself

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 and remember where on your hard disk your libraries were. However, this arrangement would make it difficult to maintain your blocks.

There are typically three classes of libraries: general usage, subject-specific, and model-specific. Many users only have general usage libraries; only users who create their own blocks need to have subject-specific and model-specific libraries.

- The major class of libraries is the “general usage” class, that is, blocks that might be used in a wide variety of your models. These libraries include the Generic, Discrete Event (if you have one of the additional modules, such as Manufacturing), and Plotter libraries that come with Extend.
- Subject-specific libraries are those which hold blocks that are only relevant to one subject, such as the Electronics libraries that come with Extend. Extend comes with a few sample subject-specific libraries, and it is very likely that you will build at least one such library if you create your own blocks.
- Model-specific libraries hold blocks that are only used in a single model. The Orbit library that is used in the Planet Dance example model is a good example of a model-specific library.

Your decision about where to put a particular new block should be based on your personal preferences. It is strongly suggested that you put new or modified blocks in a new library rather than putting them in one of Extend’s existing libraries. That way, when you get a new version of an Extend library, you will not lose the blocks you created.

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

Note It is strongly suggested that you do not add blocks to the libraries that come with Extend, or move blocks out of those libraries, because Imagine That, Inc. periodically updates Extend's libraries. For example, if you have added blocks to the Generic library, your work will be lost when you update. If you move blocks out of the Generic library to another library, the blocks in the new library will not be updated when the Generic library is updated.

Extend

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, Extend automatically saves the library. You do not need to save libraries; if you build a block and save it, both the block and the library are saved. Saving blocks is described in the section "Compiling and saving the block" on page P43.

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 Macintosh) you do not need to compile. To compile a block or a library, use the compile commands in the Define and Library menu:

- The Compile command in the Define menu compiles the block to machine code without saving or closing the block. This is useful for testing new code for syntax errors as you are building the block. This command is only enabled when the block's structure window is the active window.
- The Compile Open Library Windows command, a submenu in the Tools command of the Library menu, compiles all libraries whose library windows are open and saves the changes. As discussed above, unless you move a library from one operating system to another, it is unlikely that you will need to do this since libraries are automatically saved whenever you make a change. This command is only enabled when at least one library window is the active window.
- The Compile Selected Blocks command, a submenu in the Tools command of the Library menu, compiles the blocks selected in the library window and saves changes. It is unlikely that you will need to do this since libraries are automatically saved whenever you make a change, as discussed above. This command is only enabled when a library window is the active window and one or more blocks are selected in the library window.

Protecting the code of library blocks

If you build your own blocks and you do not want others to have access to your block code, you can protect your libraries by removing the ModL code of the blocks. A protected library can be used the same as any 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. For more information, see "Protecting libraries" on page P241.

Extend

Chapter 6: Input and Output

*An overview of the many ways that you can get data
into and out of your Extend models*

*“Many shall run to and fro,
and knowledge shall be increased.”
— Bible, Daniel 12:4*

Computer users have become accustomed to being able to move data with relative ease. As you might expect, Extend gives you many methods for transferring data.

Extend passes data to and from other Macintosh or Windows programs through all of the standard methods, and can pass data to and from non-Macintosh or non-Windows programs through text files. You can also connect Extend to hardware devices. The type of connections Extend can make are:

Extend

	Within Extend	With other Macintosh programs	With other Windows programs	Between the Windows and Macintosh versions of Extend
Text	Yes	Yes	Yes	Yes
Numbers	Yes	Yes	Yes	Yes
Pictures	Yes	Yes	Yes	Yes
Blocks and model sections	Yes	Yes (as a picture)	Yes (as a picture)	Yes
Hardware control	Yes	Yes	Yes	Yes

This chapter shows you how to use plotters and model Notebooks, and describes Extend's printing, copy/paste, drag and drop, import/export features, interprocess communication (IPC), ODBC database communication blocks, OLE embedded worksheet objects, and using Mailslots for distributed computing among Extend models running on different machines. Other Extend data transfer facilities, such as XCMDS, DLLs, and serial and device driver functions, are mentioned here but are more fully discussed in the Extend Programmer's Reference.

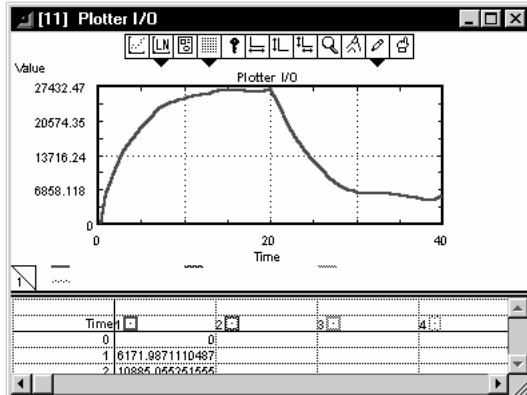
Plotters

Most of Extend's plotters have many features in common. The "Lake Pollution" model in Chapter 2: Building a Model, shows some details of how to use plotters. This section describes Extend's plotters in more detail and shows how to make plots appear the way you want. Plotters are located in the Plotter library.

Depending on their type, plotters allow you to plot 1, 2, 3, or 4 traces (plotted lines) at a time. If you build your own plotter using Extend's ModL language, you can plot up to 100 traces on a single plotter. Your choice of plotter type depends on the type of model you build (continuous or discrete event) and how you want information plotted (such as batch runs, histogram, and so on). Plotter types are described later in this chapter.

You can have more than one plotter in a model, and you can place plotters at any point in the model. Usually plotters open automatically when the simulation is run. If the plotter is set to not

open (as discussed later in this chapter), you can open it by double-clicking on its icon. When the plotter's icon is open, you see the plot for the most recent run, such as:



Extend

Plotter window (Windows)

Plot and data panes

The plotter window has two *panes*. The upper pane shows the plot (one or more traces) and the lower pane shows the data for that plot in a table. Most of the time, you are interested in seeing the plot; however, the data pane is also quite valuable. The two panes are separated by a *split bar*. Initially, about two thirds of the window is devoted to the plot and one third to the data. You can change this by moving the cursor over the split bar until it changes to \diamond , and dragging up or down.

Plot pane

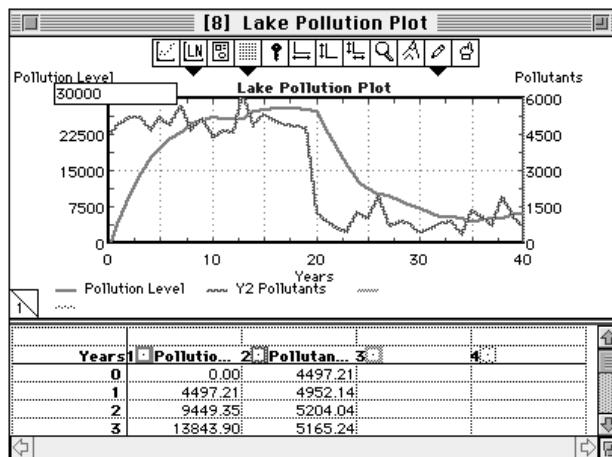
The plot pane has four pages. If the plotter was open during the simulation run, the four most recent runs will automatically be saved on those pages. If the plotter was not opened during the simulation run, only the current plot image is saved in the plot pane and you will have to use the Push Plot tool (described below) to save additional images. Click on the number at the lower left of the plot pane to see each page. If you had the Key tool (described below) on when the simulation ran, the key will be shown on the pages.

As you look at the plot, you may want to see the exact numbers that generated a point. You can scroll in the data pane, but Extend gives you a faster method. Simply put the cursor over the data item you want; the top row in the data pane contains the value that matches that point. In scatter plots, this shows the X and Y values of that point.

You change the plot axis labels and limits directly in the plot pane. To do this, select the item to be changed and type the new text. To finish, press Enter (in the numeric keypad) or click somewhere else in the window. To change many of the labels and limits, press Tab after entering each new value. If you use the Tab method to change items, the plot will redraw only after you press Enter

or click in the window. For example, to change the maximum Y axis value, you would click on that number:

Extend



Changing maximum Y axis value (Macintosh)

Data pane

You can scroll through the data pane with the horizontal and vertical scroll bars. You can select cells, rows, and columns in the data pane in the same way you select them in any data table. As with any data table, you can resize the columns in the data pane. You do this by placing the cursor directly over the column divide, and dragging the divide to the desired location while holding down the mouse button. You can also tab through the data or use the arrow keys. If you select a section of data, tabbing will move you just through the selection. Notice that, only the data for page 1 is displayed in the table.

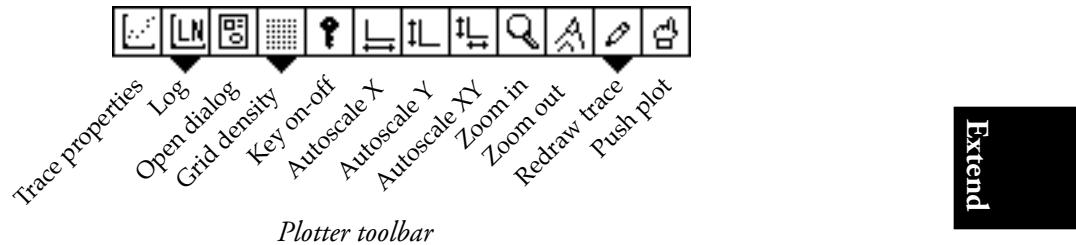
If you change the data in the data pane, those changes are reflected in the plot. You can change numbers, paste in rows, and so on. Use this capability to view how various data would be plotted, or to plot a reference line.

The labels on the columns in the data pane are changed by using the Trace properties tool, described later in this chapter. The color, pattern, and symbol for each trace is displayed to the left of its label.

Plotter tools

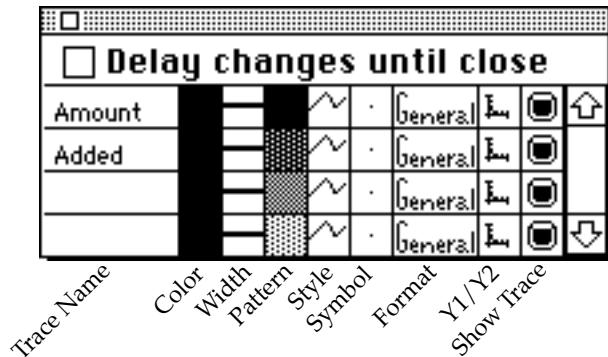
Note the tools at the top of the plot pane. These are used to change the traces and views for the plots, change the labels and formatting for the data being plotted, and to control how and when

the data is displayed. Tools with arrows beneath them are pop-up menus with multiple choices. Click on a tool to activate it. The toolbar in plotter windows looks like:



Trace properties tool

This tool lets you change many aspects of the traces and data. It brings up a small window that looks like:



Trace window (Macintosh)

If you check the “Delay changes until close” checkbox, changes you make will occur in the plot only after you close the Trace properties tool. This is useful when you are making a lot of changes, and don’t want the traces to be redrawn after each change.

Below the checkbox are 9 columns of choices:

- The *Trace Name* text at the left describes the traces and is used to label the columns in the data pane. In a discrete event plotter, each trace has two sets of labels: the name of the trace and Time. In a continuous plotter, each trace will only have one label, as shown above. To change a label, click on the text in the Trace Name area and type the text you want.
- The *Color*, *Width*, and *Pattern* choices describe how the trace looks. To change one of these items, click on the box and choose from the list that you see. You can select from 7 standard colors or choose a custom color. There are also 5 line widths and 4 patterns available.

- The *Style* choice affects the manner in which the trace is drawn. The choices are:



Trace styles

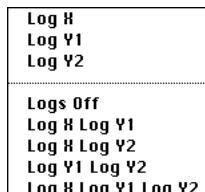
Extend

The top style indicates that the data points are connected by diagonal lines (interpolated), while the middle style indicates that they are connected by horizontal and vertical lines (stepped). The bottom style indicates that you want the points plotted with no lines between them.

- There are also many *Symbol* types, such as dots, squares, and circles, as well as a trace numbering choice (marked "#"). If you choose a non-numeric symbol, it will be drawn at each point on the trace. If you choose the # symbol, the number for that trace will be spaced evenly along the trace. Except for numbers, each symbol is drawn using the trace color, width, and thickness settings; numbers are drawn in black.
- The *Format* option sets the number format that is used for the data in the data pane, including Time for discrete event models. The four choices are General, x.xx (decimal), xxx (integer), and x.xex (scientific notation).
- The *Y1/Y2* choice tells which axis to plot the numbers against. The default choice, plots them on the Y1 (left) axis. Clicking this changes it to , the Y2 (right) axis.
- The *Show Trace* choice tells whether to display the trace at all. If you click on this choice, it turns to a closed eye and the trace is not drawn. This is useful if you have many traces and you temporarily want to hide one to make the chart clearer or if one trace is on top of another.

Log tool

The Log tool lets you choose whether to make one or both axes use a logarithmic scale. Click and hold down the mouse on the tool to display the menu. The choices that you can apply are:

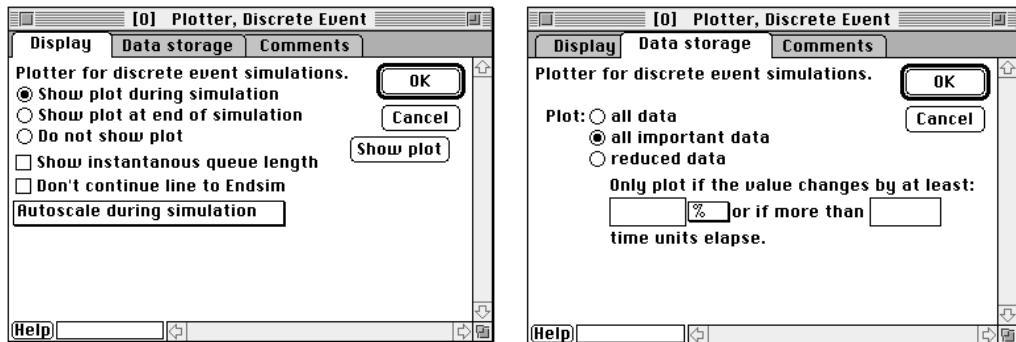


Log tool

Extend

Open Dialog tool 

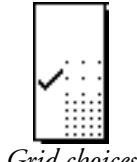
Click the *Open Dialog* tool to show the plotter's dialog. You use this tool to make changes to the way the plotter operates. For example, you can choose to not have the plot show during simulation runs, such as when you are running animation, or you can choose to plot every nth point, instead of plotting every point. The Plotter, Discrete Event, for instance, has one tab that controls how and when data is displayed and another tab that allows you to specify the granularity of the plotted data. The choices in the plotter dialogs are described in their Help and in “Plotter dialogs” on page E157.



Display and Data Storage tabs of Plotter, Discrete Event block (Macintosh)

Grid density tool 

The Grid Density tool lets you specify the type of grid behind the plot. Click and hold down the mouse on the tool to display the menu. The three choices are:



Grid choices

The top choice indicates no grid, the middle choice a light grid, and the bottom choice a dense grid.

Key on-off tool 

Clicking this tool turns on and off the key at the bottom of the plot pane. The labels, colors, patterns, and symbols that the key shows come from the entries you make in the Trace properties tool's window. The key is handy if you are not viewing the data table at the bottom of the plotter window or if you copy the plot to another document. If the key is on when you run each simulation, the key will show on each plot page.

Autoscale tools

The three autoscale tools are used to manually scale the axes to fit the data after the simulation has run. The first tool, , scales the X axis to fit all values that were measured in the simulation without changing the Y axis. The second tool, , scales the Y axes to fit all values that were measured in the simulation without changing the X axis. The third tool, , scales all axes at once. This is especially useful if a run of the simulation goes off one side of the plot. You can manually autoscale the axes while the simulation runs.

Note that you can also choose to have the data automatically scaled while the simulation runs. You do this using the “autoscale” choices in the Display tab of the Dialog Open tool.

Zoom in and Zoom out tools

There are many times you want to see a particular part of your plot in more detail, or want to zoom out and look at a larger area of the plot. The  tool magnifies an area of the plot. After you click on this icon, the cursor becomes . Drag the cross-hairs over an area of the plot and release the mouse to zoom in to match the described rectangle. The  tool zooms out by changing the axes by a factor of 2 in each direction. After using the Zoom in or Zoom out tools, you can use the Autoscale tool or the Undo command from the Edit menu to reset your axes.

Redraw trace tool

The Redraw trace tool lets you change your data on the plot and see the results in the plotter’s data table. Click and hold down the mouse on the tool to display the menu. Choose one of the four traces that you want to change. The cursor becomes a pencil, . You use the pencil to redraw the data that was plotted. After you are done drawing this, you can see the values for the drawn data by looking in the data table. The Redraw trace tool is only available in continuous nonscatter plotters, like the Plotter I/O, Conversion Function, and Input Function blocks.

Push plot tool

Each time you run a simulation with the plotter open, the previous run is automatically saved onto page 2 of the plot. Changes made to the plot after the simulation run (such as autoscale, magnify, and turning the key on) are not automatically saved. Also, if the plot was closed during the simulation run, the plot image from the last run will not be saved to page 2. After you change the view of your plot or run a simulation with the plotter closed, you may want to save a copy of that plot in one of the plotter’s plot pages. To do that, click the Push Plot tool. For example, assume that you are looking at a magnified portion of the plot and you want to save that magnified view before you run your simulation again. You would use the Zoom In tool, click the Push Plot tool, choose Undo Zoom in from the Edit menu to restore your original axis values, and run the simulation again.

When you use this tool, the current plot image is saved onto page 2. The previously saved images are each pushed up one page so that the image that was on page 4 is discarded.

Plotter dialogs

Extend comes with many types of plotters, as seen in “Types of plotters”, below. Each plotter has a dialog that you can access by clicking on the  (Dialog Open) tool at the top of the plotter. Most of the plotter dialogs have only a few choices. Some choices that some of the plotters have in common are:

Choice	Description	Extend
Show plot during simulation	Determines if the plotter is automatically opened up during the simulation. For example, you probably do not want the plotter to open if you are running the model with animation turned on.	
Show plot at end of simulation	Display plot only when simulation has been completed.	
Do not show plot	Plot remains hidden until the plot icon is double-clicked.	
Show instantaneous queue length	This plots additional data to show when a queue changes length in zero time. For example, when an item enters and leaves the queue during one event.	
Don't continue line to Endsim	This stops drawing the plot trace at the last event, not at the End Time of the model run.	
Show plot	Brings the plot window to the front if it is already open, and opens it if it is closed.	
Autoscaling	Automatically resizes the y-axis of the plot based on the maximum and minimum values observed. You can have the plot automatically scale during the simulation or at the end of the run.	
Data storage	Lets you plot all data, all important data, or reduced data. This is useful if you have plotters that store too much data, as some of it may be redundant. Reduced Data samples from the incoming stream, conserving memory.	
Plot every nth point	Lets you specify that the plotter should only draw the nth point of the data it receives. This is useful if there are many data points and the points are all very close together, or to make the plot draw faster.	

Types of plotters

Plotters are located in the Plotter library. As indicated in the following table, each plotter has a distinct purpose. Some can only be used in continuous models (“C”), some only in discrete event models (“DE”), others in both (“Both”).

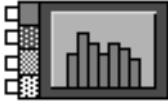
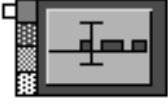
Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

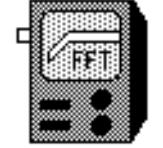
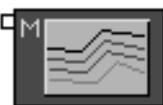
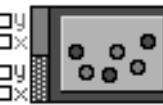
Discrete Event plotters are used in discrete event models. However, since they plot values not items, you must connect a value output (as opposed to an item output) to the Discrete Event plot-

ter. This tells the plotter what information about the items or about the status of the model you want to plot.

The MultiSim and Error Bar plotters are specifically designed to be used when you run multiple simulations for Monte Carlo or sensitivity analysis. You choose the number of times you want the simulation to run in the Simulation Setup command of the Run Menu.

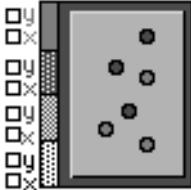
The plotters are:

Plotter	Description	Model
 <i>Histogram</i>	Creates a histogram of all the values it receives. Each bin counts the number of data values that fall within its range, or the amount of time that the incoming value was in that bin (if the Time Weighted option is used). The number of bins and the overall minimum and maximum range values are specified in the plotter's dialog. The maximum and minimum can be the entire range received or specified numbers. You can specify in the dialog whether to plot the data from each step or, if you run multiple simulations, to plot the final values of each run. The dialog also displays the number of points that fall within the specified range.	Both
 <i>Plotter, DE Error Bars</i>	Shows the mean and standard deviation of a value. This plotter is used when you run multiple discrete event simulations, such as for Monte Carlo modeling or for sensitivity analysis. The time line is divided into a number of equal intervals specified in the dialog. You can also choose to use time-weighted statistics. The plotter calculates the average mean of the value over each interval and from this determines the mean and standard deviation over all of the runs.	DE
 <i>Plotter, DE Multi-Sim</i>	Accumulates the values from up to four runs of a discrete event simulation on a single plot pane and table. In the dialog, you can choose to only plot values when they change or to plot all values. You can also specify by number which trace to show the current run on whether to automatically increment the trace number.	DE

Plotter	Description	Model
 <i>Plotter, Discrete Event</i>	Gives plots and tables of data for up to four value inputs in discrete event models. Both the value and the time the value was recorded are shown in the data table for each input. In the dialog you can specify whether to plot values only when they change or to plot all values. Use the “Show instantaneous length” option if you attach an input to the <i>L</i> connector of a queue block from the Discrete Event library and you want it to report on items that arrive and depart on the same time step (these are items that stay in the queue for zero time). You can also choose whether to autoscale the plot.	DE
 <i>Plotter, Error Bars</i>	Similar to the Plotter DE Error Bars except it is only used in continuous models. As opposed to the Plotter DE Error Bars, this plotter calculates the mean and standard deviations for the value at the exact point or step in time rather than its average over the time interval.	C
 <i>Plotter, FFT</i>	A specialized plotter used by electronic engineers, it plots both the data that is input and the FFT (Fast Fourier Transform) of the data. When you double-click the FFT icon, it shows a dialog where you can specify the number of FFT points and choose an FFT window.	C
 <i>Plotter, I/O</i>	Gives plots and tables of data for up to four time-associated inputs. In the dialog, you can specify that only every “nth” point is plotted and you can choose whether or not to autoscale the plot during the simulation. The output connectors allow the data generated in one simulation to be used as an input to another simulation. To do this, run the simulation. Then copy the plotter into the new model and connect from its output connectors.	C
 <i>Plotter, MultiSim</i>	Accumulates the results of up to four runs of a continuous simulation on a single plot pane and table. You can specify by number which trace to show the current run on and whether to automatically increment the trace number.	C
 <i>Plotter, Scatter</i>	Shows two sets of data plotted as x,y value pairs. You must connect both the x and y inputs of at least one pair in order to plot data.	Both

Extend

Extend

Plotter	Description	Model
	The same as the Plotter, Scatter except that you can plot up to four sets of data. You must connect both the x and y inputs of at least one pair in order to plot data.	Both
	Shows a moving strip chart of a specified number of data points plotted over time. The strip chart moves left as the data appears on the right side of the plotter. In the dialog, you can specify the numbers of points to show on the chart. Because it uses less memory than the Plotter, I/O, this is especially useful if you are running a long simulation and you only need to monitor the current conditions.	C
	Shows two sets of data plotted as x,y value pairs for a specific number of points. You can specify how many points to show at a time (the worm width) in the dialog. This is like the Plotter Scatter except that points are deleted. You can use this if you are generating a great deal of data but do not need to see all of it. You must connect both the x and y inputs of at least one pair in order to plot data.	Both

Copying plotted information

You can copy the information from the plotter to the Clipboard. If the plotter is the front-most window, simply choose Copy Plot from the Edit menu to copy the picture of the plot into the Clipboard. To copy the data, you must select it, then choose the Copy command.

Clearing plotted information

If the plotter is the active window, you can clear all the data associated with the current plotter by selecting the Clear All Plots option from the Edit menu. This is useful, if you want to reduce the size of your model for distribution or archiving.

Notebooks

A Notebook is a window you customize in order to organize the data in a model. Each model has its own Notebook which can contain clones of dialog items and plots, text, pictures, and drawing items. You can use a Notebook as a “front-end” to the model, to control model parameters, report simulation results, and document your model.

The most common 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, you can use it as a central

display for all of the important values in all the dialogs. Another common use is as a control panel. You clone all the dialog items that you might want to change while the simulation is running in the Notebook so that you do not need to open any other dialogs in order to make a change.

You can add text, drawings, and pictures to the Notebook that are not in the model itself. For example, you might add lines in the Notebook to separate functional sections. You might also add text to explain more fully the numbers in the items. Note that adding many drawings and pictures can reduce the amount of Extend's free memory.

To create a Notebook or open an existing Notebook, choose Open Notebook from the Model menu. You can tell whether or not the Notebook already has any items in it. If the Notebook has anything in it, the Open Notebook command from the Model menu has a check mark next to it; if there is nothing in the Notebook, the check mark does not appear. For example:

Extend

✓Open Notebook

Menu command showing Notebook with items

A Notebook can be many pages long. Page breaks and page numbers are always shown. If a model file is saved with the Notebook open, the Notebook will automatically open the next time the file is opened.

To clone an item into the Notebook, choose the Clone tool, , and drag the item from the dialog or plotter. For multiple items, drag a frame or hold down the Shift key as you select more items. Clones can be moved and resized after they are placed. You can treat a model's Notebook just like you treat a block's dialog. When you change a cloned value in a Notebook such as by selecting a button or entering a number, Extend changes that value in the original dialog as well. If you have cloned items in a Notebook and you want to find the dialog from which they came, choose the Clone tool and double-click on the item. This finds the associated dialog and opens it. Cloning dialog items is described in "Cloning dialog items onto the model" on page E76.

Note Cloning items into a Notebook on a small screen may be difficult because it is hard to have the model, an open dialog, and the Notebook showing at the same time. If you have just an edge of the Notebook showing behind a dialog, however, you can easily clone items into it. Simply drag the items to the Notebook edge and let go. Later, you can rearrange the Notebook however you please.

You can copy the contents of a Notebook as a picture which can be pasted into a word processing document. As is true for copying in general, the selection tool you use determines what will be copied (block/text, drawing layer, clone layer, etc.). For example, to select all types of items in the Notebook, choose the All layers tool () and select the portions of the Notebook that you want to copy. For multiple items, drag a frame or hold down the Shift key as you select more items, or use the Select All command from the Edit menu to choose everything. Then choose the Copy To Picture command from the Edit menu to make a picture.

Printing

One of the most common ways of exporting data is to print it and show the printed page to others. Extend gives you a great deal of control over what appears on the printed page when you choose the Print command.

Selecting what to print

Before printing, you should select what it is you want to print. Many programs can only print the active document (which, in Extend's case, would be just the model). Extend can print much more than just the model, however. The types of items that Extend can print are:

Item	Printed
Model worksheet	A picture of the worksheet as it appears on the screen. You can optionally also print the contents of the dialogs and plotters (see below).
Dialog	A picture of the dialog. If the dialog has a data table, you can select whether or not to print all of the data in the table. You can also choose to print just the top tab or all tabs.
Plotter	The plot and, optionally, the data table.
Notebook	The contents of the Notebook as it appears on the screen.
Help text	The text from the current help choice.
Text file	The text in the file.
Library window	A picture of the library window.
Structure window	All the panes from the structure window and the dialog editor window of a block (see "Structure window" on page P6).

The Print command

Extend decides which type of item to print based on what type of window is active when you choose the Print command. For instance, if a plotter is active when you choose Print, the command in the File menu reads "Print Plotter".

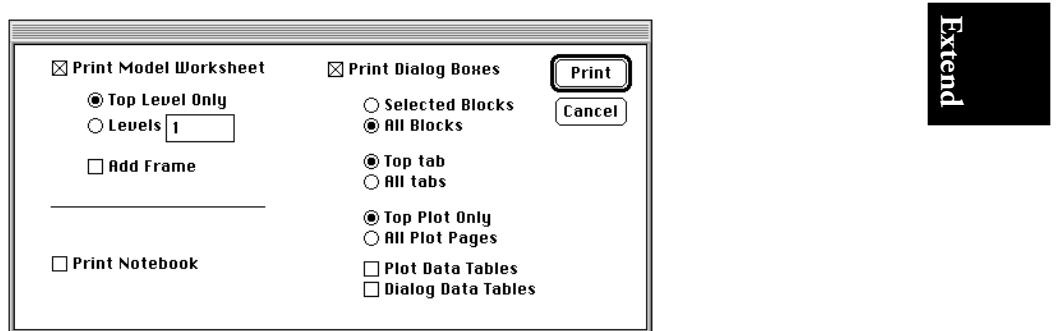
To print, choose the Print command from the File menu. If a Notebook or a library window is active, the standard Print dialog appears. If another window (such as the model worksheet) is active, two dialogs open in sequence: an Extend Print dialog and then the standard Print dialog.

- Extend's Print dialog is where you select what will be printed. As seen below, the type of dialog Extend shows for printing depends on the type of window active when the Print command is given.
- After you have selected what to print, the standard Print dialog allows you to specify particulars about the print job. For example, you can set the number of copies to be printed and (for model

worksheets and Notebooks) the range of pages. The configuration of the standard Print dialog differs depending on the operating system and the type of printer you are using.

Worksheet, dialog, or plotter active

When you give the Print command with the worksheet, dialog, or plotter active, you see the following Extend dialog before you see the standard Print dialog:



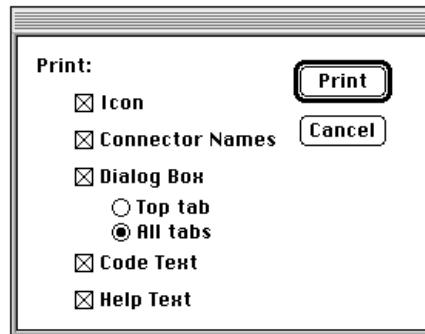
Extend's Print dialog

Depending on what type of item you are printing, some of the options might be dimmed since they would make no sense. The options are:

Choice	Description
Model Worksheet	Prints the contents of the worksheet
Top Level Only	Only shows the worksheet's top level (not the contents of hierarchical blocks)
Levels	Specifies how many levels of hierarchical blocks to print
Add Frame	Puts a border around the printed worksheet
Notebook	Prints the contents of the Notebook
Dialog Boxes	Prints the contents of the dialogs
Selected Blocks	Specifies that only the dialogs of selected blocks will be printed
All Blocks	Specifies that the dialogs of all blocks in the model will be printed.
Top Tab	Prints only the top tab of the dialogs
All Tabs	Prints all tabs in the dialogs
Top Plot Only	Prints only the visible plot
All Plot Pages	Prints all four of the plot pages
Plot Data Tables	Also prints the data tables in the plotters
Dialog Data Tables	Also prints the complete data tables in the dialogs

Structure or dialog window active

When a block's structure or dialog window is the active window, choosing the Print command gives you the following dialog before the standard Print dialog:



Print dialog for structure and dialog windows

Print and Page Setup hints

If you are printing the model worksheet or the Notebook, you can choose to print only some of the pages using the standard Print dialog. The Show Page Breaks command from the File menu allows you to see where the page breaks appear in your model. It also shows the page numbers in the upper left corner of the displayed pages to help you choose which pages you want to print.

The Show Block Numbers command in the Model menu causes block numbers to appear on their icons. Block numbers are unique identifiers for each block and appear 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.

Depending on the type of printer you have, the standard Print dialog may give you choices for grayscale/color printing. Color may come out looking better than black-and-white but takes much longer to print.

The Page (Macintosh) and Print (Windows) Setup dialogs have many settings that affect printing. For example, if you want to save paper and only want a thumbnail sketch of a large model, you can use the Setup dialog to specify a reduction of 50% of the image. You can also switch between portrait and landscape printing if your printer supports both. Extend's Show Page Breaks command comes in especially handy here. If your printer supports Graphics Smoothing and Text Smoothing, turning these off makes Extend's printed images easier to read.

Extend will automatically print default headers and footers on each page. The Preferences dialog contains a preference to enable or disable the printing of the default headers and footers (see "Preferences" on page A10).

Cut, Copy, and Paste with the Clipboard

The Clipboard is useful for passing information within Extend or from Extend to other applications. The information that is passed will be either in the form of ASCII text or graphics (Macintosh: PICT graphics; Windows: a Windows metafile). Extend uses the Cut, Copy, and Paste commands just like other programs. You can see the contents of the Clipboard with the Show Clipboard command in the Edit menu.

Note Before you choose the Copy command, it is important to select the tool from the toolbar which will allow you to copy the items you want. For example, if you choose the All layers tool and frame-select a section of a model, all items in the frame (blocks, drawing objects, clones, etc.) will be selected and available for copying. However, if you use the Block/Text tool, only blocks and text will be selected.

Extend

Copying within Extend

If you copy and paste blocks within Extend, the Clipboard also holds block parameters and connections. This allows you to paste portions of your models, including the variables in the dialogs of the blocks, to another section of your model, to other models, or to the Scrapbook (Macintosh only). (For details concerning copying large portions of models, see “Appending models” on page E167.)

You can copy drawing objects and stylized text from one part of a model to another, or from one model to another. You can also copy data from parameter entry fields and data tables and paste them into other entry fields or data tables.

Note When you copy data tables within Extend, you should not choose “Data table title copying” in the Preferences command in the Edit menu. If you do, the titles will be added to the data, causing the original data to be displaced.

Copying from Extend to other applications

Data from text entry fields and data tables can be copied from Extend and pasted into other applications such as word processors and spreadsheets. When you copy data, Extend puts the data into the Clipboard as unformatted text. You can copy data from dialogs by selecting the data and using the Copy command.

Similarly, you can copy data from a plotter’s table by selecting the cells and rows of data and choosing the Copy command. To copy the row and column titles in data tables, select “Data table title copying” in the Preferences command. Then click on one or more rows or columns and give the Copy command. Extend prompts you to select the titles you want copied.

Plot panes, Notebooks, parts of models, and dialogs are also available as pictures to be copied into other programs:

- You can copy a plot pane to other programs for use in reports or presentations. To do this, simply click in the plot pane, then choose Copy Plot from the Edit menu.

- Notebooks and models contain various types of items such as drawing objects, cloned dialog items, and so forth. As mentioned earlier, before you choose the Copy command, it is important to select the tool from the toolbar which will allow you to copy the items you want. For instance, to copy an entire Notebook, choose the All layers tool from the toolbar. Then frame-select the entire Notebook (or choose the Select All command from the Edit menu) 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 Copy to Picture from the Edit menu.

When it is not possible to copy parts of a model directly in Extend (such as the entire plotter window), use a screen capture program or:

- **Macintosh:** use the Command-Shift-3 key combination to save the selection as a picture. Or use a capture program to capture specific parts of your screen.
- **Windows:** use the Control-Print-Screen key combination to save the entire screen as a bitmap. Or, use a capture program to capture specific parts of your screen.

Copying from other applications to Extend

You can copy data from another application into an Extend parameter field, data table, or text file. If you have a lot of data, it is usually better to import it into a data table, as discussed in “Importing and exporting with text files” on page E167.

You can copy pictures into Extend in color or black-and-white to use on your model windows, Notebooks, hierarchical windows, or in the icon in a block's structure window. You can create the picture with:

- **Macintosh:** use a painting or drawing program and copy the picture to the Clipboard. Then paste it into an Extend window with the Paste Picture command from the Edit menu.
- **Windows:** use a painting or drawing program and copy the picture to the Clipboard. Then paste it into an Extend window with the Paste Picture command from the Edit menu.

In the model and Notebook window Extend will paste the picture wherever you last clicked on the window. A picture pasted from another program into Extend becomes a drawing object which can be resized and repositioned using the Draw layer tool.

Pictures that are copied, like objects created with Extend's drawing tools, always go behind Extend blocks and text. To delete a picture, select it with the Drawing Selection tool and choose Clear from the Edit menu or press the Backspace key.

Note Pictures and drawing objects (rectangle, rounded rectangle, and oval) 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.

Drag and drop editing

As discussed in “Drag and drop editing” on page E71, drag and drop is the easiest way to move a selection of text a short distance or between documents. Drag and drop can also be used to create *text clippings* (Macintosh only). By selecting a piece of text and dragging it from Extend onto the Finder (i.e. the desktop or a folder), you can create a text clipping that can be dragged into any other document of any application that supports drag and drop text. To copy a text clipping into an Extend document, simply click on the text clipping and drag it into the document. When you drag the selection, an insertion point will appear at the end of the cursor. Drag the insertion point to the desired location and release the mouse button.

Extend

Appending models

If you want to add portions of models together, you can copy the blocks from one model and paste them into another as discussed above. You can also use Extend’s Save Selected and Append Model commands which are faster and use less memory.

You may want to save a portion of a model as a template, to be later combined with other model portions to form a complete model. To do this, select the portion of the model that you want, then choose Save Selected from the File menu. Give the name of the new template. The selection tool you use determines what is saved (blocks only, graphics, and so on).

To append model segments or templates together, first open a model window and click in the window where you want the top left edge of the appended model to go. Then choose Append Model from the File menu. A dialog appears that allows you to select the model to append from. When you choose the Open button, the second model will be appended onto the first at the insertion point. Since the second model is selected, you can choose the Block/Text tool to reposition it as desired.

Importing and exporting with text files

A text file is a file of unformatted data. Text files contain written text with the styles removed and/or numerical data separated by some delimiter or separator (such as tabs or spaces). There are many situations where you want to read and write text files (also known as *ASCII* files). For example:

- You may want to share data with a database or spreadsheet program such as FileMaker or Excel. Almost every program can read and save text files.
- Most files transmitted from minicomputers and mainframes are text files.

Using text files is a lot faster and more convenient than copying when you are working with large amounts of data. You can create text files in another application such as a spreadsheet, database, or word processing program, then read those text files into Extend. The general method for creating a text file in other programs is to choose Save As from the File menu and specify the file format to be Text. You can also create text files in Extend (as discussed below), then read them into the other programs for presentation or further analysis.

Extend

How Extend uses text files

Extend can read and write standard text files in many ways.

- The easiest method for handling text files in blocks is by using the File Input and File Output blocks in the Input/Output submenu of the Generic library. These are discussed in detail in Chapter 4: Fundamental Continuous and Discrete Event Modeling and “Appendix D: Generic Library Blocks” on page A45.
- You can also use the Import Data and Export Data commands in the File menu to read and write text files in dialog and plotter data tables. *You must first select the data table columns in order to import or export the data.* These commands are described in “Import Data” and “Export Data” on page A5.
- Reporting and Tracing also generate text files if you use those features in your models. Reporting is described later in this chapter and Tracing is discussed in detail in “Model tracing” on page E218.
- If you use Extend’s Sensitivity Analysis feature, you can create a text file of data in Extend or in another application to use as the basis for the analysis. Sensitivity Analysis is discussed in “Sensitivity analysis” on page E83.
- If you are creating your own blocks with the ModL language, you can use the file I/O functions described in “I/O functions” on page P98, to read and write text files.

Working with text files in Extend

You can read, write, edit, and print text files from within Extend. This allows you to look at report files, modify data input files, and so on, without having to run another program.

For example, you can create a new text file and enter values to be used as input for sensitivity analysis or by the File Input block. You might also want to open a text file created by Extend’s reporting facility or the File Output block.

Text files are created, opened, and closed from the File menu:

- To create a new text file, choose the New Text File command in the File menu.
- To open an existing text file, choose the Open command in the File menu or click on the Open tool in the tool bar.

- To save a changed text file to disk, choose the Save Text File As command in the File menu.
- To close a text file, use the Close command from the File menu or click on the close box in the upper left corner of its window.

When a text file is the active window, many of the commands of the Edit and Define menus are available to help editing. For example, you can choose the Cut command to remove data from the text file and choose the Find command to search for particular text in the file.

Extend

You can also use the commands in the Text menu to *temporarily* improve the readability of a text file, for example, by increasing the size of the text. However, Text menu changes are discarded when you close the text file. To permanently change the font or font size used when viewing text files, use the “Text file font” option in the Model tab of the Preferences command.

When you create text files that will be used as input to Extend blocks or other programs, you should remember how the blocks or programs want the data presented. For example, some Extend blocks can take data columns separated with Tab characters, with spaces, or with other characters such as commas. On the other hand, the sensitivity analysis will only read files where the columns are delimited by Tab characters. In general, it is best to use Tab characters. Carriage return characters are always used to separate rows.

Also, remember that programs that use the Tab character to delimit columns expect only one Tab character between columns. Thus, don't put in extra Tab characters in order to make the columns line up visually; if you do, the program is sure to get confused.

Data sharing using Global Array blocks

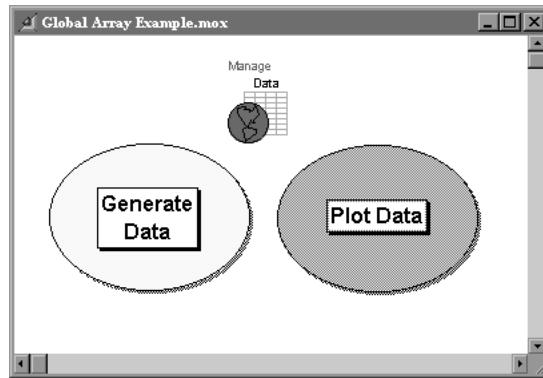
Global arrays are two-dimensional arrays that are accessible by any block in an Extend model. The global array blocks consist of the Global Array and Global Array Manager in the Generic library. These blocks allow block level access to named global arrays without having to use the ModL functions associated with Global Arrays.

In general there are several uses for these 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, such as a database.
- Accessing existing databases using ODBC, and sharing that data throughout the model.

Extend

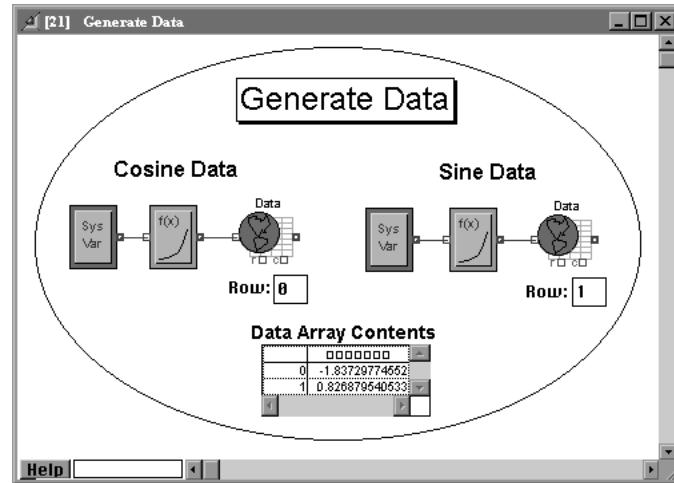
The *Global Array Example* demonstrates how these blocks may be used. At the top level, this simple model consists of a Global Array Manager block and two H-blocks called Generate Data and Plot Data.



Global Array Example model, top level

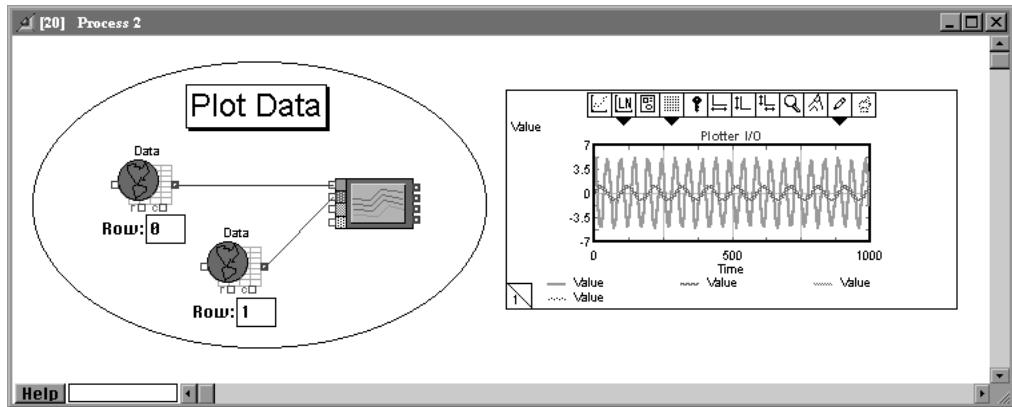
The Manager creates a 2 row by 1 column real global array called Data that is used to communicate information between the two H-blocks.

Inside the Generate Data block, Conversion Function blocks calculate values for the sine and cosine functions over time. These values are written to the Data array using the Global Array blocks. The row is determined in the dialog of the Global Array block.



Generate Data hierarchical block, opened

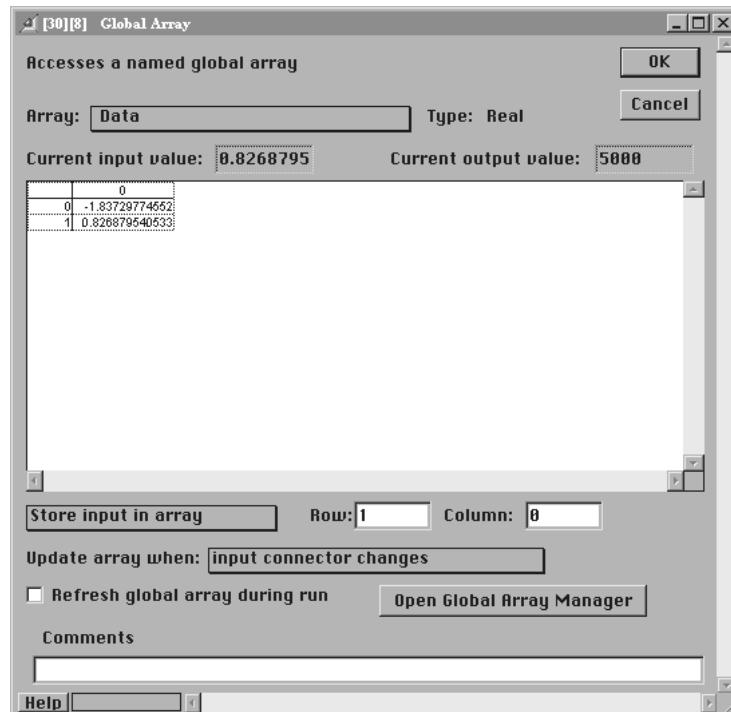
Simultaneously, the Plotter inside the Plot Data H-block is reading values from the same Data array to plot the sine and cosine functions over time.



Extend

Plot Data hierarchical block, opened

Below is the dialog of one of the Global Array blocks, showing the current state of the Data array that was defined in the Global Array Manager block.

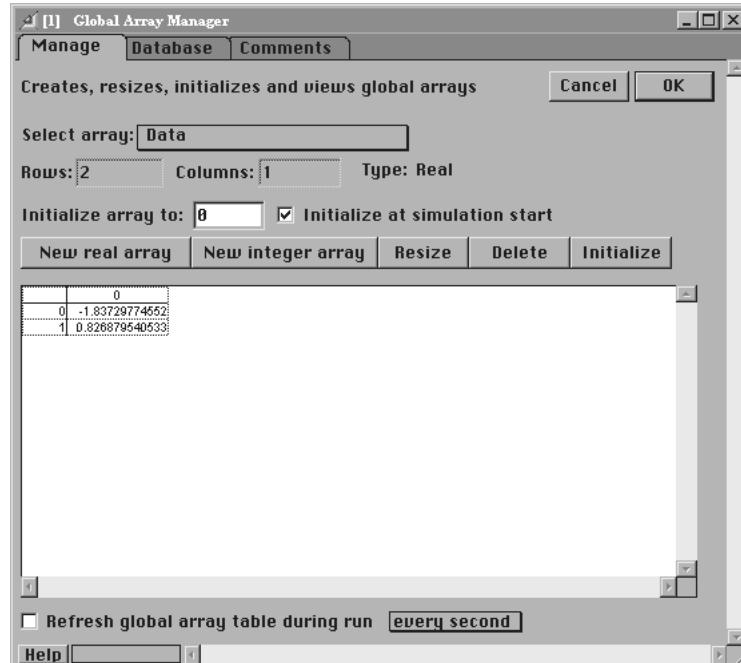


Global Array block showing array data

Extend

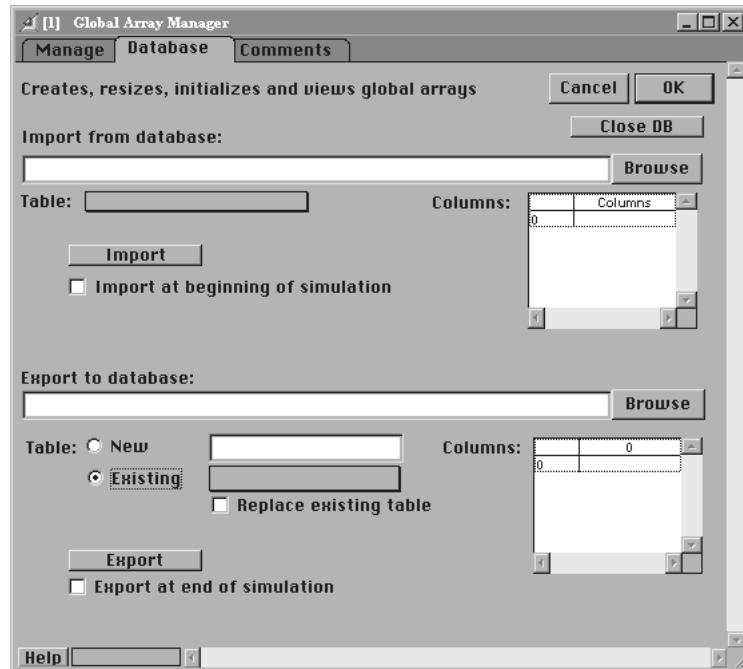
Any input data to the block can be stored in a particular row and column address. The address is also available for change via the R and C inputs. For more information on these blocks, consult their on-line help.

Global Array Manager



Global Array Manager block showing creation of data array

This block is used to create, resize, delete, and initialize a new Global Array. There are options to import and export data to and from a database that is specified on the Database tab:



Extend

Global Array Manager block showing ODBC connectivity

Once data is imported, the entire model can access the data via Global Array blocks.

Note A Global Array Manager block must be in the model to initially create the global array and initialize it at the beginning of the simulation run.

Database connectivity with ODBC

In this example, we will create an Excel database and import the data into an Extend model using Open Database Connectivity (ODBC). This is done with the ODBC functionality in the Generic Global Array blocks. For information on how to use the ODBC ModL functions in custom ModL coded blocks, see “ODBC” in the Extend Programmer’s Reference.

Building an Excel Database

Start by creating an Excel spreadsheet in a format similar to what is depicted in Figure 1. Each column should have a title with no rows between the title and the data. Change the sheet name to “TableA”, and save the file with the name “Excel Database.xls”.

	A	B	C	D
1	Col1	Col2	Col3	
2	1	6	11	
3	2	7	12	
4	3	8	13	
5	4	9	14	
6	5	10	15	
7				

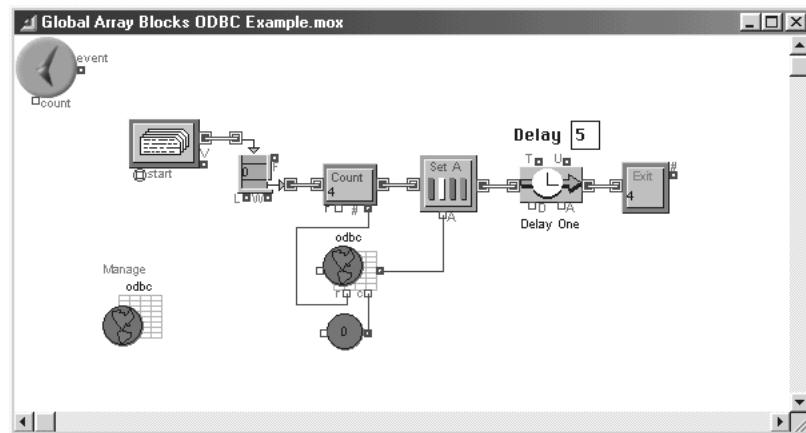
Excel Database.xls

Once the spreadsheet has been saved, it must be configured as an “ODBC Data Source” using the appropriate data source tool. The location and name of this tool varies from one operating systems to the next. For Example, on a Windows 98 machine the tool is located in the Control Panel and is called “ODBC Data Source”, while on Windows NT, the tool is located in the administrative tools system folder and is called “Data Sources”. For help in locating ODBC data source tools please refer to your Microsoft system documentation.

Once the tool has been located, go to the User DSN tab and click the Add button to create an ODBC compliant database. In the next window choose the Excel Driver option and click Finish. A new dialog is displayed requesting a name for the new database and the workbook upon which it should be based. Call the new database “Excel Database” and specify the appropriate workbook by hitting the “Select Workbook...” button.

Importing Data into an Extend Model Using ODBC

The database created above will now be used as input data for the Extend model. Open the “Global Array ODBC Example.mox” model found in the “Examples \ Manufacturing or BPR \ ODBC example” folder. This model is depicted below.



ODBC Example.mox

In the dialog of the Global Array Manager block go to the “Manage” tab and create a 5 rows by 1 column real array called “Delay Times” by clicking the “New Integer Array” button and following the series of dialog windows that automatically appear on the screen.

Now go to the Database tab. Click the Browse button and the “Machine Data Source” tab. There you should find the “Excel Database” you created earlier. Choose it and click ok.

Next, go to the Table popup menu and choose TableA. Then in the Columns table, click in the row 0 cell and select one of the columns defined in your Excel database. Click the Import button. Back in the Manage tab you should see the data from your Excel database. If it is selected, deselect the “Initialize at simulation start” check box. Click OK.

Next, go to the Global Array block and choose the “Delay Times” array from the Array popup menu. You should see the resulting information in the dialog table. Click OK and run the model. From the way this model has been configured, delay times from the Excel data base will be used to define how long each of the four items will spend in the Activity Delay (Attributes) block.

Distributed computing using Mailslots (Windows only)

Extend’s mailslots feature allows Extend applications running on two different computers on the same local area network to communicate. First, the copy of Extend running on the receiving computer opens a mailslot. The sending computer can then send a 255 character string to a receiving block on a different computer. The recipient of the message is specified by two strings: the name

of the computer (this can be found in the Windows Network Neighborhood) and the mailslot name (specified by the receiving block when the mailslot is created). For example, the receiving block on a computer named “bob” would create a mailslot named “mymail” and store the index of that mailslot with the following code:

```
MailslotIndex = MailSlotCreate("mymail");
```

A block on another could send a message to the “mymail” mailslot on “bob” with the following ModL function:

```
MailslotSend("bob", "mymail", "Hello World");
```

When the message is received by “bob”, the On MailSlotReceive message handler is called. This is a notification that mail is waiting in the mailslot queue. This message handler will continue to be periodically called by Extend until the queued mail is removed by the MailSlotRead function. The receiving computer could get the message with the command:

```
Message = MailSlotRead(MailslotIndex); // ItemString is a 255 character string
```

Note that based on the network topology, the mail message could be duplicated on the network. It may be useful to put a unique tag (serial number) on each message so that the code can identify if a particular message has already been processed.

More information on MailSlots can be found in the Extend Programmer’s Reference.

The Distributed simulation library

The Extend installation contains an example of using Mailslots to send items from one model to another. A pair of blocks from the “Distributed simulation” library are used to transfer the items. The receiving model uses the “Receive item from model” block. The sending model uses the “Send item to model” block. A common mailslot name is used to link the two blocks together.

Before the communication can begin, the receiving model must either be opened with the “Open mailslot communication automatically” option turned on or the “Open mailslot” button must be pressed. This sets up the receiving block to get the mail messages.

The simulation clock in the sending model is always ahead of the clock in the receiving model. Items arriving to the receiving model have a time tag and are stored in the “Receive item from model” block until the simulation clock matches the time tag.

There are three types messages that are sent from one model to the other. The first contains the command “Run” which will start the simulation model running. The second contains information about each item that is transferred from one model to the other. This includes a serial number for the message, the time that the item is transferred and the internal array values and attributes for the item. It is important that the receiving model has exactly the same number and names of attributes as the sending model. For efficiency, the names of the attributes are not transferred, only their values. Also, because the message is limited to 255 characters, the number of attributes that

can be transferred is limited. This limit depends on the length of the values of the attributes themselves. The third command is “End” which stops the simulation run.

Interprocess communication

The means by which applications communicate with each other and share data are collectively known as *interprocess communications*. Some communication methods have previously been discussed in this manual. For example, the clipboard provides convenient data sharing between applications running on the same operating system. The following section focuses on more extensive communication methods which allow Extend to *directly* communicate with other applications, and with other Extend models running on other machines, *while the simulation is running*. This allows Extend to work jointly with other applications on a wide variety of tasks.

Extend

Interprocess communication (IPC) provides a standard way in which one application can directly communicate with another. In Extend, you can incorporate IPC into your models using blocks from the IPC library (discussed in “Interprocess communication” on page E265), using the *Hot Link* or *Publish and Subscribe* features (discussed later in this section) or, if you program, using the IPC functions (discussed in “Interprocess Communication (IPC)” on page P102).

Communicating applications are typically categorized as *clients* or *servers*. A client application requests a service from some other application and a server application responds to the client’s request. Many applications, such as Extend, act as either a client or a server depending on the circumstances.

As a **client application**, Extend can request data and services from a server application. It does this using the IPC library or the IPC ModL functions.

- *Macintosh:* The IPC functions are implemented using AppleEvents messages (for example, Set Value, Get Value, and Evaluate).
- *Windows:* The IPC functions are implemented using dynamic data exchange (DDE) messages (for example, DDEAdvise, DDEPoke, DDERequest, and DDEExecute).

As a **server application**, Extend can receive and execute ModL code that has been sent from a client application. This means that other applications can instruct Extend to perform any task supported by the ModL language.

- *Macintosh:* Extend supports the following AppleEvents: OpenApplication, OpenDocument, PrintDocument, Quit, and DoScript.
- *Windows:* Extend supports the DDEAdvise, DDEExecute, DDEPoke and DDERequest messages. These messages accept two string parameters, *item* and *topic*. The topic parameter is the name of the Extend model worksheet you wish to communicate with. The item parameter for DDEExecute contains the ModL code to be executed. The item parameter for DDEPoke and DDERequest take the form: “variable name:#block number:row:column”. The “#” before “block number” is important to the syntax of the command and

must be included. Variable name can be the name of a dialog item, global variable, or static variable. Row and column are zero if the dialog item is not a table.

Note For Extend to communicate with another application on Macintosh or Windows, the other application must also support AppleEvents or DDE respectively.

Whether you use the IPC library or program using the IPC functions, the IPC client/server functionality in Extend is cross-platform compatible – the only difference between platforms is in how IPC is implemented. On the Macintosh, IPC is supported using Publish and Subscribe and AppleEvents; on Windows, IPC is supported through dynamic data exchange (DDE), OLE ActiveX, and Hot Links.

Hot Links (Windows only)

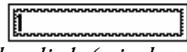
One way to share data between Extend and another program is by creating a *hot link* between the two applications. A hot link is a dynamic, one-directional, communication channel. With a hot link, when you change data in one application, the data in the other application is automatically updated. You can create links between Extend and any other application that supports hot links. There are two types of hot links within Extend: 1) links that import data from another application into your model and 2) links that export data from your model to another application.

Linking data from another application into Extend

You may wish to use data from another application as input parameters for blocks in your model. To do this, create a dynamic link between the data and any parameter field or data table in any of Extend's block dialogs, as follows:

- ▶ In the server application, copy the data to the clipboard using the other application's Copy command.
- ▶ In Extend, select the parameter field you wish to copy the data to. Note that, if you are copying to a data table, you must select the entire range of cells you want linked.
- ▶ If the server application supports hot links, Extend will recognize it and the Paste Link command in the Edit menu will be enabled. Give the Paste Link command to link the data into Extend.

If a hot link has been successfully created, the parameter will have a border around it as shown below:

Constant value = 
Parameter with a hot link (windows)

Once a hot link has been created, any changes made to the data in the other application will immediately be reflected in Extend. To remove the link, select the parameter and choose *Delete Link* from the Edit menu. There is a menu command called *Refresh links* which can be issued

when Extend is the DDE client, and the server is shut down and restarted. This command will reconnect links which have been disconnected. The *Show links* command will open all blocks with linked parameters or datatables.

Linking data from Extend into another application

You may decide to export data from your model to another application for further analysis or presentation. To create a hot link that does this:

- ▶ In Extend, select the block dialog parameter or data table cells you wish to be linked.
- ▶ Select the Copy command from the Edit menu.
- ▶ Refer to the other application's documentation to determine the way in which a hot link is created. Typically, you would choose Paste Link or Paste Special from the client application's Edit menu.



If a hot link has been successfully created, the parameter in Extend will have a border around it as shown in the previous section. Any changes made to the parameter in Extend will now be reflected in the other application. To remove the link, refer to the other application's documentation.

All hot links are saved with the Extend model. When the model is opened, Extend will attempt to re-establish the associated links.

Embedding OLE Objects and ActiveX Controls (Windows only)

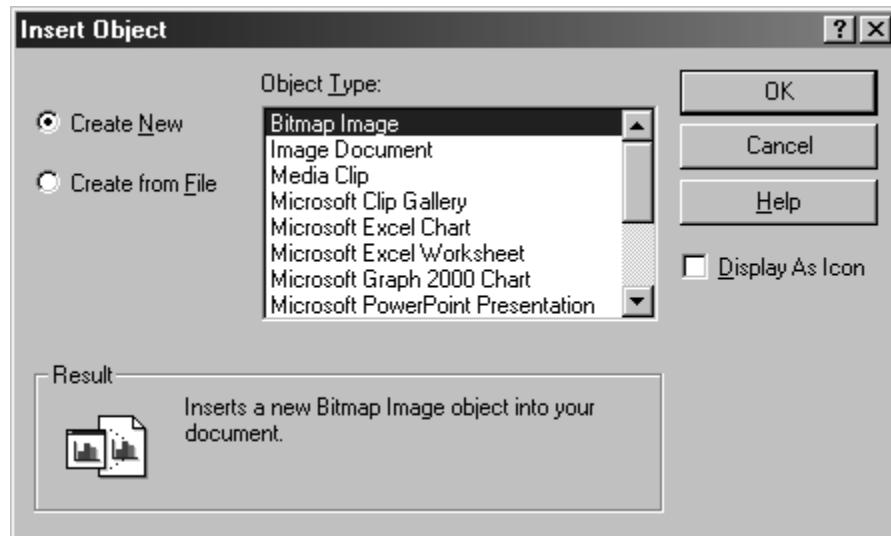
Embedded objects are external components that allow applications to have new behaviors and functionality without needing to custom code the behavior. Extend's support of this functionality makes it easy for users to incorporate objects and controls that can do things that might otherwise be difficult or impossible in the base Extend package.

Extend supports embedding of OLE objects and ActiveX controls. The difference between an embedded object and a control is that an object usually has a source application that it derives from, and a control is usually a stand-alone object that doesn't necessarily have an application behind it.

There are two places in the application where these objects can be embedded. The first place is at the worksheet/model level, where they can be included as a container object, and can be manipulated much like blocks, and other types of worksheet objects. The second is at the block dialog level, where they can be inserted into a special type of dialog item called an embedded object.

Embedding an object

To embed an object, select the *Insert Object...* command from the *Edit* menu. The following dialog will allow you to select from the insertable objects that are available on your machine:



Insert Object dialog

If *Create New* is selected, this command will create a new object of the indicated type with no data associated with it, basically an empty document of the type specified. If you select the *Create from File* radio button, it will create an embedded object with the data from that file as the starting point. The *Display As Icon* checkbox will specify that the new object should only have an icon as its representation in Extend, instead of displaying the controls interface.

The use of this dialog will have different meanings, dependent on what window you have open in Extend. If you have a worksheet or model window open, issuing this command will insert a Container Object on the model, which can be manipulated like a block, and worked with on the worksheet. If you have a dialog open with an *Embedded Object* dialog item in it, it will insert the embedded object into that dialog item.

Embedding an object on the worksheet

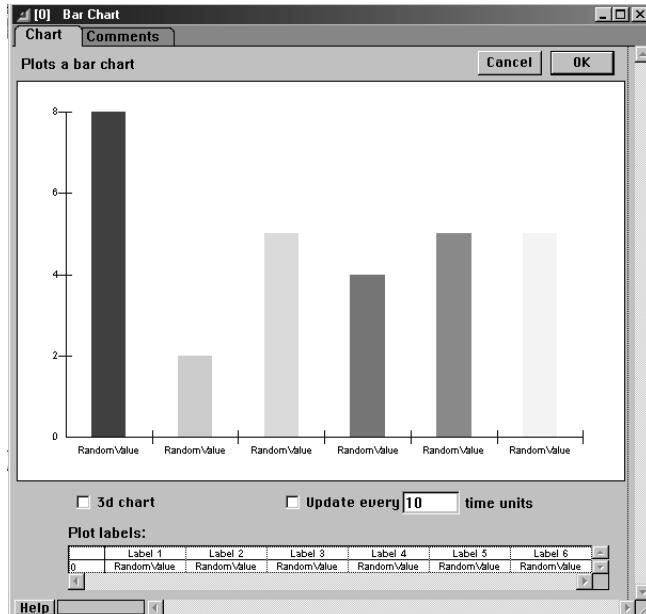
If the Object is embedded into the worksheet, it will create a container object. The container object is a worksheet object, similar to a block or an hblock. It has a block number associated with it, and can be moved around the screen in the same was as a block can. If the object responds to clicks, you will need to move it in design mode. (See “Design Mode” on page E182, below.) The block number is used to references container objects in the OLE function calls.

Embedding an Object into a Dialog Item

When a block dialog is designed, the designer can add an embedded object dialog item. This item is the location where one can embed an ActiveX control, or other kind of embedded object into a dialog. Embedded objects in dialogs are referenced via a combination of block number, and dialog item name, using the ModL functions in “OLE” in the Extend Programmer’s Reference.

Barchart example

The example model barchart.mox contains an example of the use of an ActiveX Control in a block dialog. If you double click on the barchart block in the model, you will see the following dialog:



Barchart.mox example

The plot displayed in the dialog of this block is from an embedded ActiveX control called Graphics Server.

Note: A runtime version of the GraphicsServer ActiveX control is provided with the Extend installation. This is being distributed to allow users to see this block as an example, and use the functionality of a bar chart. If the source code of this block is modified or if the Graphics Server Control is accessed through custom ModL code, purchasing a developer version of Graphics Server is required.

Graphics Server can be purchased from:

Pinnace WebWorkz
P.O. Box 4620
Seattle, WA 98104
Phone: (206) 625-6900
Fax: (206) 625-9102
<<http://www.graphicsserver.com>>

Paste Links to an embedded excel spreadsheet

A common technique for inter-application data exchange is to create data links between Extend and Excel via Paste Links (see “Paste Links to an embedded excel spreadsheet” on page E182). An embedded excel worksheet can also be data linked to Excel via paste links. This works in a fairly straightforward manner, with the following exception; Excel doesn’t support reconnecting links in embedded object when they are reopened by default. There is an option called ‘Update Remote References’ in the Excel options dialog that will enable this behavior. The following Excel VBA macro will turn this flag on:

```
Private Sub Workbook_Open()
    With ThisWorkbook
        .UpdateRemoteReferences = True
    End With
End Sub
```

Unfortunately the value of this option does not seem to be saved in embedded excel worksheet objects. This means that the macro above should be used in worksheets that are embedded into Extend models if they use Paste link connections to share data with Extend. Please see monte carlo.mox in the “Examples \ Extend \ Business and O/R” folder for an example of this.

Design Mode

The design mode menu command allows you to move or resize objects that would otherwise respond to the clicks on them in an object specific way. (For example if you click, or double click on the graphics server object when you are not in design mode, it will have the effect of “in-place-activating” the object. If you double click it in design mode, it will open a Property Pages Graphics Server dialog that will allow you to customize settings of the control.) The way objects behave both in, and out of design mode will be object dependent.

Object

The Object command in the Edit menu contains a submenu that is modified based on the currently selected embedded object. This submenu will contain commands that are derived from the objects internal definition, and will be different based on the code of the object. As an example, in the Graphics Server case, the menu just contains the single command “Property Pages Graphics Server Extended Graph” which will just pull up the property pages dialog for the graphics server object.

For information on ModL coding to control OLE objects and to use Extend as an OLE Automation Server, see “OLE and ActiveX” on page P226.

Publishing and subscribing (*Macintosh only*)

You can communicate with other Macintosh programs through the Publish and Subscribe commands if the other programs also support those features. Extend can publish data tables and plotters and can subscribe to parts of data tables.

To make Extend data available to client applications, you create a *publisher* for that data. To have Extend access data from a server application, you create a *subscriber* to that data. When you create a publisher, an *edition* file is created which contains the data you publish and is directly linked to the published data. Other applications can then share that data by creating a *subscriber* to the edition file in one or more of their documents. Subscribing puts a copy of the edition into the subscribing document. When the publisher to which the edition is linked gets updated, the changes are reflected in the edition and in all documents subscribing to the edition.

Extend

The Create Publisher and Subscribe To commands in the Edit menu let you create editions and use editions created in other programs.

You can publish two types of data:

- Data tables in dialogs and plotters are published as tab-delimited text with no formatting
- Plots are published as simple pictures

You can subscribe to one type of edition, text that is delimited by tabs that is used in an Extend data table or plotter data pane.

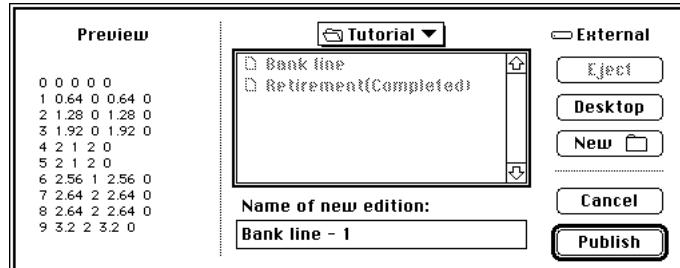
Publishing data tables and plots

The Create Publisher command is one of the easiest ways to make a direct connection between your Extend models and other programs that support publishing/subscribing. Typically, you will publish some or all of the data table in the plotter in your model since that is usually where the results of your simulation reside. You may also want to publish data tables from other intermediary blocks if they contain important data generated by the model. Publishing plots is less common but still useful if your plots describe the results of a model better than the numbers.

To publish data from a data table, first select the data you want to publish. You can select some cells or the entire table. Note that even if you select the row labels or column headings in a table, they will not be published; only the data will appear in the edition. Also, you can only have one edition per data table, although you can publish many data tables from a single model. If you select and publish the entire table, you can later add to the table, and the edition will be updated accordingly.

To publish a plot, click in the upper pane of a plotter window (the part with the picture). This selects the plot.

After you have selected the desired data in the table or a plot, choose Create Publisher from the Edit menu. You see:



Create Publisher dialog

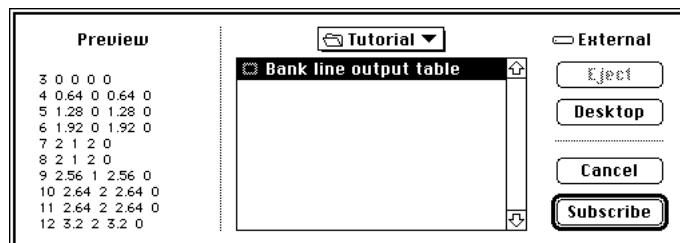
Note that the preview shows the data that you have selected in the case of a table or a reduced view of the plot if you selected a plot. Type in the name that you want for the edition and select a location for the file (you should not save editions in the Extensions folder because it will slow the launching of Extend). Then click Publish. The edition file will be saved in the selected location using the default edition icon.

When you publish a data table or plot, the data or picture in the edition is linked to the publisher, as indicated by the gray border around the selected area. When you save your model (or when the simulation run ends, if you have selected "Publishers update at end of run" in the Preferences command), the changes you made in the publisher are also made in the edition and in any documents that subscribe to the edition.

Subscribing to data tables

The Subscribe To command lets you fill in the values in a data table with values from other programs or from other Extend data tables. For example, you might set up a data table to take in variables from a spreadsheet that is generated by a different person.

An edition can have no, one, or more subscribers. In order for a document to subscribe to an edition, the edition file must already exist on disk. To subscribe to an edition that already exists on disk, select the cells in the data table that will receive the information and choose Subscribe To from the Edit menu. The dialog is:



Subscribe To dialog

In the dialog, locate and select the edition you want. The preview shows the contents of the selected edition. Choose the edition you want and click Subscribe.

As indicated by the border, the selected cells now dynamically share data with the edition; the subscriber border is darker and more granular than the publisher border. When data is changed and saved in the publisher to which the edition is linked, the changes are reflected in the edition and in all documents subscribing to the edition.

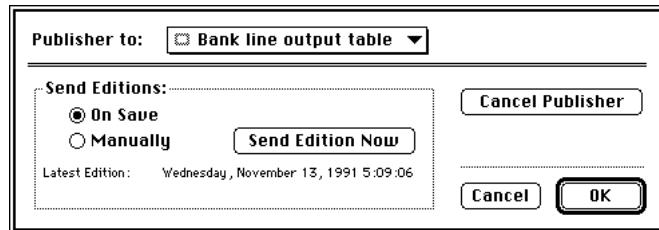
Extend

Publisher/Subscriber options

You can change some of the features of published data tables or plots and subscribed data tables by choosing Publisher/Subscriber Options from the Edit menu. It is important that you use the Publisher/Subscriber Options command to cancel publishers and subscribers, rather than just deleting or removing the edition file from the disk. Also, if you want to move the edition to another drive, you must cancel the original publisher and republish to the new drive.

Updating a publisher's edition

You can change some of the features of the edition for a published table or plot. With the published data table or plot selected, choose Publisher Options from the Edit menu. You see:



Publisher Options dialog

The Publisher To option shows you the location of the edition the selected data is published in.

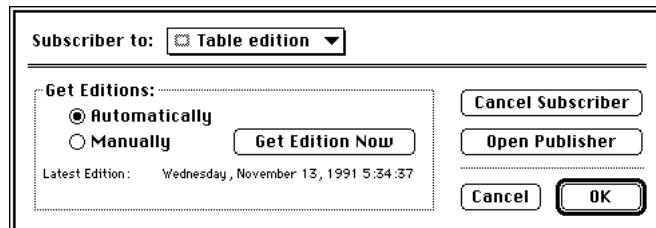
The Send Editions choices tell when you want to update the edition:

- *On Save* indicates that you want to make the changes to this edition when you save the model or when the simulation run ends. (If you choose “Publishers update at end of run” in the Preferences command of the Edit menu, editions will be updated at the end of each simulation run as well as when the model is saved.)
- *Manually* indicates that you want to only make those changes to this edition when you click the Send Edition Now button.

The Cancel Publisher button returns the selected data table or plot to non-publisher status, as indicated by the removal of the gray border. Later, when you save the Extend file, the edition is removed from disk. Always use this instead of simply deleting the edition file if you no longer want to use that file.

Updating a subscriber

With the subscribed data table selected, the Subscriber Options command lets you change some of the features of the subscribed edition:



Subscriber Options dialog

The Subscriber To option shows you the name and location of the edition the table subscribes to.

The Get Editions choices tell when you want to get updates from the edition if the model is open when a change is made to the edition. Automatically indicates that you want to get the changes in the edition whenever they are made; Manually indicates that you want to only get those changes when you click the Get Edition Now button. If the model is closed when the change is made to the edition, the data table will be updated the next time the model opens.

The Cancel Subscriber button removes the link between the table and the edition (it does not effect the edition itself, only your link to it). Always use this instead of simply deleting the edition file if you no longer want to use that file.

Open Publisher causes the program that published the information to open the document in which that information resides. This is a quick way of finding where the information actually resides and seeing it in the context of the program that created it.

Locating publishers and subscribers

If you have several publishers or subscribers in your model, or if you have merely forgotten which block has published or subscribed data, use the "Open Publishers/Subscribers" command in the Edit menu. Giving this command opens the dialogs of every block in the model that has a publisher or a subscriber.

Edition files

Extend saves edition files using the default edition icon. If you open the edition files, either by using the Open command from the File menu or by double-clicking on the edition, you see a preview of the edition. You can choose in that dialog to open the publisher, causing the publishing model and dialog to open.

Note Do not delete edition files directly. Always use the Publisher/Subscriber option, described above, to cancel publishers and subscribers rather than deleting edition files. For example, to move the edition file to another drive, you must first cancel the original publisher.

Model reporting

The report commands in the Run menu allow you to generate custom reports of model data. The commands let you choose just the blocks you want in the report, or you can choose to report on all the blocks in a model. Once the report is generated, you can view or edit the text file in Extend or export the data into another application for analysis or presentation. For example, if you have a discrete event model with queues, you might want to check each queue to see what the maximum queue length was. You could simply look in the report for those queues. There are two types of reports that can be generated: 1) the Dialogs report and 2) the Statistics report.

Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

Extend

The Dialogs report includes the final values for the input and output parameters of every chosen block, as well as the information in each block's comments field. Since this report includes the values and settings for all of the parameters in each of the selected blocks, it is a good tool for documenting your model.

The Statistics report includes the final values for the output parameters only. This report arranges the statistics in tabular form which allows for easier comparison of block results and exporting to spreadsheets.

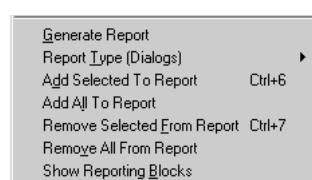
You can generate only one type of report for a given run. Both reports organize the data first by block type (see “Block types” on page P56 for a discussion on block types), then by block number allowing you to easily locate the results for a particular block. The reports are saved as text files on the disk and open automatically when the simulation is finished. If you perform a *multi-sim* run by setting the Number of runs parameter in the Simulation Setup dialog to a number greater than one, each report is appended to the current report file so that you can compare reports from earlier runs. However, if you perform two consecutive single runs, the reports will be written to separate files or written over, if you specify the same file name.

Note that model reporting only tells you the final values of the blocks in the simulation. If you want to see the values of the blocks during the simulation, use the model tracing commands described in “Model tracing” on page E218.

The reporting commands in the Run menu are:



Macintosh



Windows

Reporting commands in the Run menu

Before you generate the report, you need to choose which blocks will report. To include some blocks in a report, select the blocks and choose Add Selected to Report. To have all the blocks be included, choose Add All To Report. If you want to report on fewer blocks in the next simulation run, select the blocks you want out of the report and select Remove Selected from Report; to start over on your selection of blocks, choose Remove All from Report. You can also see which blocks you are reporting on by choosing Show Reporting Blocks; each block in the report displays the word "Report" on its icon. Reports are opened, closed, and edited just like any other text file in Extend.

Steps for reporting

- ▶ Choose the blocks you want to report, as discussed above.
- ▶ Select a report type using the Report Type command in the Run menu.
- ▶ Choose the Generate Report command.
- ▶ Run the simulation to generate the report. Extend prompts you for a name for the report file.

Note We suggest that you do not include a plotter block in your reports. Plotters write out all report data to a very large file.

Reporting example

If you select two of the teller blocks from the "Bank Line" model, the reports might look like:

Dialogs report

```
Extend Dialog Report - 8/8/97 10:39:33 AM
Run #0

ACTIVITIES
Activity, Delay          block number 3
Block Label: Teller 1
    Teller takes an average of 2 minutes to process a customer's transactions.
Input Parameters:
    Time units = generic time units
    Last Delay Used = 2 generic time units
    This block utilizes blocking
    T connector represents the amount of time since the last event
    Do not change animation pictures
Simulation Results:
    Arrivals = 31, Departures = 30
    Utilization = 1

Activity, Delay          block number 5
Block Label: Teller 2
    Teller takes an average of 2 minutes to process a customer's transactions.
Input Parameters:
    Time units = generic time units
    Last Delay Used = 2 generic time units
    This block utilizes blocking
    T connector represents the amount of time since the last event
```

```

Do not change animation pictures
Simulation Results:
Arrivals = 30, Departures = 29
Utilization = 0.98916667

```

Statistics report

Extend Statistics Report - 8/8/97 10:42:13 AM
Run #0

ACTIVITIES		Number	Name	Depart	Arrive	Util	Cost
Block	Label						
Teller 1	3		Delay	30	31	1	
Teller 2	5		Delay	29	30	0.98917	

Extend

You can edit the report file just as you would any text file in Extend; this is described in “Importing and exporting with text files” on page E167. If you program your own blocks, you can specify what data gets written in the report as discussed in “Adding Trace and Report code” on page P234.

When printing the Statistics report onto a standard 8.5" by 11" sheet and ensure that the entire page width is printed, use the Text command to change the font from the default size “12” to size “9”. The default size is set for easy on-screen viewing, but will exceed the page margins.

Communicating with external devices and instruments

In some situations, you may want to obtain data for your model directly from an external piece of equipment. For instance, if you are 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, there are several methods you can use to communicate with external devices such as scientific equipment and other hardware:

- Serial port functions
- XCMDs (Macintosh: external commands) or DLLs (Windows: dynamic-link libraries)
- Driver functions (Macintosh only; on Windows use DLLs)

Communicating from Macintosh computers

- If you want to pass data through serial devices, you can use the serial port functions. These functions can read and write any data (including real-time data) to and from the Macintosh's serial ports. This is very useful for transmitting and receiving data on a modem, for example. The three calls are very simple to use and even beginning ModL users should have no problems creating blocks which use them. See the “Serial Ports” model in the “Examples \ Extend \ ModL Tips” folder for examples of blocks that use these functions.
- XCMDs (external commands) and XFCNs (external functions) are segments of code written in any language. They can perform the same functions as drivers and are easier to use. They can

also be used to perform complex calculations utilizing specialized hardware. For more information, see “XCMDS under Macintosh”, below.

- Many hardware vendors supply device drivers with their hardware. If you are familiar with device drivers, you can use Extend functions to read from and write to external devices directly. The “ReadDevice” block in the Utilities library is an example of using driver functions to access devices. Unless you are very familiar with drivers, or have a driver supplied by a hardware vendor, it is suggested that you use XCMDS instead since they can accomplish the same results with less effort.

For more information about XCMDS/XFCNs, drivers, and serial ports see the discussions below and in “XCMDS and XFCNs” on page P237, and the functions in “I/O functions” on page P98.

Communicating from Windows computers

- 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 very useful for transmitting and receiving data on a modem, for example. The three calls are very simple to use and even beginning ModL users should have no problems creating blocks which use them. See the “Serial Ports” model in the “Examples \ Extend \ ModL Tips” folder for examples of blocks that use these functions.
- DLLs (Dynamic-link Libraries) are segments of code written in a language other than Extend’s ModL, such as Visual Basic or C++. Their standardized interface provides a method for linking between ModL and other languages. For more information, see “DLLs under Windows”, below.
- Instead of driver functions to read from and write to external devices, use DLLs, described above.

For more information about DLLs and serial ports see the discussions below and in “DLLs” on page P238, and the functions in “DLLs (Windows only)” on page P118.

Accessing code in other languages

In addition to its built-in ModL language, Extend provides two methods for accessing code written in other languages: XCMDS ( *Macintosh only*) and DLLs ( *Windows only*).

XCMDS under Macintosh

XCMDS (external commands) and XFCNs (external functions) are segments of code written in a language other than Extend’s ModL. Their standardized interface provides a method for linking between ModL and other languages.

XCMDS and XFCNs allow you to access functions that are already written in another language or solve problems that might be difficult or impossible to solve in ModL. You can use an XCMD to calculate some function, perform a task, or even access the Macintosh toolbox. Extend’s XCMD functions allow you to call the external code segments from within a block’s ModL code and per-

form operations. For example, you can pass data to the XCMD, calculate using that data, and get the results back from the XCMD. For more information, see “XCMDS and XFCNs” on page P237.

DLLs under Windows

DLLs (dynamic-link libraries) are segments of code written in a language other than Extend's ModL. This standardized interface provides a method for linking between ModL and other languages.

DLLs allow you to access functions that are already written in another language or solve problems that might be difficult or impossible to solve in ModL. You can use a DLL to calculate some function, perform a task, or even access Windows API calls. Extend's DLL functions allow you to call the external code segments from within a block's ModL code and perform operations. For example, you can pass data to the DLL, calculate using that data, and get the results back from the DLL.

Extend

Extend provides two separate interfaces for accessing DLLs. One interface allows existing DLLs to be used with Extend, the other is designed for cross platform compatibility with the Macintosh version of Extend. Either can be used to create a new DLL for use with Extend for Windows.

For more information, see “DLLs” on page P238.

Using interactivity as input and output

Do not overlook interacting with your models as a method of importing and exporting data. Although the other methods described in this chapter are more direct, you can still watch the model running and use controls or changing dialog parameters as a form of input. In fact, the most common way of getting data into your model is by interacting with it through the dialog boxes and the Notebook. Of course, it is also common to get data out of your model by watching the results as the model progresses.

Extend

Chapter 7: More Modeling Techniques

*More hints and suggestions about how to put together
your models*

*“If I had been present at creation,
I would have given some useful hints.”
— Alfonso the Wise*

You have already seen many of the techniques that you will use in your day-to-day use of Extend. The most important technique to remember is to start small, test what you have put together, refine the model, validate it, refine it again, and so on. What you are striving for is usable information, not an exact duplicate of the real-world system.

This chapter tells you about many issues that affect the way you build models. Some suggestions concern how to make your models more aesthetically pleasing while others help you find better ways to perform certain tasks. The topics are generally grouped by what is most important first.

General modeling hints

Steps in creating models

Like all tasks, you can start modeling simply by jumping into it. Also like other tasks, it is usually better to have a plan before starting. Extend makes following a plan for making a model easy.

The basic steps to creating a model are:

- *Formulate the problem*—You should define the problem and state the objectives of the model.
- *Describe the flow of information*—Determine where information flows from one part of the model to the next and which parts need information simultaneously.
- *Build and test the model*—Build the system with Extend’s blocks. Start small and enhance as needed.
- *Acquire data*—Identify, specify, and collect the data you need for the model. This includes finding not only numerical data values but also mathematical formulas such as distributions for random events.
- *Run the model*—Determine how long you want to simulate and the granularity of results, then run your model.
- *Verify the simulation results*—Compare the results from Extend to what you intended or expected.
- *Validate your results*—Compare the model to the real system, if available.
- *Analyze your results*—Draw inferences from the model’s results and make recommendations on how the system can change.
- *Conduct experiments*—Implement and test recommended changes in the model.
- *Implement your decisions*—Use the results in the real world.

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 problems may involve over-simplification in the model, false assumptions on your part 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.

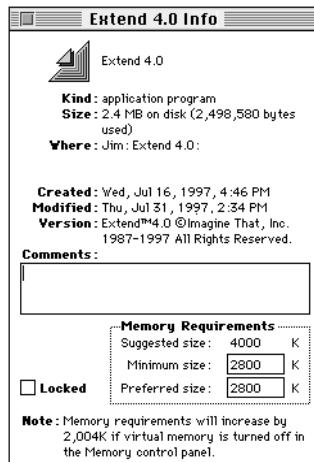
Extend

Low memory problems

You may occasionally be warned that you are running low on memory when running large models. If this happens, close any extraneous windows since these take up memory. You should also choose Preferences from the Edit menu and be sure that “Bitmap plotters” and “Bitmap blocks” are not selected ( Macintosh only). You can see how much memory you have free by looking in the upper right of the Help dialog. If you run low on memory, you should quit from Extend and increase Extend’s memory allocation, as shown below.

- Macintosh: To do this, quit Extend, exit to the Finder, select the Extend application's icon, and choose the Get Info command from the File menu.

Extend



Get Info dialog

Increase the preferred size option shown near the bottom of the dialog. Try increasing this option by 500 K increments.

- Windows:

Operating System Suggestion

Windows 95, 98, and ME	These use available hard disk space for a swap file used for virtual memory. In the System control panel, under the Performance tab, click the Virtual Memory button. Choose the "Let windows manage my virtual memory settings" radio button. If you are still having memory problems, delete any unnecessary files to increase the amount of available hard disk space.
Windows 2000	In the System control panel, under the Advanced tab, click the Performance Options button. Then click the Change button in the Virtual Memory section. Increase the Maximum Size for virtual memory until the memory problem is fixed.
Windows NT	In the System control panel, under the Performance tab, click the Change button for virtual memory button. Increase the Maximum Size for virtual memory until the memory problem is fixed.

Tool tips on the worksheet

Tool tips on the worksheet help you to quickly identify the name and block number of any block without having to open the block's dialog. When you rest the cursor above a block, a small window appears containing the block's name and local and global numbers. Tool tips on the work-

sheet can be turned on or off with the Preferences command in the Edit menu. Tool tips also exist to identify the variable name for block dialog items. This can be very helpful when programming your own blocks and is discussed further in “Accessing connectors and dialog items from the block’s code” on page P25.

Block icons – customizing

The blocks in the libraries that come with Extend (such as the Generic and Plotter 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 have the block’s icon more closely indicate its role in your particular model. However, changing 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 Extend.*

Extend

You can, of course, change the icon on *copies* of the blocks that come with Extend. And if you build your own blocks, you can give them any icon you want. However, there are two 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.
- You can add a picture to a block without affecting the original block in the library. To do this, make the block hierarchical by selecting it and using the Make Selection Hierarchical command. Then open the hierarchical block’s structure by double-clicking it while pressing the Option (Macintosh) or Alt (Windows) key or by using the Open Hierarchical Block Structure command in the Model menu, and paste the picture into the icon pane of the hierarchical block. This technique is described in more detail in the section on “Hierarchy” on page E252.

To add a picture to Extend, create the picture in a painting or drawing program and copy it to the Clipboard. With the Extend window open (model window, Notebook, or icon pane) choose Paste Picture from the Edit menu. Extend pastes the picture into the window.

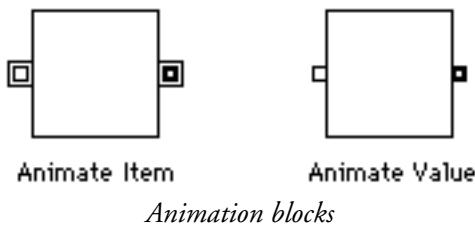
You can treat a picture like other drawing elements: you can use the Draw layer tool to drag it around the window and resize it however you want. To delete a picture, select it with the Draw layer tool and choose Clear from the Edit menu or press the Backspace or Delete key.

Animation – customizing

Many of the blocks in the Generic and Discrete Event libraries have animation built in, as described in “Appendix D: Generic Library Blocks” on page A45 and “Appendix E: Discrete Event Library Blocks” on page A55. You can see the animation in them by turning on animation in the Run menu. Animation was introduced in “Animation” on page E81.

Note: Please see “Proof Animation (Windows only)” on page E278 for more details on adding Proof animation to your models.

You may want to add your own animation to a model or to a hierarchical block’s icon. The Animate Item and Animate Value blocks in the Animation library are used for this purpose. The icons for the blocks are very simple:

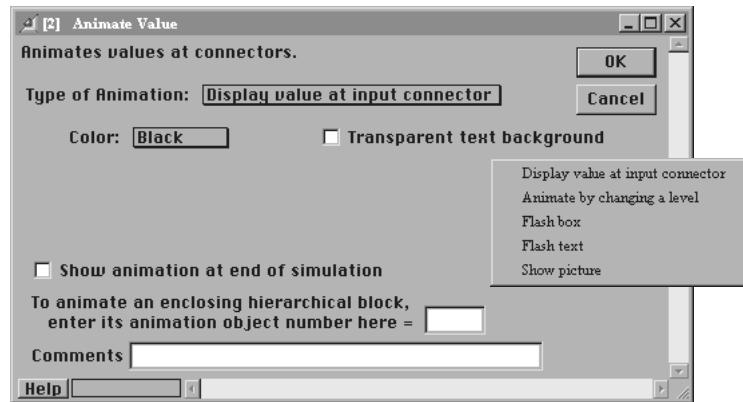


Both the Animate Item and Value blocks let you choose in their dialogs a picture or a color and pattern that will flash when they are animated. You can also use these blocks to animate hierarchical blocks, as described later in this chapter.

The *Animate Item* block can only be connected to the item connectors of discrete event blocks. When an item enters the block, the entire block flashes the color and pattern chosen, or shows a picture.

Note Discrete Event models are not available unless you have one of the Extend modules:
Extend+Manufacturing, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

The *Animate Value* block can be connected to value connectors. The block’s dialog is:



Animate Value dialog (Windows)

There are four main choices for specifying how this block animates:

- Display the value that is input to the block

- Move a level up and down between limits
- Flash an object or text when the input goes above a specified value
- Show a picture

Displaying the value is similar to the ReadOut block in the Generic library.

If you choose to animate with a level, be sure to specify values for the maximum and minimum that represent the entire range of possible values. The block will move a level up and down between the top and bottom of the block's icon.

Extend

If you choose to flash when the input is greater than or equal to a value, you can choose the shape of the animation: a box, text, or a circle. Due to the small size of the block's icon, the text must only be a few characters long. The default value, 0.5, is useful for measuring true-false values.

To show a picture, select its name from the popup menu.

Adding additional pictures are discussed in “Extensions” on page P236.

Note: Please see “Proof Animation (Windows only)” on page E278 for more details on adding Proof animation to your models.

Copying sections of a model

In some models, groups of blocks and their connections are repeated in many places. Extend makes it easy to duplicate these sections using the Copy and Paste commands. Simply select the desired group of blocks, right-click and choose Copy, or choose Copy Blocks from the Edit menu, click near the desired destination, and choose Paste Blocks from the Edit menu. You can then move the blocks as a group to the exact location you want.

For a faster alternative to copying onto the model window, select the desired group of blocks and choose Duplicate from the Edit menu. The blocks will be pasted below and to the right of the original group.

You can also use the Save Selected command from the File menu to save model portions as templates. You can then add the model portion to other models by choosing Append Model from the File menu. These commands are often faster and use less memory than Copy and Paste; they are discussed more in “Cut, Copy, and Paste with the Clipboard” on page E165.

- *Macintosh only:* If there is a group of blocks that you want to use in the future, you can copy them to the Macintosh Scrapbook using the Scrapbook desk accessory. You can then copy them from the Scrapbook at a later date and place them in different parts of your model.

Selecting cells in data tables

You can select cells in a data table or plotter by clicking and dragging. Clicking the header of a column or a row number selects the entire column or row. Clicking in the box at the upper left corner of the table selects the entire table. After you select cells, rows, or columns, you can then copy the information to the Clipboard with the Copy Data command by right-clicking or choosing the command from the Edit menu, and export it to another application.

To copy the row and column titles in data tables, select “Data table title copying” in the Miscellaneous tab of the Preferences command. Then click on one or more rows or columns and give the Copy command. Extend prompts you to select the titles you want copied.

Note If you choose to copy row or column titles, you should not attempt to paste into an Extend data table since the titles displace some of the data.

Column widths can be adjusted by placing the cursor directly over the column divide, and dragging the divide to the desired location while holding down the mouse button.

Stationery (*Macintosh only*)

When a Macintosh model file is changed in the Finder to become stationery, it becomes a template that can be used over and over. When you open a stationery document, Extend makes an untitled copy of the original model; you can make changes and additions, then save the model under another name without altering the original. Stationery files have distinctive icons.

To make Extend models into stationery:

- ▶ Select the model’s icon in the Finder.
- ▶ Choose the Get Info command from the Finder’s File menu.
- ▶ Click the Stationery pad option in the lower right corner of the Get Info dialog.
- ▶ Close the Get Info dialog.

See your Macintosh System documentation for more information about stationery.

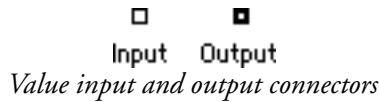
Connectors and connections

Mixing Generic and Discrete Event blocks

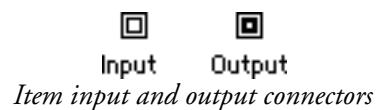
Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

As you saw in earlier chapters, you can use continuous blocks (those from the Generic library) in your discrete event models. You cannot, however, use discrete event blocks in continuous models. All blocks in the Discrete Event library require an Executive block which changes the timing of the model to match the timing expected by the discrete event blocks. This means that any model that contains discrete event blocks is a discrete event model.

As discussed in Chapter 2: Building a Model, the connectors on the blocks are visually different, indicating their type, and the type determines how they can be connected together. Many blocks use value input and output connectors to pass values:

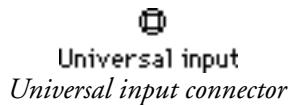


Discrete event blocks also use item input and output connectors to pass entities (called items):



Extend

Universal input connectors can be connected to either value or item connectors:



You can connect value connectors together, and you can connect item connectors together. You can also connect value output connectors or item output connectors to universal input connectors.

You cannot, however, connect a value connector to an item connector. This is because value connectors only pass values while item connectors pass unique items.

All connectors can be connected to universal connectors. Also note that if you create blocks using diamond connectors, those connectors can only be attached to other diamond connectors and to universal connectors.

Named connections

One way to avoid clutter in your models is to use named connections. A named connection is simply a text label used to represent one output in multiple locations in your model. You can imagine named connections as hidden connectors that run under the model window.

To create a named connection, type a text label near the desired output and draw a line between the output connector to the text. When you type or paste the same text near any input connector(s) and draw a line from the new connector(s) to the text name, the connection is complete. To help make your models clearer, use connection names that are relevant to the information that is being passed through the connectors. Direct connections can also be used, of course, since they help show the flow of information in your models. As discussed below, choose Show Named Connections from the Model menu to see and check the named connection links. Named connections are described in “Additional ways of connecting blocks” on page E59.

It is important that the text for named connections be spelled exactly the same, although capitalization, spaces, and returns are not relevant.

Note Named connections only connect with one level in a hierarchical 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.

Extend

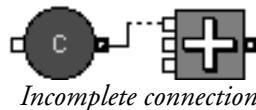
Show Named Connections command

If you use many named connections in a model, you may accidentally connect a block to the wrong named connection. The Show Named Connections command in the Model menu causes Extend to explicitly draw the connection line directly from one named connection to the other. This helps you check that you made the connections that you wanted.

You can click on the visible connection line to highlight it to its next connection, making it easy to follow the path. If you click the connection line and then use the Select All Segments command in the Edit menu, all the connections to that named connection will be highlighted.

Unconnected connection lines

Occasionally, you may have a block that is not connected to the model in the way you thought it was. This can happen if you have connections that run underneath blocks when you thought that they were connected to blocks. Extend shows incomplete connections as a dotted line segment:



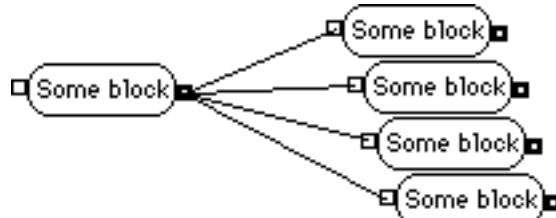
To fix these, delete the incomplete connections and reattach them to where they are supposed to be.

Connections to multiple inputs

You can connect from one output connector to as many input connectors as you want in continuous and discrete event models. However, the results in the two types of models is very different. As discussed above, value connectors pass values, while item connectors pass unique items. This means that in a continuous model, each connected input can read the value and use it in its calculations. In discrete event models, however, the first connector to pull in the passed item gets it and the other connectors do not.

Value connectors

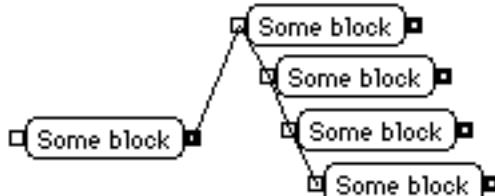
Multiple inputs for one output in continuous models is useful when many blocks need the results of a preceding block. For instance, if you have one output connector connected to four input connectors, your model might look like:



Extend

One value output connected to four inputs in a continuous model

Because these are value connectors, each input can read the value at the output of the block on the left. For aesthetic reasons, you may want to only have one connection coming from the output connector, for example when the blocks are far away from each other. Instead of four connections, you can connect to one of the input connectors, then connect the input connectors together (sometimes called *daisy-chaining*). The above example would look like:



Four value inputs connected together

Item connectors

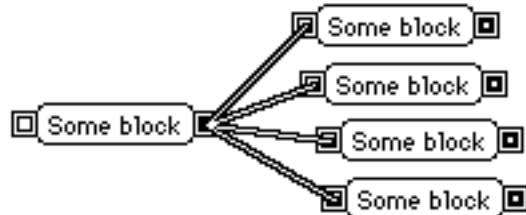
Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

In discrete event models, a single item output connected to many item input connectors is used for a very different purpose, namely parallel processing of items. This is because items (unlike values) can only be in one place at a time. In the picture below, the item will go to the first block that is free. If more than one block is free, the item will arbitrarily go to any available free block. (If it is important to specify the order in which items go to free blocks, you would use the Prioritizer block, as discussed below.) In any case, the item can only go to one block. This is very different from what happens with value connectors.

For example, in the following blocks, assume that the top block is free. An item might go to the top block unless that block was busy, in which case it would go to some other block. The blocks other than the first one may never see any items unless the first block is busy when an item comes

Extend

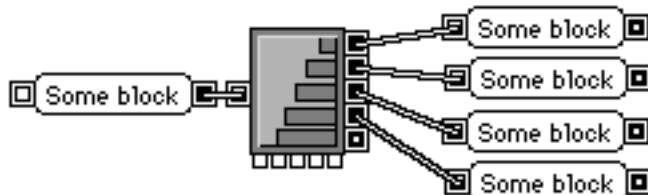
out of the block on the left. This use of parallel processing is common when you have many machines that work in parallel and items are given to the machines by something that looks for the first available machine. You can see the order of the blocks by choosing Show Simulation Order from the Model menu.



One output connected with four inputs in a discrete event model

You can also daisy-chain the blocks, as discussed above. Still, only one block would get the item.

If the priority of which block gets the item first is important, use the Prioritizer block (Routing submenu of the Discrete Event library) between the source block and the parallel blocks. For example, the model shown above would become:



Parallel blocks with Prioritizer

In the Prioritizer dialog, specify your desired priority for the parallel processes. This method is much safer than relying on parallel processing with no explicit routing order.

There are many other ways of routing items to a specific block using attributes, priorities, or load balancing techniques (see “Attributes” on page E114, the Extend+Manufacturing manual, and the Extend+BPR manual for further examples).

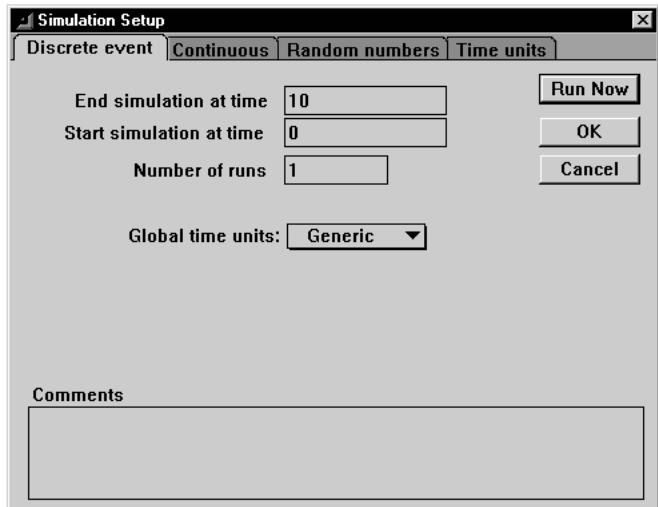
Running models

Before you run a model you need to specify how long the simulation will run and (for continuous models) the granularity or stepsize of the run. You do this in the Simulation Setup dialog.

Simulation Setup command

The Simulation Setup command in the Run menu controls the way that your simulation runs. The tabs in the dialog are:

Discrete event tab

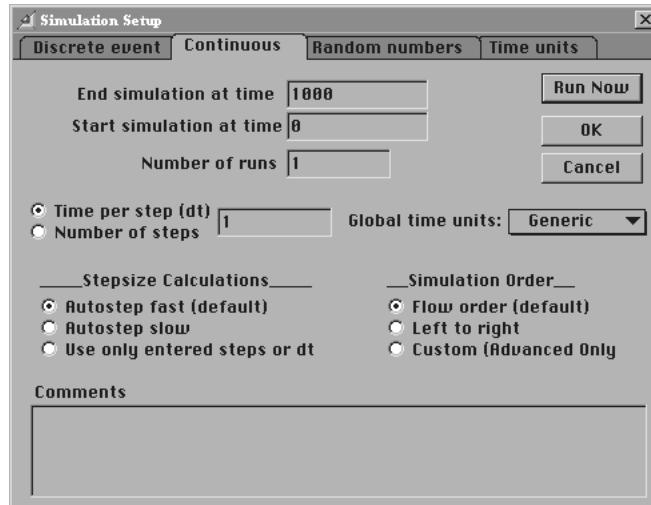


Discrete Event tab (Windows)

Choice	Description
End simulation at time	The end point of the simulation. See also “Simulation timing” on page E209.
Start simulation at time	The start point. Usually, this is 0. You can set it to a different value if your model uses the time value for some calculations. See also “Simulation timing” on page E209.
Number of runs	The number of times to run this simulation.
Global time units	Time unit for the entire model. Local time units are defined within the dialogs of blocks that contain time parameters. See also “Units of time” on page E213.
Comments	This is an optional item used for documenting aspects of the model, such as units.

Extend

Continuous tab



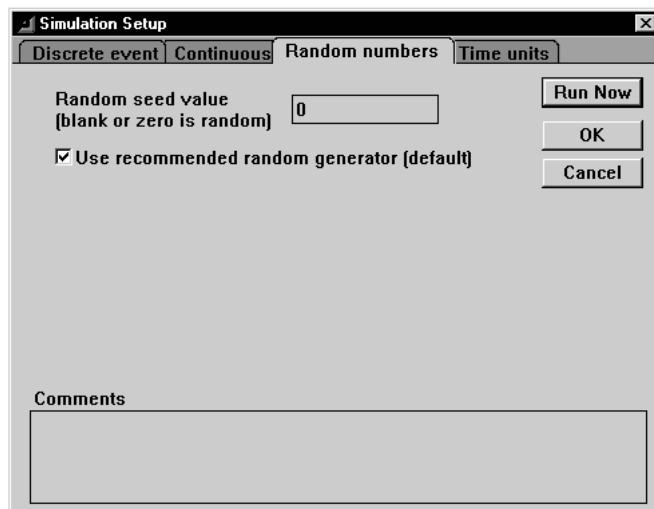
Continuous tab (Windows)

Choice	Description
End simulation at time	The end point of the simulation. See also “Simulation timing” on page E209.
Start simulation at time	The start point. Usually, this is 0. You can set it to a different value if your model uses the time value for some calculations. See also “Simulation timing” on page E209.
Number of runs	The number of times to run this simulation.
Time per step (dt)	Represents delta time, or the length of time per step. This is the default choice for determining the granularity of the simulation run. For most purposes you would 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 you enter; it is computed as: floor(((EndTime-StartTime)/DeltaTime) + 1.5) . You can see this by selecting the “Number of steps” option after you change the “Time per step (dt)”. See also “Setting dt or the number of steps in continuous models” on page E276.

Extend

Choice	Description
Number of steps	This is another method of determining the granularity of the simulation run. In most cases, this would be a number equal to the duration (length of the simulation run) so that the model calculates values once for each step, and each step would be one time unit long. A default value for delta time is automatically calculated based on the number of steps you enter; it is computed as (EndTime-StartTime)/(NumSteps - 1) . You can see this by selecting the “Time per step (dt)” option after you change the “Number of steps”. See also “Setting dt or the number of steps in continuous models” on page E276.
Global time units	Time unit for the entire model. Local time units are defined within the dialogs of blocks that contain time parameters. See also “Units of time” on page E213.
Stepsize calculations	These are only used in continuous models that change the <i>DeltaTime</i> system variable, such as electronics simulations. Generally, you always use <i>Autostep fast</i> . If your blocks change <i>DeltaTime</i> and demand more accuracy, use <i>Autostep slow</i> (which divides the calculated value for <i>DeltaTime</i> by 5). If your blocks change <i>DeltaTime</i> but you want to ignore the changes, use <i>Only use entered steps or dt</i> .
Simulation order	Generally, you always use “Flow order”. Simulation order is discussed in detail later in this chapter.
Comments	This is an optional item used for documenting aspects of the model, such as units.

Random numbers tab

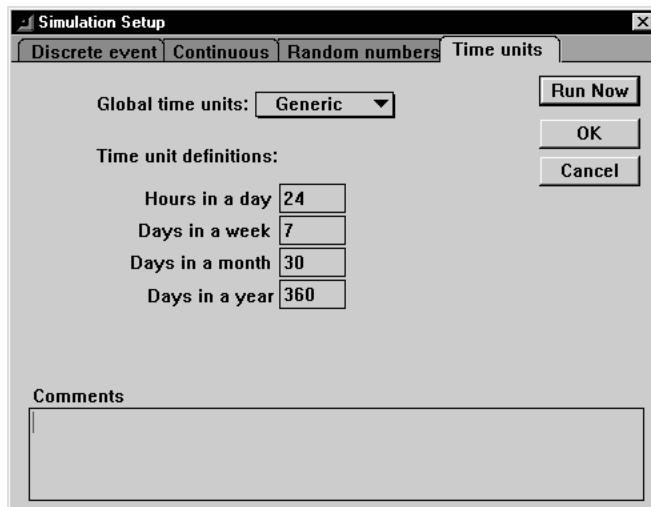


Random numbers tab (Windows)

Extend

Choice	Description
Random seed value	The “interface” for the random number generator. A value of 0 or blank uses a random seed; any other value causes repeatable sequences of pseudo-random numbers. Note that each block that outputs random numbers will generate its own independent sequence. For details about Extend’s random number generator see “Random numbers” on page E214.
Use recommended random generator (default)	When checked, Extend will use the recommended “Minimum Standard” random number generator. When unchecked, Extend will use the optional Schrage random number generator used in previous versions (backwards compatibility). Models created in versions prior to version 4 will default with this option unchecked. For details about Extend’s random number generator see “Random numbers” on page E214.
Comments	This is an optional item used for documenting aspects of the model, such as units.

Time units tab



Time units tab (Windows)

Choice	Description
Global time units	Time unit for the entire model. Local time units are defined within the dialogs of blocks that contain time parameters. See also “Units of time” on page E213.
Hours in a day	The number of hours in a day. This number is used by Extend to automatically convert local time units to the global time unit.
Days in a week	The number of days in a week. This number is used by Extend to automatically convert local time units to the global time unit.
Days in a month	The number of days in a month. This number is used by Extend to automatically convert local time units to the global time unit.
Days in a year	The number of days in a year. This number is used by Extend to automatically convert local time units to the global time unit.
Comments	This is an optional item used for documenting aspects of the model, such as units.

Extend

Simulation timing

All simulations run for a specified time and are either continuous or discrete event. Extend determines the *duration* of a simulation run based on the values entered in the Simulation Setup dialog; the duration is the period from the start time to the end time. The type of simulation you are running, either continuous or discrete event, is based on whether there are any discrete event blocks in the model. If your model contains discrete event blocks, it is discrete event; otherwise, it is continuous.

- 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. As the simulation runs, simulation time advances from start time to end time at delta time per step, calculating model data at each step. At the first step, Extend calculates what the status of the model is initially. Then it calculates the changes that take place over the next time step and determines a new set of data points. Model data is generated as a string of successive points corresponding to the steps in time. Notice that data is calculated at each step for the entire period from that step up to, but not including, the next step.
- In discrete event simulations, Extend progresses from start time through end time by a series of events where the time between events is probably not equal. Time progresses from one event to the next so time per step (delta time) and the number of steps represents the number of events in the model.

Continuous simulation example

For example, assume you want a continuous simulation to run and calculate values each year for 40 years where start time is 0 and dt is 1. The value you enter for the end time depends on whether there is integration in the model:

- If there is no integration in the model, set end time to 39. Data will be calculated at each of the 40 steps, starting at step 0 and ending at step 39. Each step's calculation would represent data for the entire year, the period beginning at that step and continuing up to but not including the next step (one delta time unit). Thus the model would calculate 40 years worth of data. If you specified start time as 1 and end time as 40, the duration would also be 40 years.
- If there is integration in the model, set end time to 40. Data will be calculated (but not output) at each of the 41 steps, starting at step 0 and ending at step 40. Each step's calculation would represent data at the beginning of the year, the period beginning at that step. However, blocks that integrate take their inputs at one step and output their results at the next step. Thus the model would calculate 40 years worth of data. If you specified start time as 1 and end time as 41, the duration would also be 40 years. In the Generic library the blocks that integrate are the Holding Tank and the Integrate blocks.

For models where delta time (dt) is other than 1, see below.

Discrete event simulation example

For example, assume you have a discrete event simulation in which the start time is 0, the end time is 60, and the time units are in minutes, such as in the “Bank Line” model in Chapter 1: Running a Model. Because this is a discrete event simulation, time per step (dt) and the number of steps are ignored. Data would be calculated for 60 minutes, the period from the start time up to and including the end time. The last event will occur sometime before or just at the end time. Unlike continuous simulations, it would not represent any period beyond the end time.

Delta times other than 1

All discrete event models, because they are event-driven, ignore the number of steps and the time per step (dt). Continuous simulations, however, require that either the number of steps or the time per step be specified. In most cases, a delta time of 1 is adequate. However, for model accuracy it may be necessary to set a delta time other than 1.

If you specify delta time as 0.5, the start time as 0, and the end time as 39, the simulation will run from time 0 to time 39 calculating data for 79 steps, each of which is one half time unit long. This is true whether your model has integration or not.

If you specify a delta time which will not result in the duration being divided into equal segments, Extend will adjust the end time to a higher value. This is to insure that the segments are of equal duration, and that the simulation will end at the end time displayed in the dialog. Alternately, you can select a new end time or delta time. For instance, in the example where end time is 39, you could specify the “Time per step (dt)” option as 2 years. Extend could then calculate data every

other year, from 0 through 38; however, the last step of the simulation would not be calculated. In this case, Extend will adjust the end time to 40. This means that model data will be calculated once every two years, starting at time 0 and ending at time 40, for a total of 21 steps.

For more information, see “Setting dt or the number of steps in continuous models” on page E276.

Simulation order

The Simulation Setup command in the Run menu allows you to choose the order in which Extend executes block data for continuous models. The choices are “Flow order” (the default) and “Left to right”. For most purposes, you would want to use the default choice, flow order. To see the order in which blocks are executed, choose Show Simulation Order from the Model menu.

Extend

Flow order

During a simulation, the blocks that compose an Extend model do calculations that generally depend on their inputs. After doing their calculations, the blocks set their output connectors to the results of that calculation so that other blocks may use their results.

In this type of system, there has to be a “first” block: a block that calculates before all of the others that depend on its results. After the first block calculates, the other blocks should calculate in the order and direction of their connections. This order is repeated for every time step or event of the simulation. To see this order, choose Show Simulation Order from the Model menu.

The rules that Extend uses to derive the order of the block calculations in continuous models are:

- 1) Blocks that generate inputs to the simulation go first. For example, Input Data or Constant blocks with only their outputs connected to inputs of other blocks would be put first.
- 2) Next, Extend executes blocks that are connected to those first blocks, in the order and direction of their connections.
- 3) Unconnected blocks and bi-directional network blocks (that have only inputs connected to inputs) are executed in left-to-right order.

For most simulations you will use Flow order; this is the default choice in the Simulation Setup dialog.

Left to right order

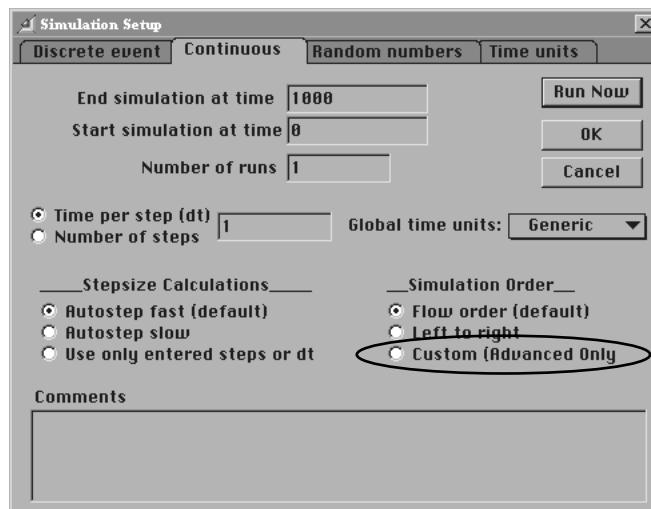
If you choose this option, Extend looks at the left/top corner of each block on the worksheet. The left-most block gets executed first, and the next left-most block gets executed second, and so on. Blocks with equal left edges get executed in top to bottom order. If your model flows to the right, and then continues at the left below that flow, this choice will still calculate all the left-most blocks first. If you use this order, be sure that blocks that calculate values are to the left of the blocks which need those values. Otherwise, there will be a one step delay in calculating the values.

Custom order

In continuous models, there are some models with multiple feedback loops that do not always come up with the desired flow order solution. This occurs because there are multiple solutions that solve the DAG (Directed Acyclic Graph) ordering problem, and it is possible for the less desired solution to be picked.

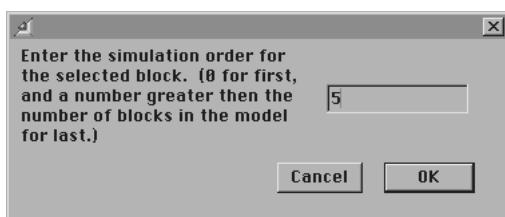
To solve this ordering problem, Extend makes it easy to start out with a flow order and then change that order for a few blocks using the Set Simulation Order command in the model menu.

Note that this menu is not enabled unless both the Custom Simulation order radio button is selected in the Run Setup dialog, and a block is selected.



Run Setup dialog showing Continuous tab and Custom order

In order to use this command effectively, the Show Simulation Order command should be checked so that the user can see which blocks need changing. When a block is selected, the Set Simulation Order command opens a dialog:



Custom Simulation Order dialog

The user can then enter a new number for that block. The block that had that order is swapped with the new ordered block.

Note that any custom order is lost if either Flow Order or Left-to-Right order is selected again.

Units of time

Time is the most common unit of measure that you will use in a model. In Extend, there are two ways to deal with time units:

- Use a *generic* time unit in the model and in each block. In this case, it is important that all time-based parameters throughout the model are based on that same unit of time.
- Establish a specific *global* time unit for the whole model. Then, in each of the blocks in your model, use parameters based on any *local* unit of time.

Extend

The default for new models is to use a generic unit of time. The Time units tab of the Simulation Setup dialog for new models specifies “generic” as the Global time unit. Each block in the new model will have its time-based parameter field followed by the phrase “time units”.

If you change the time units in the Simulation Setup dialog to be something specific, that setting becomes the global time unit for the model as well as the default local time unit within each block. You can then change the local time unit to be anything you want.

Using generic time units

For modelers who want to use the same time unit to define all the time-based parameters in the model, it may be easier to use a non-specified, generic time unit. To do this, leave “Generic” as the selection for the Global time unit in the Time units tab of the Simulation Setup dialog (all new models default to using Generic as the time unit).

When you do this, each block in the model will have its time-based parameter field followed by the phrase “time units”. In this case, you will not have the option of selecting a specific local time units for your blocks. You must therefore be sure to maintain a consistent time unit throughout all the blocks and in the Simulation Setup dialog. The comments field in the Simulation Setup dialog is a convenient place for noting what unit of time “Generic” represents in your model.

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 that are based on minutes.

Using a global time unit

The global time unit defines the units for the start and end times of the simulation and is considered to be the base unit for the entire model. Using a global time unit can often add to the conceptual understanding of your model, since it allows you to specify any time-based parameter using the unit of time that makes the most sense to you.

You establish a global time unit by selecting and defining a unit of time in the Time units tab of the Simulation Setup dialog (Run menu). You can then 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 seconds, minutes, hours, days, weeks, months or years, regardless of the global time unit.

Once you select a global time unit, all time-based parameters use it by default as their local time unit. The default time unit for the model is denoted by an asterisk (*) next to the name of the time unit in a block's dialog. As long as it is in default mode, the local time unit will be the same as the global time unit. For instance, if you change the global time unit in the Simulation Setup dialog, it will also change in every 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.

Note 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.

When the model is run, Extend will automatically convert all time-based parameters from their local time units to the global time unit. The conversions are based upon the conversion constants defined in the Time units tab of the Simulation Setup dialog. The constants default to 24 hours in a day, 7 days in a week, 30 days in a month, and 360 days in a year. However you can customize these constants to use whichever values you wish.

For example, assume you specify "days" as the Global time unit in the Time units tab and that you define a day as having 12 hours, rather than the default setting of 24. The label "days*" will appear in a popup list next to each time-based parameter in the blocks in your model. If the parameter in a block dialog is based on hours, select hours from the popup list. When the model runs, Extend will cause that parameter to be divided by 12, converting it into the appropriate value for one day.

Note The popup menu will only be present in the block dialog if a global time unit other than "generic" is selected. Otherwise the words "time units" will appear next to each dialog parameter.

Random numbers

Many blocks in Extend utilize random numbers. For example, the Input Random Number block in the Generic library and the Generator block in the Discrete Event library calculate random numbers or distributions by calling Extend's Random function. You can also specify random settings when you use Extend's Sensitivity Analysis feature, and programmers can access random number functions.

Extend's random functions produce pseudo-random numbers which are based on a repeatable algorithm known as a random number generator. This generates the uniform random numbers used in Extend's distribution functions. These 32-bit functions are seed-based and update their seed after being called. Extend supports two types of random number generators:

- The recommended, default generator known as the minimum standard random number generator. See Numerical Recipes in C, 2nd edition, pp.279. A portable and reasonably fast minimum standard random number generator that uses Schrage's algorithm, initially by Lewis, Goodman, and Miller, using new coefficients by Gerald P. Dwyer, Jr. See 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.

Extend

We highly recommend that you use the first type of generator (default) when you build models. The second generator is used mainly for backwards compatibility. That is, models developed before Extend 4.0 will automatically choose the second type of random number generator to retain results that are consistent with those older versions of Extend. You can specify which random number generator you want to use in the Simulation Setup (see "Simulation Setup" on page A29).

Random seeds

You can enter a seed for the random number generator in either the Sensitivity Setup dialog or the Random Numbers tab of the Simulation Setup command. A seed value of 0 or blank uses a random seed; any other value causes repeatable sequences of pseudo-random numbers. Each block that outputs random numbers generates its own independent sequence. For Extend library blocks that generate random numbers (such as the Input Random Number block in the Generic library or the Generator in the Discrete Event library), you can use the "Use block seed" field to set a separate seed for each block in the model.

Distribution fitting

For cases in which you have empirical data which you want to model using random distributions, Extend offers interfaces to a number of distribution fitting packages. These Companion Products can help you find the statistical distribution that best emulates your real-world data. The Input Random Number block (Inputs/Outputs submenu of the Generic library) has a Distribution Fitting tab from which you can launch a distribution fitting package, analyze empirical data, and determine the appropriate statistical distribution for a given data set. Contact Imagine That, Inc. for additional information on distribution fitting packages and other Companion Products.

Speeding up simulations

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 with smaller time increments. Although Extend runs models at extremely high speeds relative to other programs, it can never hurt to think about speed considerations.

The most important thing to remember about running speed is that one of the slowest parts of running a model is updating your screen. 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.
- Do not keep plotters open when running your model if you are concerned about speed. To keep a plotter closed, select the plotter's dialog tool (□) and, in the resulting dialog, unselect "Show plot during simulation". The Windows menu lists the plotters that are open.
- Close dialogs that are updated while the model is running. The Windows menu lists the dialogs that are open.
- The ReadOut block in the Generic library may cause the simulation to run more slowly. This is because the ReadOut block pauses while it shows you information. To speed it up, make the "ticks" count in the dialog zero. You can also unselect the ReadOut's "Show dialog" option if you don't want to see the dialog during the entire model run.
- Increase the amount of physical memory (RAM) in your computer. If your computer runs out of RAM, it will use virtual memory. Using virtual memory can slow down your model significantly.
- Calculations using a large number generic blocks connected together can slow a model down. Where possible, replace large "webs" of generic blocks with equation blocks that perform the same calculation.
- For continuous models, consider using larger time increments so that Extend 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 off if you reduce the number of steps.
- As described in "How discrete event blocks and models work" on page P194, the Executive block from the Discrete Event 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 Items 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 you want to set the number of items initially allocated to your best estimate of the number of items which will be created during the entire run. Note that unnecessarily allocating too many items will take up available memory and can also slow down the simulation.

- 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.

If you build your own blocks, see the section on profiling for information on determining what percent of the model running time is spent in any particular block.

Slowing down simulations

There are a few instances when you want a model to run slower in order to debug 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. If the model is still running too fast, click on the Slower button in the status bar: this causes animated models to run even slower. To speed up the model again, choose the Faster button.

Extend

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 from serial calls to the computer's serial ports. To accomplish this, refer to the Time Sync block in the Utilities library.

Debugging hints

Creating a model is not so fool-proof that you can assume that every model will run well on the first try. Extend gives you many ways of checking for problems in your models when you get unexpected results.

Note If you are interested in debugging ModL code in your blocks, see Chapter 5, “Using the source code debugger” in the Extend Programmer’s Reference. This discusses in detail how to debug your own blocks.

Animation

Using the animation in blocks is useful for debugging. If you know how a block is animated (you can tell which aspect will be animated by looking in the block’s help), you can watch the value or items in a block to see if something is not acting as expected. For example, if a block should show items passing through and an item never leaves the block, you can use that information to debug the model. If you run the simulation showing animation, you probably want to have the plot not open; you do this by clicking the dialog tool in the plotter and choosing to not show the plot during simulation. Also, on the Macintosh, animated models will run faster if you specify bitmap blocks in the Preferences command.

If you are using animation and the animation goes by too quickly, you can slow down your model with the Slower button in the status bar. The Slower button only slows down animated models. Speed up animation using the Faster button.

If you have animation turned on in your model, the Step Next Animation command in the Debugging command found in the Run menu tells Extend to step until the next animation change. In models where there are many steps between animation changes, this option makes going from visible change to visible change much faster.

Model reporting

The report commands in the Run menu allow you to generate custom reports of model data. The reports tell you the final values for parameters of every chosen block, as well as the information in each block's comments field.

If you build your own blocks and want to use the Report features for debugging, you need to add special code to the blocks, as described in Chapter 4: ModL Programming Techniques.

For more information about model reporting, see “Model reporting” on page E187.

Model tracing

The model tracing commands act like the reporting commands, but the output is much more extensive since the text file shows the details of block values at every step in the simulation. Model tracing is useful for finding anomalies in your models that happen as the simulation runs.

Because of the large amount of information generated by the model tracing commands, most people don't use model tracing very often. However, you can use tracing effectively to follow a single block or a few blocks to watch for values that do not match your expectations.

The trace is saved as a text file on disk and opens automatically at the end of the run. If the Number of runs field in the Simulation Setup dialog is greater than 1, each trace is added to the end of the file so that you can compare traces from earlier runs.

Tracing commands

The tracing commands, which are under the hierarchical Debugging command from the Run menu, are:



Macintosh



Windows

Tracing commands

- ⇒ You can specify individual blocks for tracing by selecting them and choosing the Add Selected To Trace command. If you want a report on every block, choose Add All To Trace. You can also see which blocks you are reporting on by choosing Show Tracing Blocks; each block in

the trace report displays the word “Trace” on its icon. Trace reports are opened, closed, and edited just like any other text file in Extend.

Steps for tracing

- ⇒ To include blocks in the trace, select the blocks and choose Add Selected to Trace.

If you want to trace fewer blocks in the next simulation run, select the blocks you want out of the trace report and select Remove Selected from Trace; to start over on your selection of blocks, choose Remove All from Trace.

Extend

- ⇒ Choose the Generate Trace command to create a trace the next time you run the model.
- ⇒ Run the simulation to see the trace results. Extend prompts you for a name for the tracing file.

Tracing example

For example, if you select the three teller blocks from the Bank Line model, the top of the tracing report might look like:

```
Extend Trace - 11/28/97 12:06:17 PM
Run #0

Activity,Delay          block number 3. CurrentTime:2.
Present Delay = 2
Arrivals = 1
Departures = 1

Activity,Delay          block number 5. CurrentTime:2.65.
Present Delay = 2
Arrivals = 1
Departures = 1

Activity,Delay          block number 6. CurrentTime:3.3.
Present Delay = 2
Arrivals = 1
Departures = 1
```

The trace shows that items are leaving the blocks approximately 0.65 minutes apart.

If you build your own blocks and want to use the Trace features for debugging, you need to add special code to the blocks, as described in Chapter 4: ModL Programming Techniques.

Blocks for debugging

The Generic and Discrete Event libraries have many blocks that are useful for debugging. Those blocks include:

Extend

Block	Library	Type	Use
ReadOut	Generic	Inputs/ Outputs	Displays the value of the input connector at each simulation step. You can watch the numbers output from another block on the ReadOut block's icon or in the dialog. In the dialog, set the time between displays.
Stop	Generic	Inputs/ Outputs	Stops the simulation when its input goes above or below a stated level and displays a message you set in the block's dialog.
Sound	Generic	Inputs/ Outputs	Plays a sound when the input is above the threshold given in the dialog.
Information	Discrete Event	Information	Displays information (arrival time, priority, and attributes) about the items that pass through it.
Show Times	Discrete Event	Information	Displays when the next event will occur for each block in the model. Watch this block's dialog during a simulation to see when each block will post a new event.
Status	Discrete Event	Information	Displays information about an item or values coming from another block. This is like the Information block except that it also shows the number of items present at the beginning of the event and the number currently available.
Timer	Discrete Event	Information	Displays the time that it takes an item to pass through a portion of the model.
Record Message	Utilities	Utilities	Records the messages sent over value connectors.
Record Item Messages	Utilities	Utilities	Records the messages sent over item connectors.

The Statistics library (available with Extend+BPR or Extend+Manufacturing), can also be used to find modeling problems. These blocks present parameters for all blocks of a certain class and are useful in finding, for example, queues that are consistently too long.

Remember that you can have many plotters at any point in your model. If you want to track values over time for debugging, add a plotter to your model and hook it up to values that you want to track. Although most models only have plotters at the right side of the model, you can put a plotter at any place you want.

Notebook

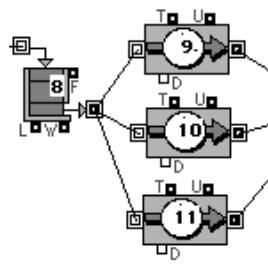
The Notebook, described in “Notebooks” on page E79 and E160, is a convenient way to see the final values for many of the dialog items in a model. This is useful in debugging because you can

group the dialog items in the Notebook by what you expect their final values to be. You can then quickly see which ones have final values that look incorrect.

Show Simulation Order command

Extend normally determines the order that blocks in a model are executed by following the path of connections. If your model is responding in a way you do not expect, you may want to see explicitly the order Extend is using to execute blocks in a model. The Show Simulation Order command puts a small number on each block indicating its order in execution. For example, you might see:

Extend



Simulation order shown

Find Block command

If your model is large, it might be difficult to find all the blocks by sight. For example, you may see in the file created by the Report block that a particular block didn't get the input you expected. The Find Block command lets you locate a block by its name, number, type, or label, or locate a text block by entering some text from the block.

Blocks and text in a model are numbered uniquely and sequentially from 0 to n-1 as they are added to the model window. This sequence does not change. If a block or text is deleted, its number becomes an “unused slot” which is available when another block or text is added to the model window.

There are two numbers associated with blocks. *Global numbers* access all blocks, including blocks inside of hierarchical blocks. *Local numbers* access a hierarchical block's internal blocks and are the same for all instances of that hierarchical block, whereas the global numbers are different for each instance. Blocks are numbered from 0 to NumBlocks-1. Block numbers show in a dialog's title bar.

Block labels are user-definable names given to a particular block in the model. Block labels are entered in the area to the right of the Help button in the bottom scroll bar of a block's dialog. Labels appear at the bottom of the block's icon.

Block labels differ from block names, which is the name you give a block when it is created. For example, Activity Delay is the name of a block in the Discrete Event library. Names appear in the dialog's title bar.

You can use the Find Block command to quickly scroll to a block and select it. The debugging blocks described above always tell you the block number in case you have many copies of a block in your model.

Watching the simulation progress

The status bar at the bottom of the screen has four buttons that help you when you are debugging a model. When the simulation is running, those buttons are:

[Stop] [Pause] [Faster] [Slower]

Debugging buttons while running

When you click the Pause button, the simulation pauses and the buttons become:

[Stop] [Resume] [Step] [Slower]

Debugging buttons when paused

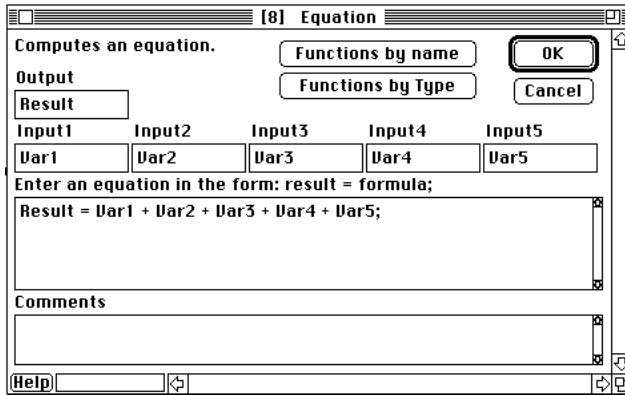
The Step Entire Model, Step Next Animation, and Step Each Block commands in the hierarchical Debugging command from the Run menu tell Extend how far to go when you have paused and click the Step button. The Step Entire Model command starts at the selected block and runs until that block would be run again. This is a good way for looking at what happens between each time a block is called. The Step Next Animation command causes the Step button to go to the next animation event while the Step Each Block command causes the Step button to simply go to the next block.

The Pause at Beginning command in the Debugging command automatically pauses the simulation after the initial processing in the model but before the first step. This gives you a chance to step from the very beginning without having to guess when to click the Pause button.

Equation editor

The section titled “Equation editor” on page E86 explains how you can use an Equation block from the Generic library (Macintosh: Generic Lib; Windows: generic.lix). This block can be a powerful tool in situations where you want to access functions not explicitly represented by Extend’s blocks, or where you want to combine the functionality of several operations into one, or

just want to create custom logic using ModL code (discussed in the Extend Programmer's Reference). Its dialog is:



Dialog of Equation block (Macintosh)

The equation in the Equation block is entered as “result = formula;”. The right side of the equation is the formula. Formulas can combine values, names and variables, operators, functions and procedures, and control statements. These are listed and explained in “Equations” on page P96.

- Values are constants, such as 31 or 5000.2
- Names and variables are used to represent values in the formula. For example, you could name the first input “revenue”, then use “revenue” in the formula just as you would use a value.
- Extend’s operators are +, -, and so on. They are listed in the Equation block help and are discussed fully in “Operators” on page P68.
- The Functions buttons in the dialog display a listing of all built-in functions and procedures, including placeholders to remind you of which arguments are required. You can copy functions from the listing to your equation with the Copy and Paste commands. You must replace the placeholders with the real arguments, and separate arguments with a comma (,). The parentheses are required to show where the arguments begin and end. Extend’s functions and procedures are discussed fully in Chapter 3: The ModL Language.
- Control statements such as “if-else”, “for”, and “do-while” provide even greater flexibility in using this block; they are discussed in “Control statements and loops” on page P70.

Of course there are some rules that you must follow as you use this block:

- Since the connectors are value connectors, the Equation block can only be connected to other value connectors.
- Equations must be of the form *result = formula;* (note the semicolon)

- The entire equation is limited to 255 characters. You can wrap your equation around in the text box, or you can press Return to go to the next line.
- You must assign a name to any input connectors you use in the formula. If an input is connected, you have to use its name in your equation. When you run the simulation, Extend will warn you if you have connected inputs that are not used in the equation. Extend will also warn you if your equation uses a connector that is not named or is not connected. You may use an input name more than once in the equation.
- Names can be alphanumeric, but they must begin with a letter or an underscore character (_). You cannot have spaces, dashes, or other non-alphanumeric characters in a name except for the underscore character. Names cannot be more than 30 characters long.
- Case is ignored, so “ACOS” is read the same as “ACos” or “acos”.
- Spaces between operators are ignored:

`Output = Input1 + Input2;`

is read the same as:

`Output=Input1+Input2;`

- You cannot use user-defined functions or procedures in the Equation block.

If you know some programming, the Equation block becomes even more powerful. You will see in Chapter 2: ModL Code Overview and Block Creation, how to use the ModL programming language in blocks; you can also use that programming language directly in the Equation block.

Sensitivity analysis

As discussed in “Sensitivity analysis” on page E83, sensitivity analysis allows you to conduct controlled experiments to explore how much of an impact a particular parameter has on model results. Extend’s sensitivity analysis features make it easy and convenient to specify the parameter you want to investigate and settings to use for the analysis.

Introduction to sensitivity analysis

Sensitivity analysis works with all numeric parameter entry items (the rectangles in blocks’ dialogs in which you type numbers). It also works with clones of those numeric items. You enter sensitivity settings for a particular dialog parameter by using a menu command or a special key as you click on the parameter or on its clone. 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 your analysis.

Once you have “sensitized” a parameter, you specify how many simulation runs you want. When you run the simulation multiple times with the Use Sensitivity Analysis command (Run menu) checked, you see the results of varying the parameter value over the settings you have chosen.

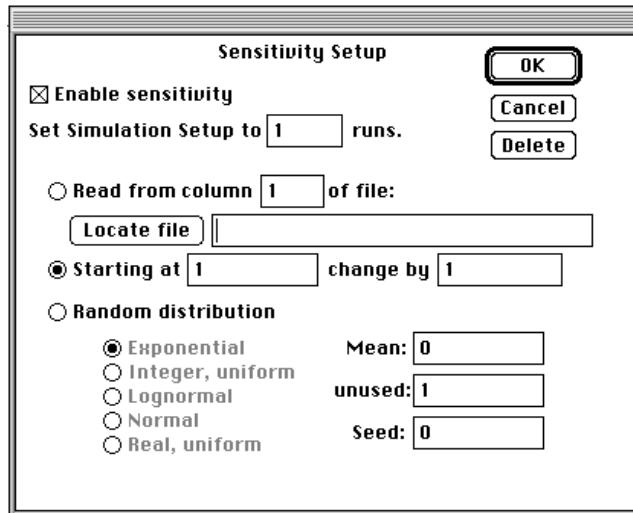
The Open Sensitized Blocks command from the Model menu shows you all the dialogs for blocks that have sensitivity settings. This is convenient if you have entered sensitivity settings for many parameters in a large model.

Steps for using sensitivity analysis

- ▶ Open the dialog of the block that has the value you want to vary.
- ▶ Click on the desired numeric entry and choose the Sensitize Parameter command in the Edit menu or,
 - Macintosh: hold down the Command key (⌘) while clicking once on the desired numeric entry.
 - Windows: hold down the Control key (CTRL) while clicking once on the desired numeric entry, or right-click the entry.

Extend

You see the following dialog:



Sensitivity Setup dialog

- ▶ Be sure the “Enable sensitivity” box is checked. As discussed below, this checkbox allows you to temporarily turn sensitivity analysis off for this parameter without losing the settings.
- ▶ Specify the number of simulation runs needed for the analysis. You can do this either in the Sensitivity Setup dialog or in the Simulation Setup dialog. Each controls the other so that the last value selected in either of them controls the number of runs.

- ▶ Choose the method of sensitizing the parameter (a file, a specified range, or a random distribution) and enter any needed values. The choices are described in “Specifying the sensitivity method and number of runs” on page E228.
- ▶ Be sure the Use Sensitivity Analysis command from the Run menu is checked. If not, choose the command so that a check mark appears next to it.
- ▶ Run the simulation.

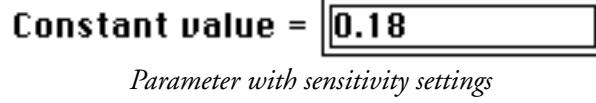
Extend

When you use sensitivity analysis, you are investigating the effect of changing one parameter upon an area of interest. You usually plot the resulting values for that area of interest on one of Extend's plotters. Although you can use any of the standard plotters (such as the Plotter I/O or the Plotter Discrete Event), it is more common that you would use a MultiSim plotter (to show up to four runs at a time in one plot window) or an Error Bars plotter (to show the mean and standard deviation of the parameters over the count of runs).

Example of sensitivity analysis

For example, assume you want to vary the outflow percentage in the “Lake Pollution” model from Chapter 2: Building a Model.

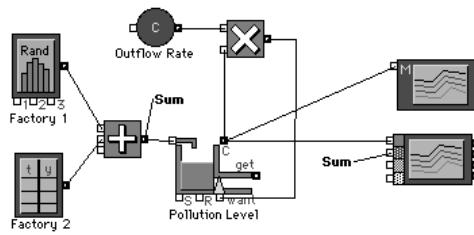
- ▶ Open the “Lake Pollution” model. This is located in the “Examples \ Tutorials” folder.
- ▶ Use the Save As command from the File menu to save the model under a new name. (This will keep your original model from being changed in case you want to refer to it later.)
- ▶ Open the Constant block by double-clicking on it.
- ▶ Hold down the Command (Macintosh) or Control (Windows) key while clicking once on the number “.18” (the number to the right of “Constant value =”). This opens the Sensitivity Setup dialog. In that dialog, choose “Starting at”, leave the initial value of 0.18, and set the “change by” value to 0.05. This will cause the outflow to increase by 0.05 for each simulation run.
- ▶ Set the number of runs to 4.
- ▶ Click OK to accept these settings. In the dialog, you see that the parameter has a border around it. This indicates that the parameter has sensitivity settings:



Leave the dialog open, or click OK to close it.

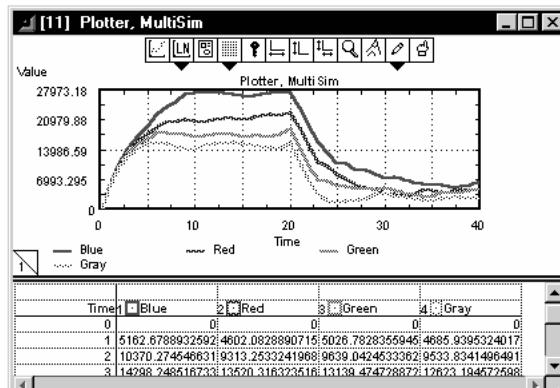
- ▶ Double-click the Plotter I/O and click the Dialog Open tool. In the dialog, choose that the plotter not open. This will keep it from opening and covering your screen as you run the simulations. The Dialog Open tool is discussed in “Plotter dialogs” on page E157.
- ▶ Add a Plotter Multisim block to the right side of the model, and connect it to the *C* connector of the Holding Tank block. This plotter shows up to four runs of data on a single graph, so you can see what impact increased outflow has on the amount of pollution in the lake. Your model now looks like:

Extend



Lake Pollution model with new plotter

- ▶ Choose the Run Simulation command from the Run menu. Extend runs the simulation four times, from run 0 to run 3.
- ▶ Click the icon to rescale the vertical axis in the MultiSim plotter. You can see the variation between the runs in the four lines.



Multisim plotter's results with sensitivity analysis results (Windows)

As you probably expected, increasing the outflow significantly reduced the amount of pollutant in the lake. Note that the level at which equilibrium was reached is reduced, but the shape of the curves is approximately the same.

Specifying the sensitivity method and number of runs

In the Sensitivity Setup dialog you can specify that a sensitized parameter will change incrementally, randomly, or from a list in a file. The three choices are:

Option	Description
Read from file	Assigns the values from a text file. This is the option you will most likely use when you are performing ad hoc experiments. Use this if you have the desired values in a text file that was created in Extend or in another program. If the file has more than one column separated by tab characters, you specify the desired column. Click on the Locate File button to enter a filename to be read from.
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. You can increase the variable with a positive number, or decrease it with a negative number.
Random	Uses a random distribution to set the parameter. This is an easy way to make a single value in your model change randomly over many simulations while keeping the value constant within a single run. Choose from one of the five types of distribution and enter the distribution parameters in the options to the right of the distribution. The <i>seed</i> is the number to use for the random number generator. As in the Simulation Setup dialog, BLANK or 0 for the seed is random.

You can specify the number of simulation runs either in a Sensitivity Setup dialog or in the Simulation Setup command's dialog. Note that the last number of runs entered, in either dialog, determines how many times the simulation will run. If you are a programmer, you can use the CurrentSense variable to control the order of use for the values set for specific runs from within a block.

Turning sensitivity on and off

You can control sensitivity globally (for the model as a whole) or locally (at the dialog parameter level):

- You use a menu command to turn sensitivity analysis on and off for the model as a whole.
- You enable, disable, and delete sensitivity settings for a particular parameter in the dialog in which the parameter appears. 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, you temporarily disable a dialog value's sensitivity so that you don't have to re-enter the number for subsequent analysis.

As discussed above, a parameter that has sensitivity settings has a frame inside of it. For example:

Delay (time units) = 8

Frame inside item indicating sensitivity analysis parameters entered

If sensitivity analysis is active for the parameter (that is, if the “Enable sensitivity” choice is checked, as discussed above), the frame is green or solid black. 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 or dotted grey. The Open Sensitized Blocks command (in the Edit menu) shows the dialogs of all blocks with sensitized parameters, regardless of whether or not the “Enable sensitivity” box is checked.

Extend

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 Setup dialog by holding down the Command (Macintosh) or Control (Windows) key and clicking on the parameter (or select the Sensitize Parameter command from the Edit menu). Then click the Delete button.

Note Editing a sensitized parameter in a block’s dialog disables the sensitivity settings for that block. When this happens, Extend automatically unchecks the “Enable sensitivity” choice. Extend assumes that if you are editing the value, you want to use that new value, not the one that was entered in the Sensitivity Setup dialog. If you want to turn off sensitivity analysis for a number for the foreseeable future, open that item’s Sensitivity Setup dialog and click the Delete button. This will help prevent accidentally changing the value in a future run of the simulation.

Reporting the results

In addition to multisim and error bar plotters, Extend’s reporting and tracing features are useful when analyzing output after using sensitivity analysis. For example, the following (edited) report was generated for the first two runs for the Lake Pollution model:

```
Extend Report - 1/11/97 4:19:46 PM Run #0
Constant                      block number 5
Constant = 0.18
Holding Tank                  block number 3
Final Contents = 6987.2061
*****
Extend Report - 1/11/97 4:19:53 PM Run #1
Constant                      block number 5
Constant = 0.23
Holding Tank                  block number 3
Final Contents = 4117.5383
*****
```

The Reporting features show final values and the Trace feature shows values at each step or event. Reporting is explained in “Model reporting” on page E187 and Tracing is discussed in “Model tracing” on page E218.

Multi-dimensional scenarios

You can enable sensitivity 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 you run the simulation 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 off:

```
5  100
5  150
5  200
...
```

In the Sensitivity Setup dialogs, you choose “Read from file” and choose the file name. For the first variable, enter “1” for the column number; for the second variable, enter “2”.

Extend

When you run a multi-dimensional analysis, you usually use a File Output block from the Inputs/Outputs submenu of the Generic library to write out the values you want to examine. In the File Output block, select “Write at end of simulation” and “run number” so that the values are written at the end of each run.

Optimization Tutorials

The discussion of optimization, starting on page E85, shows that it’s a very powerful feature that can automatically determine ideal values for parameters in your model. It does this by running the model many times with different parameters to search the solution space until it is satisfied that it has found an acceptable solution. The optimizer supplied with Extend uses an evolutionary algorithm to reduce the number of times it has to run the model before a solution is found.

Extend’s user interface simplifies setting up an optimization procedure, making it easy to add optimization to any of your models without using any other applications.

Extend facilitates this by making the optimization algorithm available within a block that is simply added to the model. This optimizer block controls all aspects of the optimization run for the user. 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.

Introduction to optimization

Optimization, sometimes known as “Goal seeking,” is a useful technique to automatically find the best answer to a problem. The “problem” is usually stated as an *Objective function*; equivalent to a cost or profit equation which the modeler is trying to minimize or maximize without going through the tedious job of manually trying different values with each model run. In fact, optimization can find the best values for your model while you sleep, and when you come in the next morning, your best results will be there!

The downside to optimization is that the model will be run many times to find a value that might be easily found manually by the modeler. This can take a long time with large models. Also, there is a small chance that optimization will converge to a sub-optimum result. 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 work, the better chance that you will get the true optimum answer to your problem.

How Optimization Works

Most optimization algorithms that can solve stochastic models (models with a random component) use an initial population of possible solutions. Each solution is tried by running the model several times, averaging the samples, and sorting the solutions. The best solution sets of parameters are used to derive slightly different solutions that might be better. Each new derived solution is called a generation. This process continues for enough generations until there are probably no better solutions in sight, and then it terminates, setting up the model with the best one found.

The problem with all optimization algorithms stem from the inability to tell when the best solution has been found, or even if the best solution has even been attempted. A good approach is to allow the optimization to continue for a “long enough” time and check to see if the population of solutions converge. After that, the user could try the optimization procedure several times to make sure that the answers agree (or are close) and that the first answer is not a false or sub-optimal one.

Steps for using optimization

Here, briefly, are the steps needed to optimize a model. The tutorial, following, will illustrate these steps.

- Open a model that you want to add optimization to.
- Open the Optimization library by going to the Library menu, selecting the Open Library command, and then selecting the Optimization library.
- Place an Evolutionary Optimizer block on the model.
- Define an a cost or profit equation (also called the objective function) that you would like to optimize.
- Determine which parameters you need for that equation.
- Drag the variables that you need onto the closed Evolutionary Optimizer block on the model.
 - For Data Table cells, set the Row,Col (e.g. 0,1) in the Optimizer's Variable table.
- Set the limits for those variables in the Optimizer's Variables table.
- Put the profit or cost equation in the Optimizer's dialog.
- If a variable needs to be constrained by other variables, add constraint equations.
- Set the optimizer defaults for random or non-random model.
- Click on the Results tab and click the “Run Now” button.

Extend

Optimization tutorials

There are two optimization tutorials:

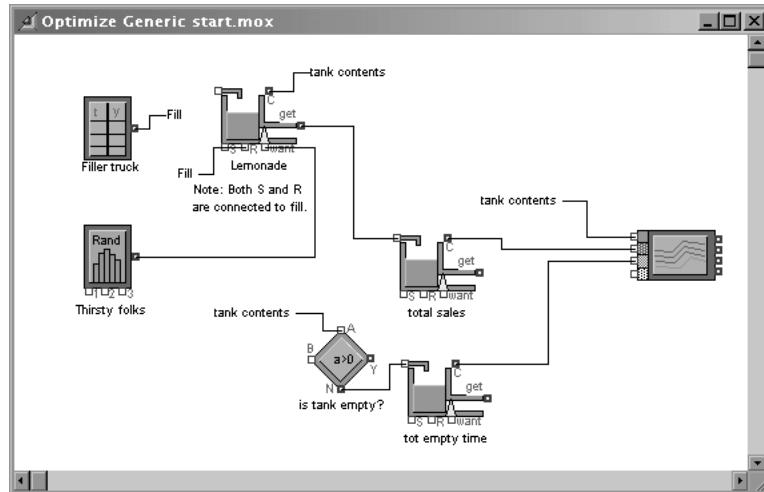
- For continuous modelers, there is a drink stand.
- For discrete event modelers, there is a machining model.

Both tutorials are quick to go through. However, since the models are very different and both have useful modeling techniques, it is advisable for discrete event modelers to go through both tutorials and just skip over the redundant parts of the second tutorial.

Optimization tutorial, continuous

Let's open up the *Optimize Generic Start* model and try setting up an optimization run from scratch.

- Open the “Optimize Generic Start” model in the “Examples \ Tutorials \ Optimization” folder.



Optimization Generic Start model

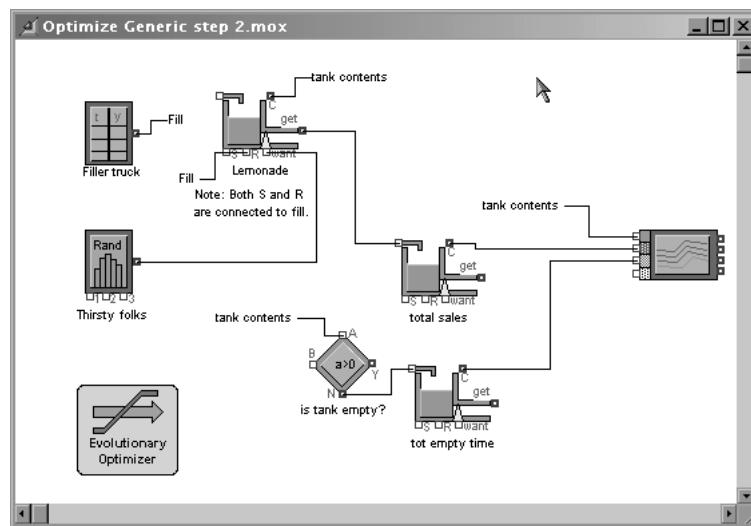
This is a model of a drink stand at a county fair. It has these features:

- A delivery truck comes periodically and exchanges the beverage tank for a new one of a specific size. There is a beverage charge plus a delivery charge to exchange the tank.
- People purchase drinks according to a random distribution.
- If our tank becomes empty, we lose sales because people get mad and go to another stand.
- If we exchange our tank too often, we lose money because the old tank gets taken away, and we lose the beverage inside it.

We want to optimize both how big of a tank we should order and how often the tanks are exchanged. It is a very simple model, but a good example to show some of the techniques that you will use in larger models.

In order to do optimizations, we need to get an Optimizer block:

- Go to the Library menu, open the Optimization library, and place an Evolutionary Optimizer on the model worksheet. Place the block at the lower left corner of the model.



Optimizer block added

The next step involves determining what variables in the model we would like to optimize. In most cases, we would like to minimize a cost or maximize a profit to get some benefit. In this model, we will try and maximize the profit we make at our beverage stand.

Setting up the objective function

We will assign costs to the beverage deliveries, we will assign a revenue to each drink sold, and we will “scold” or reduce the profit if we run out of drinks to sell. We will then derive an equation that specifies our profit for each run, and the optimizer will try and maximize that profit by changing the values of specified variables in the model until it is satisfied that the maximum profit has been reached. And it will do this automatically while we watch. Now, since the optimizer has to run the model many times, having a smaller, quicker running model is always an advantage.

Let's make some assumptions about our model:

- We sell each drink for \$2.50
- We have to make arrangements in advance for delivery size and how often deliveries will occur during the day, so the beverage company will be ready for us.

- Each tank can be from 1000 drinks to 12000 drinks and the cost is \$1 per drink.
- The delivery charge is \$1000 (that's where they get you!), so we need to get the largest tank possible to avoid those high charges.
- The drink delivery truck takes our old tank back... that's right, we lose any beverage we had in it, so we don't want too big of a beverage tank.
- If we run out of beverage, we learned that we lose about \$1000 a minute.
- The model will run for a simulation time period of 480 minutes (8 hours).
- For this model we will ignore other expenses, such as labor.

Extend

This simple stand is turning into a bit of a problem, but let's proceed.

Our equation becomes:

$$\text{MaxProfit} = \$2.50 * \text{sold} - \# \text{deliveries} * (\$1000 + \# \text{drinks} * \$1.00) - \text{time empty} * \$1000$$

Adding variables to the optimizer

In order for our optimizer to calculate this cost equation, it has to have access to some of the variables in the model. This is done by dragging clones of the desired variables onto the icon of the Optimizer block. The procedure is:

- Open a block that contains the variable parameter that you want to use in your cost equation.
- Select the clone tool.
- Drag that variable onto the *closed* optimizer block on the model worksheet
- The optimizer block will highlight when the cloned variable can be dropped onto it.

This operation adds information about the clone to the Variables table in the Set Cost tab and enables the Optimizer block to remotely read or change the value of that variable.

For our model:

- Double-click the Input Data block labeled *Filler truck* and move the dialog so that the closed Optimizer block is also visible on the model. Make sure that the Input Data tab is selected.
- Use the clone tool and drag the *Time vs. YOutput* data table to the closed Optimizer block. The block will highlight when the clone is over the block. Release the mouse when the block highlights.
- Double-click the Input Data block labeled *Filler truck* again and this time, drag the *Repeat every* value parameter (not the label or the checkbox) value onto the closed Optimizer block.

- Open the Holding Tank block labeled *tot empty time* and drag its *Display contents* value parameter onto the closed Optimizer block.
- Open the Holding Tank block labeled *total sales* and drag its *Display contents* value parameter onto the closed Optimizer block.

Now that we have placed our variables into the Optimizer block, we need to enter some limit values for the independent (input) variables so that the optimizer will know what values are legal:

- On the row for *Filler truck* with the variable name *data*, we need to tell the optimizer which cell of the data table to use. Click on the *Filler truck* or the Block number value and the Input Data block will open. Notice that the cell that holds the value of the tank (initially 5000 drinks) is at row 0 and column 1 of the data table, so for the *Row, Col* value in our optimizer block, enter *0,1*.
- For the limits in the *Filler truck data* row, enter a Min Limit of 1000 and a Max Limit of 12000 drinks. Do not enter a decimal point because we want the optimizer to understand that the number of drinks value has to be integral.
- On the row for *Filler truck repTime* (the time between deliveries), we need to enter a Min Limit of 30 minutes and a Max Limit of 480 minutes (our simulation end time).
- On the rows for *tot empty time* and *total sales*, do not enter any limits, because these values are output from the Holding Tanks, and by not putting any limits in those rows, we tell the optimizer that they are output values, not input values.

Note: We only add index values to the Row,Col column when used to access a cell within a data table.

Now that our limits are entered, let's enter the objective function as a cost equation, substituting *VarN* for the Nth row that holds the variable we are interested in:

$$\text{MaxProfit} = \$2.50 * \text{sold} - \# \text{deliveries} * (\$1000 + \# \text{drinks} * \$1.00) - \text{time empty} * \$1000$$

Where do we get the *#deliveries* factor? We need to calculate a value for the number of deliveries, but we only have a *Repeat every n minutes* value. How do we convert the repeat time variable (*Var1*) to the *#deliveries* factor:

$$\# \text{deliveries} = \text{int}((\text{endTime}-1)/\text{Var1} + 1)$$

If we divide the end time value (480 minutes) by the repeat time (e.g. 240 minutes), we will get two deliveries which seems right. But try 250 minutes... you get only 1.92, even though you have one delivery at the beginning of the day and one at the 250 minutes point, or two deliveries! We always have a delivery at the beginning, so we add 1 and truncate any fractional result with the *int()* function, giving us 2.92, which when truncated gives us 2, our correct answer. But what about a delivery every 480 minutes? That would give us two deliveries, except the last one would

happen while we were leaving... so we reduce the end time in the equation by subtracting one, and that prevents the 480 minute case (our maximum limit) from causing two deliveries. Whew...

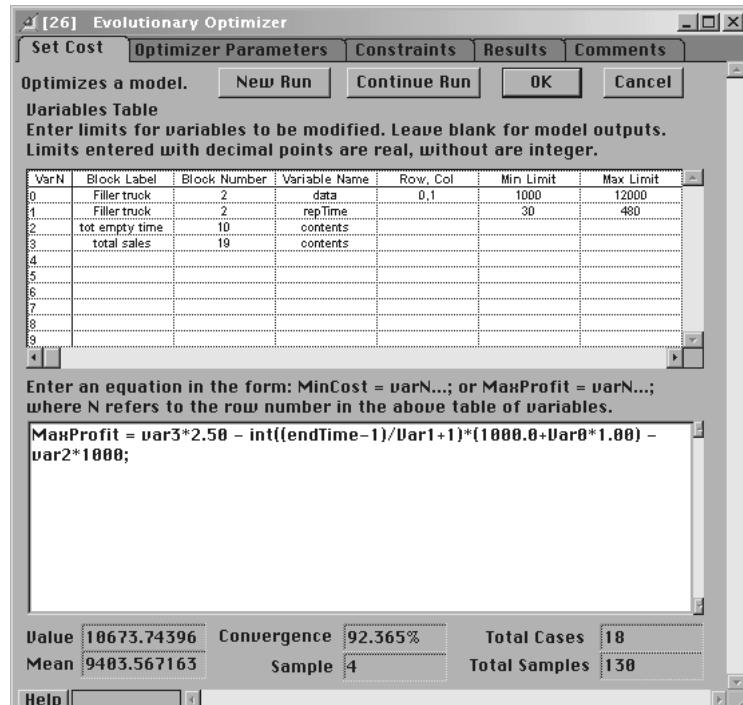
So, with our new #deliveries factor, our equation now becomes:

```
MaxProfit = Var3*2.50 - int((endTime-1)/Var1 + 1)*(1000 + Var0*1.00) - Var2*1000;
```

- Enter the above equation into the text edit box below the Variables table.

Here is our finished Cost tab for the Optimizer block, showing the dragged variables and the equation we need to enter:

Extend



Optimizer dialog Cost tab

We need to set some other parameters:

- Go to the *Optimizer Parameters* tab and click on the *Quicker Defaults, Random Model* button. This quickly sets up all the parameters for a stochastic (random) model needing multiple samples, but limits the number of samples so we can get results more quickly.

Now let's run the optimization:

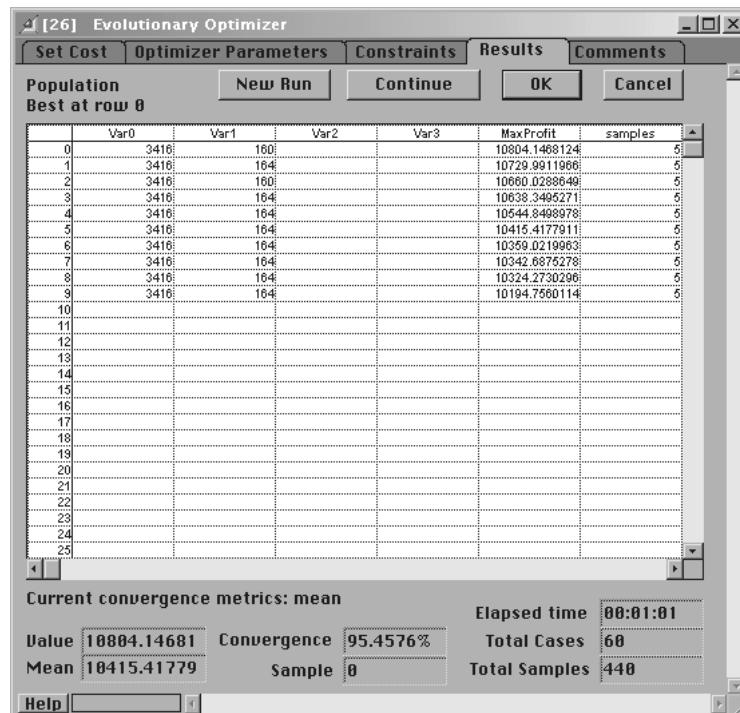
- Open the Optimizer block, click on the *Results* tab, and click on *New Run*, or click on the *Run Optimization* tool on the toolbar.

- Move the plotter over towards the right side of the screen so you can see the *Population* table.

While the run is progressing, notice how the MaxProfit value is increasing towards the top rows of the table. Notice also how the plotter is showing the MaxProfit value increasing and eventually, the convergence value will increase.

Tip: The run will complete faster if you close the Optimizer block (click OK). You can leave the plotter open to watch the values converge without slowing the run down.

When the run is finished, the Optimizer will open and display the Results tab. In any case, when the optimization run stops, the best values will have already been placed into the model.



Optimizer results

In our run, we got:

- Var0 (Filler truck data) = 3416 drinks per tank delivered
- Var1 (Filler truck repeat time) = 160 minutes between deliveries (3 deliveries)
- Var2 (tot empty time) is blank because it is an output value from the model.
- Var3 (total sales) is blank because it is an output value from the model.

- MaxProfit = \$10814.15.

Your results will vary a bit because of the randomness of the model and because we have set our convergence and samples for a *quick* result, but they should be close to these.

Constraints

Now that we have some results, we go to the drink delivery catalog to order and find that they don't deliver 3416 drink tanks! They only have specific size tanks and delivery times. How do we set up our model so that it will give acceptable results?

Extend

We use constraints! We can constrain our parameter values in almost an infinite number of ways by entering constraint equations. Here are the constraints we need to add:

- Delivery tanks only come in 1000, 2000, 3000, 4000, 6000, 8000, 10000, 12000 drinks.
- Tanks can be exchanged every 30 minutes except...
- Tanks greater than 5000 drinks can only be exchanged every 60 minutes or greater.

Here is the equation we need to granularize the drink tank sizes. We use the new variable name NewVar0 because we need to recalculate Var0 without replacing its current value:

```
// round to delivery amounts (e.g. 1k, 2k, 3k, 4k, 6k, 8k, 10k)
if (Var0 <= 4000) NewVar0 = int(Var0/1000.0 + 0.5)*1000.0;
else NewVar0 = int(Var0/2000.0 + 0.5)*2000.0;
```

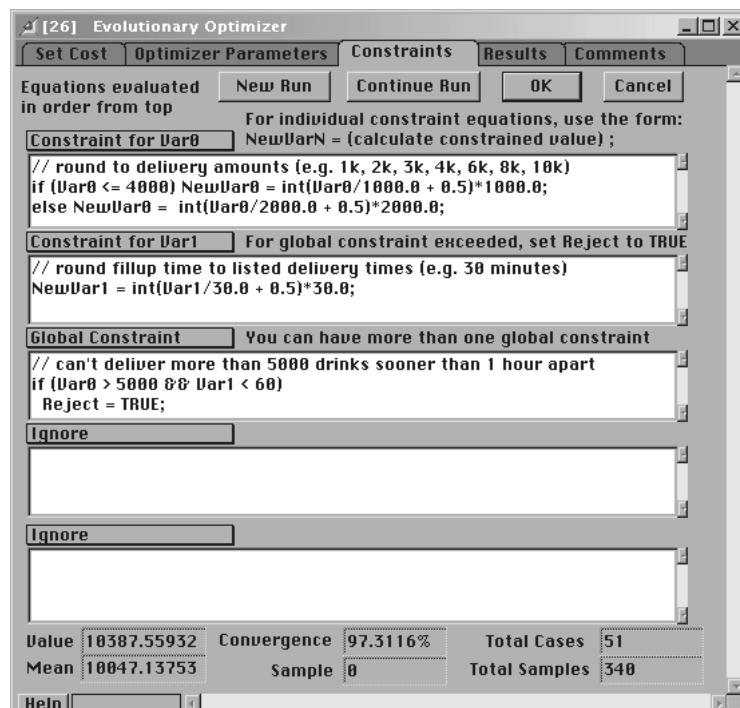
This is the equation we need to granularize the delivery times:

```
// round fillup time to listed delivery times (e.g. 30 minutes)
NewVar1 = int(Var1/30.0 + 0.5)*30.0;
```

We also need to reject cases where the drink tank is greater than 5000 drinks and the delivery time is less than 60 minutes. Here is the global constraint needed to eliminate these cases from running, because we really can't recalculate a delivery time if the delivery time is too short... we don't really know how to calculate a value to replace it with. 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.

```
// Can't deliver large tanks more often than 60 minutes apart
if (Var0 > 5000 && Var1 < 60)
    reject = TRUE;
```

Enter these equations onto the Constraints tab of our Optimizer block. In order to activate the constraint, click and set the correct popup menu choices depending on what variable you are constraining, or if it is a global constraint:



Constraints tab

Try running the optimization with the new constraints. The results show:

- Var0 (Filler truck data) = 6000 drinks per tank delivered
- Var1 (Filler truck repeat time) = 240 minutes between deliveries (2 deliveries)
- Var2 (tot empty time) is blank because it is an output value from the model.
- Var3 (total sales) is blank because it is an output value from the model.
- MaxProfit = \$10373.77.

Note that our profit has decreased slightly from our previous optimum results. This is due to the constraints we have put on the model parameters.

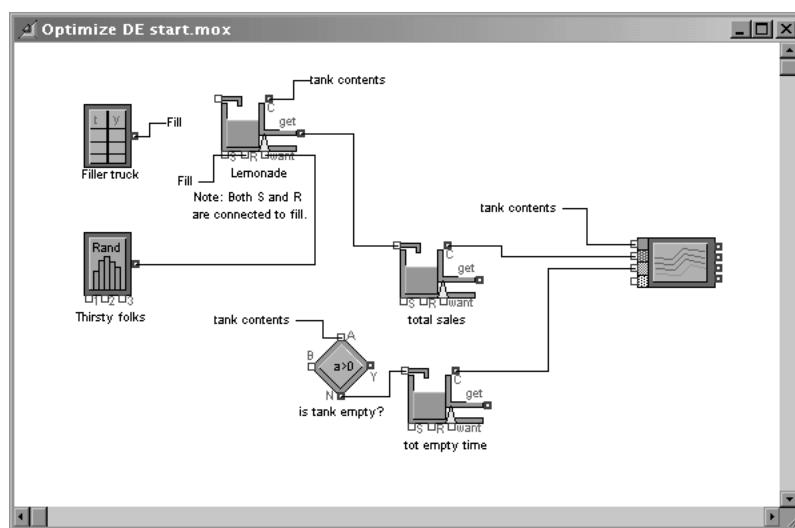
Note: More specific information about setting optimization parameters follow the next tutorial.

Optimization tutorial, discrete event

Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

Let's open up the Optimize DE Start model and try setting up an optimization run from scratch.

- Open the "Optimize DE Start" model in the "Examples \ Tutorials \ Optimization" folder.



Optimize DE Start model

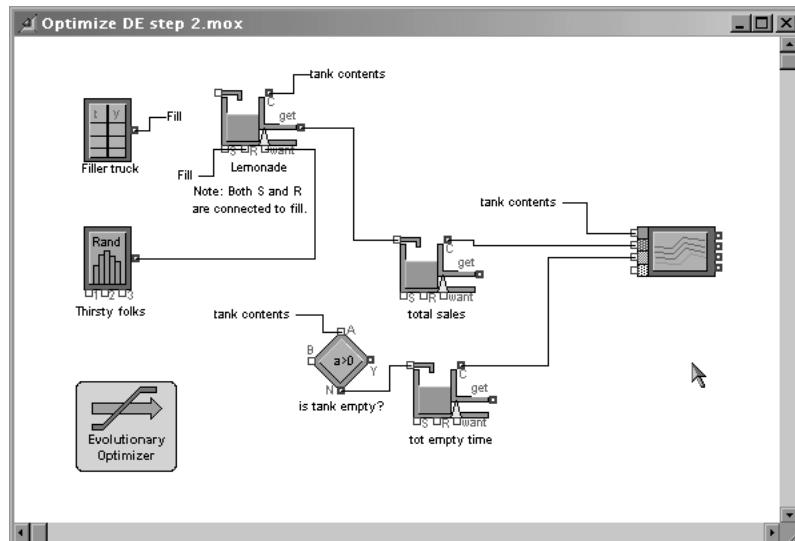
Extend

This model takes some materials, machines them, and then finally polishes them. It is a very simple model, but a good example to show some of the techniques that you will use in larger models.

In order to do optimizations, we need to get an Optimizer block for our model:

Extend

- Go to the Library menu, open the Optimization library, and place an Evolutionary Optimizer on the model worksheet. Place the block at the lower left corner of the model.



Optimizer block added

The next step involves determining what variables in the model we would like to optimize. In most cases, we would like to minimize a cost or maximize a profit to get a benefit from optimization. In this model, we will try and maximize profit.

Setting up the objective function

We will assign a cost to the machining and even to the number of slots the queue needs to maximize production, and we will assign a revenue to each item shipped. Then we will derive an equation that specifies our profit for each run, and the optimizer will try and maximize that profit by changing the values of variables in the model until it is satisfied that the maximum profit has been reached. And it will do this automatically while we watch. Now, since the optimizer has to run the model many times, having a smaller, quicker running model is always an advantage.

Let's make some assumptions about our model:

- We sell each unit for \$5000.00
- Each machine used in the Activity Multiple, for our initial machining, costs us \$100,000.00 per run.
- Our maximum queue length costs us \$10,000 per slot per run (each slot is hand finished by skilled crafts people).
- The polisher costs us \$5,000,000 divided by its polishing time per unit, squared.

That last one needs a little explanation. The polisher costs more if it polishes faster, hence the division by its polishing time. But it costs a lot more if it is even a little faster; and it isn't linear, its squared. So a polisher that works twice as fast would cost four times as much per run. This is similar to microprocessors, as they seem to cost a lot more even if they are just a little bit faster because their manufacturing yield is lower.

Our equation becomes:

$$\text{MaxProfit} = \$5000 * \text{sold} - \$100000 * \text{machines} - \$10000 * \text{slots} - \$5000000 / (\text{polishTime}^2)$$

Extend

Adding variables to the optimizer

In order for our optimizer to calculate this cost equation, it has to have access to some of the variables in the model. This is done by dragging clones of the desired variables onto the icon of the Optimizer block:

- Open a block that contains the variable parameter that you want to use in your cost equation.
- Select the clone tool.
- Drag that variable onto the *closed* optimizer block on the model worksheet
- The optimizer block will highlight when the cloned variable can be dropped onto it.

This operation adds information about the clone to the Variables table in the Set Cost tab and enables the Optimizer block to remotely read or change the value of that variable.

For our Optimize DE model:

- Double-click the Activity Multiple labeled *Num Machines* and move the dialog so that the Optimizer block is also visible on the model.
- Use the clone tool and drag the *Maximum number in activity* value to the closed Optimizer block. The block will highlight when the clone is over the block. Release the mouse when the block highlights. Close the *num machines* block.
- Double-click the Queue FIFO labeled *Holding Slots* and drag the *Maximum queue length* value onto the Optimizer block.
- Open the Input Random Number block labeled *Polishing Time* and drag its *Mean* value onto the Optimizer block.
- Drag the *Exited* value from the Exit block labeled *Sold* onto the Optimizer block.

Now that we have placed our variables into the Optimizer block, we need to enter some limit values for the independent variables so that the optimizer will know what values are legal:

- On the row for *Num Machines*, enter a Min Limit of 1 and a Max Limit of 10. Do not enter a decimal point because we want the number of activities value to be integral.
- On the row for *Holding Slots*, also enter a Min Limit of 1 and a Max Limit of 10. Since the queue length needs to be an integer, do not enter a decimal point.
- On the row for Polishing Time, enter a Min Limit of 1.0 and a Max Limit of 10.0. For this value, we need real values, so enter a decimal point for these.
- On the row for Sold, *do not* enter any limits, because this value is output from the model's Exit block, and by not putting any limits in that row, we tell the optimizer that it is an output value.

Note: We are not adding index values to the Row,Col column because this is used only for accessing a cell within a data table, and we are not using any data tables here.

Now that our limits are entered, let's enter the objective function as a cost equation, substituting VarN for the Nth row that holds the variable we are interested in:

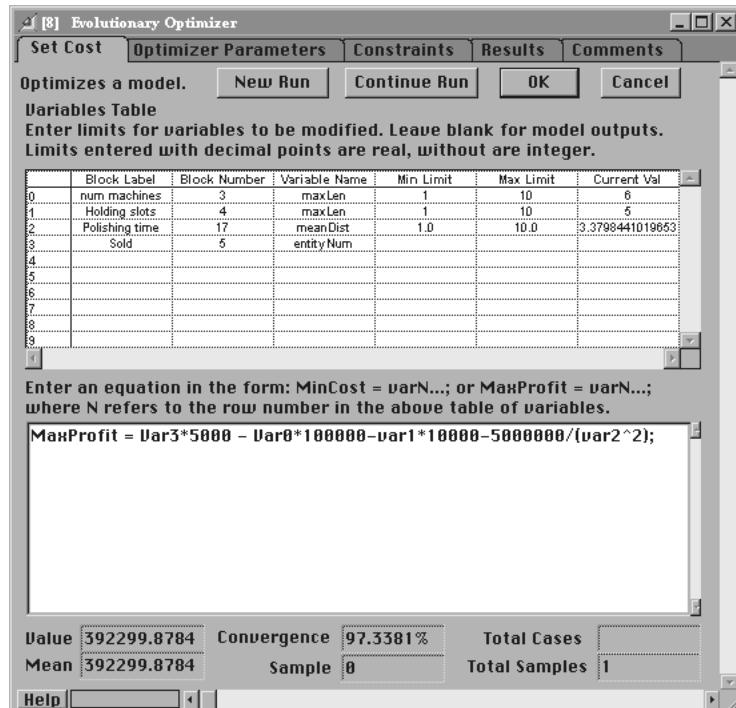
*MaxProfit = 5000*sold - \$100000*machines - \$10000*slots - \$5000000/(polishTime^2)*

gets changed to:

MaxProfit = 5000*Var3 - 100000*Var0 - 10000*Var1 - 5000000/(Var2^2);

- Enter this equation into the text edit box below the Variables table.

Here is our finished Cost tab for the Optimizer block"



Extend

Optimizer dialog Cost tab

We need to set some other parameters:

- Go to the *Optimizer Parameters* tab and click on the *Quicker Defaults, Random Model* button. This quickly sets up all the parameters for a stochastic model needing multiple samples, but limits the number of samples so we can get results more quickly.

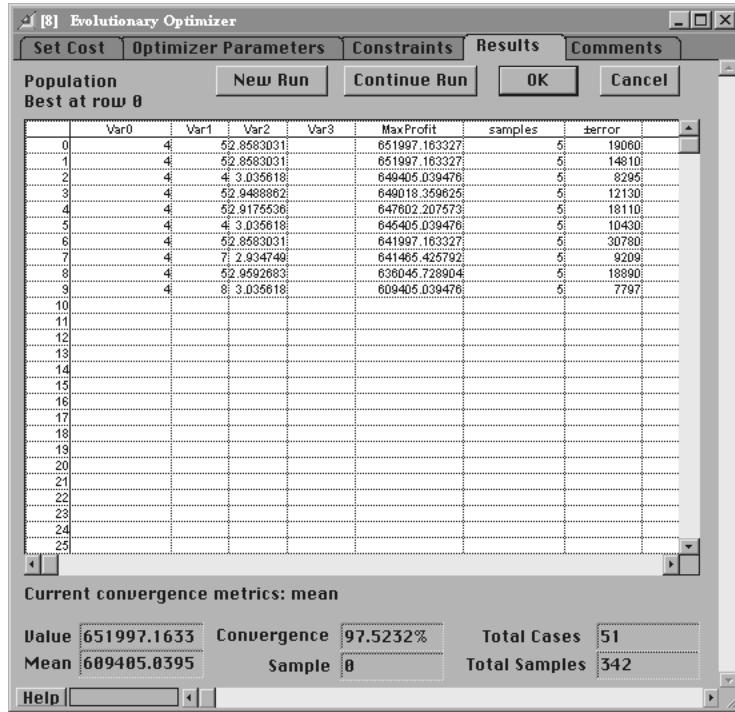
Now let's run the optimization:

- Open the Optimizer block, click on the *Results* tab, and click on *New Run*, or click on the *Run Optimization* tool on the toolbar.
- Move the plotter over towards the right side of the screen so you can see the *Population* table.

While the run is progressing, notice how the MaxProfit value is increasing towards the top rows of the table. Notice also how the plotter is showing the MaxProfit value increasing and eventually, the convergence value will increase.

Tip: The run will complete faster if you close the Optimizer block (click OK). You can leave the plotter open to watch the values converge without slowing the run down.

When the run is finished, the Optimizer will open and display the Results tab. In any case, when the optimization run stops, the best values will have already been placed into the model.



Optimizer results

In our run, we got:

- Var0 (Num Machines) = 4
- Var1 (Holding Slots) = 5
- Var2 (Polishing Time) = 2.858...
- Var3 (Sold) is blank because it is an output value from the model.
- MaxProfit = 651997.16.

Your results will vary a bit because of the randomness of the model and because we have set our convergence and samples for a *quick* result, but they should be close to these.

Constraints

Now that you have some results and you go to the catalog to order a polisher, you find out that they only come in speed increments of 0.5. For example, you can order a 1.0 polisher for

\$5,000,000, a 1.5 polisher for \$2,222,222.22, and a 2.0 polisher for \$1,250,000, but you cannot order polishers with values in between those with 0.5 granularity. What do we do?

We use constraints! We can constrain our parameter values in almost an infinite number of ways by entering constraint equations.

Here is the equation we need to granularize the polishing time. We use the new variable name NewVar2 because we need to recalculate Var2 without replacing its current value:

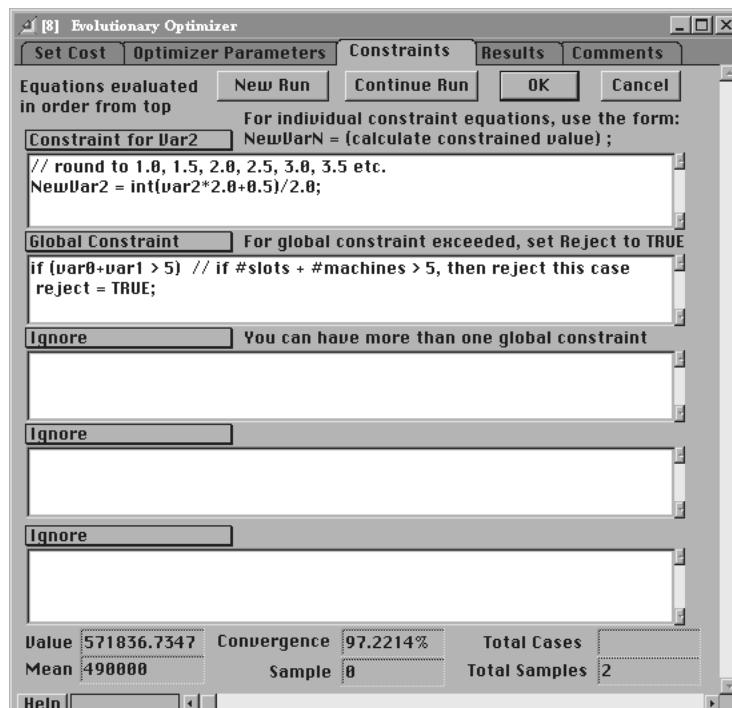
```
// round to 1.0, 1.5, 2.0, 2.5, 3.0, 3.5 etc.  
NewVar2 = int(var2*2.0+0.5)/2.0;
```

Extend

And, here is another experiment we need to try: let's limit the number of machines plus the number of queue slots to 5 or less. This is called a global constraint because we need to reject cases that don't meet our criteria, and they can involve any or all of the variables. The Reject variable, if set to TRUE, will reject that case and cause the block to calculate another possible case. If Reject is not set to TRUE, the case will be tried for the next series of runs.

```
if (var0+var1 > 5) // if #slots + #machines > 5, then reject case  
reject = TRUE;
```

Enter these equations onto the Constraints tab of our Optimizer block. In order to activate the constraint, click and set the correct popup menu choices depending on what variable you are constraining, or if it is a global constraint:



Constraints tab

Try running the optimization with the new constraints. The results show:

- Var0 (Num Machines) = 4
- Var1 (Holding Slots) = 1
- Var2 (Polishing Time) = 3.5 (note the granularity)
- Var3 (Sold) is blank because it is an output value from the model.
- MaxProfit = 580836.73.

Note that our profit has decreased from our previous optimum results. This is due to the constraints we have put on the sum of the number of machines and queue slots. Notice how the optimizer solution favored the number of machines, even though they cost more, because they are more important in generating profit than the available queue slots. These are the kinds of things we can learn from optimization!

Specifying optimization parameters

This section examines, in detail, the parameters used in the Evolutionary Optimizer block.

Variables table

When a dialog's variable is dragged onto a closed optimization block (using the clone tool), the optimization block 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.

Extend

Both lower and upper limits for a decision (optimized) variable need to be entered directly in the Variables Table. Variables that are outputs from the model, such as *Exited* from an Exit block, should not have any limits entered.

Note If the variable is an integer type, such as a number of machines, **do not enter a decimal point** in the limit. Conversely, variables that need to have real values, such as a delay time that could vary from 1.0 to 2.0, need both limits entered **with decimal points**.

Using Data Table cells in the Variables table

If a cloned data table is dragged to the closed optimization block, the *Row, Col* indexes need to be entered, separated by a comma. Since data tables are zero based and start at row zero and column zero, these numbers have to be entered starting at zero for the first cell. For example, to use the first row and second column cell of the data table, enter the *Row, Col* value:

0,1

where 0 is the first row and 1 is the second column. Limits are entered as discussed above, in "Variables table."

Objective functions

Optimizations can maximize profit, minimize a cost, or approach a constant, depending on the form of the equation. 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*dollarsPerMachine;

Or, you might want to write the equation as a cost, and minimize that cost:

MinCost = numMachines*dollarsPerMachine - thruput*dollarsPerUnit;

The variables used in the MaxProfit or MinCost equations come from the Variables Table rows, and are referenced in the equation as VarN, where N is the row number in the Variables Table. In the MaxProfit case, you might code the equation as follows, assuming the first row of the Variables

Table (row 0) holds the clone of a “number of Activities” in an Activity Multiple block, and the second row (row 1) holds the clone of “Exited” from an Exit block:

```
// each shipped unit sold gets $100 and each machine costs $10000  
MaxProfit = Var1*100.00 - Var0*10000.00;
```

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.

Just like the equation field used in the Equation block from the Generic library, equations can be multi-lined, have control (i.e. IF-ELSE, for...) constructs, and call ModL functions. To see more information on equations, see “Equation editor” on page E222.

Special objective functions - 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, we can write a profit equation as (Note that the RealAbs(x) function returns the positive absolute value of x):

For the case that the equation approaching a nonzero constant K is the solution:

```
// maximizes when equation equals constant  
MaxProfit = 1.0 - RealAbs(1.0 - equation/K);
```

For the case that the equation approaching zero is the solution:

```
// this becomes maximum when equation equals zero  
MaxProfit = ConstantValue - RealAbs(equation);
```

Note 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.

Parameters

In most cases, clicking the *default* buttons for the type of model you are optimizing (random or non-random) will quickly set all of the parameters to useful values. The most current explanation of all of the parameters will be in the on-line help for the optimizer block used.

Several of the parameters are more 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. Extend's optimizer block starts the number of samples at 1 and increases them with each generation until it is satisfied that no more are needed, but will never exceed this maximum. If Extend needs more samples, it will state that in the dialog. 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 very precise answer in a noisy model. Increasing this value (i.e. 0.9999) will cause the optimization to continue until the population converges very exactly, increasing the likelihood of a more optimum answer at the expense of run time.

Extend

Constraints

When building a model, there are almost always some parameter constraints that have to be satisfied. For example, there need to be minimum and maximum limits on the numbers of machines available. These simple limits are entered with the variables to be optimized. There could also be a dependency between two or more variables, as the maximum number of machines available might depend on the number of trucks left in a warehouse. There could also be a global constraint, where the total number of machines and people have to be below 50 because of weight restrictions on the factory's second floor. Extend's optimization block makes it easy to apply virtually any kind of constraint to your model's parameters.

Dependency constraints

Dependency constraints are useful to change a decision variable's value if the value has to be limited by the values of other decision variables. Dependency constraints are entered as equations, usually with IF or IF-ELSE statements. They are entered like this example:

```
if (Var4 > 7)
    NewVar2 = Var0+Var1; // only change it if Var4 is too large
```

Sometimes you might need the IF-ELSE form:

```
if (Var2 >= 3)
    NewVar2 = Var3-Var4;
else // var2 was less than 3
    NewVar2 = Var5/2;
```

Or sometimes you need to just modify the value of a variable. In this case we need to constrain its values to multiples of 0.5 (i.e. 1.0, 1.5, 2.0). We 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:

```
NewVar2 = Int(var2*2.0+0.5)/2.0; // 0.5 Granularity
```

In any case the NewVarN value replaces (only if changed) the old VarN value after each of the constraint equations are finished calculating, so the constraints are dependent on each other in calculation order from top to bottom of the constraint tab.

Global constraints

Global constraints are useful to reject an entire case if any or all of the decision variables don't meet a special 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
```

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 found so far is automatically placed in the model.

Hierarchy

The section "Hierarchy" on page E79 introduced how you can use hierarchy in your models to manage complexity. You may remember that a hierarchical block is a special block that usually contains other blocks (even other hierarchical blocks) connected together like a model.

Although hierarchical blocks usually contain submodels, they don't have to. They can also be useful for enhancing and documenting the model. For example, hierarchical blocks can serve as pop-up windows which reveal pictures, text, and so forth when double-clicked.

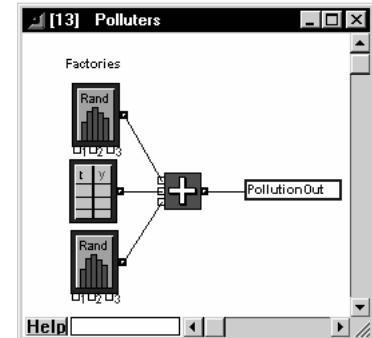
A hierarchical block is unique: it has some characteristics of a block and some characteristics of a model worksheet. Depending on how they are saved, changes to hierarchical blocks may be reflected in all instances of that block (pure hierarchy) or they may be only reflected in that one particular instance of the block in the model (physical hierarchy). Whether a hierarchical block is

considered pure or physical depends on whether you save it to a library and whether subsequent changes are saved to the master block in the library. This is discussed in more detail below.

Hierarchical blocks have two windows: the *layout pane window* (which is what you see if you double-click a hierarchical block), and the structure window. The *structure window* contains another view of the layout pane, and is where you build a new hierarchical block or make changes to an existing hierarchical block's icon, connector position, and so forth. Hierarchical structure windows are discussed in “Building a new hierarchical block” on page E255.

When you open a hierarchical block by double-clicking on it, instead of seeing a dialog, you see the layout of the submodel in the hierarchical window. For example, an open hierarchical block might look like:

Extend



Hierarchical block's layout pane (Windows)

As discussed below, there are two methods for creating hierarchical blocks:

- Select some blocks from a model and choose Make Selection Hierarchical from the Model menu
- Create a new hierarchical block with the New Hierarchical Block command from the Model menu

If you are using hierarchical blocks to organize the complexity of an existing model, you will probably use the first method more often. On the other hand, if you are building a model from the ground up, you may want to create new hierarchical blocks from scratch as well. Both methods are described in detail below.

No matter which method you use to create them, all hierarchical blocks have the following in common:

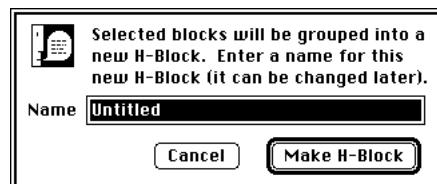
- You can copy them to other areas of your model or to other models and use them as you would other blocks.

- You can change the settings of a block within a hierarchical block by double-clicking on the hierarchical block, then double-clicking on the desired block's icon.
- You can treat hierarchical layout windows like model windows: you can add blocks from a library, create hierarchical blocks, draw items, type labels and other text, clone dialog items onto them, and so on.
- You can modify a hierarchical block's icon, connectors, or help text by selecting the block and using a menu command or by holding down the Option (Macintosh) or Alt (Windows) key and double-clicking on the block's icon. This opens the hierarchical block's structure window.
- Unlike other blocks, 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 hierarchical window without affecting the original hierarchical block. This is also known as *physical hierarchy*.
- You can also 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, if you also choose to update all instances of that block, you have *pure hierarchy*. See "Saving hierarchical blocks in a library" on page E259 for more information.

Making a selection a hierarchical block

To make several blocks into a single hierarchical block:

- ▶ Select the blocks and any desired draw items by dragging over them or by holding down the Shift key and clicking on each of them. (Remember, the selection tool you use determines what gets selected.) See NOTE, below, for information about including named connections in the selection.
- ▶ Right-click or choose Make Selection Hierarchical from the Model menu. You will see the following dialog:

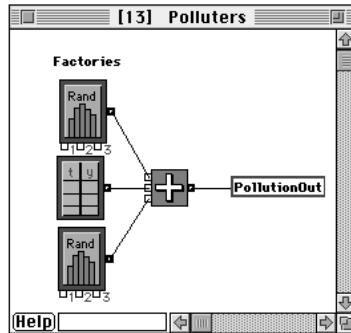


Make Selection Hierarchical dialog

- ▶ Enter a descriptive name for the block and click the Make H-block button.

Note that this action is not undo-able. If you accidentally include more blocks than you intended in the hierarchical block, you must remove them from the block's submodel and put them back in your model using the Cut and Paste commands from the Edit menu.

When you use the Make Selection Hierarchical command, Extend makes all the connections for you and replaces those blocks in your model with the new hierarchical block, including a default icon. To verify that this is so, double-click on the hierarchical block's icon and look at the submodel. The window that holds the submodel is called the *layout pane*:



Extend

Hierarchical block's layout pane (Macintosh)

Note Named connections only work on one level in a hierarchical model: the data will not be transferred between levels. This means that a named connection on one level will not communicate with a corresponding named connection in a block at a lower or higher level. If you include named connections inside a hierarchical block, you will have to provide a connector if the data is supposed to go outside the hierarchical block.

You can alter aspects of the block such as moving the connectors or adding art to the icon. See the section on “Modifying hierarchical blocks” on page E260 for more details.

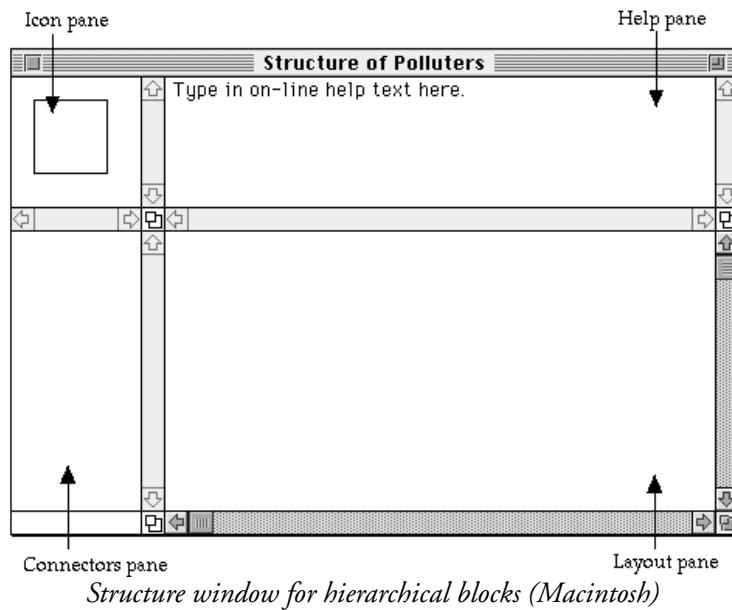
Note that when you make a selection hierarchical, the hierarchical block is saved only in the model. If you want to save it in a library, you must use the Save Block As command, described later in this chapter.

Building a new hierarchical block

To build a new hierarchical block, you must have a model window open. When you select the New Hierarchical Block command in the Model menu, Extend prompts you for a name then opens up a structure window for you.

Extend

The structure window for a new hierarchical block looks like:



This structure window is divided into four *panes*, which are work areas. The upper left pane is the *icon* pane for designing the block's icon and adding connectors. The upper right pane is the *help* pane for writing the help text for this block. The lower left pane is the *connector* pane which contains the names of the input and output connectors for the block. The large pane on the lower right is the *layout* pane, where the submodel that makes up this hierarchical block will be built. The layout pane is what you will see in the hierarchical window when you double-click a hierarchical block's icon.

As discussed in detail below, the steps in building a new hierarchical block are:

- ▶ Build the submodel
- ▶ Add connectors to the icon (and, optionally, modify the icon)
- ▶ Save the hierarchical block. You can also save it to a library, if desired.
- ▶ Connect the new hierarchical block to the other blocks in the model.

Building the submodel

To build a submodel in a hierarchical block, use the same methods as you used when you built a model in Chapter 2: Building a Model. Use the Library menu to add blocks to the layout pane, or drag them from library windows. Connect blocks in the normal fashion.

If you prefer, you can use the Copy and Paste commands from the Edit menu to copy a portion of an existing model into the layout pane to use as a submodel. Once you have built the submodel, you can modify the layout or enhance it with drawing objects or text as discussed in “Modifying hierarchical blocks”, below.

Modifying the icon and adding connectors

The hierarchical block starts with a default icon, a white square. You can modify this icon, for example by changing its shape or color, or you can delete it and create a new one. To do this, draw an icon with the drawing tools in the toolbar, or paste a picture in from another program.

Extend

Since the hierarchical block needs to be connected to the outside world, you must add the appropriate connectors. Some hierarchical blocks have both input and output connectors, while others have just one kind. Also, hierarchical blocks can have either value or item connectors, or both.

Notice that when the hierarchical structure window is the front-most window, the toolbar has additional tools from what you saw in “Tools and buttons” on page E67:



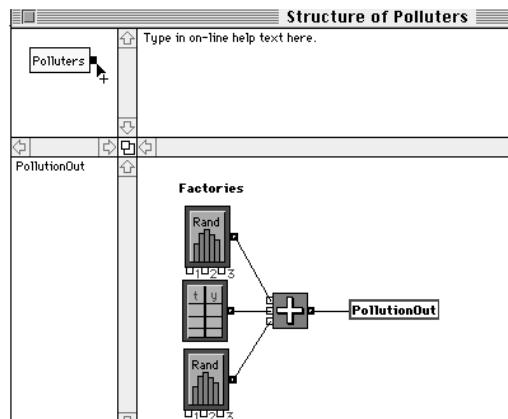
The first new tool on the palette, , is used to add animation objects, as described below. The next four icons add connectors: value () , item () , universal () , and diamond() . (Diamond connectors are used by programmers, as described in “Connectors” on page P15.)

There are four steps to adding a connector to a hierarchical block: decide the type of connector to add; add the connector; determine if the connector should be an input or an output connector; and connect the blocks in the layout pane to the connector.

- First, decide which type of connector to use. You will typically use value or item; the connector type must match the main input and output connectors for the submodel in the layout pane.

Extend

- To add a connector, click on the connector in the toolbar, then click in the icon pane at the desired position near the edge of the hierarchical block's icon:



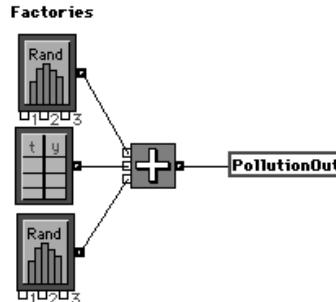
Output connector added to Polluters' icon (Macintosh)

This creates the connector in the icon pane, lists it in the connector pane, and adds a connector text object (very much like a named connection) in the layout pane. The icon pane has a grid; if you want to align a connector without using the grid, hold down the Option (Macintosh) or Alt (Windows) key as you move the connector.

Note If you choose the wrong type of connector, just click on that connector on the icon pane to select it, then click on the correct connector type in the toolbar. The connector will change to the new type.

- Every connector has a unique name. The ending of the name of the connector ("In" or "Out") defines whether it is an input or output connector. When you add connectors to the icon, they are all initially input connectors. To make one of these connectors an output connector, change its name to something that ends with "Out". You can change the connector name to whatever you want, but the name must end in "In" or "Out". To change the name of a connector, select the connector name in the connector pane. Type a new name or edit the name, and press the Enter key on the numeric keypad or click anywhere else in the connector pane to save the edited name.
- Once you have added the connector to the icon, you need to connect the connector text object in the layout pane to the appropriate blocks. You do this the same way you would connect a named connection to a block in the model window: drag a line from the connector of a block in the layout pane to the connector text. When the line thickens, the connection is made.

Named connections are described fully in “Additional ways of connecting blocks” on page E59.



Hierarchical connector connected to submodel

Saving the block

You can save the new hierarchical block so that it is saved just in the model or in a library. If you close the structure window by clicking on its close box (or by choosing Close in the File menu), the hierarchical block will be saved in the model but not in the library. See “Saving hierarchical blocks”, below, for more details.

Connecting the hierarchical block in the model

To connect the finished block to the model, first close the structure window if it is open. Then, simply connect the connectors on the hierarchical block to other connectors in your model, just as you would any other block.

Saving hierarchical blocks

Once you have built a hierarchical block, you can save it only to the model or also to a library.

Saving hierarchical blocks in the model

You can make hierarchical blocks that are saved with the model but which do not exist in libraries. When you make a selection hierarchical, the hierarchical block is automatically saved with the model when the model is saved. If you make a new hierarchical block, or make changes to the layout pane window of an existing hierarchical block, then click its close box to save it, it exists only in the model. You can copy such a hierarchical block to other parts of the model, or even to other models using the Clipboard. Each instance of the block in the model can be then be made unique by modifying it as described below. When you change one hierarchical block, the other blocks do not change at the same time.

Saving hierarchical blocks in 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 later modify that hierarchical block on the model worksheet, the changes may or may not be reflected in the master block in the library. For

example, if you modify the submodel of a hierarchical block from a library, the changes will not be saved in the master library block unless you tell Extend to save those changes, as described below.

To save a hierarchical block in a library or to cause changes made to a hierarchical block to also be made to its master block in a library:

- ▶ If it is not already open, open the structure window of a hierarchical block by holding down the Option (Macintosh) or Alt (Windows) key as you double-click the block's icon in a model window, or select the block and choose Open Hierarchical Block Structure in the Model menu. (The structure window is the active window when you are building a new hierarchical block or when you are modifying a hierarchical block's icon or connectors.)
- ▶ Choose Save Block As from the File menu. In that dialog, choose a library for the hierarchical block, and install the hierarchical block in the library. Then close the hierarchical block and choose one of the save options, as discussed in “Results of modifying hierarchical blocks” on page E263.

Note It is strongly recommended that you do not save hierarchical blocks in the libraries that come with Extend; instead, make a new library for your hierarchical blocks. This is described in “Making a new library” on page E140.

A hierarchical block that is saved in a library has its name listed in the library preceded by a special character. Like other blocks, hierarchical blocks that are saved in libraries list the name of the library in the title bar of their structure window. If no library is listed, the hierarchical block is not saved in a library. You can also tell if a hierarchical block is saved in a library by selecting the block in the model window and choosing Get Info from the File menu.

If you modify a hierarchical block that is saved in a library, the changes may not be saved to the master block in the library. Saving changes to a hierarchical block that is in a library is discussed in the section on “Results of modifying hierarchical blocks”, below.

Modifying hierarchical blocks

How you modify a hierarchical block depends on what it is you want to modify. Typically, you might want to modify the settings of the blocks in the submodel, the layout and appearance of the submodel, or the hierarchical block's icon, connectors, or help:

- You can change settings for a submodel's block by double-clicking on the block's icon in the layout pane window. You can also clone dialog items onto the layout pane as discussed in “Cloning dialog items onto the model” on page E76. You usually access the layout pane window by double-clicking on the hierarchical block's icon, although you can also access it in the hierarchical structure window.
- You can modify the submodel as you would any model: add additional blocks, create hierarchical blocks, type text and add drawing objects, or move blocks and connectors within the hierar-

chical layout pane window. You do this by double-clicking on the hierarchical block's icon to access its layout window. Then, use the Library menu and tool bar to add blocks, text, and so on. This is the recommended method to use if you want to modify the submodel only in this one instance of the hierarchical block.

- You can modify the hierarchical block's icon, connectors, or help in the block's structure window. If it is not already open, open the structure of a hierarchical block by holding down the Option (Macintosh) or Alt (Windows) key as you double-click the block's icon in a model window, or select the block and right-click or choose Open Hierarchical Block Structure in the Model menu. You can also use this method to modify the submodel in the layout pane, although this is more common when you want to save the block in a library. The procedures for making changes are the same as for building a new hierarchical block, described above.

Extend

Note If your hierarchical block is in a library, you cannot modify its structure by double-clicking on its icon in the library window. You must work directly on the worksheet, and you must use a hierarchical block originally copied from the library menu or library window.

You can add many enhancements that will improve your hierarchical blocks:

- Add text, drawing objects, and pictures (layout window or structure window must be open)
- Clone dialog items (layout window or structure window must be open)
- Change the icon (structure window must be open)
- Rename the block (structure window must be open)
- Add help (structure window must be open)
- Add animation (structure window must be open)

To access the layout pane directly, double-click the hierarchical block in the model. To access the hierarchical structure window, hold down the Option (Macintosh) or Alt (Windows) key while double-clicking the hierarchical block in a model window, or select the block and choose Open Hierarchical Block Structure in the Model menu. You can also access the layout pane by opening the structure window, although this is less common.

Adding text, drawing objects, or pictures to the layout pane

You can add text, drawing objects, and pictures to the hierarchical block just as you do in a model. To do this, open the layout window by double-clicking on the hierarchical block's icon, then use the tools in the toolbar to enhance the submodel.

Cloning dialog items to the layout pane

You can clone dialog items from the dialogs in submodel blocks directly to the layout pane. This lets you see the items when you double-click the hierarchical block's icon. The method for doing

this is identical to the normal cloning method you learned in “Cloning dialog items onto the model” on page E76.

Changing the icon

You will probably want to change the icon to better suit the purpose of the hierarchical block, since Extend starts with just a plain rectangle.

Extend

To change the icon:

- ▶ Open the structure window by holding down the Option (Macintosh) or Alt (Windows) key while you double-click on the block's icon, or select the hierarchical block and choose Open Hierarchical Block Structure in the Model menu.
- ▶ Click on the icon in the icon pane, and change or delete it (but be careful to not delete the connectors)
- ▶ Use any of the drawing tools to make the icon. You can also paste pictures from outside of Extend into the icon pane.

Note It is very important that you not delete any connectors. If you do, Extend will warn you. If you accidentally delete a connector, you can undo it or you can add another one, but you must then check the block's connections in the model.

You can change the pattern and color of the default icon without deleting it; just select it and choose new colors and patterns. If you want a different icon shape, you must delete the default icon and draw a new one. See “Drawing” on page E71, for information on how to use the drawing tools. Drawing in the icon window is just the same as drawing in the model window.

Renaming the block

When the structure window of a hierarchical block is open, you can rename the block by choosing Rename Hierarchical Block from the Model menu.

Adding help

You can change the help text by editing in the Help pane in the structure window. You can type in text, or use Copy and Paste to move it from other blocks, or even from other programs. You can change the style of text in the help pane by selecting it and giving commands from the Text menu.

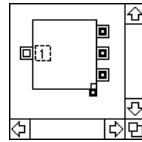
Adding animation

You can add animation to your hierarchical blocks using the Animate Item or Animate Value blocks from the Animation library described earlier in this chapter. To animate a hierarchical block you have to include the Animate Item or Animate Value block in the submodel and create an animation object on the hierarchical block's icon. The animation block in the submodel reads the values from other submodel blocks and controls the hierarchical block's animation based on those values.

To use one of those blocks to animate a hierarchical block, open the hierarchical block's structure window and follow these steps:

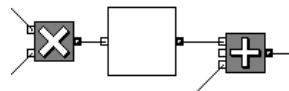
- ⇒ Add an animation object to the icon pane in the structure window (you can have more than one animation object if you wish). Animation objects are rectangular areas in an icon. To include an animation object, click on the animation tool in the toolbar, , then draw the desired size of rectangle on the icon in the icon pane. The animation object is assigned a number starting with 1. For example, a hierarchical block's icon with an animation object near the input might look like:

Extend



Animation object added to hierarchical block's icon

- ⇒ Attach the Animate Item or Animate Value block (whichever is appropriate) to the output of the block you want to animate in the submodel. For example, if you want to animate the result of a multiplication, you might use:



Example of Animate Value block

If you are using the Animate Item block, you must also attach the output of that block to the input of another block. Otherwise an item coming into the block would have no way to exit.

- ⇒ In the dialog of the Animate Item or Animate Value block, enter the number of the animation object that the block is to control. For example, if you want to animate object number 1, you would enter a 1 in the box as shown below:

To animate a hierarchical block, enter its animation object number here =
Specifying animation object 1

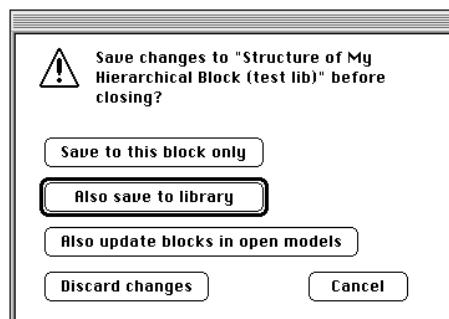
Then choose the type of animation you want. For example, you might choose to show some text on the icon.

Results of modifying hierarchical blocks

There are different results when you modify hierarchical blocks, depending on whether you modify its layout pane window or structure window, and whether the block is saved in a library:

- If you modify a hierarchical block's submodel in the *layout pane window*, 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 effect 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 only apply to that block. This has the same result as when you modify a hierarchical block's layout pane 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 is in a library, you can choose how you want those changes reflected:
 - 1) only in this instance of the block on the worksheet, choose *Save to this block only*
 - 2) also in the master block in the library (only affects blocks placed in the model from the library after the change has been made), choose *Save to library*
 - 3) 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*

When you close the structure window of a hierarchical block from a library, Extend prompts you with the following dialog:



Closing the structure window

Choose “Save To This Block Only” to save the changes just in this block or “Also Save to Library” to make the changes appear in the library holding the master block. “Also Update Blocks...” makes the changes to all copies of that block that came from the library and are used in any open models but not to models that are not open.

Interprocess communication

When modeling, you may want to use spreadsheet data as the input for your Extend models. Likewise you may want to send output data to a spreadsheet for further analysis or presentation. For example, 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 Extend.

The IPC library lets spreadsheet applications be an active part of your models. IPC blocks can send data to and from 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 spreadsheets. The *Data Send*, *Data Receive*, and *SpreadsheetCalc* blocks are designed to allow you to dynamically determine which spreadsheet cell the data is to be sent to or received from. The *Command* block allows you to instruct a spreadsheet application to perform a command or macro.

Extend

For an example of how these blocks can be used, see the IPC example in the “Examples \ Manufacturing or BPR \ Modeling Tips \ IPC” folder.

You can also share data with spreadsheets by using the publish and subscribe (Macintosh) or hot link (Windows) features. These features allow you to share information within block dialog items directly with spreadsheets. See “Interprocess communication” on page E177 for more information about publish and subscribe and hot links.

Discrete event modeling hints

Continuous blocks in discrete event models

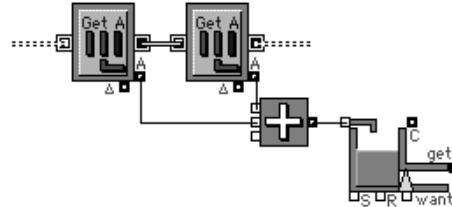
Note Discrete Event models are not available unless you have one of the Extend modules: *Extend+Manufacturing*, *Extend+BPR*, *Extend Suite*, or *Extend+Industry*.

In discrete event models, continuous blocks can be considered passive blocks. In other words, continuous blocks will not recalculate unless told to do so by a discrete event block. As discussed earlier, discrete event blocks may be connected to one or more continuous blocks. When the discrete event block needs a new value at one of its value input connectors, it will send a message out its connector to any connected continuous block telling it to recalculate. Likewise, when a discrete event block has calculated a new value at one of its value output connectors, it will send a message to any connected continuous block telling it to recalculate. These messages are then propagated to all other connected continuous blocks. (See “Value connector messages” on page P207 for a detailed discussion on messaging between discrete event and continuous blocks.)

In some circumstances, it is very helpful to understand this relationship between discrete event and continuous blocks. For example, consider the case where you want to accumulate the result of an equation performed on two or more values coming from discrete event blocks. In the following example, we would like to accumulate the result of the sum of two attribute values. Your first intu-

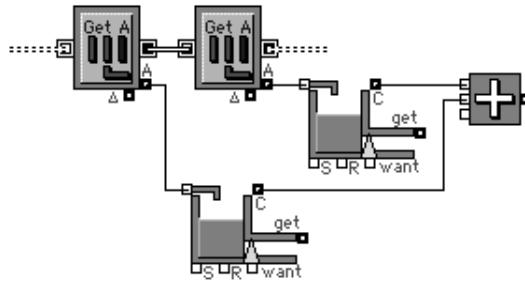
Extend

ition may be to add the two attribute values and send the result to a Holding Tank block as shown below:



Incorrect way to accumulate the sum of two attributes

The above example will give incorrect results in the Holding Tank block. Each time an item passes through one of the Get Attribute blocks, a message will be sent out the “A” connector to the Add block and the sum of the two attributes will be accumulated in the Holding Tank. This means that for *each* item that passes through the Get Attribute blocks, there will be *two* values sent (each one a sum of the two attribute values) to the Holding Tank. The model below will solve this problem:

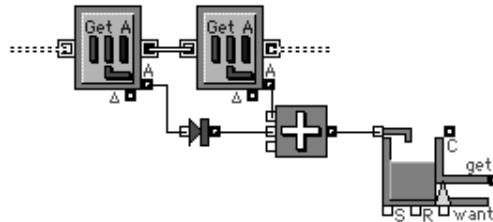


Correct way to accumulate the sum of two attributes

In the above example, each of the attributes’ values are accumulated separately. This prevents the “double counting” of the previous example, because each Holding Tank block receives only one message per item that passes through the Get Attribute blocks. The contents of the Holding Tanks are then added together.

You can also use the Stop Message block from the Utilities library (Macintosh: Utilities lib; Windows: Utils.lix) to prevent the Add and Holding Tank blocks from receiving two messages

for every item. The Stop Message block will prevent the first message from reaching the Add block but will hold the value on its output connector.



Extend

Alternative way to accumulate the sum of two attributes

Customizing animation pictures

As discussed in “Animation” on page E81, items in discrete event models can be represented by animation pictures that travel along connection lines between blocks. All items are initially represented by Extend’s default animation picture, a green circle. However, you can easily cause Extend to use another one of the pictures supplied with Extend or a picture from another source.

Extend is shipped with a number of animation pictures. These pictures are included in the Extensions folder as PICT resources (Macintosh) or bitmaps (Windows). You can also add your own custom pictures created in an external drawing package. The picture must be in the form of a PICT resource (Macintosh and Windows), a bitmap (Windows only), or windows metafile (Windows only). After placing the pictures in the Extensions folder and restarting Extend, the pictures will be made available to be used as animation pictures. Extensions are discussed further in “Extensions” on page P236.

In addition to the pictures in the Extensions folder, there are four pictures (green, yellow, red, and blue circles) programmed into the Extend application itself. These pictures will always be available for use regardless of what pictures are stored in the Extensions folder. This ensures a consistent minimum set of animation pictures with every installation of Extend.

When considering animation pictures, discrete event blocks can be divided into two groups: blocks that generate or introduce items into the model (generators and resources) and blocks that pass or temporarily hold items (passing blocks, queues, activities, etc.).

For blocks that generate or introduce items, you can define how the items that are created in the block will be represented throughout the model. The popup menu on the Animate tab of these blocks allows you to choose an animation picture to represent items originating in the block. Within the popup menu, you will be able to select from all the pictures included in the Extensions folder when Extend was last started. This includes the pictures shipped with Extend, any custom pictures that you may have added to the folder, and the four pictures included within the Extend application. When the model is run, all items that are created in the block will be represented by

the animation picture defined in the popup menu. See the online help for a specific block for a more detailed discussion on the Animate tab.

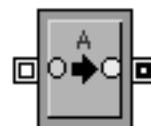
Blocks that pass or temporarily hold items can change the animation picture used by each item that enters the block. This is useful when you want to depict change in an item as it flows through the model. These blocks can change the picture in one of two ways:

- Change the animation picture of every item that enters the block to a specific picture.
- Convert only specific animation pictures to other pictures.

If and how a block will change the animation picture is defined in the Animate tab of these blocks. See the online help for a specific block for a more detailed discussion on the Animate tab.

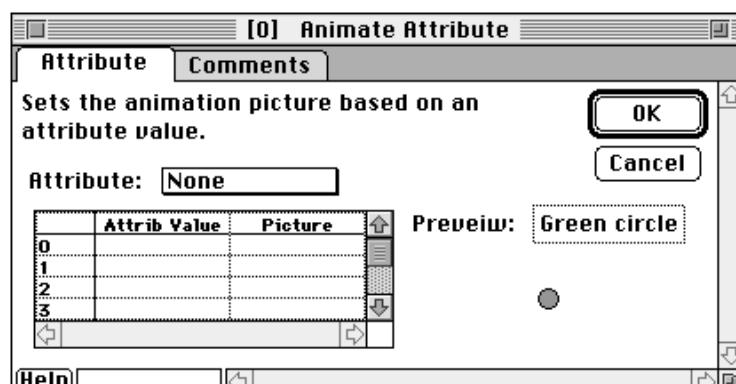
Associating animation pictures with attribute values

There are three blocks whose sole purpose is to allow you to add or customize animation without programming. The Animate Attribute, Animate Item, and Animate Value blocks are in the Animation library. The Animate Attribute block allows you to change the animation picture of an item based upon the value of an attribute.



Animation Attribute block

The block's dialog is:



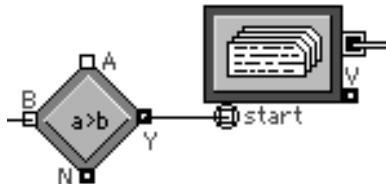
Animate Attribute dialog

You use the popup menu to specify the attribute to be read. The table has cells that are popup menus of choices and is used to define which animation pictures will be used for the different attribute values. When animation pictures are selected from the popup menus within the table, a preview of the picture will appear to the right of the table.

Generating items on demand

The *start* connector on the Program block lets you tell the block when to start sending items out. It is useful for controlling the creation of single or multiple items. If its *start* connector is connected, the Program block will only start its programs when its *start* connector views an item or has a true value (one greater than 0.5). For instance, if you connect the output of a Decision block to the Program block's *start* connector, you can use that decision to create new items.

Extend



Using a Decision block to control a Program block

Notice that when items are provided by the Program block depends on whether the *start* connector is connected to another block or not. If it is not connected, the time specified in the Program block's dialog is absolute simulation time. If the *start* connector is connected, such as in this example, the time in the dialog is relative to when the *start* connector is activated.

For example, assume you specify an output time of 4 in the first row of the Program block's dialog.

- If the *start* connector is not connected, the block sends its first item at time 4. If you start your simulation at a time later than 4 (by setting the start time in the Simulation Setup command), the item in the first line never comes out.
- If the *start* connector is connected, and it receives an item or a true value at time 3, the block sends its first item at time 7.

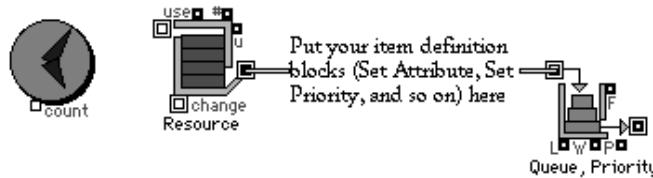
If the *start* connector is activated again before the program specified in the dialog is finished, the activation will be ignored.

Preprocessing

You sometimes want to have all your 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 item would begin traveling through the simulation as the second one was being created. There is an easy method for “tricking” Extend to create lots of items, store them in a queue, and release them.

Extend

Use the following blocks at the beginning of your simulation:



Preprocessing example

Open the Resource block (the first block after the Executive block) and set the initial value to the number of items you want to generate. In the place of the text, put the Set Priority, Set Attribute, and other blocks you might want to use to set up the items.

Put the items in a queue that will order them in your desired order (Queue Attribute for attributes or Queue Priority for priorities).

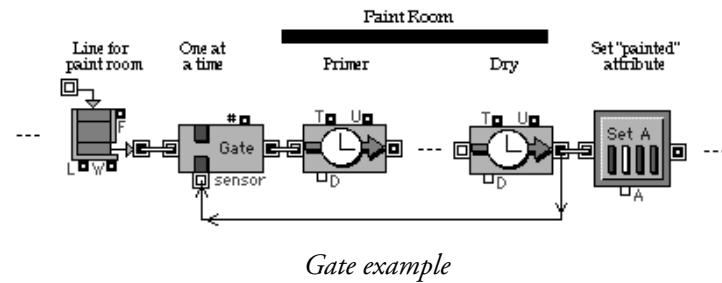
All the items will travel from the resource block to the queue block on step zero. This will assign them all values based on your item definition blocks, and output them into the model. If there are many items in your resource block, you will notice the status bar showing the word “Wait”. As soon as 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 and you have many blocks that represent the steps in the paint room. You can only have two items in the entire paint room at a time. You need a way of restricting the entrance of new items to the room until one or more items leave.

The Gate block is perfect for this because its *sensor* connector tells it when an item has reached the end of a system. The Gate block passes the first item it receives, then only lets an item pass when it sees the first item at its *sensor* connector. 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 *sensor* connector on the Gate block.

The paint room example might look like:



Extend

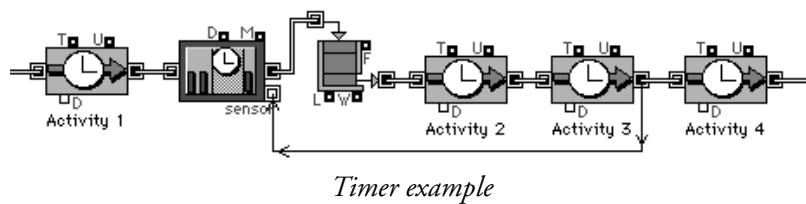
The Gate block only passes an item into the paint room when it sees an item at the end of the paint room, represented by the “Dry” Activity Delay block. Note that the *sensor* connector doesn’t accept any items; it only views them as an indicator. In the model, the item from the “Dry” block is passed to the Set Attribute block on the right; however, the connection to the Gate block’s *sensor* input lets it view the item as it passes. The *sensor* uses that as an indicator to allow the next item into the paint room.

Cycle timing

The time it 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.

The Timer block from the Discrete Event library allows you to specify a section of the model for measuring cycle time. The block adds a time tag to each item passing through and records the time in a table in its dialog. You find how long it takes for an item to get from this block to the output of another one (the target block) by attaching the *sensor* connector of the Timer block to the output of the target block.

How you connect this block in the model determines the section in which the item will be timed. Put the Timer block at the beginning of a system; at the end of the system, run a parallel connection from the output of the target block to the *sensor* connector on the Timer block. For example:



In the example, the Timer block measures the total delay caused by the blocks labelled *Activity 2* and *Activity 3*, as well as the time spent in the queue. The Timer block's dialog displays the total delay or cycle time for each item and the average (mean) cycle time for all items that pass through during the simulation run. This information is also reported at the block's *D* (delay) and *M* (mean) output connectors.

If you want the cycle time to also include the time items wait to be pulled into the block after the Timer block, you should follow the Timer with a FIFO queue (as was done above) so that the item waits in the queue and that wait time will be counted.

Note that the *sensor* connector doesn't accept any items; it only views them as an indicator. The item from the *Activity 3* block is passed to the *Activity 4* block; however, the connection to the Timer block's *sensor* input lets it view the item as it passes. The *sensor* uses that as an indicator that the item has reached the end of the measurement cycle.

Warning: Only one viewing type connector (*sensor* or status block input) can be connected to a given output connector in a model. Subsequent connectors may not get any information and so may not function properly.

Activity-based costing

Activity-based costing (ABC) 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 what factors contribute to the cost of that output. By analyzing the different cost contributors such as labor, materials, administrative overhead, rework, and so forth, you can identify which key cost drivers are candidates for reduction.

Extend can automatically track time-based and event-based cost drivers. The cost drivers are defined in the Cost tab of three different types of blocks: 1) item generators, 2) resources, and 3) activities. If a cost driver is defined in any block in the model, Extend will automatically calculate the accumulated cost for every item in the model. The accumulated cost is automatically stored in the attribute “_cost”. The current storage or resource cost is stored in the “_rate” attribute. These attributes are enabled if costing information (cost per time unit or cost per item) is specified in any block in the model.

The Extend+Manufacturing and Extend+BPR packages (or the Manufacturing or BPR library upgrades to Extend) come with additional costing blocks, more information on techniques, and example models showing how to perform ABC.

Preserving uniqueness in batching and unbatching

Batching allows multiple items from several sources to be joined as one item for simulation purposes (processing, routing, and so on). The batch blocks accumulate items from each source to a specified level, then release a single item that represents the batch. Unbatching is then used to separate items that were batched.

By default, the highest priority that was on any of the items that are batched is transferred to the output item, and all of the input items' unique attributes are combined in the output item. This is true even if the item is subsequently unbatched. As the batched items move through different streams in the model, you can simply ignore the extra attributes that are associated with the item. Thus, if a box of screws is made from a box item and many screw items, the box of screws will have the screw size attributes as well as the box attributes as it travels in the model. If it is unbatched, each screw will still have a size and a box attribute. When looking at the screw, you care only about the size attribute and can ignore the box attribute.

If it is important to not have attributes combined or priorities superseded in batched items when you subsequently unbatch them, you can select “Preserve uniqueness” in the batch block and in the corresponding unbatch block. The “Preserve uniqueness” option marks the items in the batch as unique so that an unbatch block can restore all of the items’ properties (attributes and priorities). This choice uses more memory. If not selected, the unbatching block will create items with merged properties (but will use less memory).

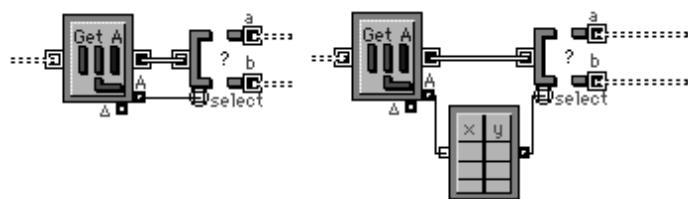
Extend

If you choose to preserve uniqueness, you must be careful when using any property-setting blocks (attributes or priorities) in the path between the batching and the unbatching blocks. Any property modifications you make between the batch and unbatch blocks will be lost once the item is unbatched, since the items are unbatched exactly as they were when they were batched. Also note that the unique identity of the items will only be restored upon unbatching. As the item travels between the batch and the unbatch block, the attributes will be combined as they are by default.

If you need to batch a variable number of items, the Extend+Manufacturing package (or the Manufacturing library upgrade to Extend) comes with additional blocks, more information on techniques, and many examples.

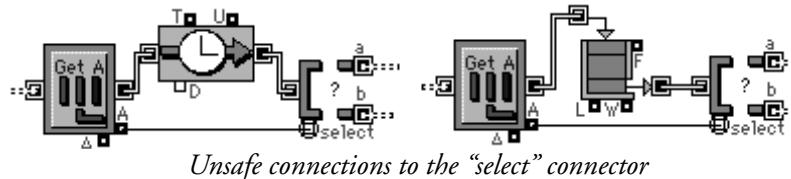
Connecting to the “select” connector

The *select* connector is used to control the behavior of the Select DE Input and Select DE Output blocks. If the *select* connector gets its value from one or more Get blocks (such as Get Attribute or Get Priority), you should avoid putting the following types of blocks between the *select* and the Get blocks: Set blocks (Set Attribute, Set Priority,) activities, and queues. These blocks can alter the value that you use for the *select* input. For example, both of the following model segments will work properly:



Safe connections to the “select” connector

However, the following model segments will not work properly:



Extend

Timing irregularities

Extend's Status Bar at the bottom of the screen assumes that the simulation runs fairly evenly with approximately the same amount of time between events. The value for the timer hourglass sometimes gets confused and 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 "Timer Wait" during the initial steps of a discrete event simulation if there is a great deal of fast activity (for instance, if you use the preprocessing suggestion shown earlier). Do not be concerned about the value in the Status Bar since it is only an estimate: the model is running correctly regardless of the value shown there.

Using scaling for large numbers of items

In many 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 you have 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 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 before, it must now take ten time units to process the same item after the scaling.

Using the Status block to track items

The *C* and *O* connectors on the Status block have similar but distinct uses for tracking items in your model. The *C* connector has a value of 1 when an item is at the watched block's output connector, waiting to be picked up by the following block. The *O* connector, on the other hand, has a value of 1 only at the instant that the item appears at the output connector; the value returns to 0 immediately thereafter. Thus, you would use the *C* connector to see if an item is being held up by the block following the watched block, but you would use the *O* connector to note the first moment an item was ready for the following block.

Continuous modeling hints

Using plotters as input to other models or applications

Extend blocks store more than just their definitions: they also store all their data. For most blocks, this is not very much information. The plotter blocks, however, store every point that was plotted in the table at the bottom of the plotter. After running a simulation, you can easily access all those points.

There are some instances where you may want to use the plotter data as input to another Extend model. For example, if you have a larger model divided up into many smaller models in different files, you may want to use this technique. To quickly pass data from one continuous model to another, you can use the Plotter I/O block. That block has four output connectors that correspond to the four input connectors. To get data out of the plotter into another model, simply select the block, copy it to the Clipboard with the Copy command, select the second model, paste the plotter at the beginning of the second model, and hook up the plotter's outputs to the places where you need the data.

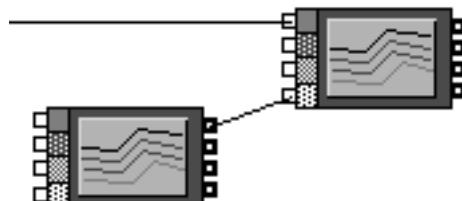
Extend

If you want to use the data in the plotter in another computer application, open the plotter, click on the columns of data that you want, copy it to the Clipboard with the Copy command, move to the other application, and paste the data there. If you want less than a full column, you can select the cells by clicking and dragging.

Using a plot line as a reference or standard

You may want to plot the current results of a continuous simulation relative to a previous run of a simulation that you are using as a standard. This is fairly easy with the Plotter I/O block. First, run the simulation to get the line that you want as a reference in the plotter block. Select the block, choose the Duplicate command in the Edit menu. This duplicates the plotter. Connect the output connector that is associated with the line you want to the input connector on the original plotter block. When you run the simulation again, the data that generated that line will be plotted as a reference line.

For example, assume that you want to replot the line that came from the top input connector of a plotter. Following the steps above would make your model look like:



Second plotter for standard line

You could also paste data directly into a column of a plotter and use that as a reference. If you want just a straight line for a reference, instead of following the above steps, simply use a Constant block from the Generic library to generate the reference line. You can also generate a reference line from a formula by connecting the output of the Input Function block (from the Generic library) to the plotter.

Note that these approaches are different than using the Plotter MultiSim blocks in the Plotter library. Those blocks allows you to plot the results of several runs of a simulation on a single plot.

Setting dt or the number of steps in continuous models

All discrete event models, because they are event-driven, ignore the number of steps and the time per step. Continuous models calculate model data iteratively; that is, at each step in a series of steps, where delta time (dt) is the time interval for each step. For the simulation results to be correct, the “Time per step (dt)” for a continuous model needs to be small enough to accurately reflect changes that occur in different parts of the model.

In most cases, a delta time of 1 is adequate. However, for model accuracy it may be necessary to set a delta time less than 1. This keeps the time interval of each iteration small, but also results in more steps, making the simulation run slower.

There are many reasons why delta time would need to be less than 1. Feedback loops and stiff equations in the blocks can require a smaller delta time to ensure that all calculations are reflected in the graph. Simulations that are run with too large of a delta time often show values jumping from very high to very low. This is known as instability or artificial chaos. Examples of models where a delta time of less than 1 may be required are:

- Signal processing models need to have their specific time per step (dt) either entered in the Simulation Setup dialog, or have it calculated by blocks in the model. For example, the Filter blocks used in the Noisy FM model calculate the stepsize based on their entered parameters.
- Differential equation models (models with integrators in feedback loops, such as those in the “Examples \ Extend \ Science and Engineering” folder) may need to have a time per step (dt) or number of steps other than 1.
- If you are building your own blocks in process control models, you might set up the blocks so that they have a Stepsize message handler that can calculate the value for the DeltaTime variable, automating this process. See the Filter library for some examples of blocks that do this.

To determine what delta time setting is reasonable, first run the simulation with a delta time of 1 (the default setting). Then run the simulation with a delta time of 0.5 (one half of the original setting). If there is no significant difference between the two graphs, then delta time of 1 is appropriate. If there is a significant difference, reduce the delta time to 0.2 and run the simulation again. Continue halving delta time until you determine a delta time which results in no significant dif-

ferences compared to the smaller delta time. The main idea is to use the largest delta time that will give accurate results, without slowing down the simulation unnecessarily.

If your model contains the Holding Tank block, you should determine whether you need to change the Holding Tank to integrate its inputs, as discussed below.

Integration vs. Summation in the Holding Tank block

The Holding Tank block's dialog gives the option to either sum or integrate its input. In general, a good rule of thumb is:

Extend

- Integrate when the value going into the Holding Tank is based on the 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 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 its 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.

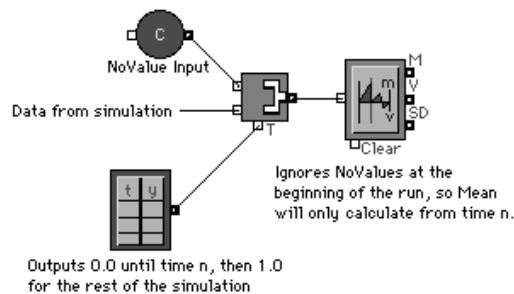
It is also important to note that if you subsequently change the delta time to something other than 0.25, the total amount in the summed Holding Tank would be different from the amounts shown above, but the total amount in the integrated Holding Tank would remain the same.

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, or when the integrating blocks are not interdependent or cross-coupled. The “Lake Pollution” model in Chapter 2: Building a Model is an example of this.
- 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 in Chapter 4: Fundamental Continuous and Discrete Event Modeling is an example of interdependent Holding Tank blocks.

Using NoValues to delay calculations

Outputting a NoValue is useful in some situations because most blocks will ignore NoValues. For example, when you want to start a Mean and Variance calculation only after a certain amount of simulation time has passed, send NoValues to the Mean and Variance block until you want calculations to occur. If you had input zeros into the Mean & Variance block, its sample size would grow, whereas NoValues will be ignored. See the example below:



Example showing how to delay a Mean and Variance calculation until time “n”

Proof Animation (Windows only)

Extend offers an interface to Proof Animation™ by Wolverine Software Corporation<www.wolverinesoftware.com>. Proof Animation is included in the Extend Suite and Industry Suite packages. Proof is a 2.5D animator that provides simultaneous animation during the simulation run. Animation generated by Extend can be “played” over backdrops drawn using Proof or imported

from CAD packages. Large numbers of objects can be moved smoothly over complex backdrops. The end result is a well-integrated simulation/animation suite.

Proof adds many new animation capabilities to Extend which include:

Feature	Benefit
Objects which move smoothly	Activities which include motion are realistically depicted
Constant ratio of viewing time to “model time”	An animation does not speed up or slow down artificially
Directional movement and rotating objects	Objects rotate, rather than sliding as they turn corners
Automatic animation of queuing	Items “stack up” when waiting in queues
Accurate physical scaling	The geometry of the system is represented exactly
Multiple animation views	Zoomed-in and zoomed-out views can be created to focus on areas of interest
Split screen animation	Multiple views can be animated at once
Isometric views	Gives a 3D appearance to the animation

Extend

Proof is designed to work with any discrete event or continuous simulation program. Imagine That, Inc. has added a specific interface to Proof which makes the transfer of simulation information to the animation much easier than it is with other simulation tools. The Proof package includes documentation on how to use Proof to its full capabilities. This tutorial covers only the portion of Proof’s features that are required to build a simple Extend/Proof simulation and animation.

Think of a Proof Animation as a model of the Extend model. Most Extend models contain far more details than can be meaningfully represented on the screen. Accordingly, as a first step, the modeler should decide which system properties are most important and how best to represent them visually. In some cases, additional detail will be needed in the Extend model to make it animate in Proof properly. This can occur when assumptions not critical to the model nevertheless appear unrealistic when animated. For example, in Extend, one Activity Multiple block can be used to represent three servers. In the animation, it would be more realistic to position each of the three servers in a unique location. The servers would then need to be modeled as three individual Activity Delay blocks for the purpose of the animation. There would be no difference in the simulation results between the two models, but additional detail is required to support the Proof animation.

Proof’s layout file contains the objects in the animation, the background of the animation, and paths on which the objects move. The Proof layout is built graphically from within Proof or it can be imported or built by other applications. Extend reads in the Proof layout information and stores the components of the Proof animation internally.

Proof commands can be sent to proof by either clicking on the Proof button in the block’s animation tab or by adding a Proof animate item or Proof animate value block. Using the Proof button

allows only one or two animation options which correspond to the behavior of the block. For example, the Proof button in the Generator block will create a Proof animation object whenever an Extend item is created. The blocks in Extend's Proof interface library (Proof animate item and Proof animate value) are considerably more flexible. Nearly all of the Proof animation command set has been implemented in these blocks and they can be used anywhere in the model. The Proof documentation includes information on all of these commands and on additional features in the Proof program that are not covered here.

Extend

Moving between Extend and Proof

- Open Proof from Extend by selecting the Load Proof menu item from the run menu.

If this option is not available Proof will need to be installed. Proof for Extend is available from Imagine That, Inc. or Wolverine Software.

- To return to Extend from Proof, select the Extend icon in the Proof toolbar. Proof usually stays loaded until Extend is closed. You can unload Proof by clicking on the “Close Proof” button in the control tab of the Proof Animation Control block. This block is in the Proof.lis library.

Building a simple Proof Animation:

This will be a simple “bank line” simulation. Customers will arrive, wait in line, be helped by the teller and leave. We will also animate the number of customers that have left the bank.

Adding a Proof animation to an Extend model requires a Proof layout. In this example, the layout will consist of classes of objects, the teller and the customer; paths, the entrance and the exit; and the background, which includes the lines over which paths are superimposed and a teller class.

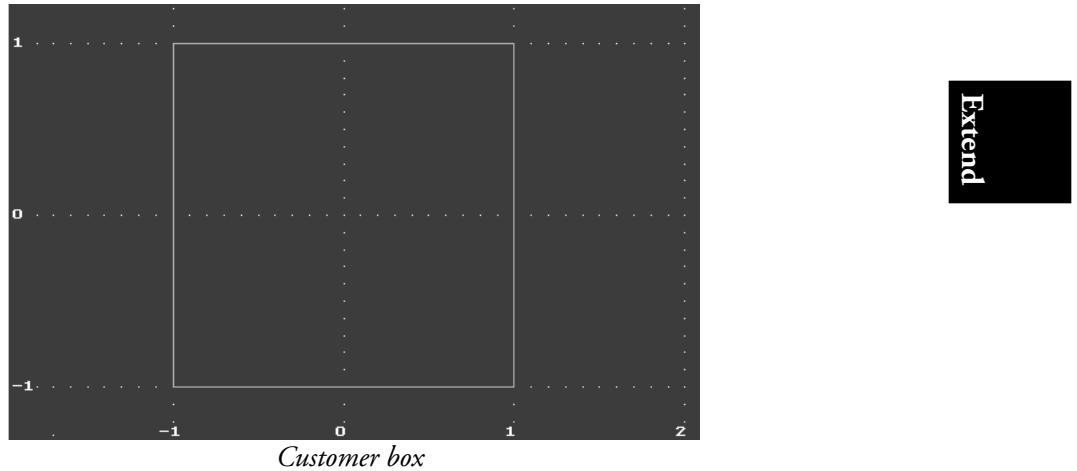
- Select “Open Proof” from the run menu.

The screen may flash when switching between Extend and Proof. This is because Proof utilizes Microsoft’s DirectX technology to obtain the highest possible animation. The first time Proof is loaded, the Proof application will run a series of tests to evaluate the version of DirectX which is installed on the computer. It will also attempt to select the best resolution. If DirectX or the video driver is out of date, it may be necessary to update these files before Proof can be loaded. DirectX can be found on the Microsoft web site at www.microsoft.com/directx. Video card manufacturers generally have a web site which contains updated drivers for their products.

When Proof is loaded, select class from the Mode menu. A class is an animation object that can be referenced by name. Two classes need to be created, one for the customer and one for the teller. To make the animation simple, the customer will be a square and the teller a larger square. In more realistic animations, more detail would be added to the classes or CAD files imported to make them look more realistic.

- Select “New” from the class menu. This will create a new class. Name this class “customer”.

- Select the polyline tool () and draw a box by clicking on 4 points around the origin approximately 1 distance units from the origin (each side will be 2 units long). Hit the “esc” key when done. The customer box should look like:



- In the Class customer dialog, set the fore and aft clearances to 1.2 and click “OK”. Fore and aft clearances determine the spacing between objects which queue up on an accumulating path. If no clearances are specified, objects can pile up on top of one another.
- To create the teller class, select new from the class menu and enter the name teller. Select a different color by clicking on the color swatch in the class dialog. Draw another box with each side 1.2 units from the origin to represent the teller.

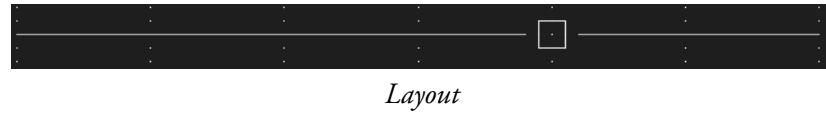
The two classes, customer and teller, that are required for this animation are now completed. The next step is to draw the layout.

- Select “draw” from the mode menu.
- Add the teller by clicking on the Place Layout Object button ().
- Give the object the name teller and select the teller class. Click the mouse approximately two thirds of the screen width from the left side of the screen to place the teller object and click “OK”.

Next, add two lines, one leading into the teller and one away from the teller. These lines will be the foundations for the paths that the customers will follow when they enter and leave the bank.

- To add the lines, click on the Line button (), select a light gray color from the new line dialog, click near the left side of the layout to start the line, and click just to the left of the teller to end the line.

- Draw a second gray line from just to the right of the teller to the right side of the screen and click “OK”. The resulting layout should look like:



Extend

Adding paths

In Proof, two forms of motion are provided, absolute motion and path-based motion. With absolute motion an object moves from its current location to a new location, at a specified speed (or over a specified time interval). The bank model uses the other form of motion, path-based motion. Paths are superimposed over lines and arcs in a layout.

To add the two paths that are needed in this model, select path from the mode menu. Click on the new path button (**+P**), give the path the name “entrance”. Select the “accumulating” option in the new path dialog. The customers will then space themselves out at the end of the line when waiting for the teller. Click on the left line. If the right side of this line is clicked, the path will go from right to left. If the left side of the line is clicked, the path will be from left to right. The direction of the path (indicated by the arrow at the end of the path) should point towards the teller (left to right). If the path points away from the teller, the direction can be changed by clicking on the “reverse active segment” button (**↔**). Use the line on the right side of the screen to specify a path named “exit”. This path should be pointed away from the teller.

Save the layout as bankline.lay. Minimize Proof or go into the run mode and click on the Extend icon in the toolbar.

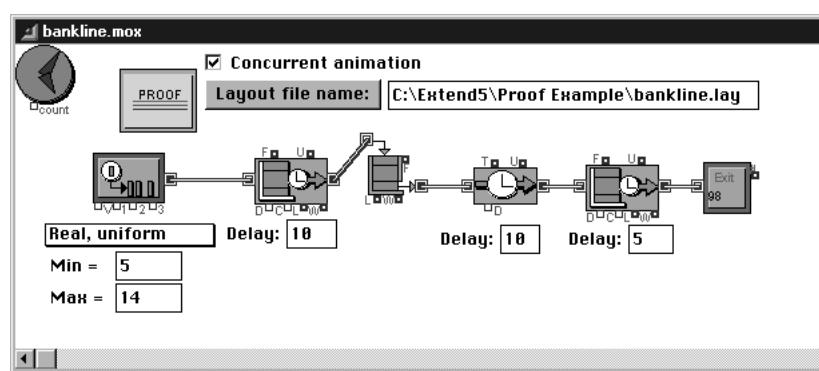
Creating the Extend model

- Generator (de.lix): creates customers
- Arrival rate uniform, minimum: 5., maximum: 14
- Activity multiple (de.lix): to simulate the time that it takes to walk into the bank
- Delay: 10
- Queue FIFO (de.lix): customers wait for the teller
- Activity: customers are helped by the teller
- Delay: 10
- Activity multiple (de.lix): to simulate the time that it takes to leave the bank
- Delay 5
- Exit (de.lix): customers leave the system
- Executive(de.lix): the discrete event clock
- Proof Animation Control (proof.lix): interfaces the Extend model to the Proof animation.
- Select the “concurrent animation” checkbox

Click the layout file button and select the “bankline.lay” Proof animation layout

Add each of these blocks to the model, connect them, and set the dialog values to that shown above. In the simulation setup (Run Menu) set the end time to 480.

This model can be found in the proof\bankline.mox file.

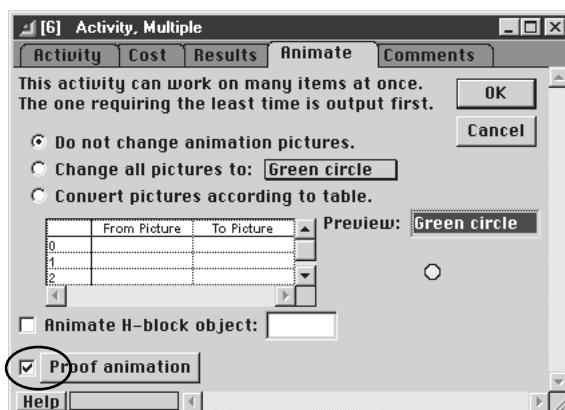


Bankline.mox model

Linking the simulation and animation

To specify animation action in Proof, open the dialog of the Extend block and select the animation tab.

- Check the checkbox next to the Proof animation button. If this is not checked, the animation information is not sent from this block to Proof. Click on the “Proof animation” button:

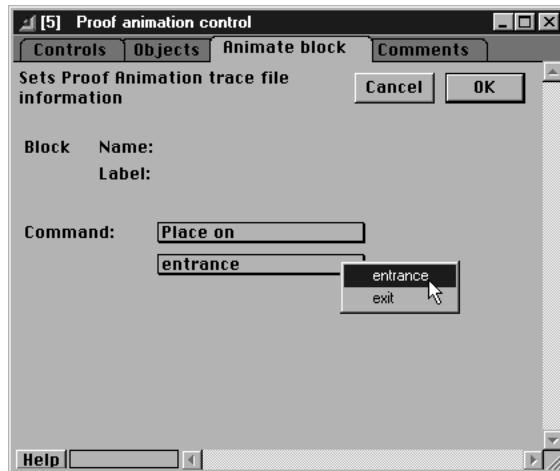


Block dialog set to Animate tab

The next dialog (from the Proof Animation Control block) should then appear.

Extend

- Select the appropriate action from the popup menu:



Proof animation dialog

Add the following animation actions to the model:

Block	Action	Result
Generator	Create customer	Creates a customer in the animation
Activity Multiple	Place on entrance	Places the customer in the entrance path
Activity Delay	Place in teller	Places the customer in the teller object
Activity Multiple	Place on exit	Places the customer on the exit path
Exit	Destroy	Destroys the customer

- Add each of these animation actions to the model.

Run the model. The animation of the system should appear and animate automatically. If the animation does not appear, check the following:

- Is the “Concurrent animation” checkbox selected in the Proof Animation Control box?
- Is the layout file specified correctly in the Proof Animation Control block?

If Proof appears, but does not animate (the title bar may flash), check the following:

- Are the animation actions selected properly for each of the blocks?

If Proof begins to animate, but begins to generate errors, check the following:

- Are the animation actions selected properly?

- Are the time values correct (if the items are too close together, the animation objects may run over each other causing warning messages to occur)?

To add more detail to the animation, add additional objects to the Proof layout file. Use the Proof Animate Item and Proof Animate Value blocks to control the new objects.

Extend

Extend



Industrial and commercial modeling

Extend+Manufacturing User's Guide

Version 5

For Windows or Macintosh

The logo for Imagine That! features the words "Imagine That!" written in a hand-drawn, cursive font. A small registered trademark symbol (®) is positioned at the top right of the "That!" part.

Imagine That!, Inc. • 6830 Via Del Oro, Suite 230 • San Jose, CA 95119 USA
Telephone 408-365-0305 • FAX 408-629-1251
Email: extend@imaginethatinc.com • Web Site: <http://www.imaginethatinc.com>

Manufacturing Introduction

In spite of recent and rapid advances in technology, many companies and institutions still suffer from outdated equipment, inefficient work practices, and a minimum of automation. This is due in part to the prohibitive amount of 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, on the other hand, provides a quick and cost-effective method for determining the impact, value, and cost of changes. Simulation models allow for time compression, are not disruptive of the existing system, and are more flexible than real systems.

A dynamic model of the processes in a manufacturing or service organization is a virtual clone of the real system. It can show how, where, when, and what tasks are performed, as well as who performs them and how much they cost. By exploring alternative strategies, simulation modelers can determine if proposed changes will accomplish corporate goals of increasing quality and customer satisfaction while decreasing cost and process time.

Simulating systems and processes

Systems are composed of mutually interacting elements, resources, and activities forming a complex whole. Systems encompass one or more processes, which are series of activities that achieve an outcome based on the inputs. For the purposes of this manual, a system is that part of the real world that is the subject of interest and processes are the method by which the system functions.

Most systems are composed of real-world elements that interact when specific events occur. Extend+Manufacturing simulates those systems using library blocks which mimic industrial and commercial operations and timing that represents the actual occurrence of events. You can use Extend+Manufacturing to create simulations of manufacturing operations, networks, service industry flows, information processing, material handling, transportation systems, and so forth.

Processes, events, and items

Although this package is called Extend+Manufacturing, as mentioned above it is useful for modeling all types of industrial and commercial systems and the processes that comprise them, from telephone networks to service industries to manufacturing systems. These systems have several things in common:

- They involve a combination of elements such as people, procedures, materials, equipment, information, space, and energy (called *items* in Extend) together with system *resources* such as equipment, tools, and personnel.
- Each process is a series of logically related *activities* undertaken to achieve a specified outcome, typically either a product or a service. Activities have a duration and usually involve the use of process elements and resources.
- Processes are organized around *events*, such as the receipt of parts, a request for service, or the ringing of a telephone. Events occur at random but somewhat predictable intervals and can be economic or noneconomic. Events are what drive most businesses.

Industrial and commercial processes therefore represent the utilization and interactivity of elements and resources driven by events. Some examples of processes, one of the events that might drive them, and the items that flow through or are consumed by the process are:

MFG

Process	Event	Item(s)	Activity
Manufacturing a product	Receipt of raw materials	Parts, labor	Assemble the parts
Filling a customer's order	Customer orders goods	The order	Process the order
Providing customer support	Customer calls on telephone	Telephone call	Route call to technical support
Emergency room admitting	Car 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

Using simulation in industry

To achieve their goals, organizations need to develop an extremely strong competitive position. One way to do this is to improve operational systems and processes by:

- Eliminating nonessential, non value-adding steps and operations
- Implementing and inserting technology where appropriate
- Improving workflow to emphasize value-adding functions
- Identifying key cost drivers for reduction or elimination.

Simulation models provide metrics for meaningful analysis and strategic planning, helping companies enhance their competitive position.

Extend+Manufacturing

Extend+Manufacturing is an object-oriented environment for building comprehensive models of industrial and commercial operations. You use this package to analyze, design, and document manufacturing, service, and other discrete process systems. Extend's iconic building-block paradigm facilitates rapid model building and the communication of ideas. Built-in performance calculations and statistical reports allow you to predict the value, effectiveness, and cost of implementing changes before you commit resources.

Extend+Manufacturing can quickly answer your questions regarding processes and operations. You use it to build a dynamic model composed of iconic blocks, run the simulation over time, and analyze the results. Then change aspects of the model and run it again to perform what-if analyses. Extend+Manufacturing is designed to let you model your current operations and test proposed changes before implementation. The resulting models make it easy to find operational bottlenecks, estimate throughput, and predict utilization. You can also use the models to look at the effects of variations such as labor shortages, equipment additions, and transmission breakdowns.



The Extend+Manufacturing files

Your Extend+Manufacturing package contains two library files and numerous example model files. The libraries are installed into the Extend Libraries folder. The example model files are installed into their own folders or directories, as discussed below.

Note that, in addition to the Manufacturing and Statistics libraries, your models will use blocks from the QuickBlocks, Discrete Event, Generic, and Plotter libraries that come with the Extend package. These libraries are shown in Appendix C.

Example and template files

This package includes detailed manufacturing, service industry, material handling, and networking models as well as model *templates* which illustrate the concepts discussed in the Extend+Manufacturing manual. These files are located in three folders: the Manufacturing, Manufacturing or BPR, and Statistics folders within the Examples folder.

The Manufacturing library

Extend's Manufacturing library (MFG) is an extension of the Discrete Event library that is shipped with the basic Extend package. This library includes blocks that are more powerful and flexible than the standard Discrete Event library blocks, as well as blocks optimized for modeling manufacturing, material handling, transportation, and other discrete systems. For instance, the library incorporates high-level modeling concepts such as variable batching, conditional routing, and preemptive operations as well as blocks that represent machines, labor, conveyors, AGVs, and so forth.

The Statistics library

The blocks in the Statistics library summarize model statistics, provide increased control over random numbers, and remove initial bias for statistical analysis. Some Statistics library blocks report

statistical information for a particular type of block in the model (activities, mean & variance, queues, or resources); others restart statistical calculations, control whether or not the random number seed is reset, or gather cost statistics.

The QuickBlocks library

The QuickBlocks library uses the other libraries and provides instant solutions to common modeling problems in the form of hierarchical blocks. Simply pull in the appropriate hierarchical block from the QuickBlocks library, enter the information for your model, and connect it to the rest of the model. You can even modify the QuickBlock or create new ones in your own library.

How the Extend+Manufacturing manual is organized

Chapter 1 of the Extend+Manufacturing manual shows some example models and typical areas of application. The Tutorial in Chapter 2 shows how Manufacturing and Statistics library blocks are used to build a typical model. Chapter 3 contains a quick overview of the blocks in the Manufacturing and Statistics libraries. Chapters 4 - 12 take detailed looks at the different aspects of modeling with Extend+Manufacturing. The blocks in the Manufacturing library are listed alphabetically and described in detail in Appendix A, while Appendix B does the same for the Statistics library. Appendix C lists other blocks you might use as you build models. As mentioned above, the examples discussed in this manual correspond to the model files included with your Extend+Manufacturing package; the example files (including their names under the Macintosh and Windows systems) are listed in Appendix D.

Before you start with this manual, it is strongly suggested that you read Chapters 1 through 4 in the main Extend manual. Once you are familiar with how Extend works, you will find it easier to understand the examples and concepts in this manual.

Installation and requirements

To install Extend+Manufacturing, follow the Installation Instructions provided with your package. Extend+Manufacturing requires additional hard disk space above what is required by the basic Extend package. The Introduction in the Extend manual lists system requirements for running Extend under the Macintosh and Windows operating systems.

New Extend+Manufacturing features in V5

Along with the new features in Extend V5, we've added some new blocks and examples. To see examples of the new blocks being used, find the block's name in the index of this manual.

- The Shift block (Discrete Event library) allows complex scheduling of resources, activities and routing.
- The DE Equation block (Discrete Event library) is useful in extracting a value from many attributes.

- The Queue Decision block (Manufacturing library) can specify a user defined ranking equation in a queue.
- The Call Center Model. Please see “Call Center model” on page M22.
- The QuickBlocks library is useful to find instant solutions to many modeling problems in the form of prebuilt hierarchical blocks. See the index under QuickBlocks to find examples.

Further reading

Computer Simulation in Business Decision Making, Roy L. Nersesian, New York, Quorum Books, 1989. Illustrates the application of simulation to different facets of planning, including inventory, logistics, personnel recruitment, and so forth.

Computer Networks, Andrew S. Tanenbaum, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1989. Condensed yet thorough exploration of the uses and structure of computer networks.

MFG

Discrete-Event System Simulation, Jerry Banks and John S. Carson II, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1984. A statistical-based exploration of the concepts of discrete event simulation. Emphasizes the application of simulation technique, design of experiments, and analysis of outputs.

Elements of Practical Performance Modeling, Edward A. MacNair and Charles H. Sauer, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1985. A collection of simulation applications in everyday life. A good resource for in-depth, not overly technical information on discrete event applications.

Handbook of Computer-Communication Standards, William Stallings, Ph.D., Howard W. Sams, Inc. (division of MacMillan), Indiana, 1990. Three-volume examination of the 7 layer OSI model, computer network standards, and DOD protocols.

Improve Quality & Productivity with Simulation, Thomas J. Gogg and Jack Robert A. Mott, Palos Verdes, California, JMI Consulting Group, 1995. An excellent overview of the fundamental principles, methodology, procedures, software, mathematics, and benefits associated with simulation modeling. Useful for both experienced and new simulation users.

Probability and Statistics for Engineers and Scientists, Ronald E. Walpole and Raymond H. Meyers, New York, Macmillan Publishing Company, 1989. Among other areas, has sections on confidence interval estimation, probability, random variables, and probability distributions.

Simulation of Manufacturing Systems, Allan Carrie, New York, John Wiley & Sons, 1988. An overview of simulation but heavy on code-based modeling of flexible manufacturing systems (FMS).

Simulation Made Easy, A Manager's Guide, Charles Harrell and Kerim Tumay, Norcross, Georgia, Industrial Engineering and Management Press, 1995. A comprehensive guide to discrete event simulation with the exception that it conspicuously ignores Extend.

M8 | Manufacturing Introduction
Further reading

Simulation Modeling & Analysis, Averill M. Law and W. David Kelton, New York, McGraw-Hill, 1991. More in depth than *Improve Quality & Productivity with Simulation* this book gives an “...up-to-date treatment of all the important aspects of a simulation study”.



Manufacturing Chapter 1: Areas of Application

As discussed in “Processes, events, and items” on page M3, discrete systems have many factors in common. For example, all discrete systems are composed of individual entities or items which flow through the system and utilize resources. Additionally, models of discrete systems often have the same objective: to determine system performance as a measurement of the effectiveness of the existing or proposed system and as a guide to improvement.

In spite of these similarities, particular types of discrete systems also have major differences. For example, the emphasis when modeling manufacturing systems tends to be on equipment issues, whereas service levels is often paramount in service system models.

This chapter discusses the common types of discrete systems:

- Manufacturing
- Service
- Material handling
- Networks

and provides example models for each system type.

Manufacturing systems

Manufacturing systems involve the processing of raw materials into finished product through a series of operations. These operations include a mix of personnel and equipment with varying degrees of automation, where the amount and sophistication of automation is usually inversely proportional to the amount of human participation.

In a discrete manufacturing system, inanimate items (for example, raw materials or subassemblies) are routed through a series of process activities, buffering queues, and storage areas until the finished product is produced. The stimulus for this production is typically either low stock levels, customer orders, or predefined quota schedules.

Types of manufacturing systems

Discrete manufacturing systems can be categorized as:

- *Flow systems* such as consumer products assembly lines or a machining line where parts are processed individually or in batches by a serially-linked group of machines until completion.
- *Job shops* (such as a metal working shop) and *project shops* (such as for custom software) where items are routed for processing depending on item type and resource availability.
- *Cellular and flexible manufacturing systems* in which families of parts are processed by groups of machines.

You simulate manufacturing systems to determine the effect of required or proposed changes. For example: moving production operations usually requires a new facility with new equipment; a new product line might mean adding new equipment to an existing facility; and the need for increased efficiency requires upgrading existing equipment. Whatever the impetus, the major model objectives are to identify bottlenecks, assess capacity and utilization, compare the performance of alternative scenarios, and develop strategies for work scheduling and job sequencing.

MFG

Important considerations

Most manufacturing operations depend heavily on the availability, capacity, and reliability of machines, so it is expected that a major portion of your modeling efforts will revolve around equipment issues. Some important considerations when designing or redesigning manufacturing systems are:

- What quantity and type of machines should be used?
- Where are the buffers located and what are their capacities?
- What is the required throughput?
- How many and what type of personnel are required?
- What are the rules for selecting a particular piece of equipment?
- How long is the setup, maintenance, and repair time?
- Are the policies regarding loading and sequencing of machines adequate?
- What is the cost of producing one unit of output?

Performance evaluation

You simulate the behavior of manufacturing systems to obtain metrics for performance analysis. Some typical measures (including their most common definitions) are:

Term	Definition
Lead time	Time from when raw materials are ordered until the product is delivered to the customer. Also called system cycle time (which differs from “machine cycle time”, below).
Makespan	A subset of lead time, this is the time it takes to change the raw materials into finished goods. Also called throughput time.
Backlog	Number of products waiting to be processed as determined by the buffer lengths.
Processing time	Time it takes for an activity to be performed.
Utilization rate	Ratio of processing time to available time. The idle time rate is 1 minus the utilization rate.
Throughput rate	Number of products produced per unit of time.
Scrap rate	Percentage of defective products produced by an operation. The <i>yield</i> rate is 1 - scrap rate.
Bottleneck	A constraint on production flow usually caused by the resource (equipment, labor, etc.) with the highest utilization rate.
Machine cycle time	Average time the equipment takes to perform one operation. Often includes factors for equipment downtime, operator fatigue, etc.
Work-in-process (WIP)	The level of in-process inventory at any point in time. The product of the throughput rate and the lead time.
Waste	Anything above the minimum amount of resources required to produce the product.
On-time delivery	How close delivery is to the projected delivery date. Note that producing something too early can be as costly as producing it too late.
Accumulated cost	The total cost of all the resources used to produce one unit of output.

Example models

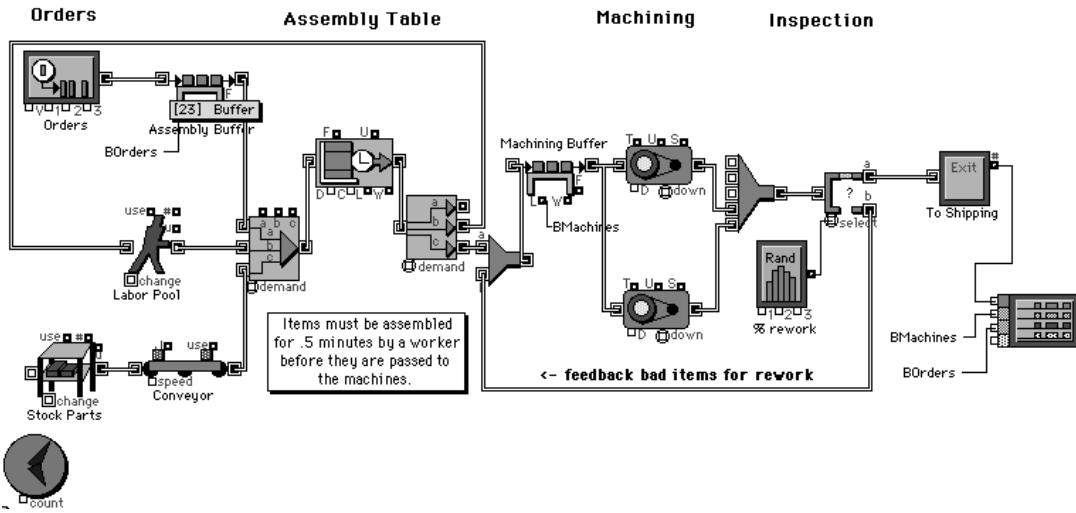
The following manufacturing models are located in the “Examples \ Manufacturing \ Applications” folder.

Assembly-Rework model

The Assembly-Rework model is an example of a simple two phase manufacturing process: an assembly phase where parts from stock are assembled by a laborer; and a machining phase where the assembled unit goes through a machining process that finishes it. The machined part is then

M12 | Manufacturing Chapter 1: Areas of Application
Manufacturing systems

inspected and either shipped or sent back to be reworked by the machines. The purpose of the model is to find bottlenecks in the process.



The particulars of this model are:

- Orders arrive at the rate of about 3 per minute (interarrival time of 20 seconds)
- It takes 3 parts to make each assembly
- Parts are brought by conveyor from the stock room as required; the conveyor moves at 60 feet/minute, has 10 slots for parts, and is 10 feet long
- There is 1 worker who takes 30 seconds to assemble the parts
- Two machines are available; they take 42 seconds to process each assembly
- 85% of the machined assemblies pass inspection and are shipped; the rest must be re-machined

Notice that the Batch block takes one order, one laborer, and three parts from stock and binds them into a single item ready to begin assembly. Since “Delay kit at” is selected for the laborer in the Batch dialog, the laborer will not be pulled into the batch block until the order and parts are ready to be assembled.

The plotter in this model is set to display the number of products successfully finished, the number of items backlog in front of the machining section, and the number of items backlog in front of the assembly process. As you will see when you run the model, the assembly buffer length steadily increases over time. In a real manufacturing system, this increasing level of backlog would indicate a serious problem.

To determine if adding another worker would help, try increasing the number for “initial labor” in the Labor block’s dialog to 2 (this is the only change you need to make, since the Activity Multiple block which represents the assembly process is already set to accommodate up to 6 workers at a time.) When you do this and rerun the model, you will see that the backlog moves to the machining section. This is a common occurrence when modeling these types of manufacturing systems: an increase in capacity in one part of the system can often showcase a weakness in another part.

In this case, a simple solution might be to increase the capacity of the machining system as well. You can do this by:

- Adding another machine to the machining section
- Simulating increased productivity by lowering the processing time of the existing machines

If you add another machine with the same capacity as the original machines, the backlogs should be reduced to acceptable levels.

MFG

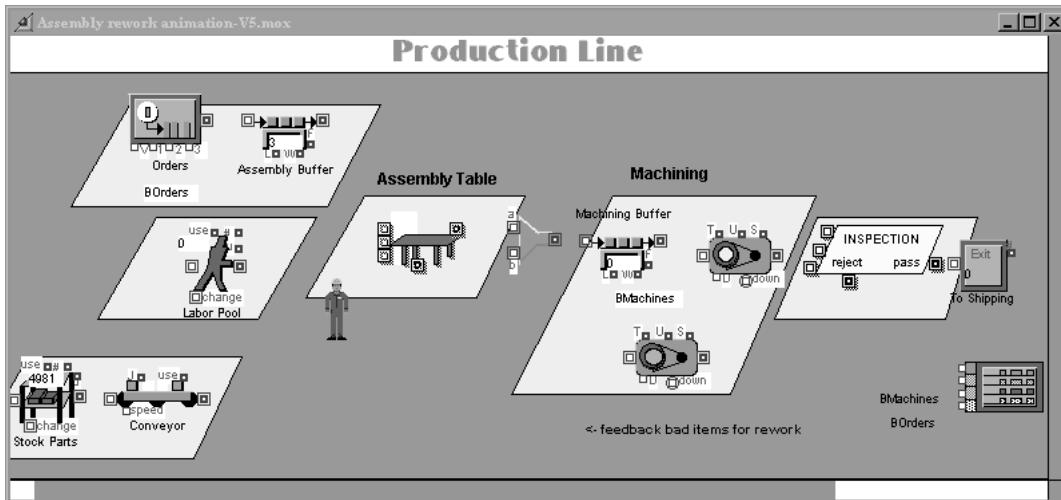
You may have noticed some changes or enhancements that could be made to this model. For instance:

- The orders could require a variable number of parts (a Batch, Variable block is helpful for modeling this)
- A more realistic model would show downtimes for maintenance and/or repairs
- Assemblies that fail inspection might not need to be re-machined for the same amount of time as uninspected assemblies

As with all models, it is important to start small and add only the level of detail that is required to meet your goals.

M14 | Manufacturing Chapter 1: Areas of Application
Manufacturing systems

There is another version of this model called “Assembly rework animation,” which lays out the model in perspective, hides the connections (using the Model menu), and displays animation:

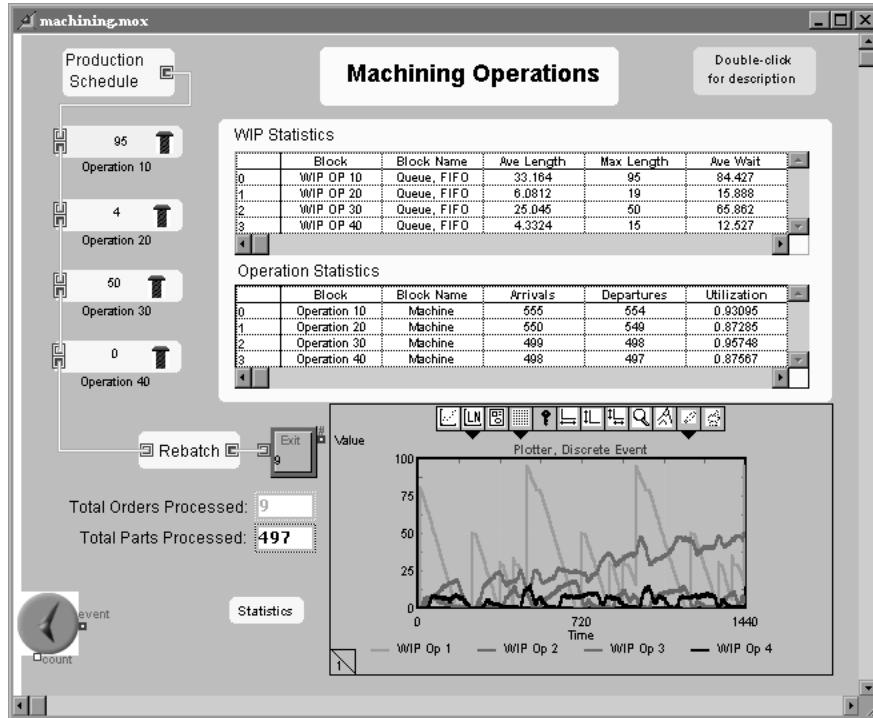


Revised Assembly Rework model

Machining Operation model

The Machining Operation model represents a basic job shop where there are four machining sections which process parts for varying amounts of time based on the part type. Each time a machine changes from processing one part type to another, it requires an additional *setup time*.

before the first new part can be processed. The purpose of the model is to accumulate statistics on the four operations and the buffers between them.



MFG

Machining Operation model

The particulars of the model are:

- Orders arrive on a fixed schedule, where each order requires the number and type of parts specified in the following table (the “Value” is the number of parts; the “Attribute value” is the part type):

Row	Output Time	Value	Priority	Attrib. value
0	0	80	1	1
1	240	50	1	2
2	360	30	1	3
3	420	30	1	4
4				
5				
6				
7				
8				
9				

- There are four machines and four types of parts. Each machine works on all four part types.
- Each machining operation processes part types for a different amount of time. For instance, the time Operation 10 takes for each part type is given in the table below, where “x in” is the part type and “y out” is the processing time:

Row	x in	y out	Up
0	1	2	≡
1	2	2.1	
2	3	1.8	
3	4	2.6	
4			
5			
6			↓

- When a part type changes, the machine takes an additional amount of time to prepare for the change. This setup time is added to the processing time for the first item after the type changes. The setup time for each operation varies from 10 to 15 minutes.

This model uses blocks from the Statistics library to collect and summarize data from various points. For easier viewing, the tables from the dialogs of the Activity Stats and Queue Stats blocks are cloned onto the model worksheet. The Generator blocks connected to the “update” connectors on the statistics blocks control how frequently they are updated. This is the method you can use to remove statistical bias, as discussed in “Other Statistics library blocks” on page M71.

MFG

To further analyze the results of the model, you might try out the confidence interval feature of the Statistics library blocks. To do this, open the dialogs of the Activity Stats and Queue Stats blocks, click on the checkbox labeled “Append New Updates”, and rerun the model. After the run is finished, open the dialog of one of the statistics blocks and click on the “Confidence intervals” button. When you do this, the range of the confidence will be calculated for each one of the reported statistics. Assuming you have created a 95% confidence interval, the result will tell you that there is a 95% probability that the true mean of the model is within the interval displayed. For more information, see “Confidence intervals” on page M202.

Services

While service industry customers may receive tangible goods (a haircut, dental services, clothing, etc.), the real “product” of service systems typically is, well, *service*. Thus the goal of most service systems is to provide an intangible and perishable benefit to customers on demand.

Similar to manufacturing systems, items in service systems (customers, paperwork, etc.) are routed by resources (clerks, accountants, etc.) through processing areas (emergency room, fast food counter, etc.). However, unlike the usually predictable arrival rate found in manufacturing systems, service industry arrival rates are often volatile. Where long waits in manufacturing systems can result in spoilage and higher inventory carrying costs, the primary waiting cost in service systems is lost business and ill will. And unlike manufacturing systems, where the emphasis is on inanimate equipment, services are people-intensive, both in the delivery (service personnel) and in the receipt (customers). Thus service industries seek to meet a highly variable rate of demand with a hopefully constant rate of service where both the demanders and the providers of the service have wills of their own.

Types of service systems

There are at least as many different service systems as manufacturing systems and each segment views their own providence as unique. However, service systems could be classified as:

- Financial (banks, credit unions, brokerage house)
- Healthcare (hospitals, clinics, dental office)
- Retail (stores, supermarkets, repair shops)
- Utilities (telephone, gas, cable TV)
- Professional (accounting, tax, legal)
- Delivery and transportation (mail systems, airlines, moving services)

Important considerations

Models of service operations are most often built to analyze four issues:

- Staffing (an evaluation of the queue length and wait times at various staffing levels)
- Process improvement (to streamline the processing of transactions such as paperwork and telephone calls)
- Order fulfillment (supplying the customer with what is wanted, when it is wanted).
- Cost (to reduce processing costs and therefore increase profits).

Performance evaluation

Most service industry performance measures are an attempt to arrive indirectly at the real performance measure: customer satisfaction. An intangible which derives from the commingling of multiple activities and interactions, customer satisfaction is almost impossible to quantify. However, the activities that contribute to it, or which take away from it, can be measured. For example, how long a customer waits for service will have an immediate effect on satisfaction levels. And whether a customer has to call just once or many times for problem resolution can cause a long term effect when it comes to future purchases.

The major performance measures in service systems are:

Term	Definition
Cycle time	Time from when the customer enters the system to when the customer leaves.
Waiting time	A subset of cycle time, this is the time the customer waits before receiving service.
Queue length	The number of customers waiting in line at any one time. Average queue length and maximum queue length are often reported and analyzed separately.
Service time	The time it takes to provide service to one customer.
Utilization rate	Busy time. The ratio of service time to available scheduled time. The idle time rate is the inverse of the utilization rate.
Service rate	Number of customers who receive service per unit of time.
Error rate	Percentage of service problems as determined by customer complaints, observation, etc.
Balking rate	The percentage of time the customer decides not to enter the service system (usually due to long line lengths).
Reneging rate	The percentage of time the customer leaves the line before obtaining service (usually due to a long wait time).
Non value-adding activities	Anything above the minimum amount of resources required to provide service.
On-time-service	How close the providing of service is to the promised time. As is true for manufacturing systems, too early can be as costly as too late.
Accumulated cost	The total cost of all the resources used to process one item.

Example models

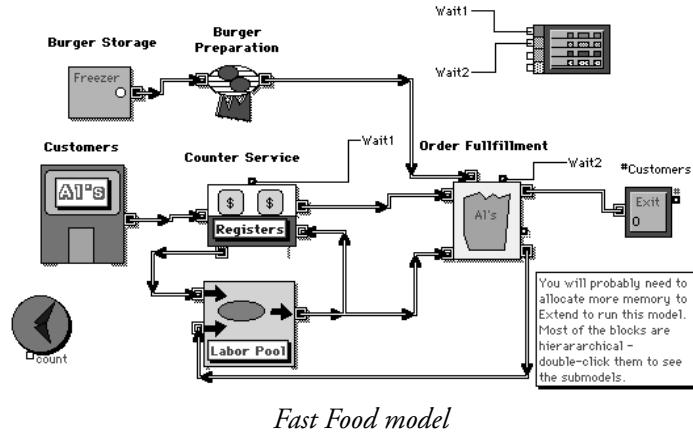
The Fast Food model, Ferrari Agency, and Call Center models are located in the Applications folder in the Manufacturing folder.

Fast Food model

The Fast Food model is an example of a service business where the object is to provide customers with food as quickly as possible, minimizing wait time.

The model is divided into two sections, representing the front service area and the kitchen in the back. In one section, customers arrive, place orders for hamburgers, wait for them to be served, then exit. In the other section, workers take orders, collect the cash, and forward the orders to the cook. The cook then prepares the hamburgers and gives them to the workers who fulfill the orders. The separate functions of the model are represented by hierarchical blocks.

Note  *Macintosh:* This model may require more memory than is initially allocated to Extend. See your Extend manual for information regarding memory allocation.



Fast Food model

MFG

The particulars of the model are:

- The average interval between customer arrivals is based on historical information. This is entered as a table where the “Y Output” is the time between arrivals:

Row	Time	Y Output
0	0	5
1	30	1
2	60	0.5
3	90	1
4	120	5
5	1000	5
6		
7		
8		
9		
10		

- Historically, 30% of customers order 1 hamburger; 50% order 2; and 20% order 3.
- There are 200 frozen hamburgers (10 bags of twenty burgers) in the stock room when the model begins.
- An automated storage and retrieval system maintains a stock of defrosting hamburgers.
- It takes 2 minutes to take an order and 15 seconds to fulfill the order once the burgers are cooked.
- Workers at the front can both take and fulfill orders.
- The store opens with 3 workers. Thirty minutes later (at the start of lunch hour) 3 more workers arrive. After ninety minutes, 1 of the workers leaves.

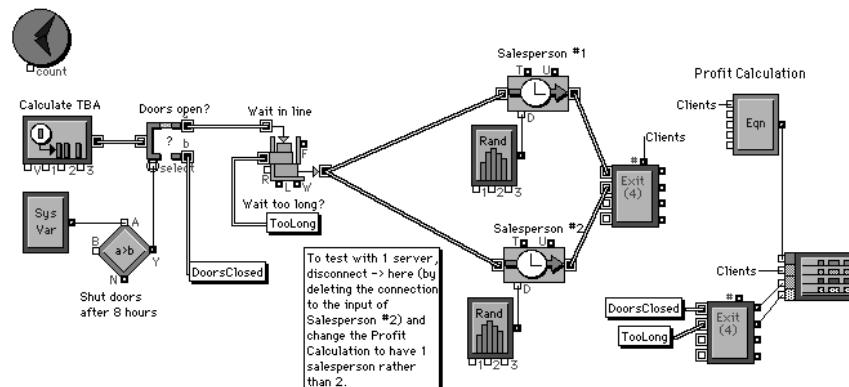
- There is only 1 cook in the back making hamburgers and it takes 3 minutes to broil each hamburger. The cook tries to maintain an inventory of at least 5 cooked hamburgers so customers waiting in line can be quickly served.

Customers are timed as they wait for the cashier (counter service hierarchical block) and as they wait for their order to be fulfilled (order fulfillment hierarchical block). Both the counter service and fulfillment blocks warn if too many people are waiting on line. When the model is run, the plotter shows the waiting times during the period that the restaurant is open.

The model tries to maintain a fixed quantity of cooked hamburgers, so during the lunch rush the waiting time becomes dependent on how fast the kitchen can produce hamburgers. You could make the required number of cooked hamburgers a function of the length of the customer line, so that the model would “predict” when more hamburgers were needed, indicating that a smaller, less expensive kitchen could provide the same speed of service. However, could this optimization be implemented in a real restaurant? To do this, someone or something in the restaurant would have to determine how many customers were in line, estimate their hamburger requirements, and constantly feed that information back to the kitchen.

Ferrari Agency model

The Ferrari Agency model represents an automobile sales room. The owner wants to determine the optimum number of salespeople to employ with the objective of maximizing daily profit.



Ferrari Agency model

Some of the model specifications are:

- The agency is open for 8 hours. Clients cannot enter after the doors have closed; however, clients who enter before the doors close are waited on until their service is finished.

- The owner has historical information regarding the time between arrivals of clients as shown in the table:

Row	Time	Probability
0	3	0.2
1	13	0.6
2	30	0.1
3	90	0.1
4		
5		
6		

Probability table for time between arrivals of clients

- A table of service times for the salespeople is provided:

Row	Value	Probability
0	2	0.1
1	7	0.2
2	15	0.5
3	30	0.15
4	50	0.05
5		
6		
7		

Probability table of service times (in minutes)

MFG

- Clients are served in order of arrival and salespeople must finish with one client before serving another. Clients leave if they have to wait longer than 10 minutes to be served.
- Each salesperson costs the agency \$75 per day. Each client waited on represents an expected profit of \$210 (because you expect 5% of the clients to purchase a car, and the average profit per car is \$4200).

Although an eight hour day is actually 480 minutes (8 hours * 60 minutes), salespeople can wait on customers after the doors have closed. Therefore, the End Time is set at 490 minutes to show the last customers being served.

The model starts out with two salespeople. Run it like that, and note the profits.

- ⇒ To see the effects of having just one salesperson at the agency, disconnect the second salesperson from the Queue Reneging block so that no more items flow to that part of the model. Then, change the 2 to a 1 in the Profit Calculation equation block and run the model again.
- ⇒ To see the effects of having a third salesperson, add another Activity Delay block to the original model and connect from the output of the Queue Reneging block to the new Activity Delay block and from there to the Exit block. Then change the 2 to a 3 in the Profit calculation equation block. (Using a block to represent each salesperson is just one method you can use to vary the number of available salespeople. As seen in “Adding another auto inserter” on page M54, there are other methods.)

The model is notable for its use of the System Variable and Queue Reneging blocks:

- The System Variable block (from the Information submenu of the Generic library) allows you to control what happens in the model based on model variables such as the number of simulation runs or simulation time. In this case, the block is used to “shut” the agency doors at the appropriate time.
- The Queue, Reneging block (from the Queues submenu of the Manufacturing library) is used for routing items out of the line if the wait is longer than 10 minutes.

Call Center model

This model is more complex than the models discussed earlier in this manual. It is an excellent illustration of the creative use of hierarchical blocks, customized animation of hierarchical block icons, and the use of the Notebook as a control panel or executive interface. The purpose of the model is to determine how many resources are required to support the desired service level.

MFG

Call centers are a good application for simulation. They are easy to model and information is generally available about the service times, arrival rates, renege times, and the paths that the calls take through the center. Also, the type of information produced in a simulation model is very useful in evaluating the quality of service in a call center. This model illustrates the use of hierarchical blocks to organize the structure of the model, a more sophisticated routing of items than previous examples, and the use of the notebook to serve as a user interface for a model.

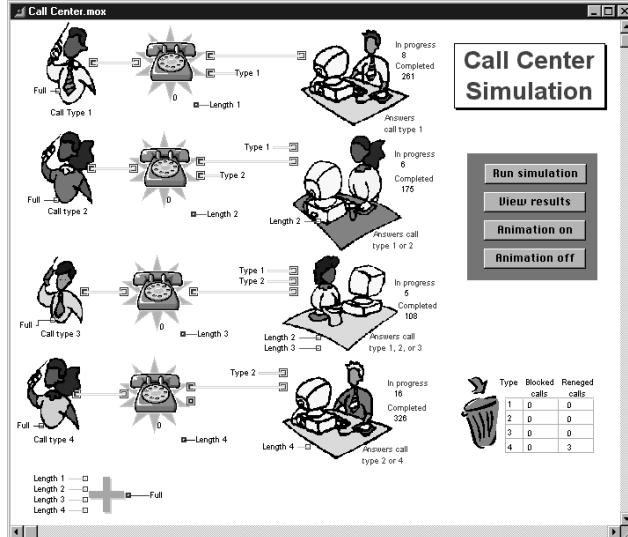
About the model

In this call center, there are four types of calls arriving at random intervals and four types of agents who are able to answer the calls. Each agent type is specialized in a particular call type. However, some agents are able to answer calls of a different type.

The following table shows the specialized call type (X) and the optional call type (O) for each agent:

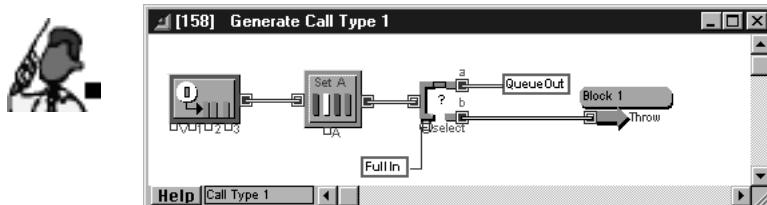
Call Type	Agent 1	Agent 2	Agent 3	Agent 4
1	X	0	0	
2		X	0	0
3			X	
4				X

An agent will answer their specialized call type if any calls of that type are waiting in the queue. If there are no calls in their specialization, they will answer a call of their optional call type.



Call Center model

Calls arrive, and wait in a queue for an agent. If the queue is full, the call will be blocked, the caller has received a busy signal. To determine if the queue is full, the lengths of each of the individual queues for each call type is added together and this sum is compared to a maximum allowable total number of calls waiting. The Generate Call Type 1 hierarchical block illustrates the use of the Select DE Output block to route the item elsewhere if the queue is full. Note that a Set Attribute block is used to set the call type, call time and the standard deviation of the call time attributes.



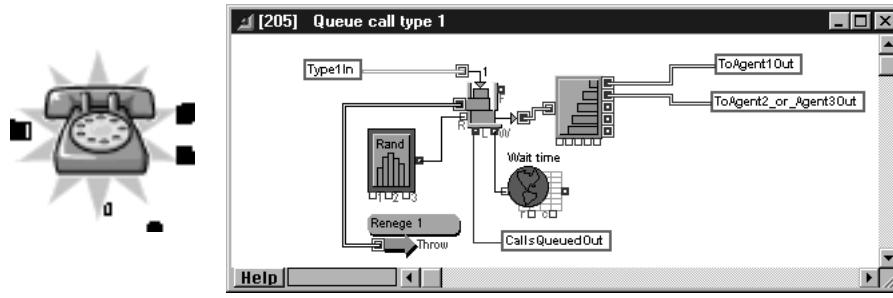
Generate Call Type 1 hierarchical block

If a specialized agent is available the call will go to that agent, otherwise, if an optional agent is available, the optional agent will be chosen. A call will wait in a queue if no agents are available. If no agents are available for this call type, the call will wait. If the caller waits too long, they will renege or give up.

The “Queue Call Type 1” hierarchical block shows the Reneging queue which permits the calls to expire after a certain amount of time (in this case the time is a random value from the Input Ran-



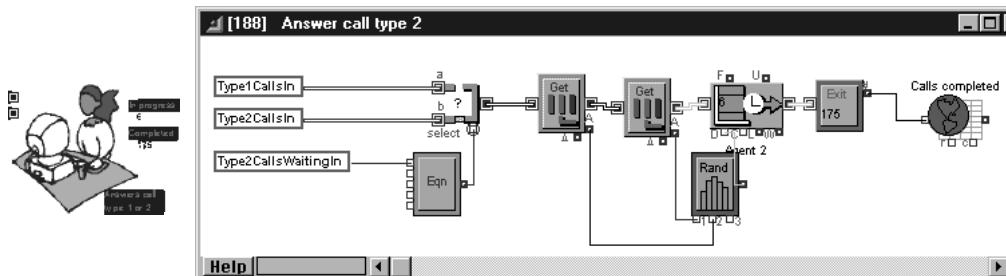
dom Number block) and leave the block through an alternate connector. The Prioritizer block attempts to send the calls to the specialized agent first. If no specialized agents are available, the Prioritizer will attempt to send the call to an optional agent.



Queue call type 1 hierarchical block

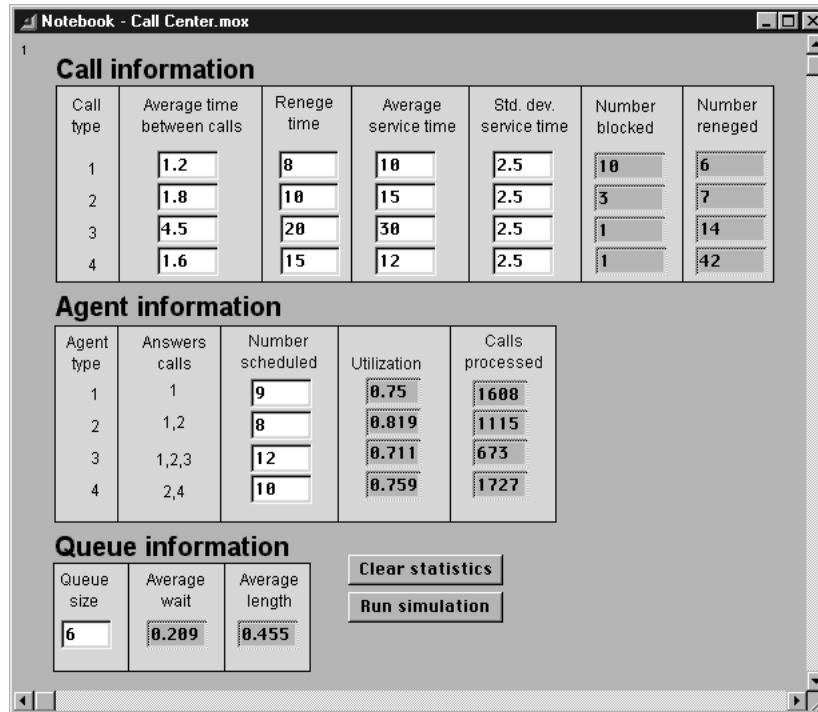
MFG

The agents will give priority to calls in their specialized area even if optional calls have been waiting longer. The Answer Call Type 2 hierarchical block shows an equation and Select DE Input block used to choose the calls within the agent's specialty if any are waiting and optional calls (in this case type 1 calls) otherwise.



Answer call type 2 hierarchical block

Extend's notebook is used to display the input parameters and the results of the model.



Notebook

Material handling systems

Material handling is the movement of items from one location to another using a resource or resources. Although material handling could be, and often is, modeled as part of a manufacturing process, the blocks that represent material movement can be used for other purposes. In addition, material movement frequently constitutes a significant (and often non-value-adding) portion of the time parts stay in a manufacturing system. Hence it is considered here as a separate topic.

Types of material handling systems

Unless they are moved manually, most items are moved using a *fixed-path method* (such as a conveyor) or by utilizing *transporters* (such as AGVs or cranes) which have more or less free movement. Blocks in the Manufacturing library specific to material handling include:

- *AGV* (Automatic Guided Vehicle Systems, used with route blocks)
- *ASR* (Automated Storage and Retrieval Systems, used with route blocks)
- *Conveyor Belt* (a linear conveyor that can be accumulating or non-accumulating) and *Conveyor Carousel* (a non-accumulating rotating conveyor)

- *Crane* (an overhead transportation device with both a destination and a return length)
- *Route* (path for AGVs and ASRs when they transport items) and *Route, Delay* (path for AGVs and ASRs when they do not transport items)
- *Transporter* (ground-level industrial vehicle with both a destination and a return length of trip)

Important considerations

When modeling material handling systems, the characteristics of the vehicles are usually of most concern:

- Speed (in the Manufacturing library, speed is always expressed as “feet per time unit”) or circulation time (for circulating conveyors)
- Length of destination trip (in the Manufacturing library, the length is always in feet)
- Whether there is a return trip and how long it is (in the Manufacturing library, the length is always in feet)
- The total number of slots (for conveyors) and the number of slots before the output (for circulating conveyors)
- Number of carriers required
- Disposition and routing of empty vehicles
- Required amount of buffering before and after

As you will see in “Implied processing time” on page M131, the amount of simulation time it takes to move items is determined by the entries you make in a material handling block’s dialog regarding speed, length, and so forth.

Performance evaluations

Some typical measures of performance for material handling modeling are:

Term	Definition
Delivery time	Time from when the item is ready to be picked up to when it is delivered to its destination point.
Response time	Time from when a vehicle is requested to when it is available.
Queue size	For accumulating conveyors, the number of items waiting to be picked up.
Load movement time	Time from when materials are picked up to when they are delivered.
Load capacity	The number of items that can be accommodated (usually a single item or one batched item.) For conveyors, capacity is determined by the number of slots and the conveyor speed.
Utilization rate	Busy time. Ratio of time moving items to available time (conveyors also consider slot numbers.) The idle time rate is 1 - utilization rate.
Throughput rate	Number of items that can be moved per unit of time.
Blocking rate	The percentage of time items cannot move because the next process is not ready to receive them.
Bottleneck	A constraint on movement caused by a utilization rate that is too high.

Example models

There are two material handling examples in the Extend+Manufacturing package ( Macintosh: Transportation 1 and Transportation 2. The files are located in the Transportation folder within the Processing folder within the Manufacturing folder. These two models are discussed in the section “Material handling and transportation blocks” on page M147.

In addition, the “Assembly-Rework model” on page M11 and the “Fast Food model” on page M18 both contain blocks for materials movement.

Communication systems

A communication system is a collection of autonomous entities interconnected across a shared resource so that they function in a specific manner. Discrete event communication systems are usually established to provide rapid and efficient transmission of data, where the data being transmitted is digitized into bits. For example, when you make a telephone call, your voice does not travel over the telephone lines. Instead, it is digitized into data bits which travel the line and are reassembled into the sound of your voice at the other end.

The connections between communicating networked entities can be by wire, lasers, microwaves, communication satellites, or other means. Communication systems transfer data using either the *simplex* (one directional), *half-duplex* (only one direction at a time), or *full-duplex* (both directions simultaneously) methods.

Types of communication systems

Discrete communication systems can be categorized as either *computer networks* (such as a local area networks or wide area networks) or *telecommunication networks* (such as a telephone or satellite system). These systems have the following characteristics in common:

- A sender that addresses data to a specific location or broadcasts it for general access
- The activity of transmitting the data
- A receiver which acknowledges reception of the data
- A transmission medium and equipment such as transmitters and receivers
- Routing mechanisms or algorithms

MFG

Important considerations

You build communication system models to evaluate how an existing or a proposed system will perform under various protocols and equipment configurations. The purpose of the models is typically to determine if the system will meet the specified requirements, to eliminate or minimize transmission bottlenecks, and to reduce the costs of implementing or modifying the system.

Communication models typically have five major focuses: the *interarrival time* of data packets or messages; network *service time*; the number of servers; the *queueing discipline* used (priority, FIFO, etc.); and the amount of *buffer space* in the queues. As you may expect, many of the other modeling issues revolve around data concerns:

- Error detection, control, and reporting
- Determining when the sender is overwhelming the receiver with data
- Disassembling and reassembling of the data or message packets
- Collision avoidance
- Transmission speed
- The effect on data flow when there are changes in routing algorithms and equipment
- Determining the optimum number of servers to maintain a required level of service
- Evaluating system performance under increased transmission loads

Performance evaluation

Some important measures of performance are:

Term	Definition
Message delay time	Time from when the message originates to when it is delivered. Also known as <i>transit time</i> or <i>message latency</i> .
Connection establishment (or release) delay	The amount of time between the issuing of a request for connection (or release) and the requestor's receipt of confirmation of its occurrence.
Buffer length	Length of the queues for the originating, intermediate, or destination I/O buffers.
Contention queue	The number of messages contending for the same channel at the same time.
Processor delay	Delays at originating, intermediate, and destination nodes
Network utilization rate	The ratio of used line capacity to the amount available.
Throughput	The number of messages of user data transferred per time unit.
Bottleneck	A constraint on network flow, usually caused by the resource or node with the highest utilization.
Transmission rate	Ratio of messages successfully transmitted to total number transmitted. The <i>residual error rate</i> is 1 - transmission rate.

Example models

The CSMA LAN and CSMA Packet models are located in the Applications folder in the Manufacturing folder. *These models are supplied courtesy of Doug Shannon at TRW.*

Note  *Macintosh*: These models may require more memory than is initially allocated to Extend. See your Extend manual for information regarding memory allocation.

Overview of computer networks

Computer networks are a collection of computers which are interconnected for the purpose of exchanging information. The widespread use of computers coupled with expectations of instant access to information means that a computer today is more likely to be part of a network than not.

Companies network their computers so that resources, such as information, files, and equipment, can be made available without regard to the physical location of either the resource or the person accessing it. By serving as backup systems, networked computers provide increased reliability; by allowing incremental changes when increased performance is required, they facilitate corporate growth. But perhaps the most important contribution of networked computers is in the area of "work groups": networked computer files allow users at different locations to work on projects and contribute information concurrently.

Note The architecture of computer networks is complex, is described in most network texts, and is beyond the scope of this model. The following is meant as a brief overview only.

Computer networks are often categorized as local area networks (LAN) or wide area networks (WAN):

- LANs have computers located in the same room, the same building, or the same campus. Local networks have a length of not more than a few kilometers, are owned by a single organization, and have a data rate of at least several Mbps.
- WANs (also called long haul networks) have computers located in same city, country, or hemisphere. These networks have a wide range of data rates and are owned by multiple organizations.

There are several standards for LANs recognized by the Institute of Electrical and Electronic Engineers (IEEE), including CSMA/CD, token ring, and token bus. CSMA/CD stands for carrier sense multiple access/collision detection. Ethernet, which is a trademark of Xerox Corporation, is a product which implements CSMA/CD protocols.

In a CSMA/CD network, a node with a data packet to transmit queries the channel for activity. If it senses that the channel is idle, the node sends its data. If it senses that the LAN is busy, the node may try again as soon as the channel is idle or it may wait a while, depending on the protocol.

When a node starts transmitting data, it sends a message indicating that the channel is now busy. However, because of the physical length of the LAN, there is a delay in propagating this message to all the nodes. This period of time is called the *slot time*. During the slot time, the LAN is vulnerable to transmission errors resulting from data collision. If two or more packets of data collide, they add together and produce garbage.

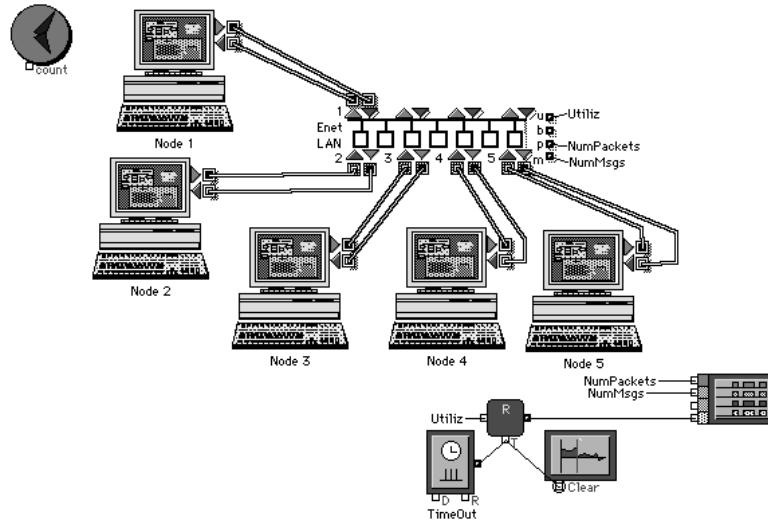
When nodes sense a collision, they stop sending their data, pause for a random amount of time, and retry. Since the period before retry is random, if there is not a lot of traffic in the channel one of the nodes will find the channel idle and send its data packet successfully. If the channel is very busy when the retry is attempted, the node will back off again. Each time the nodes do this, they back off further. It is interesting to note that this behavior sometimes allows a node with a newer message to “sneak in” and send its data ahead of the others. Low fidelity models sometimes use LIFO queues to model this situation.

The International Standards Organization has developed a seven-layer OSI Reference Model as a framework for computer network architecture. Most modeling is of the physical layer and *protocols* (the rules for interpreting messages passed between layers).

The CSMA_LAN model

The CSMA_LAN model simulates a simple network architecture (e.g. an office) with five computer terminals which access one LAN. These terminals randomly generate message or file traffic

across the LAN to a specified terminal or server. The purpose of the model is to determine the utilization of the LAN based on system configuration and activity.



CSMA model

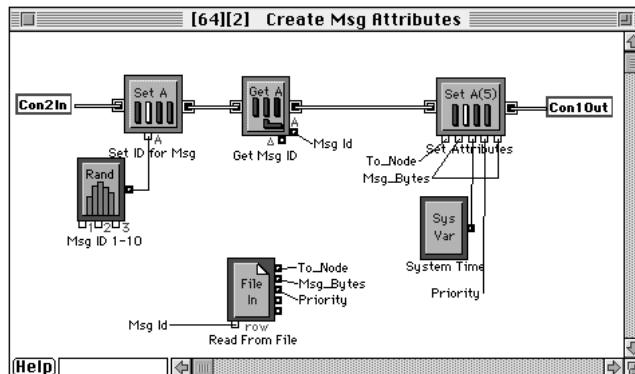
MFG

The particulars of this model are:

- There are 5 nodes which generate message and file traffic at random intervals of from 1 to 30 seconds.
- The message size and the packet size are the same.
- The nodes function in full-duplex mode.
- The nodes send messages with various sizes. Messages of from 1000 to 54,000 bytes represent simple application level protocol messages or small files; message sizes of 300,000 to 1,000,000 bytes represent bigger files.
- Three of the nodes send messages at a frequency of 1-10 seconds; two of the nodes send files at a slower frequency of between 20-30 seconds.
- Every node is connected to a LAN which has a data transmission rate of 1.25 million bytes per second.
- The node number for the intended receiver, the length of the packet size (in bytes), and the priority of the message (although not used in these models) are given in the table:

Destination node	Message size	Priority
5	4000000	1
4	1000	2
3	4000	1
2	1000	1
1	1000000	2
3	1000	1
1	45000	2
4	1000	1
5	300000	2
1	54000	2

Each of the five nodes in the model is represented by a hierarchical block with a computer icon (if your computer is fast enough, run the model with animation on to see the computers flash when a message is sent.) Within each node, another hierarchical block (“Create Msg Attributes”, shown below) generates messages at random intervals and specifies the characteristics of the message.



Submodel in Create Message Attributes block (Macintosh)

Each computer node has a different mean for the exponential distribution depending on its estimated frequency of message sending. Attributes are used to specify the message ID, the intended receiver, the message length (in bytes), and a priority for each outgoing message.

The File Input block (from the Inputs/Outputs submenu of the Generic library) is used so that the model can read data from an external file (for example, from Excel) where it can be more easily modified and controlled. This is an advantage in situations where you duplicate blocks, want different data for their parameters, but don't want to have to enter the specific data for each block. In this model, the data is *not* read from a file but has been entered directly into the data table within

the File Input block. To read from a file, create the file, then check “Read at beginning of simulation” and provide the file name and path in the dialog of the File Input block.

Note that this is a low-fidelity model which ignores much of the complexity of LAN and terminal I/O protocol. For example, it would not be representative of a heavily loaded system or if messages were widely different in size and very large. There is no random backoff following collisions; instead that situation is modeled as a LIFO queue. There are no leading edge situations (two LANs in a row performing parallel activities) and no true collision. However, the fidelity of this model is sufficient to estimate LAN utilization and rough latency (time delays) due to LAN contention. Thus it is appropriate to use this modeling approach if the LAN performance is a small part of a larger system performance model.

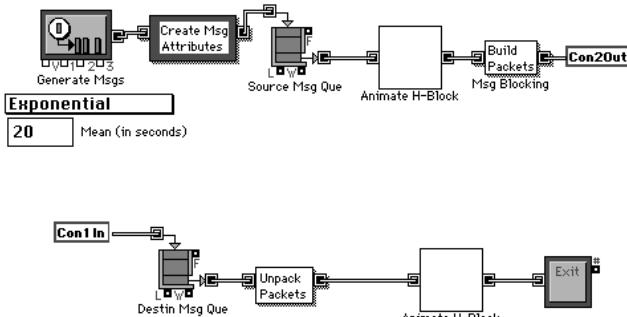
By making improvements to this model, the results could provide more accurate performance estimates and additional data on LAN activity such as lost message counts, I/O queue lengths, and latency distribution for delayed messages, as discussed below.

MG

The CSMA Packet model

The CSMA Packet model (CSMA_PAC) is an improvement on the fidelity of the CSMA LAN model. This variation provides for simulating message/file packetization and reconstruction at the destination node. However, as you will see when you run this model, this added complexity has a trade off in simulation performance. Since the execution time is multiplied by the larger number of packets generated per message, the model takes significantly longer to run.

This model is almost identical to the CSMA LAN model, discussed above, except the LAN block has some modifications and there are additional blocks in each of the nodes for building and unpacking packets and for identifying the last packet in the transmission, as shown below:



Network Node hierarchical block in the CSMA Packet model

Notice that traffic statistics across the LAN show packet counts; however, packets are still rushed to the LAN without the correct Ethernet protocol. If the purpose of your model was to focus on the performance and complexities of the LAN in detail, you would need greater model fidelity. This can be accomplished by upgrading the LAN protocol in the model (a simple LIFO queue in these models) to sense activity and collision and the Ethernet random exponential backoff algo-

rithm. Other potential improvements include blocking a terminal's I/O until previous packets have been received and modeling the leading edge of messages/files to allow correct transmittal across multiple networks.

The Network Crash model that comes with Extend, in the “Examples \\ Manufacturing or BPR \\ Science & Engineering” folder, illustrates a method of simulating messages crashing and destroying each other if they overlap.

Manufacturing Chapter 2: Tutorial

The Manufacturing library (MFG) is designed to help you build models of commercial and industrial processes; the Statistics library reports model-wide metrics for performance analysis. The Manufacturing and Statistics libraries are used in concert with other Extend libraries, especially the Discrete Event, Generic, and Plotter libraries.

The Manufacturing library is based on Extend's Discrete Event library and contains many extensions to that library. To use the Manufacturing library to the fullest extent, first establish at least an intermediate understanding of Extend and the Discrete Event library. Familiarity with the Extend Generic library is also helpful since there are many cases in which blocks from all three libraries are used in the same model. We suggest that you refer to the main Extend manual, particularly Chapters 1 through 4, for more information on how to use Extend and the Discrete Event and Generic libraries. Chapters 1 through 3 in the main manual provide a brief general tutorial for using Extend. Chapter 4 provides very important information about using the Discrete Event and Generic libraries.

The following tutorial will introduce you to most of the important blocks in the Manufacturing and Statistics libraries. The models mentioned in this tutorial are located in the "Examples \ Manufacturing \ Circuit" folder.

Description of the problem

A circuit card manufacturer wants to increase the capacity of its production line to accommodate a new product. Specifically, it is being proposed that a second automatic component insertion machine, with the same capacity as the original, be installed. This is based on the assumption that the existing automatic insertion machine is the bottleneck in the production process and will cause even more throughput problems when the new product line is introduced.

The existing process is as follows:

- The company uses five types of circuit cards which arrive in fixed-size batches according to a predefined schedule. The schedule for each card type repeats every 120 minutes. For example:

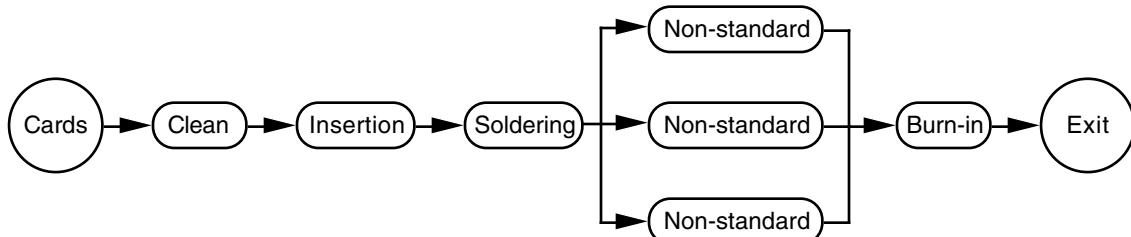
Card type	Arrives at ____ minutes from start	Batch size
1	0	20
2	20	30
3	40	25
4	60	30
5	80	25
1	120	20
2	140	30
...

- The first operation is a single cleaning station that requires at least 36 seconds, at most 54 seconds, and usually 48 seconds to clean each piece.
- The clean cards are put onto an automatic insertion machine which installs most of the electronic components. This machine can work on up to 6 circuit cards at one time and requires 5 minutes to process each card.
- Once the components have been installed they are soldered into place in a 10 foot long wave solder machine which has a capacity of 30 cards on its belt and moves at 1 foot per minute.
- There are three stations which install any nonstandard components that the automatic insertion machine is unable to handle. The amount of time required for this operation varies by card type:

Card type	Process time (minutes)
1	2.5
2	2.0
3	2.5
4	3.0
5	2.0

- The final step is a burn-in test. In this process, the cards are combined into groups of 24 which are put into an oven and power-cycled for 20 minutes.

A simple diagram of this process would look like:



Before you build this 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 jump in and immediately start building the model. However, you will find that you will end up with more useful models if you begin the model-building process by asking yourself 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. In this example, you are trying to determine the equipment requirements for a production schedule.
- *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. For example, in the model of this circuit card manufacturing process you should not include the inventory levels of the individual components because this information is not important to the goal of the model.
- *Where is the required data?* In this example, all of the required data is given to you. However, it is usually not quite this easy. 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 (such as the maximum capacity of the burn-in process) or if the data is from a statistical distribution (such as the amount of time required to clean the cards). 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 running Extend, you should 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. Again, this tutorial example is rather straightforward: the global time units are hours (although you will be able to choose other time units to define parameters locally), stations are the resources and time delays, and circuit cards flow through the facility.

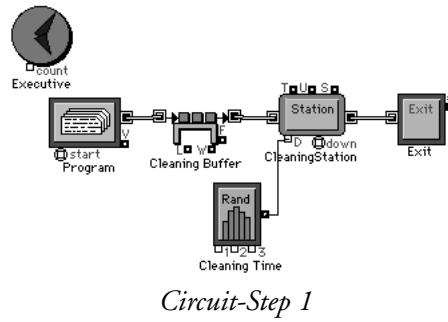
- *What alternatives will be investigated?* Although the model may lead you into new, unexpected directions, you should try to think ahead so that you build a model that is easy to change from one alternative to the next. In this case, you at least need to be able to easily add another automatic insertion machine.

Building the first part of the model

In building a simulation model, it is easiest to start with a simple subset of the process and add detail until you arrive at a completed model. This allows you to test at various stages while making the model building process more manageable. This is the process that you will use while building the model of the circuit card manufacturing facility.

Note The following models are located in the “Examples \ Manufacturing \ Circuit” folder. The blocks you will use to build these discrete event models come from the Discrete Event, Generic, Manufacturing, and Statistics libraries. In the following examples, blocks from libraries other than the Manufacturing library have their source library identified.

Begin by scheduling the arrivals of cards and modeling the process of cleaning them. If you follow the steps in this section, your model will look like:



Define a global time unit

Start by selecting a global time unit for the model. Since the circuit card arrival schedule repeats itself every 120 minutes (2 hours), we will most likely want to run the model for at least 4 hours to witness any bottlenecks that may occur as a result of the arrival schedule. Choose “hours” as the global time unit in the Discrete Event tab of the Simulation Setup dialog from the Run menu and set the simulation to end at time “4”.

Lay out the basic model

It will only take a few blocks to model the arrival and cleaning of the cards.

Executive block



The first block in *every* discrete event model is the Executive block from the Discrete Event library. This block must be to the left of any other block, so it places itself at the upper leftmost edge of the model window when you select it from the library.

The Executive block automatically controls the timing of Manufacturing models such that time advances when events occur. You do not need to enter any data into its dialog or connect it to any other blocks.

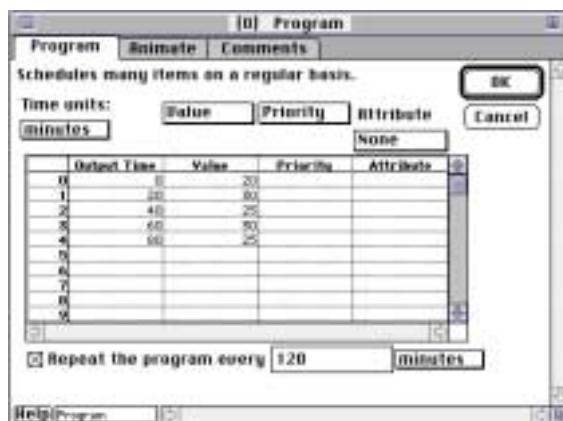
Program block



This Discrete Event library block, located in the Generators submenu, schedules the arrivals of items at specific times. It is particularly useful when you know exactly when the items will arrive, such as in this example where you have a receiving schedule. If you had less information, you could use a Generator block (also from the Discrete Event library) to create a random sequence of arrivals.

MEG

As you will see in “Scheduled arrivals” on page M85, the Program block outputs items at specified times. In the dialog of the Program block, select minutes as the local time unit. Enter the receiving schedule for the circuit cards, specifying the arrival time and the number of cards in each shipment:



Program block dialog (Macintosh)

As seen in the Program block's dialog, the following columns store information:

- *Output Time*: The time (in minutes) each batch of circuit cards arrives.
 - *Value*: The number of circuit cards represented by each item that is output. Item Values are discussed in detail in the main Extend manual and in Chapter 5 of this manual. In brief, each item

generated by the Program block can have a Value; the default is a Value of 1, representing just the item itself. A Value greater than 1 represents a group of cloned items. For example, by giving the item at time 0 a Value of 20, you are telling the Program block to output one item that represents 20 identical circuit cards at time 0.

- *Priority:* Not used in this model.
- *Attribute:* Leave blank for now.

Also, the following settings help define how the Program block will work:

- *Time units:* The time unit used to define the times in the Output Time column. This should be set to minutes. Note that you will not be able to change this to minutes if you have not defined a global time unit, as mentioned in “Define a global time unit” on page M38.
- *Repeat the program every...:* Check the checkbox for this option, select minutes as the time unit, and enter “120” to indicate that the schedule is repeated every 120 minutes. Since the schedule repeats, choosing this option saves you from having to enter the data for the entire day.

Note that since it is unimportant to the model, the batches of cards are not identified by type. This is an example of keeping the model simple by excluding unnecessary detail.

Buffer block



The Buffer block from the Manufacturing library Queue submenu stores an unlimited number of items until a downstream process is ready for them. In this example it represents the work in process before the cleaning operation.

Connect the item output of the Program block to the item input of the Buffer block. (Remember that you usually want to follow a Program block with a Buffer or queue-type block because the Program block “pushes” items out even if the next block is unable to accept them. Without a queue or buffer which accepts all the items the Program block generates, some items may be destroyed.)

The Buffer block will automatically hold items for cleaning, up to the maximum specified in its dialog. There is no data to enter in the dialog, but you should label this block “Cleaning Buffer”.

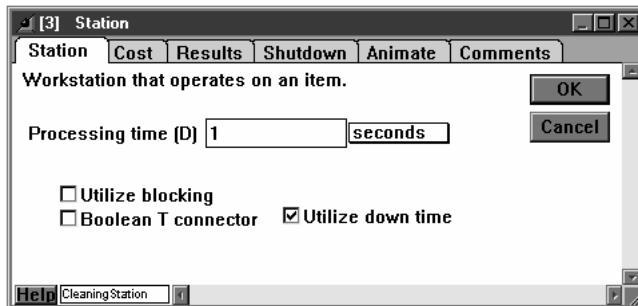
Note When an item with a Value greater than 1 enters a buffer or queue block it is immediately decomposed into separate identical items. For example, when the Program block sends a circuit card item with a Value of 20 to the Cleaning Buffer, the item will become 20 distinct circuit cards each with a Value of 1 and the same original properties (attributes, priorities, etc.), if any, as the original item.

Station block



This block acts like a time delay with a single unit of capacity and can be found in the Manufacturing library Activities submenu. The Station block usually represents a step in a process performed by an operator or machine. In this example it represents the cleaning operation where one circuit board is cleaned at a time.

Connect from the item output of the Cleaning Buffer to the item input of a Station block and label the block “Cleaning Station”. When the Cleaning Station is ready, it will pull a circuit card item out of the Cleaning Buffer and process it. When the cleaning operation is completed, the Cleaning Station will output the cleaned card and pull in a new one for processing.



Station block dialog (Windows)

As noted at the beginning of this chapter, the time required to clean a card is less than 1 minute. It is therefore easiest to define the delay time for this station using seconds as the time unit. Select seconds in the popup menu located next to the processing time.

You could set a constant processing or delay time in this block just by entering a number in its dialog. However, the information given at the beginning indicates that the time it takes to clean a card is random. To model this, you use an Input Random Number block to set the processing delay to a different random value for each circuit card that is cleaned. Since we have chosen seconds as the time unit for this station, the numbers received from the Input Random Number block will be in seconds.

Input Random Number block

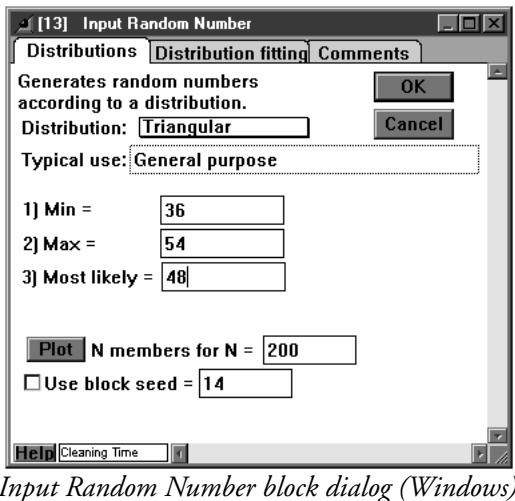


The Input Random Number block (from the Inputs/Outputs submenu of the Generic library) generates random values from a statistical distribution. Typically this is used for processing times, random decisions, and random equipment failures. In this case, it represents the time to clean the cards.

In the model, the Input Random Number block is labeled “Cleaning Time”. By connecting the output of this block to the “D” connector on the Cleaning Station you specify that the processing

time for the cleaning operation will be a random value output by the Input Random Number block.

For information about random numbers and about selecting and using distributions, see “Random numbers and probability distributions” on page M194. For this model, the specific random distribution is specified by selecting a *triangular* distribution in the Cleaning Time block’s dialog and entering a minimum of 36 seconds, a maximum of 54 seconds, and a mode (or most likely value) of 48 seconds. This corresponds to the original information on page M36 concerning the length of the cleaning process. When you have entered the data, the block’s dialog will look like:



Input Random Number block dialog (Windows)

Exit block



The Exit block from the Routing submenu of the Discrete Event library is used to pass items out of the model. This block represents the point where items permanently leave the domain of the model.

The Exit block is the last block in this model. Most models will have at least one Exit block to dispose of the items created during the course of the simulation. Exit blocks automatically count the items as they leave the model; there is no need to enter data in their dialogs.

Running the model

If you have connected all the blocks and entered the dialog information correctly, you should have a working model of the arrival and cleaning process as seen in “Circuit-Step 1” on page M38.

To verify that the model is working as expected and that you can account for all the items in the model, try running it. Once the simulation has completed running, you can examine the simulation results by double-clicking on the Cleaning Buffer, Cleaning Station, and Exit blocks.

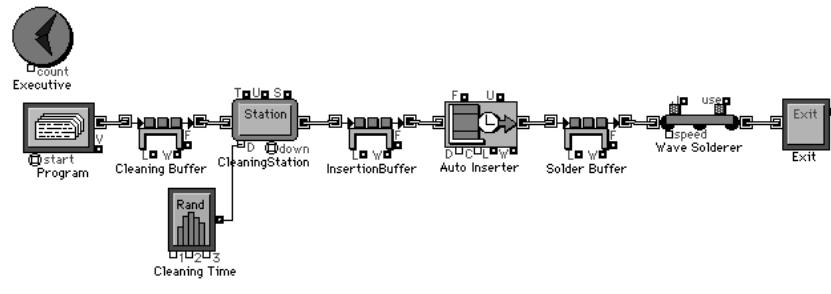
Another verification technique is to run the model with animation on, as you will see in “Measuring performance and debugging models” on page M203. Since animation slows model performance considerably, it is not necessary to run with animation on for the entire time. You can stop the simulation before it ends or turn animation off after the simulation has been running for a while. The important consideration is to determine that portions of the model work as expected.

Notice that there is a backlog of cards shown in the Results tab of the Cleaning Buffer. This is because a batch of cards arrived at the final time period of the simulation and there is not enough time for the cleaning operation to process all the cards in the batch. If you examined the length of the Cleaning Buffer over time, you would see that it empties at the end of each 2 hour period. This indicates that the backlog of cards is temporary and does not require increased Cleaning Station capacity. Another indicator of this is that the Cleaning Station’s utilization indicates that it is not working at full capacity over the simulation run.

Adding two more operations and two more buffers

Next, you need to represent the automatic insertion and the wave solder operations. In order to prepare the model for the addition of the new operations, delete the connection between the Cleaning Station and Exit blocks and drag the Exit block to the right.

When you are finished with this section, your new model should look like:



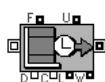
Circuit-Step2

Adding a buffer

Circuit cards that leave the Cleaning Station are stored in a buffer until the next operation is ready to process them. This helps prevent a potential situation termed “blocking”, where cards have been cleaned and are ready for electronic components but the automatic component insertion machine is already busy processing other cards. Blocking is discussed in “Blocking” on page M97.

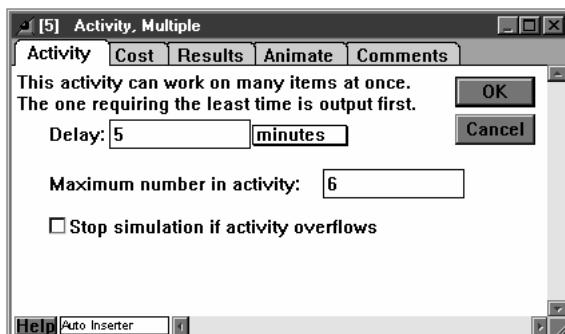
To represent the buffer, connect a Buffer block (discussed in “Buffer block” on page M40) to the output of the Cleaning Station. Label the Buffer block “InsertionBuffer”.

Activity, Multiple block



The Activity, Multiple block from the Activities submenu of the Discrete Event library specifies a time delay with a capacity of more than one. This block is commonly used to represent several processes that occur in *parallel*.

In this example you use the Activity, Multiple block to represent the automatic insertion machine, which processes several boards at a time. In the block's dialog, specify that the maximum number of items in the activity is 6, corresponding to the capacity of the automatic insertion machine. The delay of 5 minutes is the amount of time that the machine works on each card. The time unit for the delay time is minutes. After you enter the data and label the block "Auto Inserter", its dialog should look like:



Activity Multiple dialog (Windows)

Since the Cleaning Station processes circuit cards for a random amount of time, the cards will arrive at the Auto Inserter randomly. The block keeps track of when each card arrives and passes each one out exactly 5 minutes after it comes in. This is true even when there are fewer than 6 cards in the Auto Inserter at a time; each card is passed out based on its arrival time and the delay time set in the block.

Adding another buffer

As was discussed in "Adding a buffer" on page M43, there is a potential for blocking between operations. To represent a queue or buffer between the insertion and soldering operations, connect a Buffer block (discussed in "Buffer block" on page M40) from the output of the Auto Inserter.

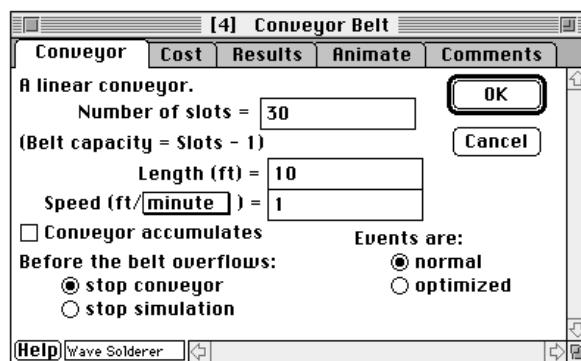
Conveyor Belt block



The Conveyor Belt block from the Manufacturing library Activities submenu represents a linear conveyor typically used for materials handling. In this model you use this block to represent the wave soldering process.

The wave soldering operation works somewhat differently than the auto insertion operation. The insertion operation is an example of parallel processing where several cards can be processed at the same time. Although the soldering machine can hold up to 30 cards on its belt, it is a *linear* or *serial* operation and only one card is soldered at a time. A Conveyor Belt block can model the behavior of the soldering machine.

In the Conveyor's dialog you can specify the maximum number of parts on the conveyor at any time, the length of the conveyor, the speed of the conveyor, and whether or not it is an accumulating conveyor. Its dialog is:



Conveyor Belt block (Macintosh)

In this example, set the capacity of the conveyor to 30 indicating that 30 circuit cards can be on the wave solder machine at one time. Set the length to 10 ft. and the speed to 1 foot/minute, as indicated by the original information you were given. For clarity, label the block "Wave Solderer".

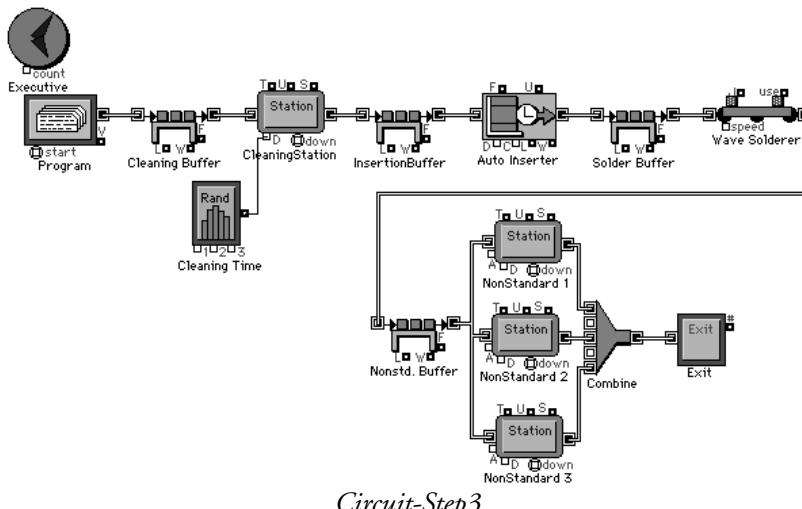
Note The time the Conveyor Belt block takes to process items is dependent on the values you enter for its length, speed, and capacity. The actual delay for each item in this block is calculated as:

$$(Length/Speed) * (Belt capacity - 1) / Belt capacity$$

For example, the delay in this example is $(10/1)*(30-1)/30$ or 9.667 minutes. See "Implied processing time" on page M131 for more information.

Adding a buffer and the nonstandard insertion stations

The next step is to add the nonstandard component insertion stations. When you have completed this section, your model should look similar to:



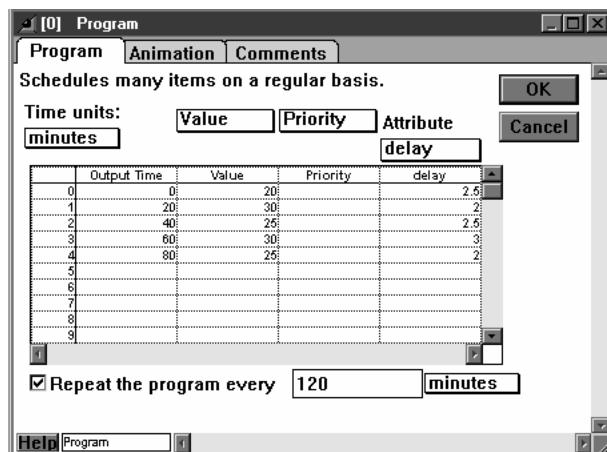
The nonstandard buffer

To represent the typical system where the output of one process is stored until the next process is ready, add another Buffer block (discussed in “Buffer block” on page M40) after the Wave Solderer operation. This will allow soldered circuit cards to accumulate between the Wave Solderer and the nonstandard insertion operations without blocking. Label this block “Nonstd. Buffer”.

Using attribute values to specify a processing time

As you saw at the beginning of this tutorial, the time it takes for the three stations to insert non-standard components varies by card type. A simple and convenient method for modeling this is to specify the processing time for each type of card as an attribute, then have a delay-type block read the attribute value and use that as its delay time.

To do this, return to the Program block (discussed on page M39), click on the Attribute popup menu, and select “New attribute”. Enter “delay” as the attribute name and click OK. Then enter the nonstandard process times as attribute values in the Program block’s dialog, as shown below:



Program block dialog with attribute values entered (Windows)

MFG

Remember that, although the Program block generates batches of circuit cards, each individual circuit card will retain the same original properties (such as attribute values) as the original.

Station (Attributes) block

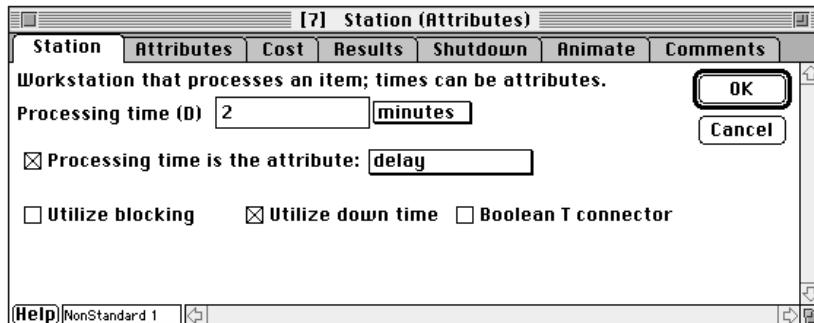


The Station (Attributes) block from the Manufacturing library Activities submenu enhances the functionality of the Station block by providing the option of using an attribute value as the delay time. This is a convenient method for having a delay time be unique depending on the item processed.

Note An easy way to put three identical blocks in a model is to enter the information into the dialog of one block, then duplicate (or copy and paste) the block twice so that there are three blocks.

In this example, you will use the value of the attribute named “delay” (which was set in the Program block) as the processing time for the circuit card. To do this, add a Station (Attributes) block

to your model and label it “NonStandard 1”. Choose in the block’s dialog that the “Processing time is the... attribute *delay*” and set the time units for the delay to minutes, as shown below:



Station (Attributes) dialog where processing time is an attribute value (Macintosh)

MFG

After duplicating the first block twice, change the labels of the two new blocks to “NonStandard 2” and “NonStandard 3”, respectively. You should now have three Station (Attributes) blocks in your model.

Notice that the output of the buffer is connected directly to the inputs of the three nonstandard insertion stations. This is an example of a *simple parallel connection*, as discussed in “Simple parallel connections” on page M127. This direct connection causes Extend to pass cards to the first available station. If more than one station is free when a card is ready, it is unpredictable which station will get the card. Since it is unimportant in this model which station gets an available card, this method of parallel processing is appropriate for this model. In other models, you might want to explicitly specify the order for processing as discussed in “Manufacturing Chapter 7: Routing” on page M107.

To determine total processing time by item, you could select the “Modify the attribute...” and the “add the processing time” dialog choices in this block’s Attribute tab. This would cause an item’s delay time to be accumulated as an attribute value each time an item was processed. The final attribute value could then be read at the end of the simulation to determine total processing time. In this model, however, this information is not necessary.

Combine (5) block

 The Combine (5) block from the Manufacturing library Routing submenu merges up to five streams of items into one stream. This block does not hold or process items and there are no dialog entries to make.

Connect each of the outputs of the nonstandard insertion stations to an input on the Combine block. This block does not batch items together, as does the Batch (Variable) block discussed below, but rather just merges them into one stream. You do this so that all cards that have finished

processing, from whatever source, will be immediately available to the block following the Combine block.

Connect the output of the Combine block to the Exit block.

Running the model

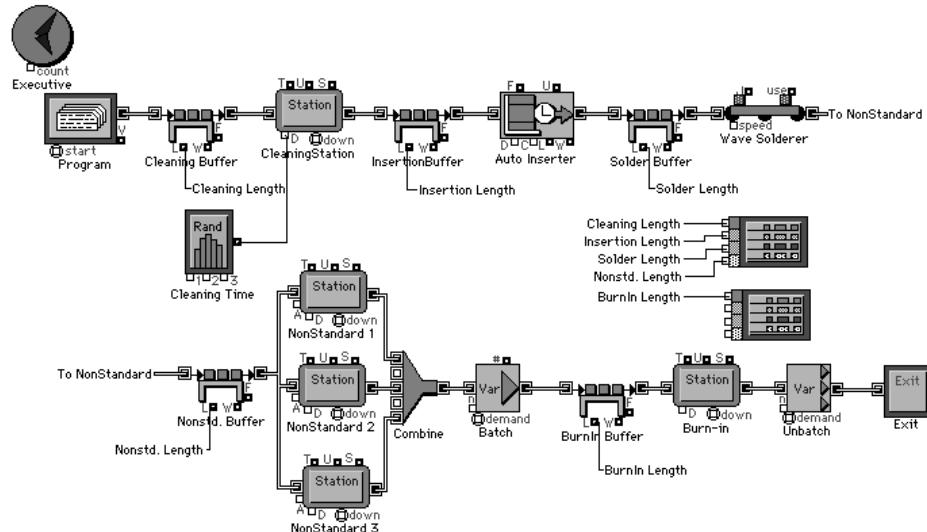
At this point, the model is more interesting. If you run the model now, you should begin to see the interactions between the different processes in the circuit card manufacturing system.

As before, you should try running the model with animation on for at least a portion of the run time to get a useful picture of what's happening in the model. Then run the model with animation off (for increased speed) to get measurements for verification.

Completing the model of the existing process

The final operation is the burn-in station where the circuit cards are combined into groups of 24 and power cycled in an oven for 20 minutes. To model this, you will add blocks that batch and unbatch and a block that represents the burn-in process. In addition, you will add plotter blocks to graphically report results as you run the model. So that you can insert these additional blocks, disconnect the Exit block from the Combine (5) block and move the Exit block to the right side of the model.

Once you have completed the model, it should look like this:



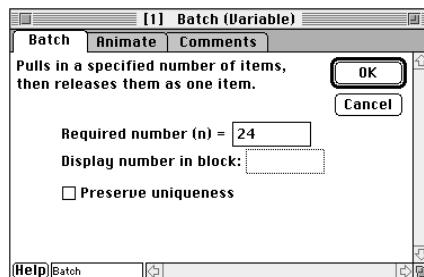
Circuit-Step4

Batch (Variable) block



The Batch (Variable) block from the Manufacturing library Batching submenu allows a variable number of items to be joined into one batched item. In the dialog, you can specify how many items are joined. If “Preserve uniqueness” is checked, the properties of the original item (attributes, etc.) are maintained and can be restored at a later point in the model.

Although the Batch (Variable) block is most often used to batch a single stream of items where the quantity in each batch changes, it is also useful when the batch quantity is fixed. Since all items in this model come from a single source (the Combine block) the Batch (Variable) block is a better choice than the standard Batch block from the Discrete Event library. The dialog of the Batch (Variable) block is:



Batch (Variable) block dialog (Macintosh)

You use this block to combine the circuit cards into groups of 24 (using the “Required number” dialog item) for the burn-in process. Here, “Preserve uniqueness” is not needed because the items do not contain any information that you need to maintain.

Adding a burn-in buffer and station

You could batch the circuit cards after they leave the nonstandard insertion stations, then send them directly to the burn-in station. However, without a buffer between these two operations, there is another potential for blocking, as discussed earlier.

Buffer block

Put a buffer (labeled “BurnIn Buffer”) between the nonstandard insertion stations and the burn-in operation so cards can be accumulated without affecting the operation of the line.

Notice that, whether you place the BurnIn Buffer before or after the Batch (Variable) block, there would be no affect on how the process operated. (There would, however, be differences in the queue statistics since a Buffer in front of the Batch block would gather statistics on individual cards, and a Buffer after it would gather statistics on batches of cards.) For this example, connect the output of the Batch (Variable) block to the BurnIn Buffer as shown in the model above.

Also notice that this is a batch of 24 items being treated as one item, not one item with a Value of 24. Since the batch has an item Value of 1, it is processed as one item. This is very different from what happened with the outputs from the Program block on page M39, each of which had Item Values greater than 1.

Station block

Use another Station block (discussed in “Station block” on page M41) to represent the burn-in operation and label it “Burn-In”. Since the burn-in time is fixed, in the dialog of the “Burn-In” station set the processing or delay time to 20 minutes.

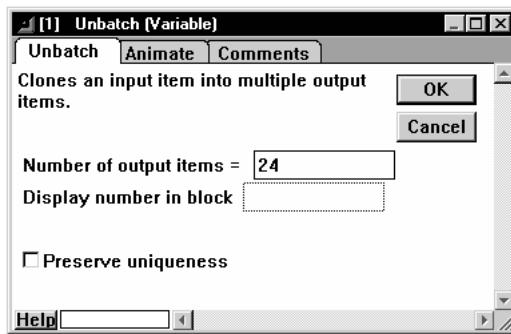
Unbatch (Variable) block



The Unbatch (Variable) block from the Manufacturing library Batching submenu performs the reverse operation of the Batch (Variable) block, splitting each item into multiple items. Notice that if the item being unbatched was previously batched with the “Preserve uniqueness” option, checking “Preserve uniqueness” in this block would restore the original item’s properties.

MFG

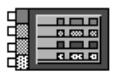
When you enter “24” in its dialog, the Unbatch (Variable) block will output 24 identical items for each item that it gets. Since the “Preserve uniqueness” option was not checked in the Batch (Variable) block, you should not use it here:



Unbatch (Variable) block dialog (Windows)

Connect the output of the Unbatch (Variable) to the input of the Exit block that you moved to the right earlier. The Exit block will count all the cards that have finished processing during the simulation run.

Plotter, Discrete Event block



The Plotter, Discrete Event block (from the Plotter library), plots up to four value inputs at a time. Use the “Show instantaneous queue length” option in the plotter’s dialog if you connect from an “L” output of a queue or buffer block and you want to report items that arrive and depart on the same time step.

There are two categories of data reported by every simulation model – the dynamic information that is generated as the model runs (showing trends), and the static information that is reported at the end of the simulation run (showing final results). Up to now you have mostly looked at block dialogs, examining the results at the end of the simulation run. Since block dialogs report their information in real time, you could leave dialogs open while the model runs; however, that is often confusing and slows the simulation unnecessarily. An alternative is to graph block information using plotters from the Plotter library. Plotters are useful for illustrating trends in a model and can be placed anywhere you want.

For this model, you will plot the number of items in the buffers over the simulation run. Since each plotter block graphs a maximum of four inputs, and there are five buffers in the model, add two Plotter, Discrete Event blocks to the model. Connect from the “L” output of each buffer to an input on a plotter either directly or by using named connections. This will cause the plotters to show the queue length for the buffers as the simulation runs. Label the traces in the plotter dialogs to show which buffer length is being graphed (working with plotters is discussed in the main Extend manual).

MFG

Analyzing the model and running the simulation

You now have a model of the “as is” system. However, building a model is only one step in the process of simulating a system. Once the model is built it has to be tested and analyzed. You should run the model, examine its performance, and analyze the results to determine if the model is behaving as you intended and if it conforms to the real system. This process is known as model verification and validation.

Model verification

As discussed in “Model verification” on page M77, verification is the process of ensuring that a model works as expected. You have already performed part of the verification process by building the model in stages and with minimal detail, then running it at each stage to observe the results. Another simple verification technique is to make sure that you can account for all the items in a model. For example, you could turn on animation and track the items as flow through the model or calculate the number of cards you expected to receive and see if that number corresponds to the arrivals of cards in the Cleaning Buffer.

Model validation

Once the model is verified you need to validate it to determine that it accurately represents the real system, as you will see in “Model validation” on page M77. For this model, since all information is provided, an important validation technique is to make sure that model parameters conform to the information given at the beginning of this chapter.

Running the model of the “as is” system

Run the model for a simulated 4 hours. As you can see, there are not many circuit cards left in the buffers, indicating that the process is working fairly well as it is currently configured. However, if

you double click on the Auto Inserter block, you should see that its utilization is over 90% and that it has the highest utilization of any operation in the model. These are indications that the automatic insertion operation could indeed be a bottleneck when the new product line is added.

Experimenting with the model

Once you are satisfied with the existing model, you can use it to explore the effects of modifying the system and examine alternatives that may improve system performance. This is collectively known as *conducting simulation experiments*. There are several methods for conducting experiments:

- Sensitivity analysis: Determining how much of an impact a parameter or group of parameters has on model results.
- Scenario analysis: Comparing alternative system configurations.
- Optimization: Finding the optimum combination of inputs that will produce the desired output.
- Monte Carlo simulation: Repeatedly running the simulation and statistically analyzing the results to determine with a certain level of confidence what the expected performance of the system would be.

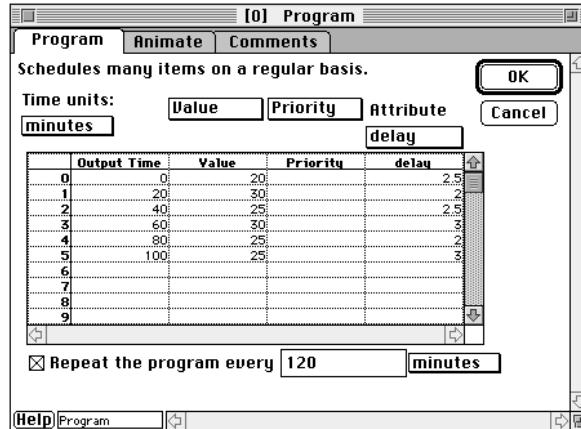
MFG

In the next section you will compare alternative system configurations based on the effect of adding another product line. This should help you determine what changes to the circuit card manufacturing line will be required to allow the system to accommodate the new product.

Adding the new product line

The new circuit card will arrive at time 100, has a batch size of 25, and requires 3 minutes of non-standard insertion processing time. To add the new card type to the receiving schedule, modify the

Program block using this information. When you have completed the changes, the Program block's dialog should look like:



Program block dialog with new product at time 100 (Macintosh)

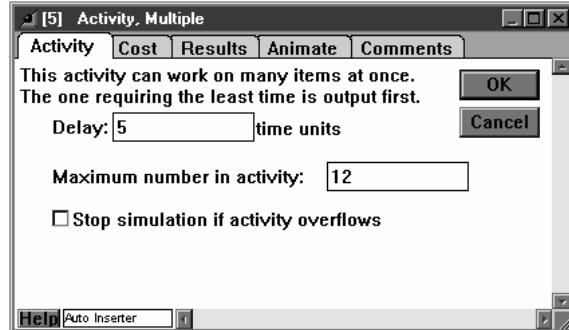
When you run the model you will quickly see that the Insertion Buffer fills up faster than it empties. This is another indication that automatic insertion station is a bottleneck in the process.

Adding another auto inserter

As stated in the system description at the beginning of this chapter, adding another automatic insertion machine is one of the alternatives you have been asked to investigate. There are two ways to do this:

- 1) You could add a second automatic insertion station in the model. To do this, you would duplicate the Activity, Multiple block (labeled "Auto Inserter"), connect the output of the Insertion Buffer to it, and use a Combine block to combine the output of the two insertion stations so cards can flow from both of them to the Wave Solderer.
- 2) An easier method for modifying the model would be to double the "maximum number in activity" parameter in the dialog of the existing automatic insertion operation. Changing this number from the original 6 to 12 circuit cards gives the same operational effect as adding

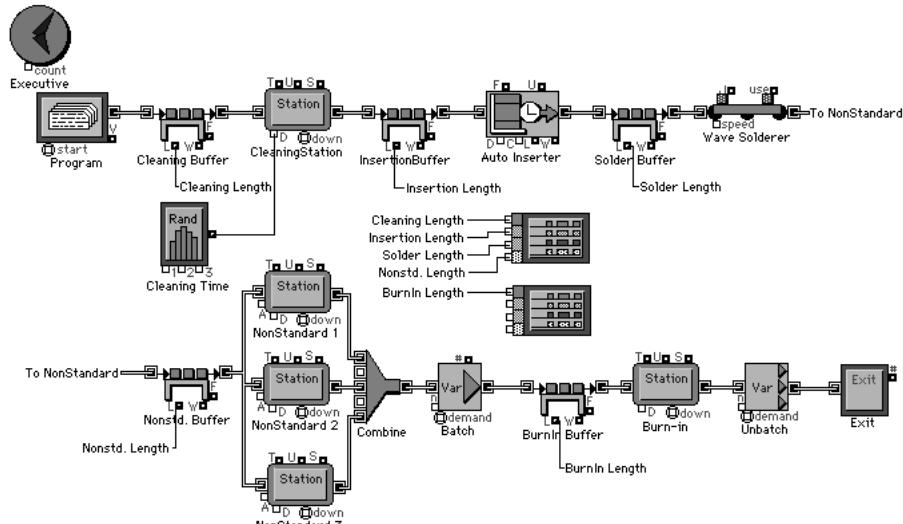
another insertion station without having to change model structure. If you choose this method (which is recommended), the dialog for the Activity, Multiple block should look like:



Activity, Multiple block dialog with doubled capacity (Windows)

MFG

Once you have completed the model, it should look like this:



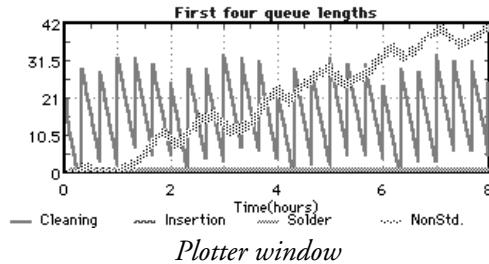
Circuit-Step 5

Notice that, since you have only made changes to block dialogs, this model looks exactly the same as the model in Step 4.

Analyzing the results

When you run the “Circuit-Step5” model you can see that the automatic insertion stations are able to process all of the circuit cards that arrive. The next task, however, is to see if the other operations are able to process their workload effectively.

If you run the model for 8 hours you will see a plot of the buffer lengths that looks like this:



The plot shows that the length of the nonstandard buffer steadily increases as the model runs. This clearly indicates that the nonstandard buffer is filling with circuit cards faster than the non-standard components operation can process them. The logical step would be to add a fourth non-standard components operation. If you do that, you will see that the bottleneck has just moved further down the line and that the burn-in operation is now impacted also. This indicates that an increase in capacity is necessary there as well.

So far, the results of modeling this system indicate that, by itself, adding another automatic insertion operation will not add sufficient capacity to accommodate the new product line without bottlenecks.

Adding animation

Turn on animation by selecting the Show Animation command in the Run Menu. When you run the simulation, Extend's default animation picture, a green circle, will be animated along the connection lines between the blocks.

In the Animate tab of blocks that generate items, you can choose the animation picture that will represent your items. In this model, the Program block generates batches of 20 to 30 circuit boards. In the Animate tab of the Program block, choose the box picture to represent the batches as shown below:

Items will appear as:



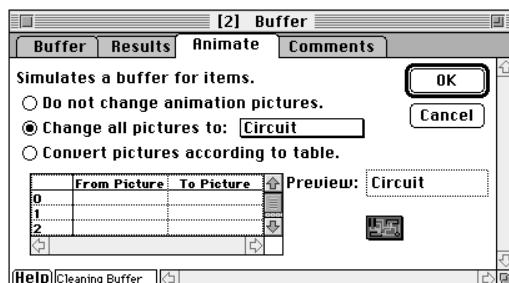
Setting the animation icon in the Program block

As the items flow through the model, they will be represented by the picture that appears below the popup menu.

If you want to change an item's animation picture, select the Animate tab of any block through which the item passes. In this tab, you can change the animation picture for every item that passes

through the block to a specific picture, or change the animation pictures according to a conversion table.

In this model, batches (represented by the box picture) enter the first Buffer block and are automatically separated into individual circuit boards. You can have the Buffer block change the animation picture for each item exiting the block to a circuit board as shown below:



Changing animation picture of each item to “Circuit” (Macintosh)

During the Burn-In phase, the circuit boards are batched together in groups of 24. Using the technique described above, you can change the animation picture of the items exiting the Batch (Variable) block to a box representing the batch and change the animation picture back to a circuit board when the items are unbatched.

To prevent the plotters from automatically opening up at the beginning of the simulation run, click on the Open dialog tool at the top of the plot window. Select “Do not show plot” in the dialog of both plotters.

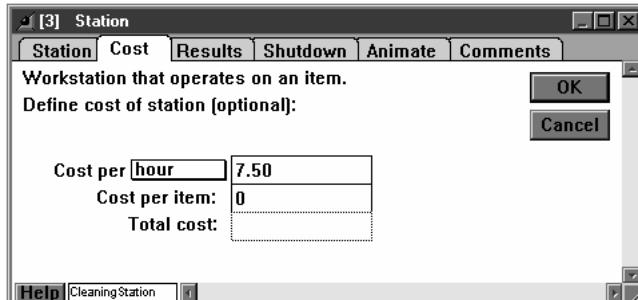
Note that running the simulation with animation turned on will slow down the execution time of the model.

Calculating production costs

Extend’s activity-based costing feature can help you calculate the cost of producing the circuit boards. (See “Manufacturing Chapter 11: Activity-based costing” on page M177 for a detailed discussion on activity-based costing in Extend.) The table below lists the cost of running each of the machines:

Machine	Cost per hour (Variable cost)	Cost per item (Fixed cost)
Cleaning Station	7.50	0.00
Auto Inserter	3.00	1.00
Wave Solderer	10.00	0.00
Non-Standard Inserters	9.00	1.25
Burn-In Station	6.50	0.00

In the Cost tab of each block listed in the table above, fill in the Cost per hour and Cost per item as shown below:



Cost tab of Station block (Windows)

MFG

Note To ensure that each circuit board retains its own unique identity and cost information, “Preserve uniqueness” must be checked in both the Batch (Variable) and Unbatch (Variable) blocks.

When the simulation is run, production costs will be tracked with each circuit board. To collect the cost data, add a Cost Stats block anywhere in the model and a Cost By Item block after the Unbatch block and before the Exit block.

Cost Stats block



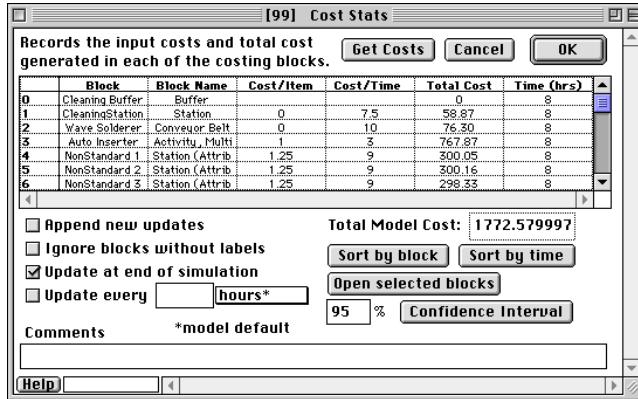
Place this block (from the Statistics library) anywhere in the model and it will report the following statistics for all costing blocks in the model: Block Number (or label, if a label is entered in the block), Block Name, Cost Per Item, Cost Per Time Unit, and Total Cost.

Cost By Item block



This block (from the Statistics library) views and displays the cost of the items 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.

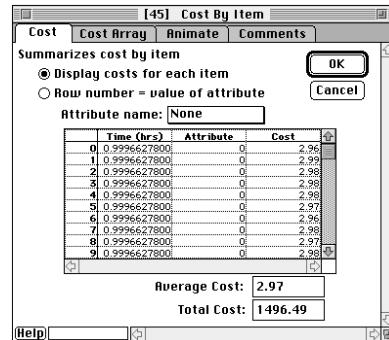
After the simulation is run, the cost generated in each activity, queue and generator-type block is collected and displayed in the Cost Stats block as shown below:



Dialog of Cost Stats block (Macintosh)

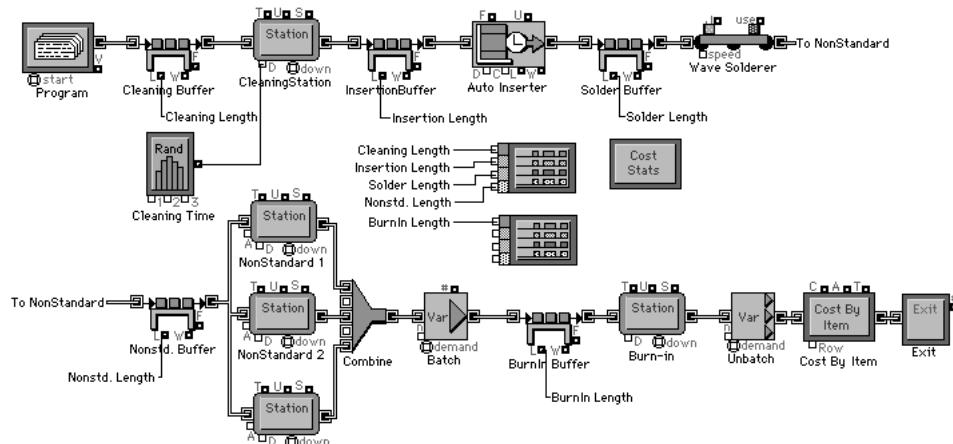
MFG

The cost of each item that passes through the Cost By Item block will be displayed in the block's dialog as shown below:



Dialog of Cost By Item block (Macintosh)

Once you have completed the model, it should look like this:



Circuit-Step 6

How confident can you be in the results?

Since there are random elements in this model, a single run is merely one estimate of the performance of the model. You can get a better idea of the range of system performance by running the model multiple times, as you will see in “The length and number of simulation runs” on page M74.

The Statistics library provides blocks for summarizing the results of multiple runs and generating *confidence intervals* based on those results. Confidence intervals are useful tools for determining how reliable your results are; they are discussed in the section “Confidence intervals” on page M202. You generate a confidence interval by running a model several times using a Statistics library block to collect a series of observations about a statistical variable. You can use a Queue Stats block, for instance, to gather utilization information for all the queues and buffers in a model. When you specify a confidence level (say, 95%) and click on the “Confidence Interval” button in the dialog of the Queue Stats block, the dialog reports confidence interval information for utilization.

By adding the following Statistics library blocks to the circuit card model, you will be able to collect a series of observations on the model performance and statistically analyze the results.

Clear Statistics block



This block (from the Statistics library) clears the statistics in various blocks in a model at periodic intervals or in response to a specific event. This block is used to eliminate statistical bias caused by the warm-up period, which is discussed in “Non-terminating systems” on page M75.

Activity Stats block



This block (from the Statistics library) reports the following statistics for all activity-type blocks in a model: arrivals, departures, utilization, and time of observation. You can place the block anywhere in the model.

Queue Stats block



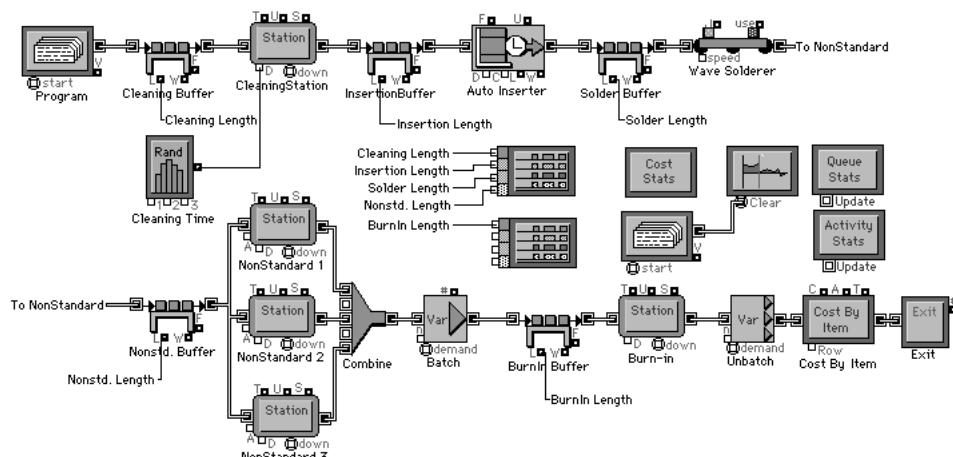
This block (from the Statistics library) reports the following statistics for all queue-type blocks in a model: average and maximum queue length, average and maximum wait time, and time of observation. You can place the block anywhere in the model.

In the dialogs of the Activity Stats and Queue Stats blocks, select “Append new updates” and “Update at end of simulation”. These entries will cause a new set of observations to be appended for each simulation run and will prevent the table from being updated until the simulation ends (so there will be minimal impact on simulation speed). This allows confidence intervals to be calculated.

Adding another Program block

You also need to add another Program block (discussed on page M39), and connect its item output to the “Clear” input of the Clear Statistics block. In the Program block’s dialog, set one item to arrive at 30 minutes. When this item enters the Clear Statistics block, it will reset the statistical accumulators in the model after the first 30 minutes, eliminating the start-up bias.

When you have added these blocks, your model should look like:



Circuit-Step 7

Set the number of runs to 10 and the simulation end time to 8.5 (8 hours plus the 30 minute warm up period) in the Simulation Setup dialog, then run the model. When the 10 runs are complete, the table of results in the Activity Stats block should look something like this:



Block	Block Name	Arrivals	Departures	Utilization	Time (hrs)
0	CleaningStation	Station	621	620	0.99249
1	Wave Solderer	Conveyor Belt	627	616	0.4297
2	Auto Inserter	Activity_Multi	627	620	0.5065
3	NonStandard_1	Station (Attrib)	192	191	0.98785
4	NonStandard_2	Station (Attrib)	192	191	0.98819
5	NonStandard_3	Station (Attrib)	190	189	0.98264
6	Burn-in	Station	24	23	0.97083
7	CleaningStation	Station	621	620	0.98837
8	Wave Solderer	Conveyor Belt	627	617	0.42993
9	Auto Inserter	Activity_Multi	627	620	0.50653
10	NonStandard_1	Station (Attrib)	192	191	0.98854

Table in Activity Stats block

When you click the “Confidence Interval” button in the dialog of the Activity Stats and Queue Stats blocks, the blocks will generate confidence intervals based on the observations and the confidence level you select. For example, at a 95% confidence level, the Activity Stats block will look something like the following:

Block	Block Name	Arrivals	Departures	Utilization	Time (hrs)
0	Auto Inserter	Activity_Multi	626.7±0.2800	619.9±0.1833	0.5065±0.0001
1	Burn-in	Station	24.00±0.000	23.00±0.000	0.9716±0.0005
2	CleaningStation	Station	621.0±0.3865	620.0±0.3865	0.9908±0.0012
3	NonStandard_1	Station (Attrib)	192.1±0.1833	191.1±0.1833	0.9888±0.0004
4	NonStandard_2	Station (Attrib)	192.5±0.3055	191.5±0.3055	0.989±0.00049
5	NonStandard_3	Station (Attrib)	190.4±0.2993	189.4±0.2993	0.9833±0.0004
6	Wave Solderer	Conveyor Belt	626.8±0.2444	615.7±0.5499	0.4297±0.0001
7					
8					
9					
10					

95% Confidence level

This shows that you can be 95% confident that the utilization of the Cleaning station is between 98.96% and 99.20%. In other examples, you may see broader ranges. Note that a confidence interval is only as accurate as the model is. If, for example, the model is a poor representation of the actual system, the values derived from the confidence interval will be inaccurate as well.

Final analysis

As it turns out, merely increasing the capacity of the Automatic Insertion machine was insufficient to accommodate the addition of the new product type; two other operations needed increased capacity as well. This additional cost may have an effect on whether or not the new product line is introduced. By using simulation, you were able to predict the expected initial impact of the new

product (insufficient capacity at the Automatic Insertion machine), and also determine other impacts that were not foreseen.

You may have noticed that there are several factors that could have been included in this model but weren't. For example:

- Equipment downtime
- Labor
- Tracking circuit cards by type
- Processes outside the scope of the model, such as ordering the circuit cards or quality inspections
- Storage and retrieval (material handling) operations

Although most of these factors would be present in the real-world system, they were not considered important to the goal of the model and were therefore ignored.



Manufacturing Chapter 3: Overview of the libraries

The Manufacturing library is a toolkit for modeling discrete industrial and commercial systems. You use the blocks in this library, in conjunction with other Extend blocks, to create models that show how operations and processes work or how you would want them to work. You use the Statistics library blocks to accumulate and report important statistical information about the models you build.

In order to use the Manufacturing and Statistics libraries and Extend to the fullest, you first need to be familiar with discrete event modeling concepts. The main Extend manual describes events, items, and other concepts used in discrete event modeling. You should read and be familiar with those terms before you proceed.

Chapters 4 and 7 in the main Extend manual briefly discuss modeling concepts you should be aware of as you build models and how some of these concepts are applied to solve real-world problems. Remember that you build a simulation model and study its behavior in order to draw conclusions about the real-world system. How you build the model and what data and assumptions you use, directly affects the outputs you obtain and your analyses of those results.

Throughout this manual, references to blocks in libraries other than the Manufacturing library will be identified by library:

- Animation library
- Discrete Event library
- Generic library
- Plotter library
- QuickBlocks library
- Statistics library

The appendices of this manual also provide descriptions of each block in the Generic, Discrete Event, Manufacturing and Statistics libraries.

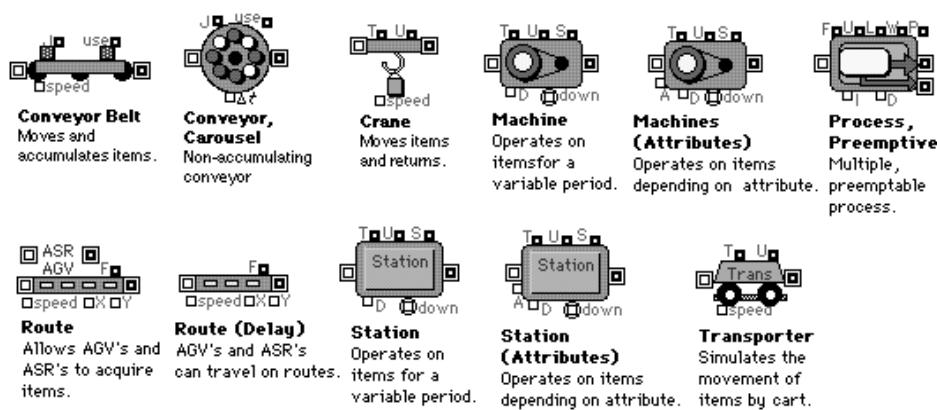
Manufacturing library blocks

The Manufacturing library (MFG) is composed of six different types of blocks: activities, batching, generators, queues, resources, and routing blocks.

Activities

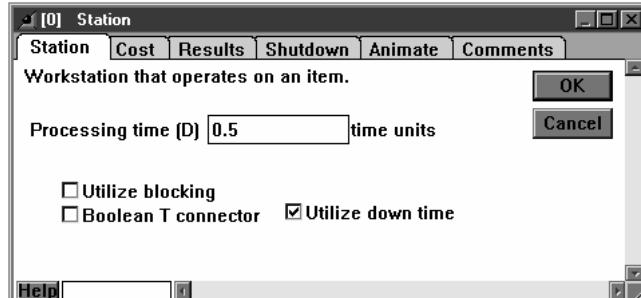
Activities are events that happen to resources (discussed below) or groups of resources, such as machining, waiting on customers, or moving items along a conveyor belt. Activities involve a delay or processing time that represents the amount of time it takes to perform the task in the real world.

The activity blocks regulate the flow of items through the model. Each activity block pulls in an item, processes it, then holds it until the next block is ready for it. The activity blocks in the Manufacturing library are: Conveyor Belt, Conveyor Carousel, Crane, Machine, Machine (Attributes), Process Preemptive, Route, Route (Delay), Station, Station (Attributes), and Transporter, as shown below:



Activity blocks in the Manufacturing library

Some blocks in the Discrete Event library, such as the Activity Delay, also simulate activities. A typical dialog of an activity block is:



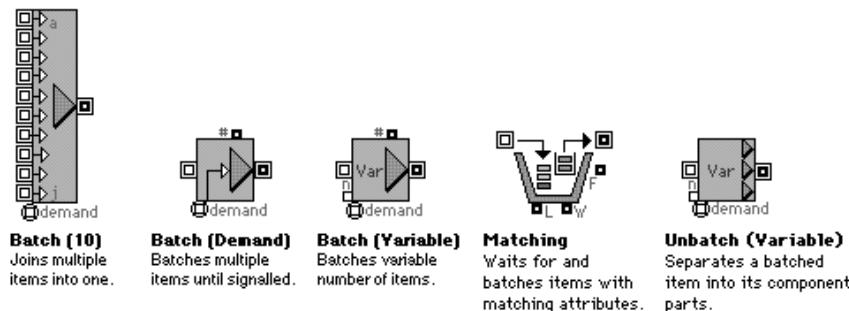
Station block dialog showing processing time (Windows)

MFG

Batching and unbatching

The *batch* blocks represent situations where items are permanently or temporarily assembled or joined together to form one unit. Blocks that *unbatch* correspond to disassembling one unit into its original components or into duplicates of itself. The concept of batching and unbatching is discussed in detail in “Manufacturing Chapter 9: Batching and Unbatching” on page M151.

The Discrete Event library contains the Batch and Unbatch blocks. The Batch (10), Batch (Demand), Batch (Variable), Matching, and Unbatch (Variable) blocks in the Manufacturing library provide even more flexibility; they are shown below:



Batching blocks in the Manufacturing library

Generators

Generators provide items. The Schedule block provides items according to a schedule and is typically used in conjunction with the *change* connector found in resource blocks to dynamically control the number of resources available throughout the run. The DownTime (Unscheduled) block

provides items at random intervals and is typically used in conjunction with the *down* connector on some activity blocks to simulate breakdowns and unscheduled maintenance.



Generators in the Manufacturing library

Queues

Queues hold resource items and release them in a specified order. The three queue blocks in the Manufacturing library (Buffer, Holding, and Queue Reneging) each hold items in FIFO (first-in-first-out) order. The Holding block accumulates items until a specified number has been reached, then releases them one at a time. The Queue, Reneging block has the additional feature of allowing items to leave prematurely, based on model conditions. The blocks look like:



Queueing blocks in the Manufacturing library

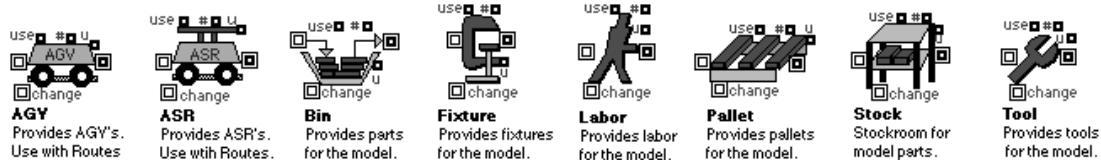
The Discrete Event library contains additional queueing blocks such as the Queue Attributes and Queue Matching blocks. For more information about queues, see “Queueing disciplines” on page M95.

Resources

A *resource* is a supply of items, such as tools, customers, or labor, that are involved in operations. Resources can be unlimited: you can provide a steady stream of resource items to the process for the duration of the simulation. Resources can also be limited: you can specify how many of a particular resource will be available to the process at any one time or even during the entire simulation run. Resources can be reused if you wish. This is most common with labor, where the worker works on a task for some time, then returns to the pool of available labor.

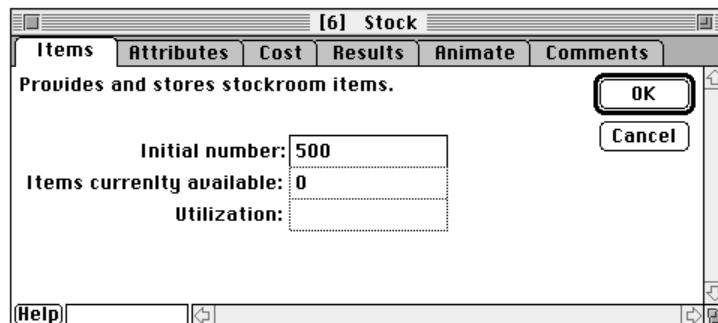
In Extend, resources can be modeled either explicitly as an item from a resource-type block or implicitly using an activity-type block (discussed above) which represents both the resource and

the activity the resource is performing. The resource blocks in the Manufacturing library are AGV, ASR, Bin, Fixture, Labor, Pallet, Stock, and Tool, as seen below:



Resource blocks in the Manufacturing library

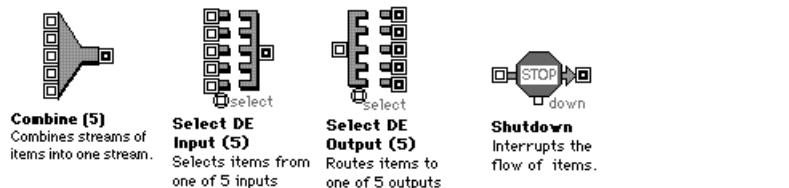
Resources can be generated at random or by schedule, or they can be provided directly from a passive source. For instance, you can attach a Generator block from the Discrete Event library to a Stock block to simulate parts being delivered to a stockroom at random intervals. If the items for your operations are delivered at regular intervals, you can connect a Program block (from the Discrete Event library) or Schedule block (from the Manufacturing library) to the Stock block. To specify a passive source, you would use a Stock block with the amount available entered in its dialog, without connecting a Generator, Schedule, or Program, such as:



Setting the amount available in a Stock block (Macintosh)

Routing

The remaining blocks in the Manufacturing library are routing blocks that help control the flow of items. They are: Combine (5), Select DE Input (5), Select DE Output (5), and Shutdown:



Routing blocks in the Manufacturing library

Statistics library blocks

The blocks in the Statistics library are divided into three groups: blocks which report statistical information for particular blocks by type, blocks that report statistical information for items that pass through the block, and blocks which aid in refining statistical data collection.

The “Stats” blocks

The “stats” blocks accumulate data and calculate statistics for particular blocks in the model. In addition to the block label, block name, and the time the information was observed, each of these blocks displays metrics that are specific to their block type, such as utilization or average wait time.

These blocks record their information in tables in their dialogs. One row of data is recorded for each block in the model each time an observation is made. Observations may overwrite previous observations (restart at the top of the table) or be appended to existing values.

To use these blocks, place them anywhere in the model. You can choose in their dialogs to update the statistics continuously (which significantly slows simulation time) or only at the end of the simulation run. You can also force a periodic or scheduled update of statistical information by connecting a block (such as a Schedule block) to the “Update” input connector. 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.

The “stats” blocks are Activity Stats, Cost Stats, Mean & Variance Stats, Queue Stats, and Resource Stats:



The Cost By Item block

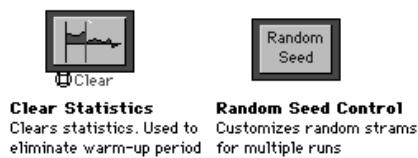
The Cost By Item block reports on the cost information of every item that passes through the block. The data is recorded in the table in the block’s dialog. The block can be instructed to record one row of data for every items that passes through it or to organize the data according to an attribute value. There is also a tab which displays the contents of the cost array for each item. This

tab is included primarily for debugging purposes. See the main Extend manual for a description of the cost array.



Other Statistics library blocks

The blocks in this group are Clear Statistics and Random Seed Control, as shown below:



MFG

- The Clear Statistics block clears and resets statistical accumulators in specified blocks at periodic intervals or in response to a system event. When you start a simulation run, the model often starts out empty. For example, queues have no items and operations have nothing to process. In this situation, gathering statistical data from when the simulation first starts could skew the results. The *warm-up period* is the amount of time a model needs to run before statistical data collection is valid. The Clear Statistics block is used to eliminate the statistical bias caused by the warm-up period.
- The Random Seed Control block controls when and if the seeds used throughout a model are re-initialized.

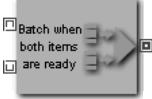
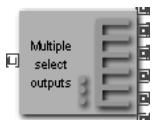
QuickBlocks library blocks

The QuickBlocks library provides instant solutions to common modeling problems in the form of hierarchical blocks. Simply pull in the appropriate hierarchical block from the QuickBlocks library, enter the information for your model, and connect it to the rest of the model.

Block	Function	Block	Function
	This configuration of blocks allows the size of the batch in the Batch (Variable) block to be defined by the value of another item's attribute.		This models a different delay for each item based on an attribute value. The attribute is used in a lookup table to calculate the actual delay.

M72 | Manufacturing Chapter 3: Overview of the libraries
 QuickBlocks library blocks

MFG

Block	Function	Block	Function
	Reads in a set of attributes from a file. The row read in is based on a Product ID attribute.		Both items that will be forming a batch wait outside of the batch block before the batch is formed. The batch is only created when both items are available.
	Dumps the contents of a queue at specific times. An example of this would be pulling all of the mail out of a mailbox so that it can be sorted.		Routes items to a large number of locations based on an attribute value. The Select DE Output (Discrete Event Library) and the Select DE Output (5) (Manufacturing Library) are both designed to be combined together to accommodate this.
	Models randomness for processing times.		The time required for an item to renege from a queue is random. This often occurs in service systems where the time that a customer will wait before abandoning the queue is a random value.
	When items arrive to a sequence of queues before identical operations (such as check-out lines in a supermarket) the most commonly used selection rule is to choose the shortest queue.		When a variable number of items are batched together, assigning an attribute to the number of items in the batch makes it easy to unbatch the same items later.
	The arrival rate changes by time of day, month, or year.		

Manufacturing Chapter 4: General Modeling Concepts

This chapter discusses general concepts that apply to the model as a whole. Time units, the length and number of simulation runs, model verification, and model validation will all be considered.

Time units

As described in Chapter 7 of the main Extend manual, there are two ways to deal with time units in Extend:

- Use a generic time unit in the model and in each block.
- Establish a global time unit for the model and use any time unit to define time-based parameters in the individual blocks (local time units).

This section assumes that a global time unit (a unit of time other than “generic”) has been established. You do this by choosing a global time unit in the Simulation Setup dialog from the Run menu.

The default time unit

Once you select a global time unit, all time-based parameters in the blocks in your model will be set to the *default time unit*. The default time unit will always be equal to the global time unit for that model. For example, if you change the global time unit from “hours” to “minutes”, all time units set to the default will also change from “hours” to “minutes”. The default time unit is denoted by an asterisk (*) next to its name in a block’s dialog, as shown below:

Delay: **hours***

Time-based parameter set to the default time unit

All time-based parameters for any new blocks added to the model will also be initially set to the default time unit.

The default time unit provides consistency when adding new blocks to the model, since every new block will initially be using the same time unit. It also allows users to easily convert a model built

using generic time units to a model using a global time unit, without affecting the simulation results. You do this by selecting a global time unit in the Simulation Setup dialog.

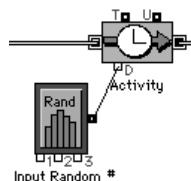
However, in most cases you will not want to keep all parameters in default mode. Explicitly selecting a time unit (even the same unit of time as the global time unit) for each local parameter in the model is a safer choice. Then, if you subsequently decide to change the global time unit, you will not accidentally affect the simulation results.

For example, assume you initially set your model's global time unit to "hours" and that you specify 2 as the delay time for an Activity, Delay block. However, instead of explicitly selecting "hours" as the local time unit for the activity block, you leave it in default mode (in this case, "hours*"). If you later change the global time unit from hours to days, the delay time for the Activity, Delay will become 2 *days*, adversely affecting simulation results. By explicitly selecting a local time unit (a time unit without the * after it) for all time-based parameters, you can prevent this type of error.

MFG

Setting time-based parameters using connectors

Some time-based parameters can be set using a connector value. For example, you can set the delay time of an Activity, Delay block by connecting an Input Random Number block to its "D" connector, as shown below:



Setting a time-based using a connector

In these situations, the value sent to the input connector must be defined in the time unit specified in the receiving block. For instance, assume that you want the delay in the Activity, Delay block to be approximately 30 minutes. If the local unit of time for the delay is minutes, you would set the Input Random Number block to generate numbers with a mean of 30. However, if the Activity, Delay block used hours as its local time unit, the Input Random Number block should be set to generate numbers with a mean of 0.5.

It is possible that you may be in a situation where you set numbers in one column of a block based on the time unit for that block, and set numbers in another column based on the time unit for a second block. An example of this is described in "Specifying the custom parameters" on page M83.

The length and number of simulation runs

An important consideration when building a model is *how long and how often should the simulation be run?* Your entries for the "end time" and "number of 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)

Following is an overview of these issues. A comprehensive discussion about terminating and non-terminating systems is beyond the scope of this manual. For more information, including how to calculate the number of runs and the run length, refer to *Simulation Modeling & Analysis* and *Improve Quality & Productivity with Simulation*; these books are listed in the section “Further reading” on page M7.

MFG

Terminating systems

Some systems obviously lend themselves to a natural determination of end time. For instance, most service operations and job shop projects 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 a walk-in customer service center which 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 M202, 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 exam-

ple, 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 2 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 your model simulates processes operating at their normal or steady-state level. In your models, start-up conditions may be unrealistic or nonrepresentative of the actual system and may bias simulation results. To overcome this, you could wait until after the warm-up period to gather statistics, reset statistics using blocks in the Statistics library (as discussed in “Clearing statistics” on page M205), or run the simulation for a very 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 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, you would typically model the 8 hours that a bank was open 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) for a total of 8 hours 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 the 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.

There are several excellent sources for determining the length and number of simulation runs. For more information, refer to the simulation and statistics texts mentioned in “Further reading” on page M7.

Model verification

The process of debugging a model to ensure that every portion operates as expected is called model verification. In the Tutorial, you performed part of this verification process by building the model in stages and with minimal detail, then running it at each stage to observe the results. A common verification technique could be termed *reductio-ad-absurdum* (reducing to the absurd) reducing a complex 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 which interact to see how they run on their own

MFG

Other methods for verifying models include making sure that you can account for all the items in a model, animating the model or portions of the model, or using diagnostic blocks from Extend’s libraries (there are several debugging-type blocks described in Chapter 7 in the main Extend manual.) For more information, see “Measuring and verifying simulation results” on page M203.

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 help-

ful to compare nonessential results which 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 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.

For more information, see “Measuring and verifying simulation results” on page M203.

Manufacturing Chapter 5: Items and Values

As discussed in the Introduction to this manual, items are what flow through the model. Informational values provide information about items and model conditions.

An *item* is a process element or object that is being tracked or used in a model. For example, items can be parts, customers, workers, telephone calls, data packets, and so on. Items are individual entities and can have unique properties that are specified by their attributes and priorities (discussed later). An item can only be in one place at a time. Items change state (physically move, are transformed, or are delayed) when events occur, such as a part being assembled, a customer arriving, and so on. In discrete event models, items arrive, are delayed and transformed, then exit the model.

Values provide information about items and about model conditions. Values tell you the number of customers in a queue, how many products have been manufactured, and how frequently data packets are sent. They also report delay time, operation utilization, and costs of processes. When you use a plotter in a discrete event model you are plotting information about items, not the items themselves. For example, when you connect from the “#” connector on an Exit block to the input of a plotter, you are displaying the time that each item left the model and how many items have exited.

Note Do not confuse informational values with *item Values* such as the “Value of item (V)” set in the dialog of the Generator block from the Generators submenu of the Discrete Event library. Each item can be cloned into duplicates of itself, for example to represent a group of items arriving at the same time. This is known as the number of items or *item Value*. See “Choosing an item’s Value” on page M82 for more information about item Values.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Change Attribute	Attributes	Discrete Event
Downtime (Unscheduled)	Generators	Manufacturing
Generator	Generators	Discrete Event
Get Attribute	Attributes	Discrete Event
Input Data	Inputs/Outputs	Generic
Input Function	Inputs/Outputs	Generic
Input Random Number	Inputs/Outputs	Generic
Machine (Attributes)	Activities	Manufacturing
Program	Generators	Discrete Event
Schedule	Generators	Manufacturing
Set Attribute	Attributes	Discrete Event
Set Attribute (5)	Attributes	Discrete Event
Set Priority	Attributes	Discrete Event
Station (Attributes)	Activities	Manufacturing

The models discussed in this chapter can be found in the “Examples \ Manufacturing \ Items” folder.

Item generation

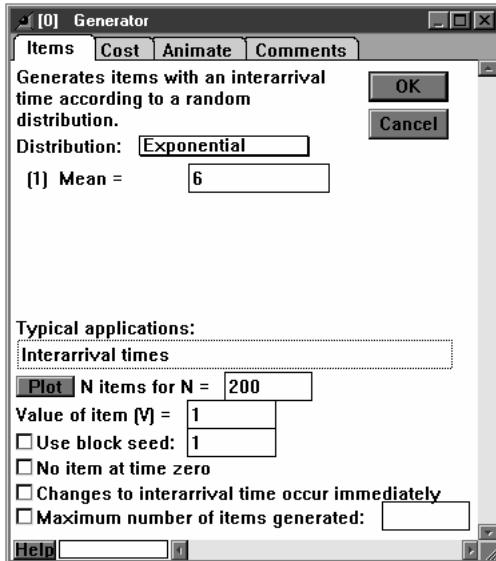
You can specify item arrivals using the Generator, Program, Schedule, DownTime (Unscheduled), or Input Data blocks.

Generating arrivals at random intervals

The Generator block is the most common method for generating random arrivals. The interval between item arrivals determines how frequently events occur; this is known as the *inter-arrival time*.

Choosing a distribution in the Generator block

The Generator block outputs items with a specified interarrival time. Its dialog contains several distribution choices in the “Distribution” popup menu as well as a table for entering data for an empirical or user-defined distribution.

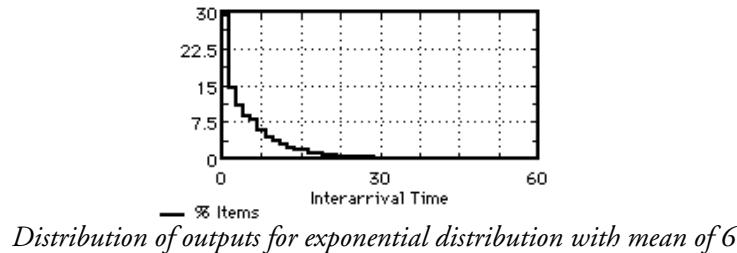


Generator dialog showing exponential distribution with a mean of 6 (Windows)

Each distribution is described fully in the Generator block’s Help and in the main Extend manual; they are also discussed briefly in the section “Random numbers and probability distributions” on page M194 of this manual. In the Generator block’s dialog, the parameter arguments for each distribution are labeled (1) for the first argument, (2) for the second argument, and (3) for the third argument, if used. For instance, for the exponential distribution selected above, the first argument is the “mean”. Because the second and third arguments are not used, the input boxes for those parameters do not show up in the dialog.

When you choose a distribution in the dialog of the Generator block, you are specifying both the interval between item arrivals (the interarrival time) and the characteristics of the rate of arrival. For example, when you choose an exponential distribution with a mean of 6, one item will arrive approximately every 6 time units for the duration of the simulation. However, the *shape* of the

exponential distribution dictates that it is more likely that the time between arrivals will be between 0 and 6 than between 6 and 12:



When building models, it is common to use a constant or uniform (integer or real) distribution in the early stages 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 your real-world processes.

MFG

Choosing an item's Value

You can choose that one item is input at each arrival event – this is the most common case and the Generator defaults to this option. Alternately, you can specify that a number of items is released at each event. You do this by using the “Value of item (V)” option in the dialog, changing the item’s Value to a integer number greater than 1.

For example, assume the simulation time units are in minutes and you want to show that one part arrives approximately every 6 minutes. To do this, select the Exponential distribution, enter a 6 in the entry box labeled “(1) Mean”, and leave the “Value of item (V)” set to 1, as seen in the Generator dialog above. To show that 3 part/items arrive every 6 minutes, keep the settings at Exponential with a mean of 6, but change the “Value of item (V)” entry to a 3. When you do this, the block will output one item with an Value of 3 approximately every 6 minutes.

As discussed in the main Extend manual, the blocks that follow the Generator block determine how the item with a Value greater than 1 is treated. For instance, if an item with a Value of 3 goes directly into a queue, buffer, or resource block, it will be split into three items each with an item Value of 1. However, if the item goes directly into an activity-type block, it will be treated as a single item with a Value of 3. In most cases, you will want to follow the Generator block with a queue or buffer, which will decompose the item into three separate items. For more details, see the main Extend manual.

Note In most cases, you probably will not want to generate more than one item at each event. For example, rather than inputting 3 items every 6 minutes as discussed above, you would probably want to generate 1 item every 2 minutes. This is because, unless they are in one box, it is not common to see three parts arrive at exactly the same time; parts are more likely to arrive at slightly different times.

Using the Downtime (Unscheduled) block

As described in “Interrupting or shutting down activities” on page M143, various activity blocks can be shutdown by sending an item as a signal into the “down” connector. The value of the item can be used to determine the length of the shutdown period. The Downtime (Unscheduled) block is an excellent way to generate items for this purpose. Within this block, you may specify a distribution for the time between shutdowns as well as a separate distribution for the duration of the shutdown.

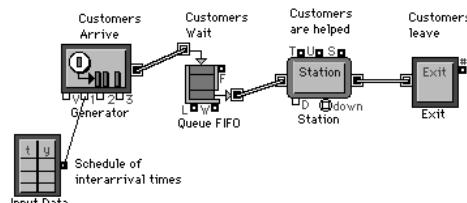
Generating arrivals with custom intervals

You might want to show items arriving randomly where you customize some aspect of the distribution. You do this using an Input Data block from the Inputs/Outputs submenu of the Generic library to modify one of a distribution’s arguments.

Note The Time Dependent Arrival block in the QuickBlocks library (QuickBlocks.lix) gives you a quick way of implementing this in your model. See the Quick Blocks library for other solutions.

Specifying the custom parameters

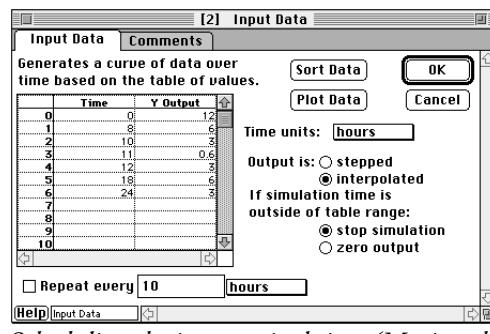
In the simplest case, you use an Input Data block to specify one or more of the parameter arguments for the distribution selected in the Generator block. This is an example of dynamically changing a value, as discussed in “Changing parameters dynamically” on page M201. For example, you can use the Input Data block to specify the mean of an exponential distribution so that it changes based on the time of day:



Custom intervals model

In the dialog of the Generator block, choose the exponential distribution so that items (in this case parts) arrive exponentially. Notice that the mean of the distribution is specified by the parameter labeled “(1) Mean =”. When you connect the output of the Input Data block to the “1” input connector of the Generator block, you are specifying that the mean of the distribution will change during the simulation run, overriding any entry in the dialog. In this case, this allows you to dynamically change the average time between arrivals (the interarrival time).

Notice that a smaller mean value indicates that there is less time between arrivals, so parts arrive more frequently. Since the mean is smallest from time 10 until time 12, the values in the Input Data dialog cause parts to arrive more frequently for that period. The Input Data block dialog is:



Scheduling the inter-arrival time (Macintosh)

MFG

Choosing the correct time units for the columns

The table in the Input Data block is used to define the output value (“Y Output” column) at a given simulation time (“Time” column). The time units popup menu at the right of the dialog is set to hours and represents the unit of time used to define the values in the “Time” column; this block is set to output values for the period from 0 to 24 hours. This corresponds to the simulation start and end times specified in the Simulation Setup dialog.

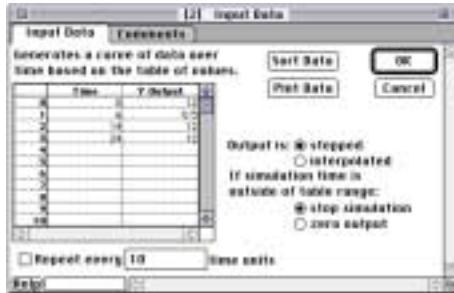
In this model, the “Y Output” values are sent to the Generator block and are used to calculate the average time between arrivals. As noted in “Setting time-based parameters using connectors” on page M74, when a value passed to a block’s input connector is used to set a time parameter (as in this case), the value sent must be defined in the time unit that the receiving block expects. Since the Generator block is using minutes as its local time unit, the “Y Output” values of the Input Data block must also be stated in minutes. For instance, the value of 6 in the Y Output column represents a mean or an average of 6 minutes between arrivals.

Note Do not set the interarrival time to “0” for an Exponential distribution. The Generator 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 Generator block when you vary the mean of the arrival intervals over time. The Generator’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 Generator 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 Generator 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.

Assume you connect an Input Data block to the “1” input connector of a Generator block, varying the interarrival mean according to the following table:



Dialog of Input Data block

Both the “Time” and “Output” columns 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. In this example, it is possible for an item to be generated at time 0 with an arrival time of 14 or more. If that happens, the Generator will ignore the fact that the mean should have changed to 0.5 between hours 6 and 14, and the number of arrivals will be much less than expected.

MFG

In the dialog of the Generator block there is a check box labeled “Changes to interarrival time occur immediately”. When checked, it will cause the Generator block to immediately respond to changes to any input parameter. In the case of the above example, the Generator 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 and interpolating from the original arrival time.

Scheduled arrivals

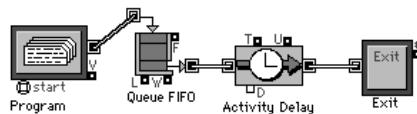
You can easily schedule item arrivals to occur at fixed intervals using the Program block from the Discrete Event library. For example, assume you receive 500 shipments of parts a week, but that almost all of them are received on Wednesday. You can enter the arrival times (“Output Time”) and the number of shipments arriving (“Value”) in the dialog of the Program block, a portion of which is below:

Row	Output Time	Value	Priority	Attrib. value
0	1	50		
1	2	50		
2	3	500		
3	4	50		
4	5	50		
5				
6				
7				
8				
9				

Scheduling the arrival of orders (Macintosh)

When you use the Program block, you enter the actual arrival times rather than entering the time between arrivals as was done with the Generator block; the Program block automatically calculates

the time between arrivals based on your entries. In this example, there are five arrival events (one event each on days 1 through 5), each with an item value of either 50 or 300. The table indicates that on the third day (Wednesday) you receive 300 shipments while you only receive 50 shipments on each of the other days. Remember that since the Program block always pushes items, you would follow it with a queue or buffer block to avoid losing items, as shown in the model:



Scheduling fixed interval model

MFG

Like the Generator block, the Program block outputs an item with a Value greater than 1 as if it were a group of items all arriving at the same time. When the item goes to a queue or buffer block, it becomes multiple copies of itself. For example, as each item is input from the Program block to the Queue, FIFO block, it will become either 50 or 300 units, depending on its Value.

The Schedule block is also used to generate items based on a schedule. This block is typically used in conjunction with resource and activity blocks to model scheduled shift changes and shutdowns. See “Scheduling resources” on page M167 and “Scheduling activities” on page M135 for examples of how to use the Schedule block.

The Shift block is very useful in scheduling and in setting up shifts. See “Shift block used to schedule” on page M137, and “Using the Shift block” on page M169.

The “start” connector

The Program and Schedule blocks both have *start* connectors that you can use to control the output times of the data entered in their dialogs. How the timing in the blocks works depends on whether or not start is connected.

- If the start connector is not connected, the timing of the schedule entered in the dialog is absolute simulation time compared to the time entered in the Simulation Setup dialog. For example, start time 2 in the Program dialog would occur at simulation time 2, as long as simulation time 2 happens. If the starting time entered in the Simulation Setup dialog is 0, the Program would begin 2 time units after the simulation began. However, if the starting time entered in the Simulation Setup dialog is 4, the program at time 2 will never occur.
- If the “start” connector is connected, such as to a Decision block from the Decision submenu of the Generic library, the time entered in the dialog is relative to when the “start” connector is activated. For instance, assume simulation starting time is 0 and the Program block is scheduled to start at time 11. If the “start” connector gets activated at simulation time 5, the Schedule block would start its program at time 16 (5 plus 11).

The “start” connector is activated whenever it gets an item or sees a true value (a value >0.5). Once “start” is activated, it causes the entire schedule to happen. If “start” is activated while the schedule is happening, the new starting message is ignored.

Attributes

Attributes play a very important role in a discrete event simulation. As discussed in the Extend manual, an attribute is a quality or characteristic of an item that stays with it as it moves through the model. Each attribute consists of a name and a numeric value. The name identifies some characteristic of the item (such as “size”) while the value gives the dimension of the characteristic (such as “15”).

- You can attach up to 300 attributes to any item flowing through a simulation.
- You can find which block uses an attribute, rename, and delete attribute names in the Executive block’s Attributes tab. See “Managing attribute names” on page M91.
- You use the Set Attribute, Set Attribute (5), Change Attribute, Get Attribute, and DE Equation blocks to set, modify, and check attributes.
- You use the Queue, Attribute and Queue, Matching blocks (from the Queues submenu of the Discrete Event and Manufacturing libraries) to release items from queues based on attributes.
- The Queue Decision block (Manufacturing library) allows sorting (ranking) and conditional release based on an equation which can use attributes.
- The Matching block (from the Queues submenu of the Manufacturing library) pulls in items and batches them based on attribute values.

MFG

Most of the resource-type blocks let you specify default attributes for items in their dialogs.

Setting attributes

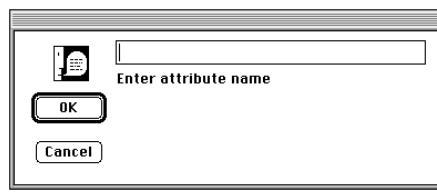
You can attach many attributes to any item flowing through a simulation. There is a limit of 300 unique attribute *names* for a model, but you can use any or all of those 300 attribute names for any item in the model. If you try to add a new attribute that causes the number of attributes to exceed the limit, Extend will warn you.

Attributes are meant to be unique; if you attempt to add a new attribute with exactly the same name as an existing attribute, Extend 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.

You usually set attributes for items just after they have entered the model from the Generator, Program, or Schedule blocks, then have all items enter a queue or buffer. The most common method for assigning attributes to an item is to pass items through the Set Attribute or Set Attribute (5) blocks (the Set Attribute (5) block allows more flexibility in setting attribute values). Another

method is to set an attribute in the dialog of a block that has a default attribute setting, such as the Bin, Fixture, and Labor blocks (from the Resources submenu of the Manufacturing library). You can set up to 7 attributes to an item with each Set Attribute block, and up to 5 attributes with each Set Attribute (5) block. You can use more than one Set Attribute block to assign attributes to an item.

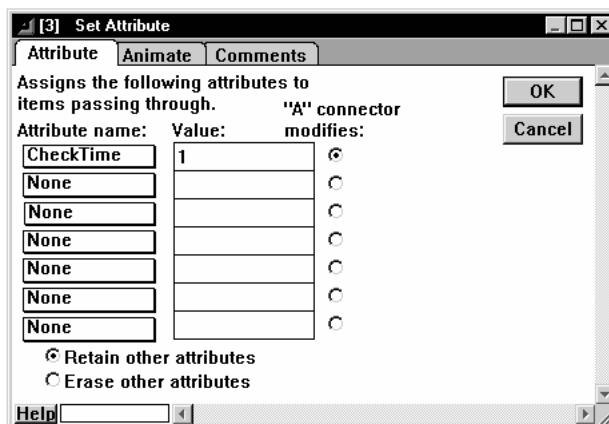
All the attributes in a model are accessible from popup menus in the block dialogs. To create a new attribute, select “New attribute” from the attribute popup menu and type the name of the new attribute in the New Attribute dialog.



New attribute dialog (Macintosh)

MFG

To set an attribute, select the attribute in the popup menu (or create a new attribute), then enter the value of the attribute into the box to the right of the popup menu, as shown below. As each item arrives in the block, the attribute information indicated in the dialog will be assigned to that item.



Setting the attribute of an item in the Set Attribute dialog (Windows)

To set attribute values dynamically, use the value input connectors on the Set Attribute and Set Attribute (5) blocks to override the attribute value set in the dialog. For instance, you can modify the values for up to 5 attributes using the value input connectors on the Set Attribute (5) block. This is often done using an Input Random Number block or Input Data block.

Changing attribute values

If you have an existing attribute value that you want to modify, the most flexible way to do this is with the Change Attribute block. This block allows you to modify the value of an existing attribute by using a constant value, a value coming from a connector, or the value of another attribute present on the item. You can either multiply, divide, add, or subtract the two values to arrive at a new attribute value.

The Machine (Attributes) and Station (Attributes) blocks also allow modification of the value of the attribute selected in the popup menu in their dialogs. This is convenient when the modification is based on the processing that the machine or station is performing on the item.

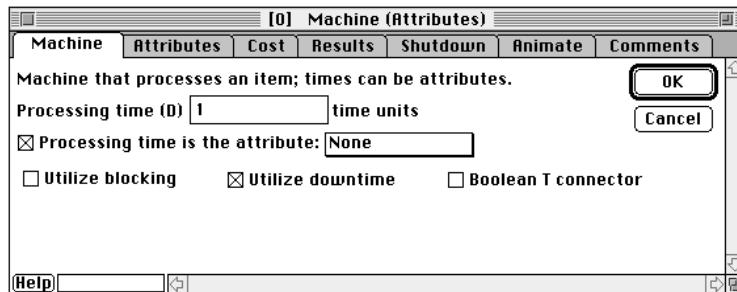
The DE Equation block can set an attribute value based on a user defined equation in that block.

If you just want to dynamically change the existing value of an attribute, use the Set Attribute block or the Set Attribute (5) block, as discussed in “Setting attributes”, above.

MFG

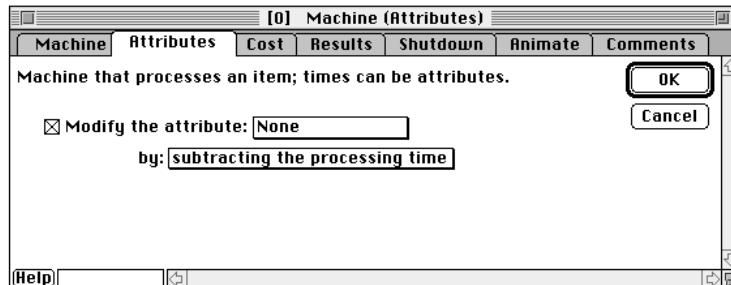
Reading attribute values and reporting changes

In order to manipulate an item based on the attribute value, usually to route it or process it, you will need to read the item’s attribute value. The most common method for reading attributes is to select the attribute by name from the list in the attribute popup menu in an attribute-reading block, such as the Get Attribute block or the Machine (Attributes) block. For example, the dialog of the Machine (Attribute) block is:



Machine (Attribute) dialog (Macintosh)

In the dialog, you can specify that an item's attribute value be used as its processing time. You can also modify the attribute value, for example by decreasing it to represent the time this block took to process the item.



Attributes tab of Machine (Attributes) dialog (Macintosh)

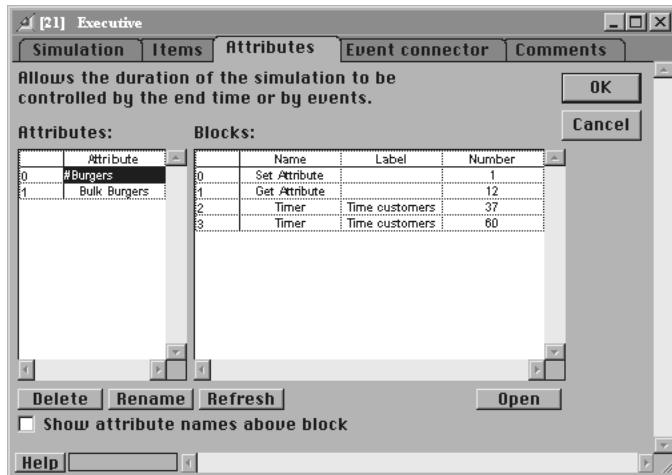
MFG

The Get Attribute block also has a Δ (*delta*) connector for reporting when an attribute value changes. The " Δ " connector outputs a 1 when an item's attribute value (for the attribute specified in the dialog) differs from the previous item's attribute value. Otherwise it outputs 0. You use this, for instance, to determine when there is a new type of item or when an attribute value used for processing time changes. For example, you can have an attribute called "Type" with values which specify the item type. When the value of Type changes, indicating a new type of item, the " Δ " connector outputs 1. This is shown in the section "Adding setup time" on page M133.

For every model, there exists a set of attribute names that have been created for use by the items in that model. Not all items will make use of every attribute name in the model. 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 Attribute block). Within the dialog of a Get Attribute block, you can choose to have the Δ connector output a 1 when the attribute is not used by the item. In this case, the Δ connector will output a 1 when the value of the specified attribute changes (as it does normally), and it will also output 1 when the attribute is not used by the item. Otherwise, it will output a 0 when the attribute has the same value as the attribute for the previous item.

Managing attribute names

The Executive block's Attributes tab shows each block that uses a specific attribute name and can be used to delete or change attribute names.



Executive block's Attributes tab

This is very useful when your modeling needs change. It is a central place where all attribute name management can be done.

Using attributes

Attributes are commonly used to specify amount of work time required, routing instructions, operation times, or part quality in statistical process control. They are also used to reflect progress in the amount of work done, cumulative values, or a change in status or quality. The Machine (Attributes) block and the Station (Attributes) block can use attribute values to determine the amount of operation time needed for an item. Like the Change Attribute block, they can also modify existing attribute values.

Assigning properties

Attributes are useful for assigning properties to items and then using those properties to specify a processing time or to route the items for processing. For example, you can use attributes to specify things such as "needs final check" or "ship unchecked". There are two ways of using attributes for this:

- Set an attribute value to the amount of time it takes to process the item, where 0 means that the item doesn't need any further processing, as described in "Custom processing time" on page M130.
- Set an attribute value so that it dictates the route. For instance, you can set the attribute to a yes-or-no (1 or 0) value indicating whether or not a particular process is applied to the item, as in

“Routing decisions based on attributes” on page M115. You can also have attribute values indicate which of several processes will be performed, as in “Routing decisions based on attributes” on page M115 or “Machines that can only process certain types of items” on page M122.

Accumulating and tracking information

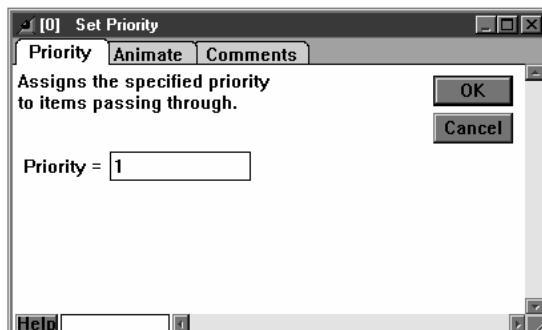
Attributes are also useful for accumulating and manipulating information about items, such as cost, total processing time, weight, or quality. The Machine (Attribute), Station (Attribute), and Change Attribute blocks can modify existing attribute values by adding or subtracting a percentage or a value. You add to an existing attribute value to provide cumulative information such as time, as shown in “Accumulating data using attributes” on page M206. You can also subtract from attribute values to indicate a change in status or quality or to show the remaining processing time for an item which undergoes several operations, as shown in “Cumulative processing time: time sharing” on page M131.

MFG

Attribute blocks are often used in conjunction with other blocks to track information about items through the model. For instance, the section “Variable batching and unbatching” on page M159 shows how to use the Set Attribute and Get Attribute blocks to keep track of the number of items in batches which have varying sizes. The section “Adding setup time” on page M133 illustrates how a change in attribute value can be used to indicate a change in item-type, necessitating a reconfiguring of machines.

Priorities

Priorities allow you to specify the importance of an item. The lowest value (including negative values) has the top priority. You can assign priorities to items in discrete event models and manipulate items based on those priorities. The Set Priority block is used to assign priorities to items:



Setting an item's priority (Windows)

You usually assign priorities just after items enter the model from the Generator, Program, or Schedule blocks, then have all items enter a queue or buffer. 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.

In Extend, items can only have one priority. (If you need multiple levels of priorities, you should use attribute values instead.) When a new priority is added to an item that already has priorities, the new priority prevails. When items are batched, the highest priority of the items prevails in the resulting batched item.

Blocks in the Discrete Event and Manufacturing libraries pass items' priorities with the items. Most of the resource blocks let you specify default priorities for items. The major priority-manipulating blocks are in the Discrete Event library, as shown below:

- The Set Priority block assigns a priority to an item.
- You can view an item's priority with the Get Priority block.
- The Queue Priority block releases highest priority items first.
- You can select the input that has the highest priority with the Select DE Input block from the Routing submenu of the Discrete Event library or the Select DE Input (5) block from the Routing submenu of the Manufacturing library.

Notice that the Prioritizer block (from the Routing submenu of the Discrete Event library) does not assign or use items' priorities. It allows you to prioritize its output connectors so that an item will be routed to the first available connector with the highest priority. As shown in “Explicit ordering” on page M114, the Prioritizer block prioritizes the *path* an item will take rather than the item itself.

The section x shows how priority values are used to determine if one item should preempt another.

Note 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, Priority block if they have to wait there with other items.



Manufacturing Chapter 6: Queueing

Queues provide buffers for items awaiting further processing. The Manufacturing and Discrete Event libraries have several blocks which allow you to specify queues or waiting lines. This chapter discusses the use of these queues.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Buffer	Queues	Manufacturing
Holding	Queues	Manufacturing
Queue Attributes	Queues	Discrete Event
Queue Decision	Queues	Manufacturing
Queue FIFO	Queues	Discrete Event
Queue LIFO	Queues	Discrete Event
Queue Matching	Queues	Discrete Event
Queue Priority	Queues	Discrete Event
Queue Reneging	Queues	Manufacturing
Queue Resource Pool	Queues	Discrete Event

Queueing disciplines

There are several predefined scheduling algorithms, also known as *queueing disciplines*, in the Discrete Event and Manufacturing libraries.

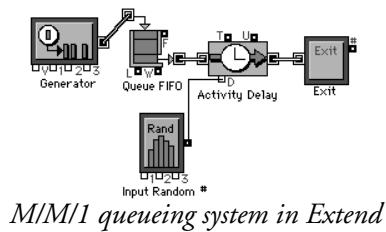
- The Queue FIFO block represents a first-in, first-out or first-come, first-served queue.
- The Queue LIFO block represents a last-in, first-out queue.
- The Queue Priority block reads items' priorities and passes the items out highest priority (lowest priority number) first.
- The Queue Attribute and Queue Matching blocks allow you to define custom scheduling algorithms based on item attributes.

- The Queue Decision block allows a user defined equation to decide the sorting order. 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.
- The Queue Resource Pool holds on to items until a specified number of resource pool units become available.
- The Buffer block acts like a FIFO queue.
- The Queue Reneging block allows you to specify how long an item will wait in the queue until it reneges, or prematurely leaves. This can also be useful to facilitate “Jockeying,” where an item can move back and forth between lines to try and get into a shorter line.
- The Holding block accumulates items until a specified number has been reached, then releases them individually.

MFG

Queue/server systems

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 inter-arrival 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 inter-arrival times/ exponential service times/ single server: unlimited queue length/ infinite population/ first-in-first-out service*. In Extend, the Generator block from the Generators submenu of the Discrete Event library is used to provide items at exponential inter-arrival times (and many other inter-arrival times as well). The Queue FIFO block holds the items, releases them first-in, first-out, and can have a maximum queue length specified in the dialog. The Activity Delay (from the Activities submenu of the Discrete Event library), Machine, and Station blocks (from the Activities submenu of the Manufacturing library) represent servers: you specify an exponential service time by connecting an Input Random Number block (Inputs/Outputs submenu of the Generic library) to the *D* (delay) connector on those blocks. A typical M/M/1 system expressed using Extend blocks would look like:



M/M/1 queueing system in Extend

Queueing considerations

Once items are generated for the model, it is common that they will be held in a queue or buffer block. The models discussed in this chapter can be found in the “Examples \ Manufacturing \ Queues” folder.

Choosing a queue

The Discrete Event and Manufacturing libraries allow you to select the type of queue (FIFO, LIFO, priority, or queueing by matching attribute names and/or values) required for your model. The Buffer, Holding, Queue Reneging and Queue Resource Pool blocks act like FIFO queues. The Queue Decision block allows a user defined sorting rule, the holding block has the additional feature of accumulating items then releasing them individually, the Queue Reneging block allows items to renege or prematurely leave the queue if they have waited too long, and the Queue Resource Pool block holds on to items until a specified number of resources pool units are available.

It is important to remember that, except for a FIFO queue, there must be other items in the queue at the same time to allow the queueing disciplines to work appropriately and affect the order of the items. For example, if you choose the Queue Priority, and there is never more than one item in the queue at a time, the effect of queueing based on priority is negated.

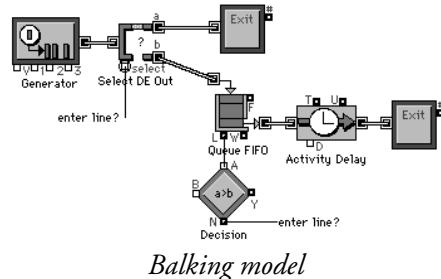
Blocking

Blocking occurs when an item is finished processing, but is prevented from leaving the block because the next process or operation is not ready to pick it up, or the next resource block is full. Blocking is most common in serial operations such as when there are several activities in a row performing tasks. In that case, each activity has the potential for blocking arriving items. Blocking also occurs when activities are not preceded by queues, 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 M126, “Successive ordering” on page M113, and “Machines that can only process certain types of items” on page M122 illustrate potential blocking situations.

Balking

Sometimes customers enter a facility, look at the long line, and immediately leave. This is an example of *balking*. Balking is typically represented by having a Decision block (from the Decision submenu of the Generic 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 block routes the item out of the model before it enters the queue.



Balking model

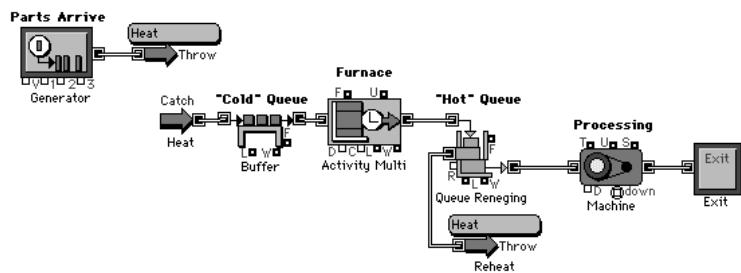
MFG

In this example, the Decision block monitors the length of the queue. If the queue length is less than the threshold defined in the dialog of the Decision block, the “N” output of the Decision block will output a 1. This instructs the Select DE Output block to route the item through its bottom output connector. If the queue length is greater than the threshold, the “N” output will output a 0 and the Select DE Output will route the item out its top connector to the Exit block.

Reneging

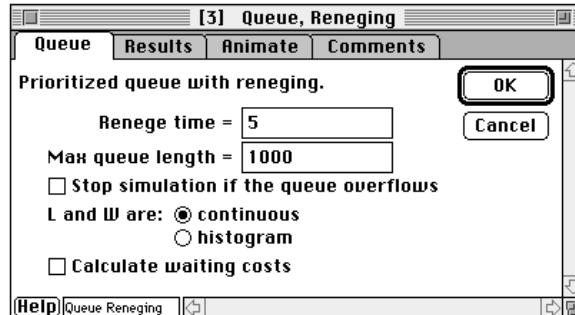
Reneging occurs when an item, having entered a queue, leaves before it reaches the output. For example, telephone callers who are put on hold will hang up without getting help if they feel they have waited too long for assistance. You can simulate this by using the Queue Reneging block as shown in the following example.

In the following model, parts wait in a buffer until they can be heated by a furnace, then wait for processing. If too much time passes before a part is processed (such that it cools down), the part is sent back to the buffer to wait for reheating:



Reneging model showing parts being reheated if not processed promptly

The dialog of the Queue Reneging block specifies how long each part will wait before it must be reheated:



Queue Reneging block with "Reneg time" set to 5 (Macintosh)

To set the reneging time dynamically or based on model conditions, connect to the block's "R" input connector.

MFG

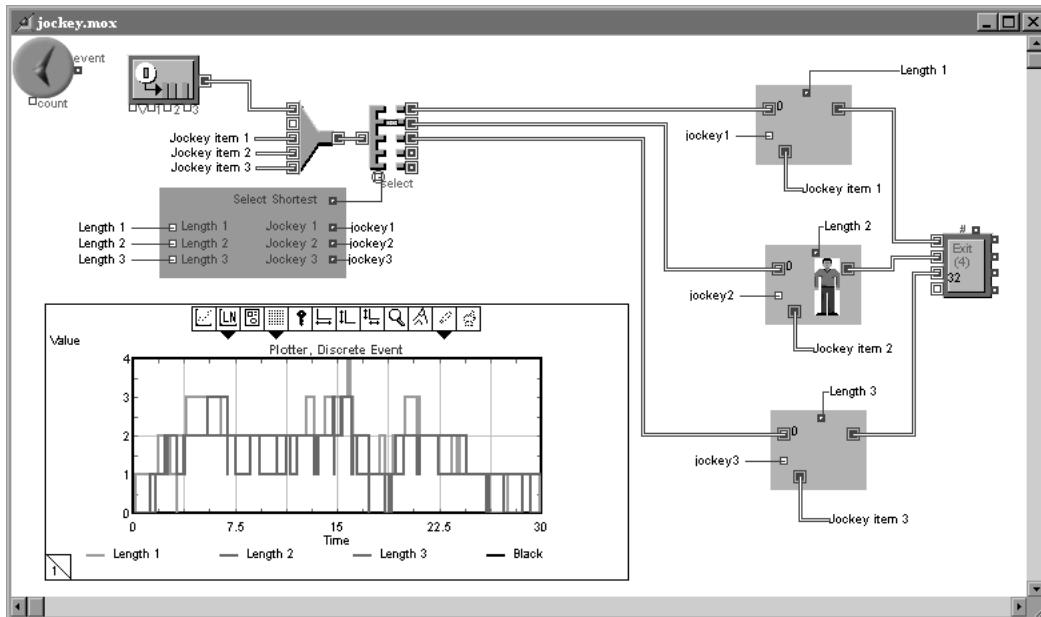
As seen in the Results tab, the Queue Reneging block automatically counts and reports how many items have reneged. Items that renege leave through the item output connector on the left of the block. Reneging items can be routed back to the original line (as in the example above), routed elsewhere in the model, or they can exit the model.

Jockeying

You can see a good example of jockeying if you go to the supermarket. Watch as people leave the end of a slow moving cashier's line to try and get onto a faster line.

The Queue Reneging is useful in building a model of this type of behavior. Normally, items renege if they have spent too much time in the queue, but the Queue Reneging block has a con-

nector that can force reneging of the last item in the queue. The Jockey.mox model is a good example of this:



Jockey.mox model showing hierarchical blocks that calculate the shortest queue

The leftmost hierarchical block forces the longest queue to renege its last item and then routes that item to the shortest queue.

Sorting using the Queue Decision

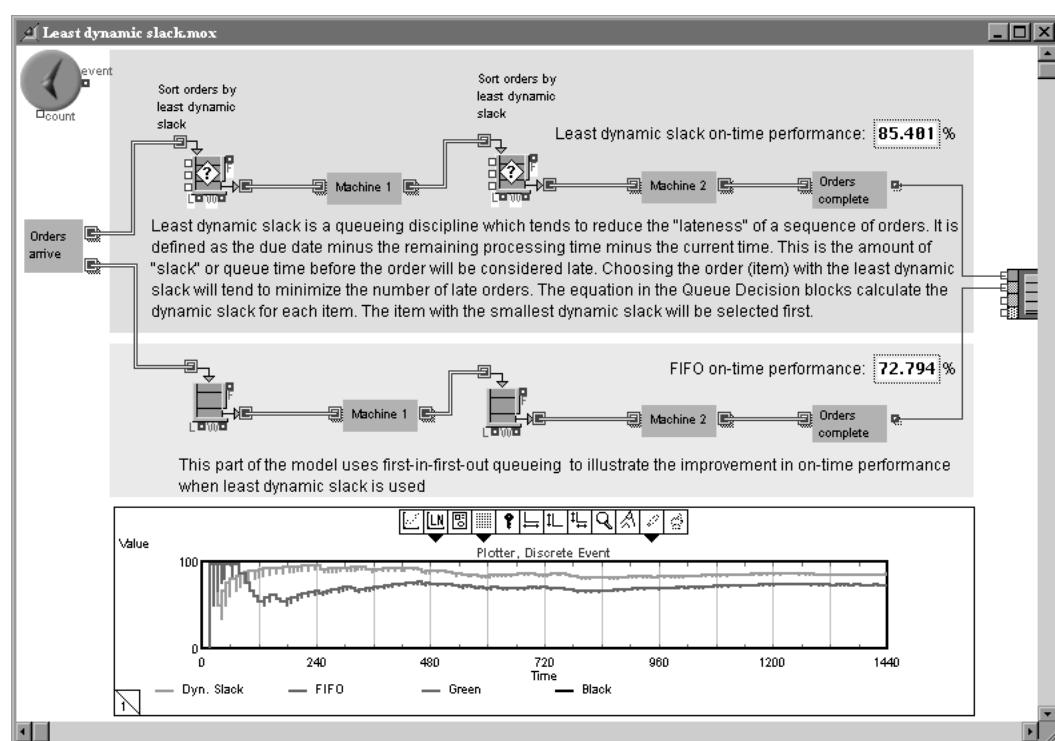
The Queue Decision block uses an equation to determine the ranking rule of the queue. The equation is written as a comparison rule which is evaluated for each item in the queue. The item that has the highest or lowest equation result (specified in the dialog) is placed at the front of the queue. If the result of the equation is within a certain tolerance, a secondary equation is used as a tie breaking rule. Optionally, if a blank is returned by the equation for all of the items, nothing will be released from the queue. A number of variables can be used to determine the ordering of the queue these include:

Variable	Uses
Attribute	An attribute on the item currently being evaluated
Priority	The priority of the item evaluated
Arrival time	The time that the item arrived to the queue
Queue rank	The rank of the item as if this were a FIFO queue

Variable	Uses
Connector	One of three input connectors
Best result	The best (highest or lowest) equation result so far

Least dynamic slack

A least dynamic slack ranking rule uses the remaining process time and the duedate of the item to sort the items in the queue. Slack is the due date less the remaining processing time. In essence, it is the “float” that is available before the item is due to be completed. In using slack to sort the items priority is given to those items that are closest to being late. The example model “least dynamic slack.mox” illustrates the improvement in on-time performance that can be achieved when sequencing orders by least dynamic slack instead of first in first out.



MFG

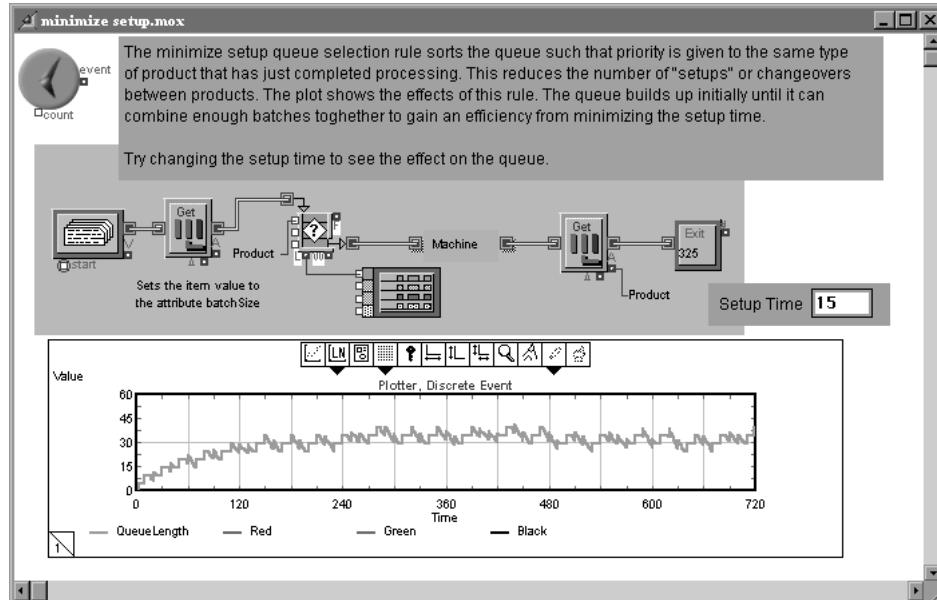
Least Dynamic Slack model

Minimize setup

In some systems, setup time, or the changeover from one product to another, can add significant delay to the processing of items. In this 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. The model “Minimize setup.mox” compares the attribute “Type” of each item in the queue to the “Type” attribute on the item that has just

M102 | Manufacturing Chapter 6: Queueing
Queueing considerations

completed processing. The first item with its “Type” attribute equal to the item that has just completed processing is released first. If no item in the queue have the same attribute value as the item that has just completed processing, the first item in the queue is selected.

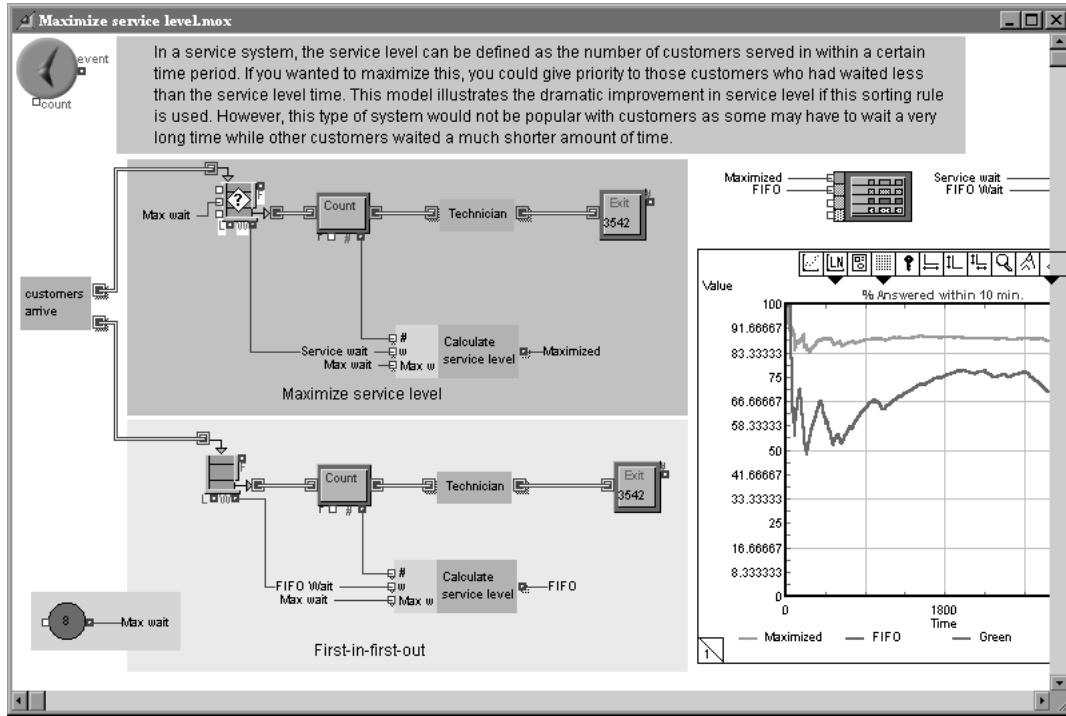


Minimize Setup model

Maximize service level

In some service systems technicians are rated on the percentage of customer requests fulfilled within a certain time period. The measurement is the percent of the customers served within this time period. The “Maximize service.mox” model uses a Queue Decision block to sort the queue into two priority levels. If the customer has waited longer than 10 minutes, they are placed at the

back of the line, otherwise the customer priority is the customer that has spent the longest time (up to 10 minutes) in the queue.



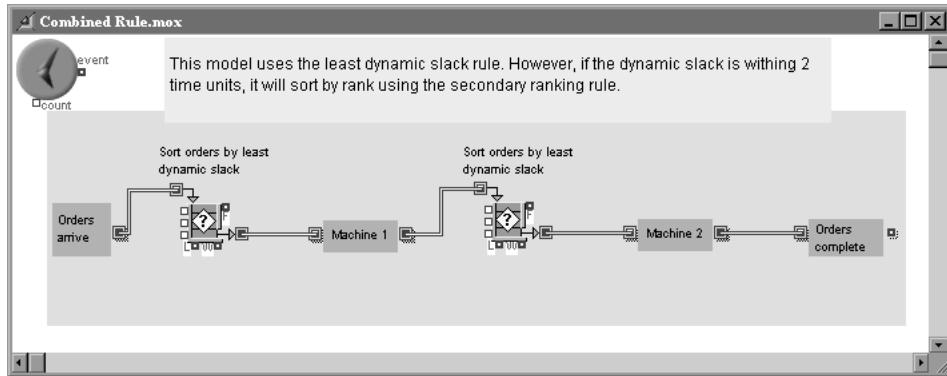
Maximize Service Level model

Combined rules

Because of the tie-breaking feature of the second equation, the Queue Equation block can be used to model situations where two items are considered equal with the primary sorting rule and a secondary rule is used to determine the item with the higher priority. The “Combined Rule.mox”



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.



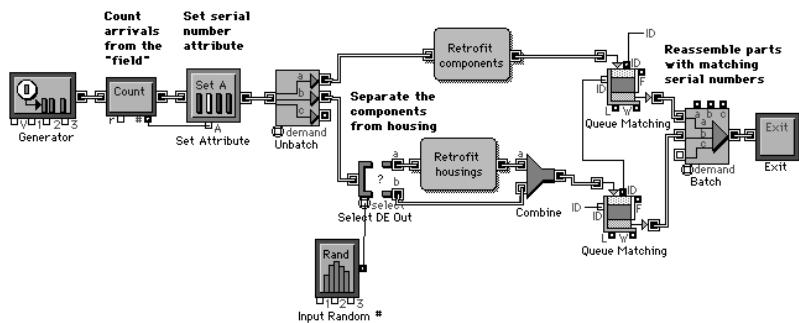
MFG

Combining Rules model

Matching queues

The Queue Matching block releases items only if they use a specified attribute *and* the attribute's value matches the value at the "ID" connector. This 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.

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.



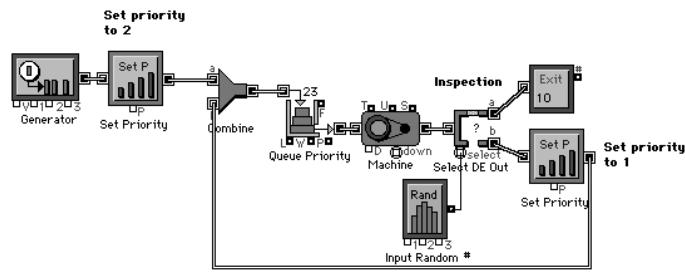
Matching model showing disassembling and reassembling of components

The Count Items block (from the Information submenu of the Discrete Event library) counts each electronic system as it arrives and outputs the total number that have passed through the

block. The Set Attribute block uses this value to set a “serial number” attribute for each electronic system. After refurbishment, the system is reassembled using its original parts.

Priority queues

When the block downstream can accept an item, the Queue Priority block will search through the contents of the queue and release the item with the highest priority. In the following example, items enter the model and immediately have their priority set to 2. After the machining processes, each item is inspected for flaws. If the item does not pass inspection, its priority is set to 1 and it is sent back to the Queue Priority where it waits to be re-machined. When the machine can accept a new item, the Queue Priority will release the item with the highest priority. In this case, any item waiting to be re-machined will be released first. 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 Priority.



Priority model - top priority given to items waiting to be re-machined

Holding

The Holding block accumulates items until a specified number has been reached, at which point the individual items are released. This is useful if you have a machine that has a capacity to process multiple items at once, but cannot accept new items once processing has begun. In cases such as this, it is common to gather items waiting to be processed until you have an amount equal to the machine’s capacity, then release the items into the machine for processing.

For example, consider an oven that can bake 20 silicon wafers at once. In an effort to minimize possible contamination, the oven cannot be opened once the baking process has begun. You would use a Holding block to accumulate a total of 20 silicon wafers. The wafers could then be released into the oven which is modeled using an Activity Multiple block (from the Activities submenu of the Discrete Event library) with its capacity set to 20.



Manufacturing Chapter 7: Routing

As you build models, you will frequently encounter situations where you want to manipulate items coming from several sources or send items out on various paths. Depending on your purpose, there are several methods for accomplishing this. Remember, items can only be in one place at a time.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Activity, Service	Routing	Discrete Event
Catch	Routing	Discrete Event
Combine	Routing	Discrete Event
Combine(5)	Routing	Manufacturing
Decision	Decision	Generic
Get Attribute	Attributes	Discrete Event
Input Random Number	Inputs/Outputs	Generic
Machine	Activities	Manufacturing
Max & Min	Math	Generic
Prioritizer	Routing	Discrete Event
Select DE Input	Routing	Discrete Event
Select DE Input (5)	Routing	Manufacturing
Select DE Output	Routing	Discrete Event
Select DE Output (5)	Routing	Manufacturing
Set Attribute	Attributes	Discrete Event
Throw	Routing	Discrete Event
Unbatch	Batching	Discrete Event

The models discussed in this chapter can be found in the “Examples \ Manufacturing \ Routing” folder.

Items from several sources

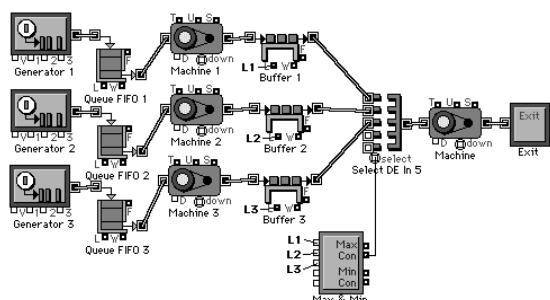
Depending on your modeling needs, you may want to merge different streams of items into one stream of individual items or join separate items into a single item.

Overview of merging, joining, and selecting multiple input streams

- To *merge* items from several sources into one stream, where each item remains separate and retains its unique identity, use the Combine, Combine (5), or Throw and Catch blocks. You then typically direct the single stream into one or more queues. For example, you can use this to represent traffic merging into one lane or people accessing one hallway from several offices. A Combine block is used to merge items requiring more processing in “Cumulative processing time: time sharing” on page M131 and to reroute preempted items in “Preemption” on page M141. “Using the Throw and Catch blocks” on page M109 illustrates using a Catch block to merge multiple streams of parts into one stream.
- To *join* items from various sources and process them as one unit, use the batch blocks, as described in “Manufacturing Chapter 9: Batching and Unbatching” on page M151. 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 Combine blocks, which only merge items so that each item remains separate and is separately processed.
- To *select* one item for processing from several sources based on some decision, use the Select DE Input block or the Select DE Input (5) block. The decision can be a logical decision (choose every other item to route to the top assembly 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 blocks and are modified by blocks connected to their “select” input connectors. Using the select input blocks is shown below in “Balancing multiple input lines”.

Balancing multiple input lines

To even out the queue lengths of multiple input lines, use a Select DE input (5) block controlled by a Max & Min block which checks the length of each queue. For example, assume you have three loading docks that fill up as trucks unload, and you want to take items from the dock that is most full first. The model, with machine blocks representing the process of unloading the trucks, is:



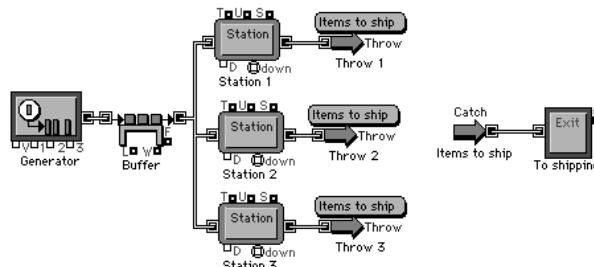
Input line balancing: choosing the longest queue

On the Max & Min block, the top *Con* output connector tells which of the inputs has the largest value, indicating the longest queue. This tells the Select DE Input (5) block which buffer to retrieve the next item from. In the dialog of the Select Output (5) block, be sure to choose that the value at the “select” connector chooses an output and that the top output is selected by a value of 1.

Using the Throw and Catch blocks

The previous example discussed routing items using connections to blocks which are nearby and at the same level of hierarchy. Sometimes, especially in large models, this is not convenient. The Throw and Catch 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 the Throw and Catch blocks are used as an adjunct to routing, not a replacement for the methods described previously.

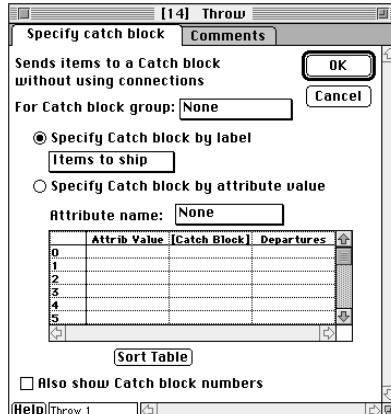
Throw and Catch blocks pass items without connections and can even be used deep within nested hierarchical blocks to send items to other hierarchical blocks. For that reason, they are often more convenient to use than the Combine block. In the following example, three Throw blocks route items to a Catch block which sends the items “to shipping”:



Model using Throw and Catch blocks to route items

Within the dialog of the Throw block, you have the option of routing all items to the Catch block specified in the popup menu (as shown below) or routing items to different Catch blocks depen-

dent on the value of a specified attribute (see “Attribute routing using the Throw and Catch blocks” on page M117).



Throw block dialog showing Catch block selected (Macintosh)

The popup menu in the Throw block’s dialog displays the labels of the different Catch blocks, allowing you to select the appropriate block. Therefore, as you add Catch blocks to your model, you must enter a label in the field at the lower left hand corner of the Catch block’s dialog. Only Catch blocks with labels will appear in the Throw block’s popup menus.

If you are working with a large number of Catch blocks, you may want to organize them into groups. To do this, select or create a group name using the “Catch group” popup menu in the Catch block:

Catch group: **group1**

Catch group popup menu in Catch block

The “For Catch block group” popup menu in the Throw block can be used to select a Catch block group. Once selected, only the labels of the Catch blocks in the selected group will appear in the “Specify Catch block by label” popup menu.

Items going to several paths

In many cases, you will need to route items from one stream to one of many different streams or separate an item and route its components individually.

Overview of parallel processing, unbatching, and selecting multiple output streams

- Taking a stream of items and routing them to a group of activities or operations is called *parallel processing*. In parallel processing, each item is handed off to one of several activities, such as a Machine 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.

- For situations where one item is 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 M156.
- The Select Output blocks are 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 you can direct patients requiring injections to the “shot room”. The blocks that take an input item and route it to one output path out of many are the Select DE Output block, the Prioritizer block, the Throw block and the Select DE Output (5) block. The use of these blocks is described in “Successive ordering” on page M113, “Explicit ordering” on page M114, “Routing decisions based on attributes” on page M115, and “Simple routing” below.

Note The Select DE blocks expect integer values for comparison and will round non-integer values. For example, the value 0.003 would be rounded down to a zero. If you use non-integer values for an attribute value that is also used as the processing time, you would need to convert the attribute values to integers before they go to the Select DE Output block. This is discussed in “Cumulative processing time: time sharing” on page M131.

Simple routing

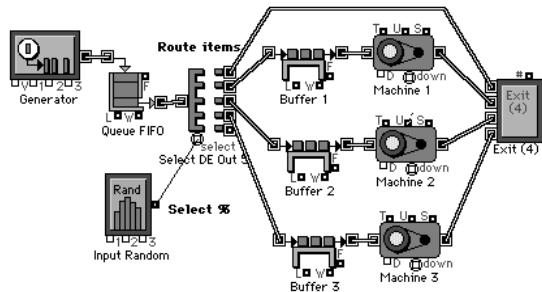
The simplest routing example is where a percentage or specified number of items are routed to one section of the model, while the rest are routed to another. An example is a formal processional line, where the first person in line goes to the left row of seats and the next person goes to the right.

The Select DE Output blocks are most appropriate for routing a number of items onto one path or another. You choose which output connector an input item should be routed to based on logic in the dialog or the use of the “select” connector. The dialog has options for toggling or for changing the outputs after a given number of items have passed.

For instance, you can specify that one out of every 4 items goes out the bottom output of the Select DE Output block. Or you can toggle the outputs from top to bottom so that items are handed off in successive order; the example “Successive ordering” on page M113 uses this method to choose parallel processors.

M112 | Manufacturing Chapter 7: Routing
Items going to several paths

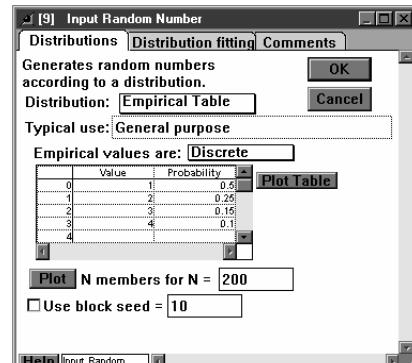
The “select” connector allows more flexibility than the dialog settings. For example, you can connect an Input Random Number block to the “select” connector and choose the Empirical distribution to specify a percentage of items flowing from each output:



Simple routing based on the select connector value

MFG

Indicate in the dialog of the Select DE Output (5) block that the top output is chosen by a “select” value of 1. By default, the outputs below the top output are selected in order by adding one for each connector from the top. In this example, the top connector would be selected by a value of 1, the second by a value of 2, the third by a value of 3, and so on. The Empirical distribution table in the Input Random Number block is set as follows:



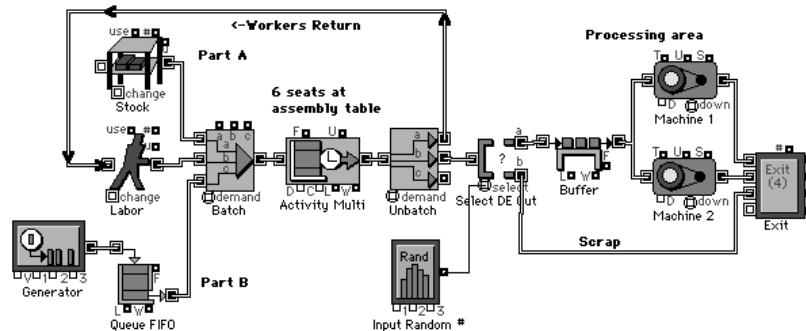
Setting the Empirical distribution (Windows)

These settings cause the select connector to output 50% of the items at the top output, 25% at the second output, and so forth.

Notice that there are buffers in front of the machines. If you omitted the buffers, there would be a possibility that the items would be blocked. For example, if the top machine were due to receive an item, but was still processing, the other machines would have to wait for items until the top machine finished processing and pulled in its item.

Scrap generation

An important aspect of manufacturing systems is the generation of *scrap*. Many manufacturing processes create an expected but irregular quantity of waste or bad items. This can be accomplished in Extend by randomly routing some items out of the normal processing stream. This extension to the model described in “Unbatching” on page M156 has a Select DE Output block that determines whether the items coming from the Unbatch block should continue to the rest of the line or be discarded. The Input Random Number block uses the Empirical Distribution to specify that one out of every ten items becomes scrap. When you connect to the “select” connector on the Select DE Output block, approximately one out of ten items will go to the bottom connector, and thus exit the model as scrap.

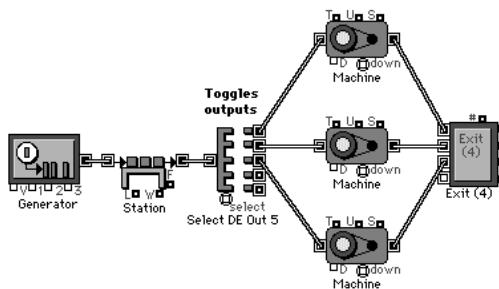


Scrap generation added to model

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 section.

Successive ordering

To hand items to operations in successive order regardless of whether another operation is free, use the Select DE Output block or Select DE Output (5) block to toggle from the top to the bottom of the output connectors. An example of selecting three successive machines is:



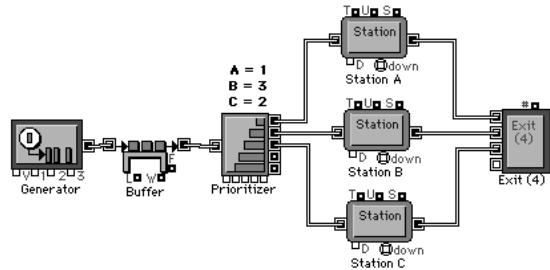
Parallel processing in successive order (with possible blocking)

Since the “select” connector is not used, by default the Select DE Output (5) block will toggle its outputs from top to bottom (choose outputs in successive order starting from the top output). In this situation, even if the third machine is the first one ready to accept an item, it will only get an item after the first and second machines have each pulled in an item. Also note that an item is only pulled from the Buffer block when a machine has finished processing, potentially causing blocking in the system. The example “Balancing multiple output lines” on page M119 shows a solution for this.

Explicit ordering

If you have 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).

The Prioritizer block allows you to specify the priority of each output. Note that this is different from assigning a priority to an item, since the Prioritizer essentially prioritizes the path, not the item. For example, assume that you want Stations A and C to get most of the items for processing, and Station B to only get items if both A and C are busy. The model is:



Parallel processing in explicit order

In the dialog of the Prioritizer, assign the highest priorities (which are the lowest numbers) to the top and bottom outputs, and the next lowest priority to the middle output:

Enter priority for output:	
a =	1
b =	3
c =	2
d =	
e =	
Top	
.	
.	
.	
Bottom	

Prioritizer block dialog showing output priorities

When you do this, Station A has first priority on items. If Station A is busy, Station C will get the item. Only if Station A and C are busy will Station B get an item.

You can assign the same priority to multiple outputs. If you do this, however, the highest priority will go to the topmost output that 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. This is rarely what you want, since the outputs would not always have an equal chance of being chosen.

A better method for assigning equal priority to two or more outputs is to connect an Input Random Number block to the value inputs of the Prioritizer block. Set the distribution of the Input Random Number block to be an Integer, Uniform distribution with a minimum of 1 and a maximum of 10. With this method, each output would then have an equal chance of being chosen.

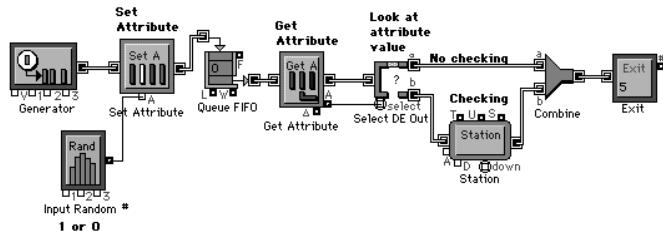
Routing decisions based on attributes

You may want to route items based on some characteristic of the item, such as its size, quality, age, or requirements. To do this, assign an attribute to the item and read that attribute value to route the item.

MFG

Attribute routing using the Select DE Output block

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 Input Random Number block, as shown below:



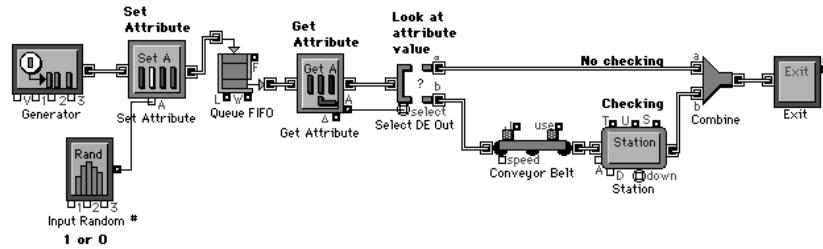
Routing based on attributes

The Empirical distribution in the Input Random Number block specifies that 25% of the items do not require checking (0 for attribute value) and 75% do (1 for attribute value). The “A” output from a Get Attribute block is connected to the “select” connector of a Select DE Output block. The Select DE Output block 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 Combine 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

M116 | Manufacturing Chapter 7: Routing
Items going to several paths

if checking is needed. All those steps would be between the Select DE Output block and the Combine block:



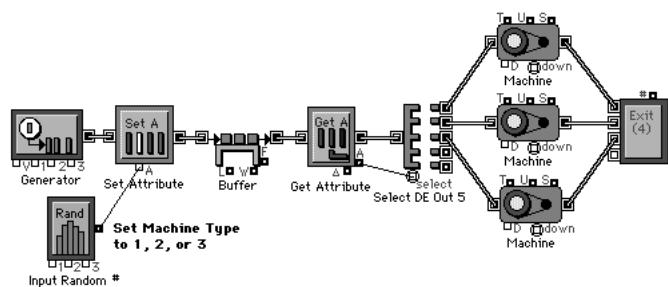
Transportation only if checking

MFG

The example “Machines that can only process certain types of items” on page M122 is another instance of using attributes to route items.

Note The default behavior of the Select DE Output block is to determine the path the next item will take before it has exited the preceding residence block (see the main Extend manual for a discussion of residence and passing blocks). For this decision to be made, the item’s attribute value must be set before it has reached the residence block. Since the Set Attribute and Get Attribute blocks are both passing blocks, the purpose for the Queue FIFO block in the above model is to provide a residence block for the items after the attribute value has been set. Alternatively, you can make the Select DE Output block accept the item before determining its path by selecting “Item enters Select block before decision is made” in the dialog of the Select DE Output block.

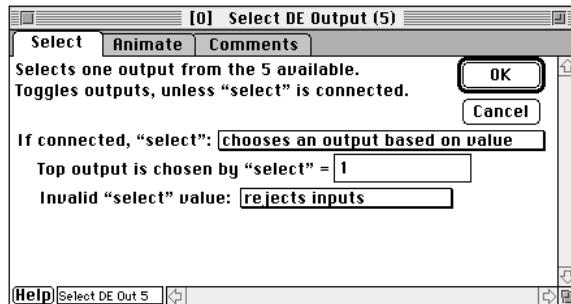
The same technique used in the example above can be applied to choosing a machine based on an attribute or priority value at the “select” connector. For instance, if you have an attribute that gives the number of the machine that an item is to be processed by, you use a Get Attribute block to extract that information and pass it to the “select” connector, as shown below:



Parallel processes selected by attribute value

In the model, the Set Attribute block assigns an attribute value of 1, 2, or 3 to the items passing through, corresponding to the number of the machine that will process that item. The dialog of

the Select Output (5) block indicates that the value at the “select” connector chooses an output and that the top output is selected by a value of 1. In this instance, a value of 2 at “select” will cause the item to be processed by the middle machine, machine 2.



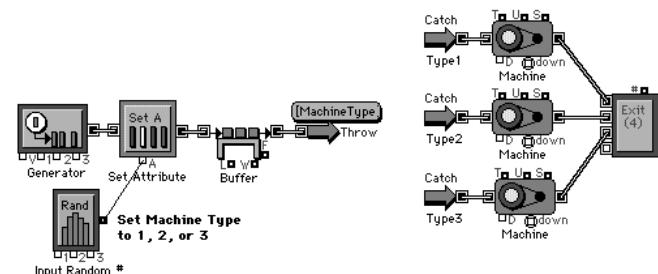
Choosing an output based on the select connector value (Macintosh)

MFG

If you have more than five machines, you can place two Select DE Output (5) blocks in parallel and assign them to different ranges of attribute values as shown in “Extended routing” on page M120.

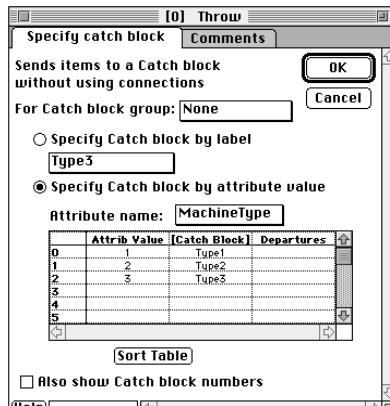
Attribute routing using the Throw and Catch blocks

As described in “Using the Throw and Catch blocks” on page M109, the Throw block can be used to route items to a specific Catch block. The Throw block can also be used to route items to different Catch blocks depending on the value of an attribute. The previous example, built using Throw and Catch blocks rather than the Select DE Output (5) block, is shown below:



Parallel processes selected by attribute value using the Throw and Catch blocks

In this example, the Throw block reads the attribute “MachineType” and routes the items to the appropriate Catch block according to the table in the Throw block’s dialog, shown below:



Dialog of Throw block (Macintosh)

To associate an attribute value with a specific Catch block, type the value into the “Attribute Value” column and select the appropriate Catch block using the popup menu in the “[Catch Block]” column.

Conditional routing

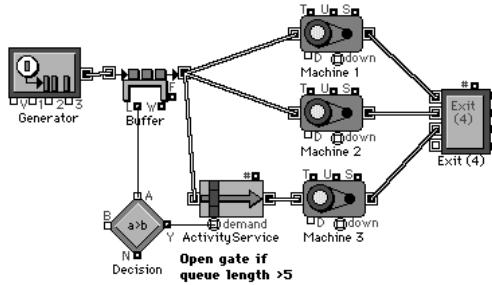
Sometimes you will want to route items based on the current conditions of the model. For example, you may want to monitor queue lengths to determine whether or not a machine will be brought on-line or to balance the use of parallel lines.

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 M135) or based on some other factor such as the backlog of work. 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.

Extend can simulate this easily using the Decision block and the Activity Service block. The “L” output of the buffer 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. An example of this is:



Routing based on model conditions

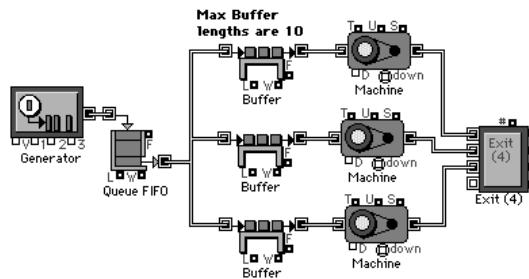
The dialog of the Decision block specifies that the “Y” connector outputs a true value (1) when the value at the “A” input is greater than 5. This activates the “demand” connector so that the Activity Service block lets items through to the third machine (until then, it will not accept items). When the Buffer holds 5 or fewer items, the Activity Service block 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 shut down operation until processing resumes.

When you bring a system on line, it may cycle on and off too frequently. See “Item preemption and process interruption” on page M141 for some methods for avoiding this.

Balancing multiple output lines

Operations are often preceded by buffers before each operation, such as a staging area for each machine (as compared to the single staging area for all machines as in the previous example.) This would look like:

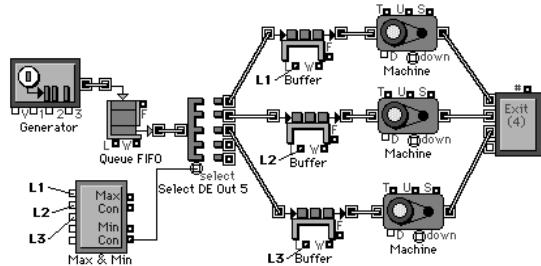


Parallel processing with buffering

M120 | Manufacturing Chapter 7: Routing
Items going to several paths

In this case, the first buffer connected receives all the items until that buffer reaches its maximum, then the next buffer starts to fill (unless the first machine kept up with the flow of items, in which case the next buffer will never receive any items). This is rarely what you want.

To even out the use of the machines, you can use the “Successive ordering” method shown earlier, keeping buffers before each machine. However, if the machines work at different speeds, this will cause the buffer of the slowest machine to fill more rapidly than the other buffers. A better method would be to check the length of the queue in each buffer and give the next item to the queue that is shortest. The Max & Min block is excellent for this:



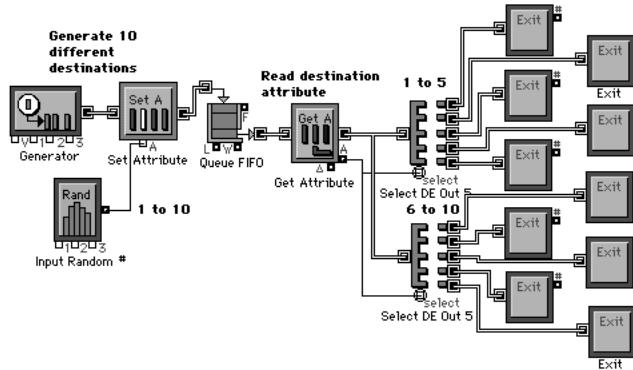
Output Line balancing - choosing the shortest queue

On the Max & Min block, the bottom *Con* output connector tells which of its inputs has the lowest value, indicating the shortest queue. This tells the Select DE Output (5) block which buffer to hand the next item to. In the dialog of the Select Output (5) block, be sure to choose that the value at the “select” connector chooses an output and that the top output is selected by a value of 1.

Extended routing

In some instances, you may want to route items to more than five destinations. For example, patients in an emergency room may require one of several different treatments: laboratory work, x-ray, cast room, burn treatment, poison center, operating room, and so on. To do this, you use a

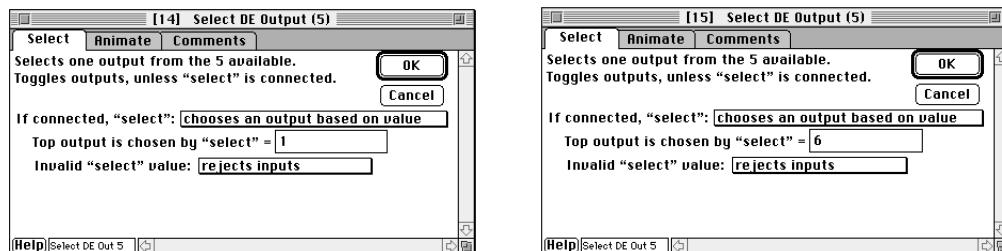
Select DE Output block to route to two or more Select DE Output (5) blocks in two or more stages.



Routing to 10 destinations

MFG

The model uses item attributes to specify a destination for routing, where each item gets an attribute value from 1 to 10. Both Select DE Output (5) blocks get the attribute value from the Get Attribute block. The top Select DE Output (5) block routes items with attribute values from 1 to 5, and the bottom Select DE Out (5) block routes items with attribute values from 6 to 10:



Top Select DE Output (5) block processing destinations 1-5 (left) and Bottom Select DE Output (5) block processing destinations 6-10 (right)

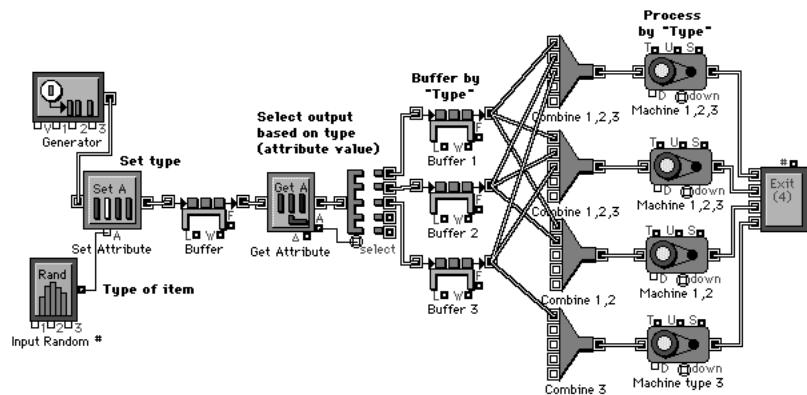
Note Be sure that “Invalid ‘select’ value rejects inputs” is selected in the dialog of both blocks.

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 models of stereos 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 DE Output (5) block and the Combine block are very handy in such situations.

Assume that you have such an assembly line. 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, you have 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. You would use the following blocks in your model:

MFG



Processing by type

The Get Attribute block looks at the “Type” attribute and outputs its values at its “A” output connector. The Select DE Output (5) block chooses an output based on the value at its “select” connector; the top output is selected by a value of 1 (for type 1), and so forth.

Notice the use of the Buffer blocks in the above model. The buffers are used to queue the items by type, with the top buffer for type 1, etc. Without the buffers, 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 DE Output 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.

Note As an alternative to using the multiple (and messy) connection lines between the buffers and the machines, you could use named connections, which are discussed in the main Extend

manual, or the Throw and Catch blocks, as you saw in “Using the Throw and Catch blocks” on page M109.

MFG

M124 | Manufacturing Chapter 7: Routing
Items going to several paths



Manufacturing Chapter 8: Processing

This chapter will discuss different ways to connect multiple activity blocks and to control processing time and availability of items and resources.

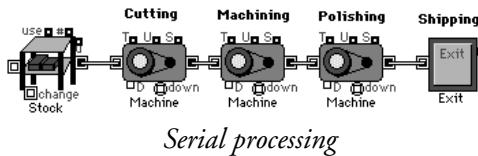
The following blocks will be the main focus of this chapter:

Block	Type	Library
Activity Delay (Attributes)	Activities	Discrete Event
Activity, Multiple	Activities	Discrete Event
Activity Service	Routing	Discrete Event
AGV	Resources	Manufacturing
ASR	Resources	Manufacturing
Conveyor Belt	Activities	Manufacturing
Conveyor, Carousel	Activities	Manufacturing
Crane	Activities	Manufacturing
Decision	Decision	Generic
Downtime (Unscheduled)	Generators	Manufacturing
Get Attribute	Attributes	Discrete Event
Input Data	Inputs/Outputs	Generic
Input Random Number	Inputs/Outputs	Generic
Machine	Activities	Manufacturing
Machine (Attributes)	Activities	Manufacturing
Process, Preemptive	Activities	Manufacturing
Program	Generators	Discrete Event
Queue, Priority	Queues	Discrete Event
Route	Activities	Manufacturing
Route (Delay)	Activities	Manufacturing
Schedule	Generators	Manufacturing
Select DE Output	Routing	Discrete Event
Station	Activities	Manufacturing
Station (Attributes)	Activities	Manufacturing
Transporter	Activities	Manufacturing

The models (and any sub-folders) discussed in the chapter can be found in the “Examples \\ Manufacturing \\ Processing” folder.

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 such as governmental processes. A simple example of serial processing is an assembly line, where several processes are performed on one part prior to shipment:



MFG

Since you have many machines in series without buffers 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 M97). Serial processes can cause the entire activity to be slowed down 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, you can put a queue or buffer in front of each machine to represent a holding area, as shown in “Simple routing” on page M111.

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 5 machines that can each process parts arriving from the stockroom. With the Manufacturing 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 a block

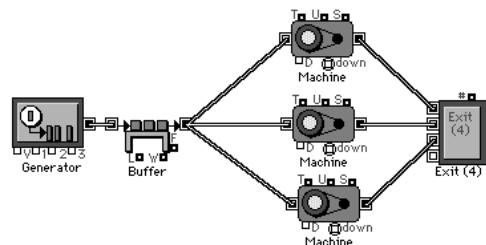
You can use the Activity, Multiple block or the Process, Preemptive block to represent several activities that occur in parallel when you do not need to show each activity as a separate block. The Process, Preemptive block has the additional feature of being able to cause an item to be preempted by another, higher priority, item.

These blocks take in items (up to a specified maximum) and process them for a specified time starting from when they arrive. The item with the smallest processing time and earliest arrival time is passed out first. For example, you would use these blocks 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

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 Extend to pass items to the first available activity. However, if more than one activity is free when an item is ready, it is unpredictable which block will get the item. For instance, a buffer that holds items for three Machine blocks would look like this:



Simple parallel processing

MFG

The first free machine will get the item. If two machines are free when an item comes out of the buffer, it is not clear which block will receive it.

Note Unless it is completely unimportant in your model, you should always explicitly state the ordering for parallel activities. See Manufacturing Chapter 7: Routing 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. You model this delay explicitly using activity blocks such as Machine or Station blocks or implicitly by specifying the length and speed in the material handling blocks (such as in the Crane). (The models discussed in this section can be found in the “Examples \ Manufacturing \ Time” folder.)

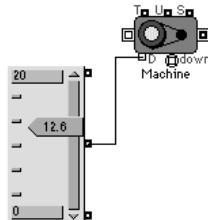
Processing time can be static or can vary dynamically depending on model conditions. It can be random, scheduled based on the time of day, customized depending on the item that is being processed, or any combination of these. You can set processing time through a block’s dialog, by using the “D” input connectors, by reading the delay time from an item’s attribute, or by specifying a length and speed for the process.

Fixed processing time

You can set the delay in the dialog of the activity-type block 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, you enter

the value 5 as the processing time in the dialog. This is most common when you are building the early stages of a model and are using constant parameters so you will get repeatable results.

Another method for having a fixed processing time is to connect a Slider control to the “D” input of an activity block, such as for the Machine block below:



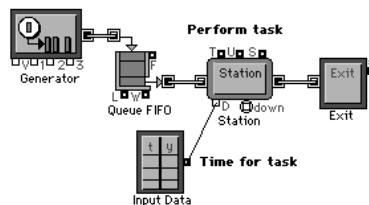
Slider control used to set machine processing time

MFG

You can manipulate the Slider control with each simulation run, or within a simulation run, to see the effect of various processing times.

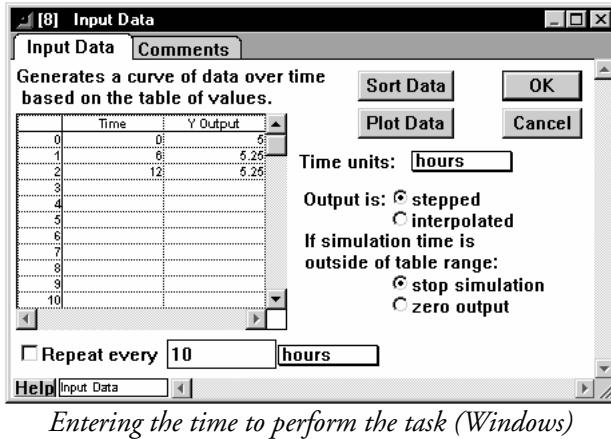
Scheduled processing time

If you know that an activity 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. 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, you can connect an Input Data block to the “D” connector of an Station block, as shown:



Scheduled processing time

The time to perform the task is entered in the “Y Output” column of the Input Data block’s dialog, shown below:



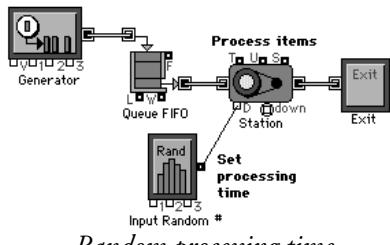
Entering the time to perform the task (Windows)

MFG

Notice that the time units for the Time column are in hours and that the time the worker takes to perform the task changes at 6 hours. Also notice that since the output of the Input Data block is connected to the “D” connector of the Station block, the Y Output column should be defined in the same time units that are set for the delay parameter in the Station block (minutes). See “Setting time-based parameters using connectors” on page M74 for a discussion on time unit consistency.

Random processing time

A common requirement for activities is to set a random processing or delay time. This is easily accomplished by connecting an Input Random Number block to the “D” connector of an activity-type block. For example, you can connect an Input Random Number block to a Machine block:



Random processing time

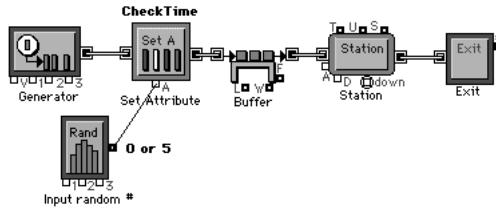
In the dialog of the Input Random Number block, select the distribution required, for example Normal, and specify the value of the parameters, such as a mean of 2 and a standard deviation of 0.2. The processing time will then be normally distributed and the Machine block will process each item for approximately 2 time units.

For more information about random numbers, see “Constant values versus random variables” and “Random numbers and probability distributions” on page M194.

Custom processing time

As you saw in “Using attributes” on page M91, attributes can be used to specify how long a specific item will be processed by a particular activity. The Activity Delay (Attribute) block, Machine (Attribute), and Station (Attribute) blocks can use attribute values to specify a processing time.

In the simplest case, you set an item’s attribute value to the desired amount of processing time, then use an attribute manipulating block, such as a Station (Attribute), to read the attribute value and process the item for that period of time. The model is:



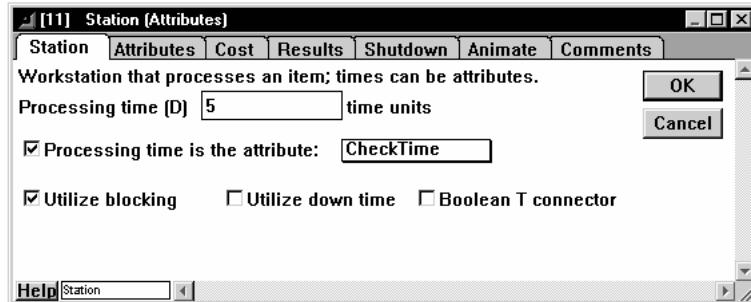
Custom processing time

Use the Set Attribute 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 an attribute value of 5, for instance, and items that ship unchecked have an attribute value of 0. In the dialog of the Set Attribute, choose that an input at the “A” connector modifies the attribute value of the “Checktime” attribute. That value (0 or 5) is provided by the Input Random Number block using an Empirical distribution where 25% of the items have a value of 0 and 75% have a value of 5. The Empirical Table from the Input Random Number block dialog looks like:

	Value	Probability
0	0	0.25
1	5	0.75
2		
3		
4		

Setting a custom processing time (Windows)

All items then go through the checking step, easily represented by a Station (Attribute) block. In its dialog, this block indicates that the “Processing time is the attribute ‘Checktime’”:



Processing time based on attribute value (Windows)

MFG

Implied processing time

Some Extend blocks allow you to specify length, speed, or other factors which indirectly result in a processing time. For example, the Conveyor Belt, Crane, Route, and Transporter blocks allow you to specify a length (in feet) and a speed (in feet per time unit). In addition, you can set the number of slots in the Conveyor Belt and the circulation time, and the number of slots before the output in the Conveyor, Carousel block.

The settings in these material handling blocks result in delay times for items. For example, if you set the Crane to be 10 feet long with 0 feet for the return and a speed of 1 foot per time unit, it will take 10 time units for an item to travel by crane. However, if you set the Conveyor Belt block to be 10 feet long with 10 slots and a speed of 1 foot per time unit, it will take 9 time units for an item to travel from start to finish. This is because when the item reaches the final conveyor position, slot 10, it has completed its journey. To determine the actual delay for items, see the Help for these blocks.

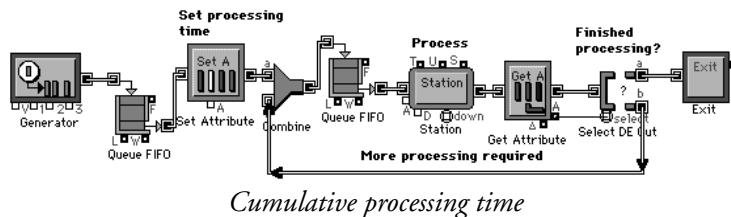
Cumulative processing time: time sharing

You can also set an item’s 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.

To do this, use an attribute-manipulating activity block such as a Station (Attributes) block and specify that the attribute value be decreased by the amount of processing time. When you do this, each activity subtracts its processing time from the attribute value, so that the value remaining represents the processing time left. You can use a Get Attribute block after each station to determine if the item was fully processed or not, and therefore whether it should proceed to the next station 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, with periods in between where nothing is happening. Since the time units are so small, the periods when there is no processing occurring are typically not noticed, and each job appears to be processed continuously.

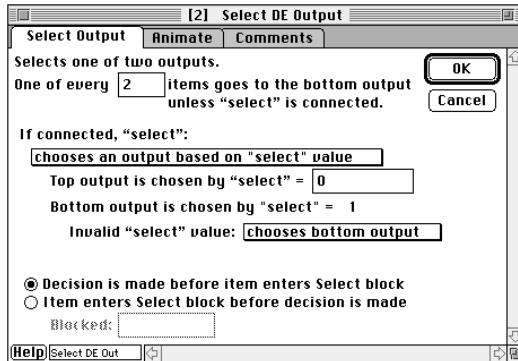
A simple time sharing model has one activity that processes each job for a very short period of time, then sends the job back to the queue so it can be processed again, until the total required processing time for that job has elapsed:



The attribute value specified in the dialog of the Set Attribute block determines the total processing time required for each job, in this case 2 milliseconds. The Station (Attribute) block processes the job for a fixed time (1 millisecond) and subtracts that time from the attribute value. The Get Attribute block looks at the attribute and outputs its value at the “A” connector.

Notice that the “A” output is connected to the “select” input of the Select DE Output block. As each job enters the Select DE Output, its attribute value is read by the “select” connector. In the dialog you choose that the output connector is selected based on the value at “select”, that the top

output is chosen by a select value of “0”, and that invalid values cause the job to go to the bottom output. This causes values other than 0 to go to the bottom connector. The dialog is:



Select DE Output dialog (Macintosh)

MFG

When the value at “select” is greater than zero, indicating that there is processing time left, the job is sent out the bottom connector and back to the queue to wait for more processing. When there is no more processing time left, the job is completed and exits the simulation through the top output.

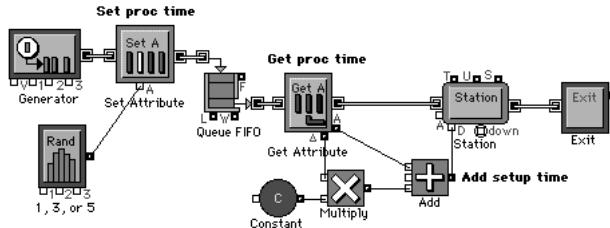
In the time-sharing model, the time units are integers representing milliseconds. This is because the Select DE blocks expect integer values for comparison and will round non-integer values. For example, the value 0.003 would be rounded down to a zero. If you use non integer values for the processing time, you need to convert the attribute values to integers before they go to the Select DE Output block. You can do this with a Conversion Table block from the Math submenu of the Generic library.

Adding setup time

A common problem in manufacturing is that often a machine must 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 you saw in the example “Custom processing time” on page M130. However, you need to add an additional amount of time for processing whenever the type of item changes.

The model uses the Δ connector of a Get Attribute block to determine when the attribute value changes, indicating a different type of item passing through. You use this indication to add a setup

time to the attribute value, resulting in a longer processing time for the first item each time the type changes. The model is:



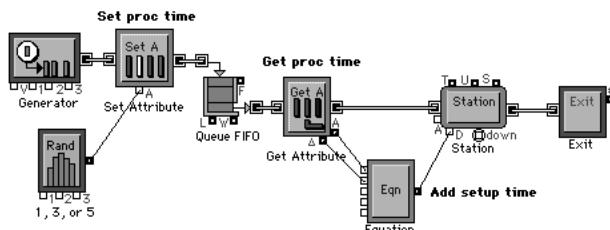
Setup Time 1: Adding a setup time

MFG

The “ Δ ” connector outputs 0 as long as the value of the specified attribute stays the same, and outputs 1 (for True) when the attribute value changes. A change indicates the arrival of a new type of item. As long as the attribute value does not change, the Constant block (from the Inputs/Outputs submenu of the Generic library, specifying a setup time of 2 time units) 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 setup time is added to the attribute value.

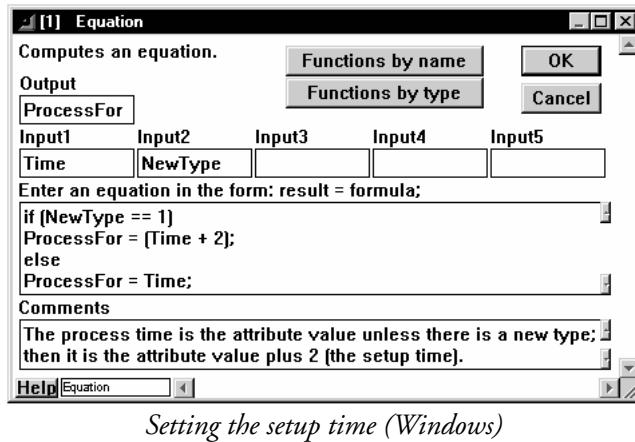
Note that each item still has its original attribute value. You do not change the attribute value in this model, you only use it to determine a processing time. The processing time (whether equal to the attribute value or equal to the attribute value plus the setup time) is input at the “D” connector. The Station (Attribute) 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, you can also use the Equation block from the Math submenu of the Generic library as follows:



Setup Time 2: Setup time from Equation block

The dialog of the Equation block is:



Setting the setup time (Windows)

MFG

Bringing an activity on-line

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 \ Manufacturing \ Bringing On-line” folder.

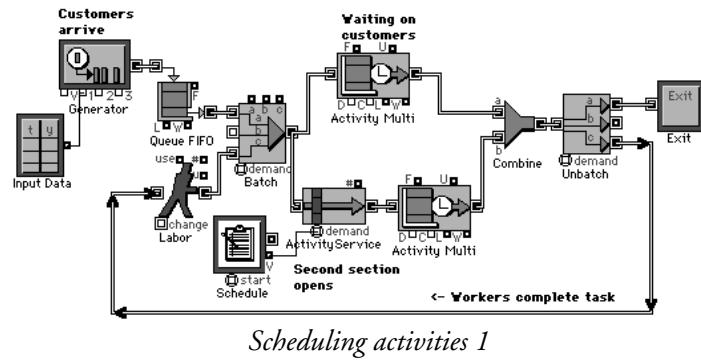
Please also see “Using the Shift block” on page M169 for examples of activity and resource allocation that are tied together and scheduled as “Shifts.”

Scheduling activities

Activities can be scheduled. This is most common when you bring an activity on-line based on the time of day. The following example shows you how to schedule the availability of a portion of an operation.

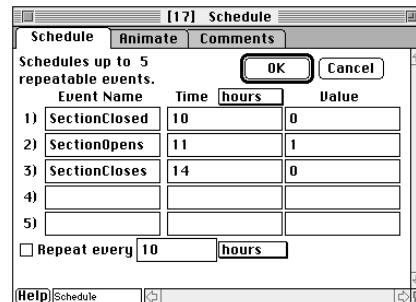
For this example, assume that you are modeling a coffee shop that opens at 10 AM and closes at 6 PM. Customers arrive exponentially throughout the day, with most customers arriving during the lunch period, that is from 11 until 2. You have 8 service people and 2 dining sections available. The simulation time is in hours, and the simulation is run for 8 hours, from time 10 (10 AM) until time 18 (6 PM).

Assume that you have two service counters, but only open the second counter from 11 until 2 PM. You connect a Schedule or Input Data block to an Activity Service block, allowing customers access to the second counter only during certain hours. The model is:



MFG

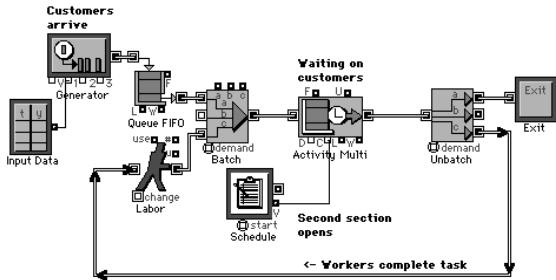
The Activity Service block only allows items through when its “demand” connector is activated. You usually do this by connecting a value connector, such as the “V” connector on the Schedule block, to “demand”. As long as the value connector is true (outputs 1), the Activity Service stays open; when the value is 0, for false, it closes. The dialog of the Schedule block is:



Opening and closing the counter (Macintosh)

Notice that in the above example the connection to “demand” is from the value connector, not from the item connector. As explained in the Extend manual, universal connectors such as “demand” treat item and value connectors differently. Value inputs cause “demand” to be activated as long as the value is true (1) and close when the value is false (0). Item inputs cause “demand” to accumulate a demand for a number of items. If you had connected from the item output on the Schedule block to “demand”, only one customer would have gone to the second counter (at time 11). (The item output could be used to bring a system on-line for a fixed number of items.)

In the above example, you could have also used the “C” connector on the Activity Multiple block to control the capacity of the block to simulate the opening of the second counter. The model looks like:



Scheduling activities 2: Modeling both counters using Activity Multiple

MFG

Each counter has the capacity to serve 5 customers at once. By doubling the capacity of one Activity Multiple block during the period between 11 and 2 PM you can model both counters being open. The portion of the Schedule block that controls the capacity of the Activity Multiple is:

Event Name	Time	hours	Value
1) SectionClosed	10		5
2) SectionOpens	11		10
3) SectionCloses	14		5
4)			
5)			

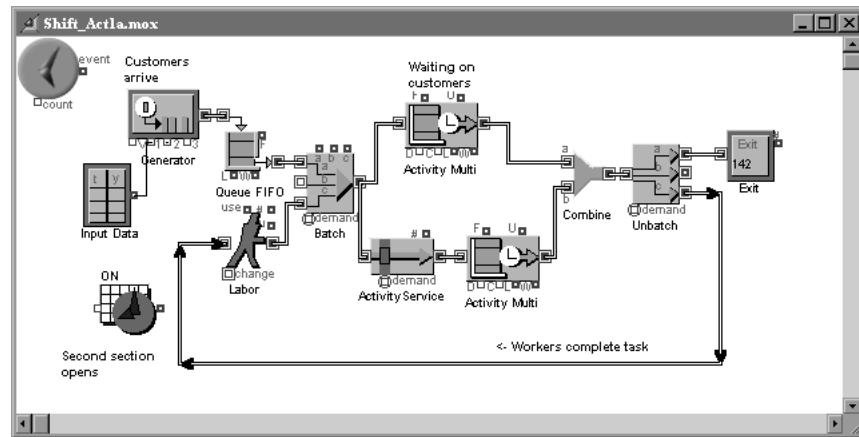
Scheduling Activity Multiple capacity

Shift block used to schedule

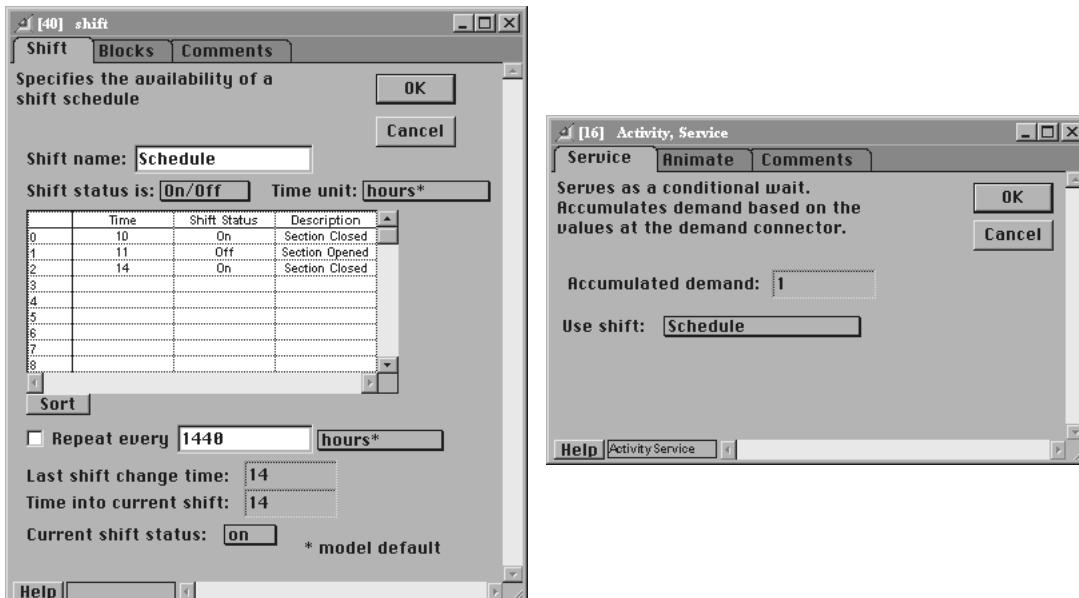
Yet another approach to this would be to use a Shift block to control the Activity Service. The Shift block would contain the same information as the schedule block.

M138 | Manufacturing Chapter 8: Processing
Bringing an activity on-line

Examples of use of the Shift block can be found in the “Examples \\ Manufacturing or BPR \\ Shift” folder.



Scheduling using the Shift block



Dialogs of the Shift block and Activity Service block showing shift use

The “schedule” shift is selected in the Activity Service block. As long as the shift is on, the Activity Service block will allow items through, when the shift closes down, items will no longer be permitted to pass through the Activity Service block.

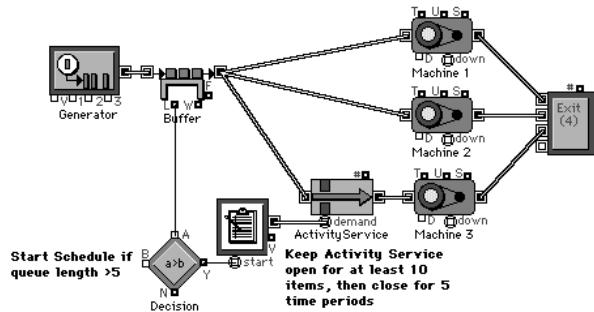
Controlling the flow of items to an activity

As discussed in “Bringing a system on-line” on page M118, you can use logic constructs in Extend to route items to a portion of the model that will then start operating, and you can bring a system on-line based on the time of day. However, the method shown in those examples may cause the activity being brought on-line to cycle on and off frequently. You probably do not want this to occur, as it results 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, you there are two main ways to control the flow of items to the activity: by specifying the number of items which will be processed or by specifying the amount of time the activity will be on-line.

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 which originally activated it. This reduces the number of times the machine turns on and off. To do this, add a Schedule block before the Activity Service block in the model “Conditional routing” on page M118 (in the “Examples \\ Manufacturing \\ Routing” folder), such as:



Bringing a system on-line for a number of items

When activated, the Schedule block puts out a single item with a value of 10. This causes the Activity Service block to stay open until 10 items pass through, before shutting off again. The table in the Schedule’s dialog looks like:

Event Name	Output Time	Value
1) OpenPath	0	10
2) DelayRestart	5	0

Activating the demand connector with an item

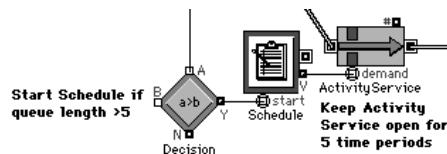
There are two items of special note in this model: how the “start” and “demand” connectors are activated.

- Since the *start* connector is connected, the Schedule block runs in relative simulation time (begins its schedule relative to when start is activated), as explained in the section “The “start” connector” on page M86. Once “start” is activated (gets an item or sees a value greater than 0.5) it causes the entire schedule to happen; subsequent activations are ignored until the schedule is complete. The second line of the schedule means that the Schedule block will ignore any start messages for 5 time units. This 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 Service block could be activated constantly.
- The *demand* connector is activated when it gets an item, causing the Activity Service to open and allow items through. The number of items allowed through before the Activity Service closes is determined by the value of the item at “demand”. For instance, each value of 10 creates a demand for 10 items before the gate is shut.

MFG

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. This is also easy. To do this, use the Schedule block to output values to the “demand” connector on the Activity Service block. Connect from the Schedule block’s value (*V*) connector instead of its item connector:



Bringing on-line for a fixed time

In the table in the Schedule’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:

Event Name	Output Time	Value
1) OpenPath	0	1
2) ClosePath	5	0

Activating the demand connector 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”, it will stay activated as long as the value it receives is greater than 0, which in this example is for 5 time units.

Note Please also see “Using the Shift block” on page M169 for examples of activity and resource allocation tied together and scheduled as “Shifts.”

Item preemption and process interruption

In discrete processes, it is common that there will be interruptions and changes in plans caused by the occurrence of a higher priority event. As a general rule, items in Extend are preempted and processes are interrupted:

- *Preemption* occurs when an item being processed is superseded by another item. An example of this is when one job is superseded by another, higher priority job. The preempted item (in this case, the current job) could be finished by another piece of equipment, finished later by the original piece of equipment, or never finished. The Process, Preemptive block allows you to specify which item will be preempted: the item that has been processed for the longest time, the item that has taken the least amount of time, or the item with the lowest priority. Items that are preempted can be routed back to the original Process, Preemptive block for later processing or routed elsewhere in the model.
- When an activity is *interrupted*, it temporarily stops processing. A process that is interrupted can resume when the interruption ends. The “down” input on the Machine and Station blocks is used to interrupt or shut down processes.

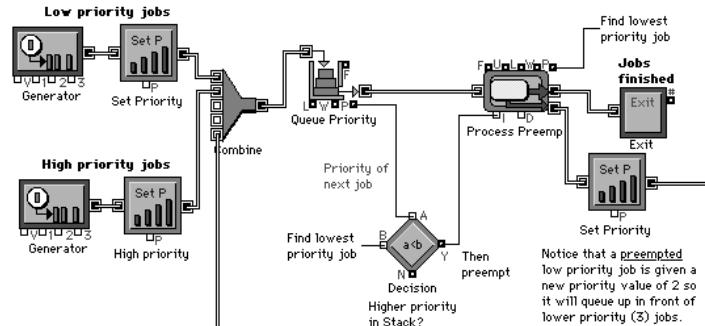
Preemption

The Process, Preemptive block allows you to model situations where one item preempts (prematurely supersedes) another item. You can preempt the item that has been in the block the longest time, the shortest time, or which has the lowest priority (the highest value is the lowest priority). You can specify that preemption occur only if the block is already processing the maximum number of items and you can store the preempted item's remaining processing time as an attribute for subsequent processing. Items in the Process, Preemptive block exit one at a time from either of the two item outputs: the top output under normal conditions; the bottom output if the item is preempted.

You use the Process, Preemptive block in conjunction with other blocks which give the priority of waiting items, compare that priority to the items being processed, and trigger the preemption. Items are only preempted if the “I” input of the Process, Preemptive block gets a True value (a value greater than 0.5).

For instance, to preempt based on priority there must be one block which reports the incoming item's priority value and another block which compares the priority of that item to items already

in the Process, Preemptive block. The following model causes an item to be preempted if the Process, Preemptive block is full and a higher priority item is next in line to come in:

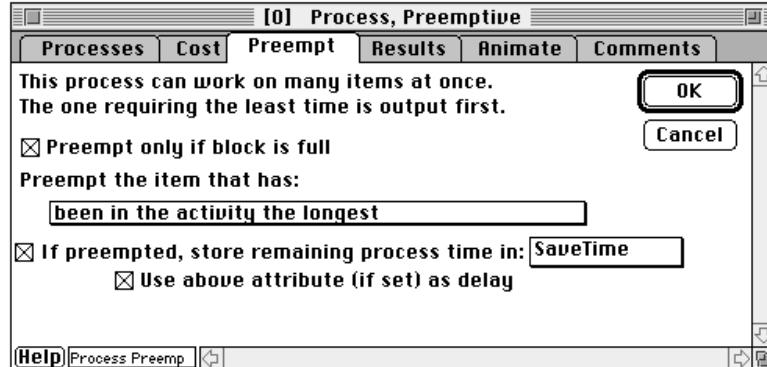


MFG

Preempting Processes model

The Decision block reads the priority of the first item waiting in the Queue, Priority block and compares it to the priority of the items in the Process, Preemptive block. If the priority of the waiting item is higher (has a lower value), the Decision block sends a value of “1” to the “I” input of the Process, Preemptive block. This causes the item with the lowest priority to be sent out the lower item output connector.

The settings in the Preempt tab of the Process, Preemptive block determine how the preemption will occur:



Preempt tab of Process, Preemptive dialog (Macintosh)

As the dialog indicates, if the item waiting in the Queue, Priority block has a higher priority (a lower priority value) than one of the items already being processed, *and* if the Process, Preemptive block is at full capacity, the item being processed will be preempted. In addition, the amount of processing time that remains is attached to the preempted item as an attribute named “SaveTime”. Since “Use above attribute (if set) as delay” is also checked, when the preempted item returns to the Process, Preemptive block it will process only for the time indicated by “SaveTime”.

Notice that the model also uses a Set Priority block (from the Attributes submenu of the Discrete Event library) to change the priority of the preempted item so that it will take precedence over lower priority items.

Interrupting or shutting down activities

Employee breaks, equipment maintenance, inventory-taking closings, and tool failures all involve breaks in activities for a period of time called *downtime*. Discrete event systems frequently consist of many activities that perform multiple tasks, and shutting down activities should be considered in the model to avoid being overly optimistic in predictions. The models discussed in this section can be found in the “Examples \\ Manufacturing \\ Shutting Down” folder.

You can shut down activities at a scheduled time, such as for coffee breaks or machine maintenance, or it can be a random occurrence, such as for equipment failures. You can also shut down activities based on some factor in the model, such as when the buffer after the machine is full. The duration of the downtime can either be a constant or a random value.

MFG

The *down* connector on the machine and station blocks is used to shut activities down for a period of time. The *S* output on those blocks outputs a 1 while the activity is shut down. In Extend, the *time between failures* (TBF) is the inter-arrival time of successive shutdowns and the *time to repair* (TTR) is the downtime duration.

When the “down” connector gets an item or sees a value greater than 0.5, it causes the block to be shut down. You can specify in the dialog that the value at “down” (either the value of an item input or the value from a value input) should be interpreted as the duration of the downtime, or that the activity is shut down until the connector value returns to 0. After a shutdown, the activity ignores all further shutdown messages until it is active again.

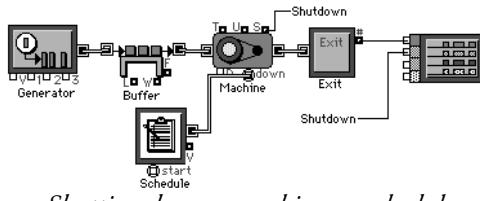
When a machine or station is shut down, you can handle the item which is currently being processed in one of three ways (as specified in the Shutdown tab of the block):

- The item can be discarded, such as when parts are spoiled by the machine going down.
- The machine 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.
- The machine can finish processing the item once it has been returned on-line.

Scheduled shutdown

The Program block, the Schedule block, and the Shift block are often used to schedule an activity to shut down (the Program block is more flexible, but the Schedule block uses less memory).

For example, to schedule downtime for a machine, connect the item output from a Schedule block to the “down” connector of a Machine:



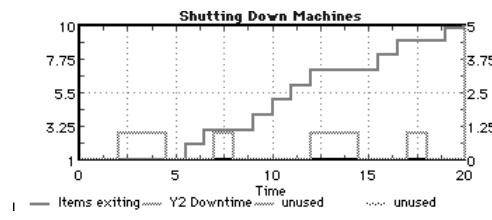
Shutting down a machine on schedule

In the dialog of the Machine, choose “use down connector value as duration”. You can schedule multiple shutdowns and the entire schedule can be repeated at regular intervals. For example, to schedule machine maintenance at time 2 lasting for 2.5 time units and another at time 7 lasting for 1 time unit, you enter the following in the dialog of the Schedule block:

Event Name	Output Time	Value
1) shutdown	2	2.5
2) shutdown2	7	1

Dialog of Schedule block

You can also specify in the dialog that the schedule be repeated every 10 time units, meaning that there would also be a downtime from time 12 until time 14.5 and from time 17 until time 18, and so forth, as shown in the plot:



Scheduled downtimes

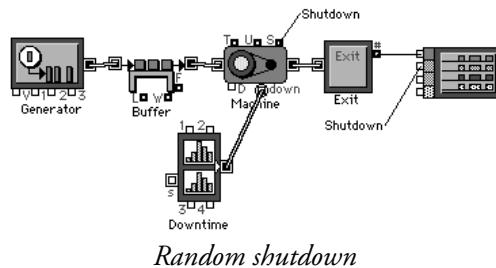
The “Items exiting” line in the plot shows the number of items processed, while the “Downtime” line, which is plotted against the Y2 axis, shows when the machine is shutdown and the duration of the downtime. The S connector gives a value of 1 while the machine is shutdown and a value of 0 while it is not shutdown.

Note that because the “start” connector of the Schedule block is not connected, the schedule performs in absolute simulation time. If you connect to the “start” connector, the schedule will begin relative to when the start connector is activated. The “start” connector is discussed in the section “The “start” connector” on page M86 and in the main Extend manual.

Random shutdown

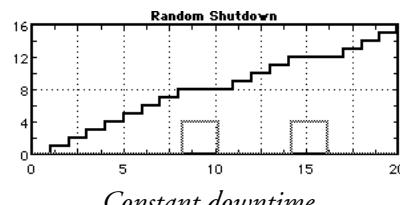
You can shut down activities randomly for a fixed duration and you can shut down activities randomly for a random duration.

For example, assume you want a machine to shut down approximately every 7 time units, with a constant duration of 2. You do this by connecting a Downtime (Unscheduled) block to the “down” connector of a Machine:



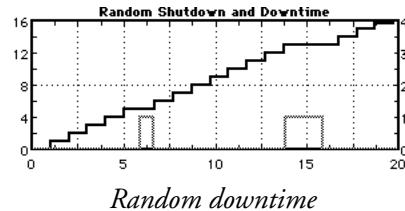
MFG

In the dialog of the Downtime (Unscheduled) block, choose a Normal distribution with a mean of 7 and a standard deviation of 1 for the Time between failures (TBF) parameter. This means there is a time between failures of approximately 7 time units, since when the “down” connector gets an item (about every 7 time units) the Machine will shut down. To set the constant duration of the downtime (time to repair), choose a Constant distribution with a value of 2 for the Time to repair (TTR) parameter. Since the “down” connector is set to use the value it receives as the duration, when the “down” connector gets an item the Machine will shut down for 2 time units. To prevent a downtime occurring at the beginning of the simulation run, enter 1 as the “A” parameter in the “First TBF after start...” field. The graph should look something like:



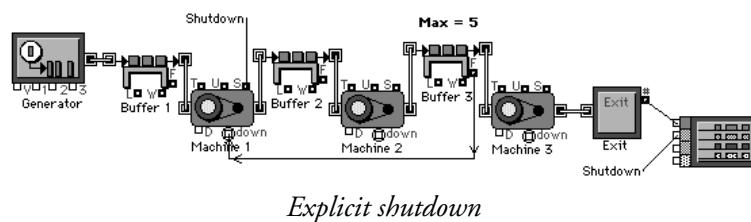
To have a random shutdown with a random downtime duration, choose a distribution for the TTR parameter in the dialog of the Downtime (Unscheduled). For example, select a Normal distribution and use a mean of 2 and a standard deviation of 1. When you do this, the Downtime (Unscheduled) block supplies a random value for the item it generates. The duration of the down-

time will then be normally distributed with a mean of 2. A plot of the model should look something like:



Explicit shutdown

The previous examples have shown how to shut down a machine in isolation from other events in the model. You can also shut down activities based on model factors, for example if a buffer downstream from the machine reaches a limit. To do this, connect the *F* connector from the Buffer block to the “down” connector on the Machine:



Explicit shutdown

You specify the Buffer’s limit in its dialog; whenever the Buffer is full, the “F” connector outputs 1. Since you want the Machine to stay shutdown for as long as the Buffer is full, choose in the dialog of the Machine to “shut down while down connector value >0.5”.

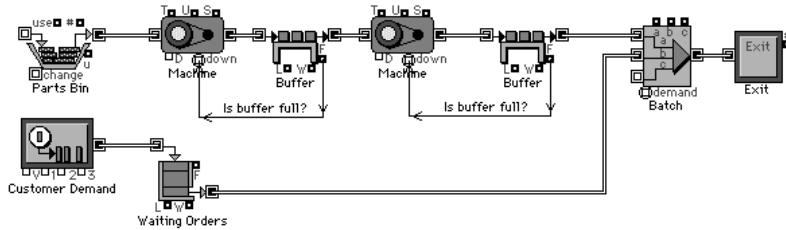
This is a fine example of how you can have downstream factors affect upstream activities. If you examine the model closely, you see that the last machine is processing so slowly that Buffer 3 quickly reaches its limit of 5 items. Since the Buffer cannot take in any more items while it is full, the middle Machine is blocked (cannot process a new item until an item is removed from Buffer 3). However, the first Machine continues to process items, filling Buffer 2. By explicitly shutting down the first Machine, you affect where items are stockpiled and which Machines are shutdown when an activity is blocked.

Note Please also see “Shift block used to schedule” on page M137 and “Using the Shift block” on page M169 for examples of activity and resource allocation that are tied together and scheduled as “Shifts.”

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

This is modeled in Extend by monitoring the buffers between machines and having that information regulate processing. To do this, set the buffer capacity to the number of kanbans and connect the “F” (full) output from the buffer back to the preceding machine’s “down” connector. When the buffer has remaining capacity, its “F” connector will output 0 (zero) and the preceding machine will be authorized to produce parts. If the buffer is full, its “F” connector will be 1 and the preceding machine will be shut down until the queue is reduced.

If you run this model with animation on, the machines will flash a blue circle while they are shut down.

Material handling and transportation blocks

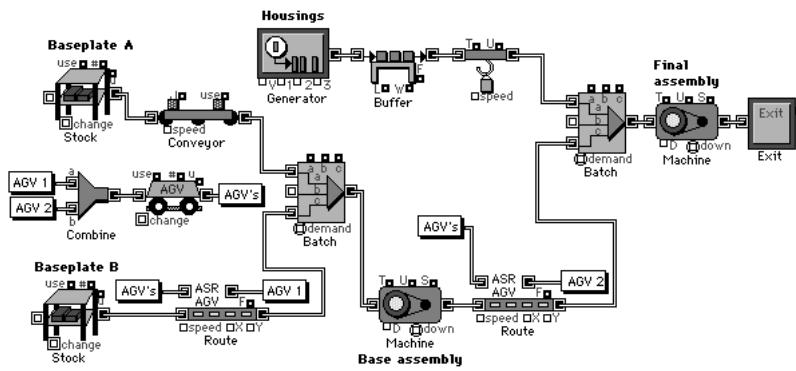
The Manufacturing library has two groups of blocks that represent the movement of items: independent (transporter) blocks and fixed path (routed) blocks. The models discussed in this section can be found in the “Examples \ Manufacturing \ Transportation” folder.

- The *independent* blocks are the Conveyor Belt, Conveyor Carousel, Crane, and Transporter blocks. You can use the independent blocks anywhere in your model and you can set the speed and length of each one depending on your real-world system.
- The *fixed path* blocks are the AGV, ASR, Route, and Route (Delay) blocks. These blocks model systems in which vehicles (either automatically guided vehicles or auto-storage and retrieval vehicles) move along predefined routes. The AGV and ASR blocks provide the vehicles; you specify the number available in their dialogs. There can be more than one vehicle along each route; the maximum number of vehicles is set in the dialog of the Route block. You set the Route length and speed in the dialog, or through input connectors which override the dialog values. The length and speed determine how long the AGV takes in its journey (its delay).

To use AGVs (or ASRs) you need only one AGV block to provide vehicles and one or many Route blocks to provide the routes. Put a Route block between each pair of steps where an item is transported by AGV or ASR. Connect the output of the AGV block to the Route block’s “AGV/ASR”

input. Connect the flow of model items to the unlabeled item input on the Route block (items enter the Route block at the item input and exit at the item output). Connect the Route's AGV/ASR output to the input of the AGV block (this can be accomplished easily with a named connection, as shown below).

An example containing various types of transportation/material handling blocks is:



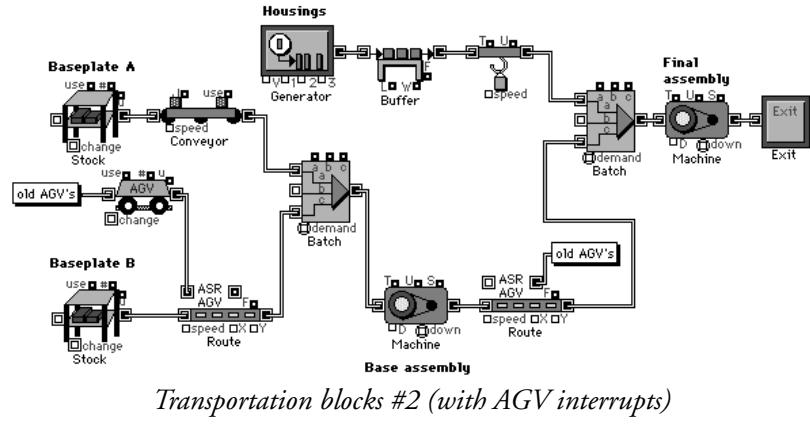
Transportation blocks #1

The model shows how two baseplates are assembled, joined with a housing, then moved to a loading dock. Baseplate A is moved on a conveyor belt to the Base assembly machine while Baseplate B is moved there on an AGV. The two baseplates are joined by a Batch block for processing. After base assembly, they move by AGV to the machine that will join them with the housing. Since the housing is very heavy, it is moved to the final assembly machine by a crane.

This model uses two Route blocks, the first of which picks up the AGVs and Baseplate B and transports them to the Base Assembly machine. The second Route block transports the assembly on an AGV to the Final Assembly machine. You specify in the dialog of each Route block the maximum number of vehicles allowed on the route at the same time, in this case 3 vehicles on each route.

To simulate movement, Route blocks attach AGVs to the items they transport. If the AGV is released at each route, as it is in the model above, the item is left at the end of the route. It can then be processed and continue to another route for transportation. To model the situation where

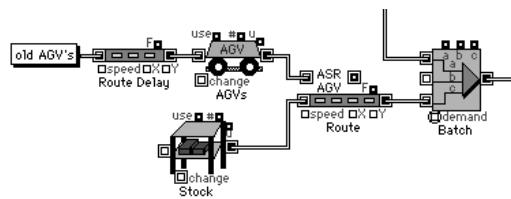
the AGV transports the item to an activity, then waits there to continue transporting the item again, you should not release the AGV until the last route, as shown below:



MFG

In this model, there are AGVs present off the routes. This is because the AGVs transporting Baseplate B stay with it while it is being assembled and continue with it to the next Route block. This means there can be more AGVs in use than are reported in the dialogs of the Route blocks. However, the total number of AGVs is still limited by the number provided by the AGV block. The delay associated with an AGV when it is off the route is the total time it takes the item to be processed.

Notice that in the above models there is no time associated with the return of the AGVs. The Route (Delay) block is useful for when the AGV or ASR move without transporting items, for example when they return to the source and you want to model how long the return takes. In the Route (Delay) dialog, enter the total length of the return route. You would connect it to the input of the AGV resource block as shown here:





Manufacturing Chapter 9: Batching and Unbatching

In every manufacturing process, items pass through the factory and work is performed on them. At the beginning, precursors of the final products come into the process, usually as raw materials, subassemblies, and packaging. These are made available to the process at times and in quantities that are different for each manufacturing facility. These precursors do not exist in a vacuum, however. 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. For instance, in a plant that assembles circuit boards, the chips and the bare boards come together in an early step and travel as an assembled unit through the rest of the process.

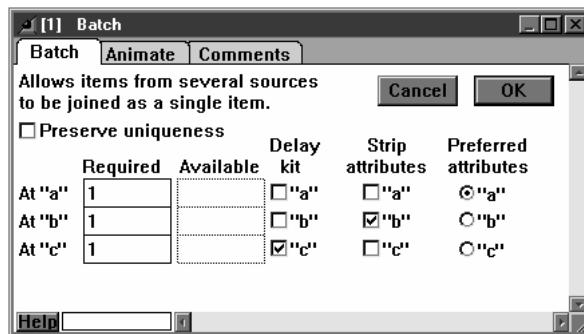
These same concepts apply to other discrete processes. For instance, in an emergency room model, doctors are temporarily *batched* with their patients during medical diagnosis. In the same model, a technician, a diagnostic machine, and a patient would be batched for the duration of x-ray treatment. This chapter will discuss how to join, or batch, items together. See Manufacturing Chapter 10: Resources for a discussion on modeling resources.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Batch	Batching	Discrete Event
Batch (Demand)	Batching	Manufacturing
Batch (Variable)	Batching	Manufacturing
Batch (10)	Batching	Manufacturing
Matching	Batching	Manufacturing
Program	Generators	Discrete Event
Unbatch	Batching	Discrete Event
Unbatch (Variable)	Batching	Manufacturing

Batching

Batching allows multiple items from several sources to be joined as one item for simulation purposes (processing, routing, and so on). The batch blocks accumulate items from each source to a specified count, then release a single item that represents the batch.



Batch block dialog (Windows)

Batching is used to accomplish two slightly different tasks, kitting and binding. Both kitting and binding are merely modeling concepts used to describe specific types of batching. How you construct your model (whether the batched items remain joined or not) determines whether the items are considered kitted or bound.

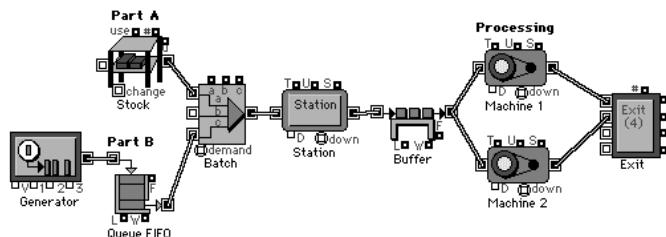
- *Kitting* occurs when a specified number of items are physically joined together, then released as a single item. This is most common when simulating assembly work or packaging operations. You may or may not unbatch the item at some other point in the model; most often you would not. For example, you can batch one manual, two promotional pieces, and four disks to make a software package that is later processed and shipped as one product.
- *Binding* items is used to reflect a situation where one item is required to be temporarily associated with one or more other items as they progress through a portion of the model. Items that are batched for this purpose are usually separated from each other later in the simulation, although they do not have to be separated if you do not care to track the disposition of one or more of the original items. This type of batching is common when simulating labor movement or material handling, and bound items are most often resources. For example, a ship attempting to dock might require two tugboats to guide it through the docking process. You batch the ship and the tugboats, then direct them to an activity block representing the docking process. After this, you unbatch the tugboats and the ship and send them on separate paths in the model.

When the situation you are modeling requires that one item be assembled from two or more precursor items, you use the batch blocks to kit the items together. These blocks are used to take inputs, often from different streams, and join them for processing. In some situations you know in advance how many of each item is required to make one item; in some situations the number of items batched depends on other model factors. The blocks that are used for batching are the

Batch, Batch (10), Batch (Demand), Batch (Variable), and Matching blocks. The models discussed in this section can be found in the “Examples \ Manufacturing \ Batching” folder.

Simple batching

The batching example below joins one of Part A with three of Part B; the assembly then travels as one item through the rest of the manufacturing process:



Simple batching model

MFG

“Part A” and “Part B” are joined by the Batch block according to the dialog, a portion of which is shown below. Note that the Batch block will not release an assembly until it has received one item (Part A) from the top input and three items (Part B) from the bottom input.

Required
At “a”
1
At “b”
0
At “c”
3

Portion of the Batch block dialog

The rest of this model illustrates a simple manufacturing process. The parts take a specified amount of time to assemble, as indicated by the Station block. Each assembly item waits in a buffer for a machine to become free. The Buffer block acts as a collection point that is always querying the two machines and handing the assembly to the first available machine. The first free machine takes the assembly from the buffer, processes it, and sends it out to the Exit (4) block.

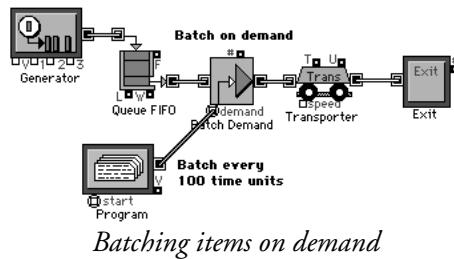
Batching identical items

There are some processes that require you to batch together many identical items into a single package. A common example is packing finished pieces into a box. Although you can just use one row of the Batch block, the Batch (Demand) and Batch (Variable) blocks are specifically designed for this purpose.

The Batch (Demand) block

The Batch (Demand) block collects items until the “demand” connector signals that they should be batched and released. The batch is released as one package composed of the number of items that had been accumulated at that point. You use this when batches are made in a time-dependent fashion or when outside factors determine how many items go into each batch. For example, if

you are 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 truck-load of boxes) is then released.



In seen in the example, you use the Program block to trigger the “demand” connector at scheduled times. A portion of the Program dialog is shown below.

Row	Output Time	Value	Priority	Attrib. value
0	100	1		
1	200	1		
2	300	1		
3	400	1		
4	500	1		
5				
6				
7				
8				
9				

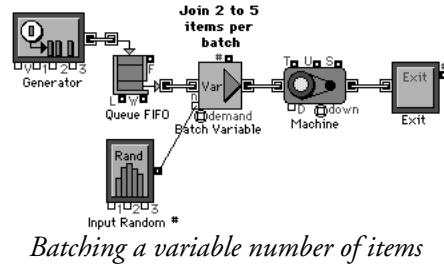
Portion of the Program block dialog (Macintosh)

Since “demand” is activated whenever it gets an item, the Batch (Demand) block will release its stored items at time 100, 200, and so on. A variation on this model, using the Information block (from the Information submenu of the Discrete Event library) to verify model data, is shown in the section “Getting information about items” on page M205. For more information about the demand connector, see page M155.

The Batch (Variable) block

If you want batches that include a variable number of items, use the Batch (Variable) block. This block makes batches that are sized by the “n” connector or the dialog choice. You connect the “n” connector to the output of a block or equation that determines the size of the batches as the model

runs. For example, if an assembly line produces items in different sized boxes that fill a truck, you can decide on the batch size (the number in the truck) based on an equation of the size.



Merging attributes

By default, when items are batched all of the input items' attributes are combined assigned to the output item, and the highest priority (lowest priority value) of any of the input items is transferred to the output item. Some blocks, such as a Batch block, provide options that allow you to control how the attributes are combined. For example, you can strip the attributes of all items entering specific connectors, or you can specify which connector has priority when multiple items have the same attribute. You can also preserve the uniqueness of individual items' properties when you unbatch them, as discussed in "Preserving uniqueness of attributes and priorities" on page M160.

MFG

Delaying kits

When you use the Batch block, you can specify that items at one or more inputs will not be brought into the block until one or more of the other inputs has its requirements filled. You do this when you have a limited resource, such as a technician, that you don't want to have restricted while waiting for other required items to arrive. You can either let all items waiting to be batched enter the block whenever they arrive (the default mode), or you can force the items at specified inputs to wait outside the block. Each input for which "Delay kit at..." is selected in the dialog will have its items wait outside until all required items for the unselected inputs are in the block. For example, in the Batch block dialog on page M152, since the "Delay kit at c" option is selected, items at the "a" and "b" inputs are pulled in until the quantities required at each are filled, then the "c" item is immediately pulled in. At least one of the "Delay kit at..." choices in the dialog must be left unselected.

The demand connector

The batch and unbatch blocks have *demand* connectors that can be used to control the operation of the block. The "demand" connector is a universal input connector. This means that it can accept inputs from either value output connectors or item output connectors. The presence of an input at the demand connector is used to trigger batching or unbatching.

- If a *value* output connector is connected to the demand connector, it is used as a true/false indicator, triggering batching or unbatching. The actual value from the value connector is ignored; what is considered is whether or not it exceeds a value of 0.5. Value connectors outputting a

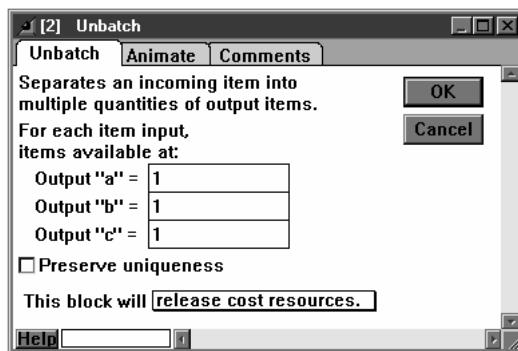
value less than or equal to 0.5 are considered false, while a value above 0.5 is considered true. For example, a value connector sending the number 4 to the demand connector of a Batch block will cause the block to pull in available items and batch them. If the value had been 0.3, the Batch would not have pulled in items.

- If an *item* connector is connected to the demand connector, any item it receives is considered the same as a true value and causes the block to be activated. The demand connector takes into consideration the Value of the item at demand. For example, if an item connector sends an item with a Value of 4 to the demand connector of the Unbatch block, the demand connector will cause the block to pull in 4 items to be unbatched. If required items are not immediately available, the block will remember the demand amount and decrement it only as items become available and are pulled in for unbatching. Note that this demand feature only applies to item Values; as mentioned above, it does not apply to the values coming from value connectors.

MFG

Unbatching

Unbatching is usually used to separate items that were temporarily batched. It can also be used to duplicate or clone items, even items that have never been batched.



Unbatch block dialog (Windows)

- There are cases when you want to batch items only temporarily; this is called binding, as discussed above. Generally, items that are bound are subsequently unbatched. For instance, you can move an item on a pallet through part of your process, take the item off the pallet, move the item on through the line, and move the pallet back into the pallet storage area. It is also common to have resources (especially labor) that are batched with other resources, delayed, then returned to the resource block. For example, you can have a worker, a job order, and some parts that need to come together, be delayed (as the worker assembles some of the parts), and then move the parts on in the factory while the worker goes back to get the next order and parts. To do this you first batch the items using one of the batch blocks discussed above, specifying in the batch dialog how many of each item is required from each input. You can then use an unbatch block to separate the items, routing them appropriately into the rest of the model. When you

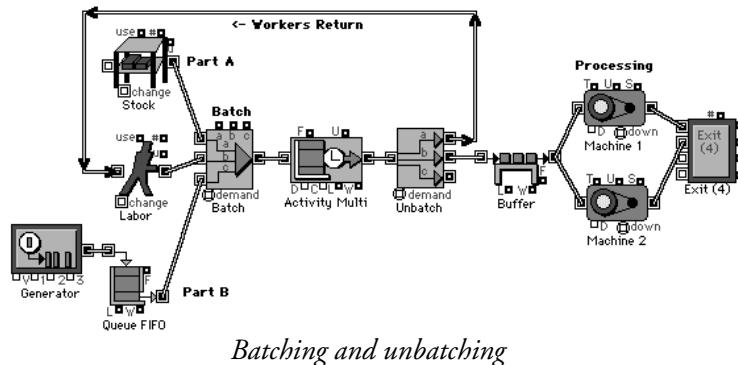
unbatch items in this situation, you specify the same number for each output in the unbatch dialog as you specified for the inputs in the batch dialog.

- Unbatching can also be used to duplicate an item into many *duplicates* of itself, even if the item has not been previously batched. For instance, you might want to simulate order processing procedures where the order is received, entered into the computer, and printed. You can unbatch each resulting invoice package into 3 duplicates that represent a packing slip, an invoice, and a shipping copy. As another example, you can have one item represent a group of items throughout the modeling process, then unbatch the group at the end of the model to determine how many individual items were actually processed. The numbers specified in the dialog of the unbatch block determine how many duplicates of the item you want at each output.

The blocks that are used for unbatching are the Unbatch and Unbatch (Variable) blocks. The Get Attribute block (from the Attributes submenu of the Discrete Event library) also has an option that allows you to duplicate items based on an attribute value.

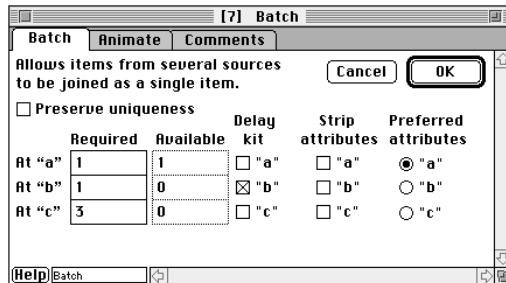
Simple unbatching

The following model is an extension of the example “Simple batching” on page M153, with the addition of unbatching:



Before the parts are passed to the machines, they must be processed by a worker, who is represented by a Labor block. Although not actually joined with the parts, the worker is bound tempo-

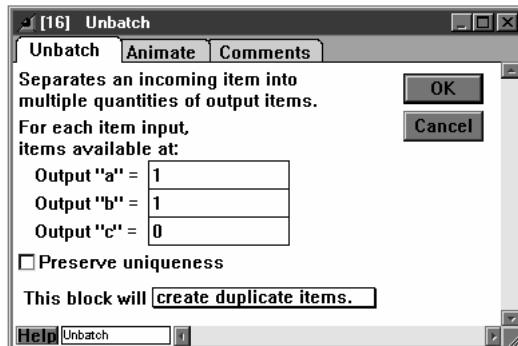
rarily with them by the Batch block. The Batch dialog is modified to require one item at input "b" (that is, one worker) before the assembly item is passed out of the block:



Binding a worker (Macintosh)

MFG

In the dialog, "Delay kit at b" is selected, indicating that a worker will not be pulled into the block until the other required items are present. (This is especially useful for conserving resources, such as when you model workers being needed at more than one location in the model). Each worker performs a task (represented by an Activity Multiple block from the Discrete Event library) on the batched item and, when finished, puts the item in the buffer and returns to the labor pool ready to perform the task on another unit. This is modeled by the Unbatch block, which takes a single item (the output from the activity) and creates two items. One item represents the worker who is sent back to the labor pool and the other represents the assembled item that is passed to the buffer:



Unbatching 1 assembly and 1 worker (Windows)

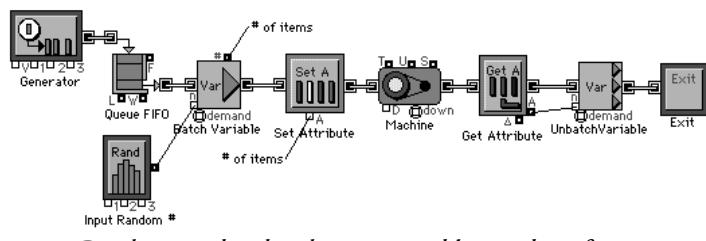
Notice in the example that the worker is batched at the "b" input, but unbatched at the "a" output. This is fine as long as you are not preserving the uniqueness of item properties (discussed in "Preserving uniqueness of attributes and priorities" on page M160). Since you are not doing that in this model, it does not matter which output is used for unbatching a particular item. What is important is that the correct quantity be output and that it be correctly identified in the model. It was simply more convenient to route the workers out the top output on the Unbatch block. If you

select “Preserve uniqueness” in the Batch and Unbatch block, you must have the outputs correspond to the inputs or preserving uniqueness will not work correctly.

Also notice that the work performed by the laborer is represented by the Activity Multiple block. The Activity Multiple block allows more than one item to be processed at a time, and it releases items in the order of their elapsed delay times. Since the Batch block might be ready to release a second assembly item while the first one is being processed, you use an Activity Multiple block instead of a Station block so that all items can be processed when available. This has the real-world equivalent of having more than one labor station available at a time, similar to a workbench large enough to accommodate many people and parts.

Variable batching and unbatching

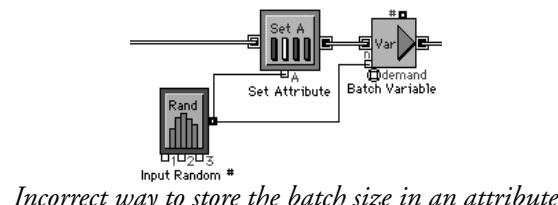
If you batch a variable number of items, and want to unbatch that same number of items, use the Batch (Variable) block. To keep track of the number of items a batch is composed of, assign an attribute to each batch. Set the attribute value to be the number in the batch, and use that value as the “n” input in the Unbatch (Variable) block:



Batching and unbatching a variable number of items

Notice that the Set Attribute block is placed immediately after the Batch (Variable) block. This insures that the batched item's attribute value corresponds to the number of items for that batch. Attaching the “A” output on the Get Attribute to the “n” input of the Unbatch (Variable) causes each batch to separate into its original number of items.

You might think that it would be easier to assign the batch size to the attribute by directly connecting the Input Random Number block to the Set Attribute block, as shown below. This, however, would not work properly. The Input Random Number block would generate a number when an item passes through the Set Attribute block and would generate a new number when the item reached the Batch Variable block.



Incorrect way to store the batch size in an attribute

Note The size of the batch is set when the first item for that batch arrives in the Batch block.

Preserving uniqueness of attributes and priorities

By default, the highest priority that was on any of the items that are batched is transferred to the output item, and all of the input items' attributes are combined in the output item. This is true even if the item is subsequently unbatched. Some blocks, such as the Batch block, provide options that allow you to control how the attributes are combined. For example, you can strip the attributes of all items entering specific connectors, or you can specify which connector has priority when multiple items have the same attribute. As the batched items move through different streams in the model, you can simply ignore the extra attributes that are associated with the item. Thus, if a box of screws is made from a box item and many screw items, the box of screws will have the screw size attributes as well as the box attributes as it travels in the model. If it is unbatched, each screw will still have a size and a box attribute. Since you care only about the size attribute when looking at the screw, you can just ignore the box attribute.

If it is important in your models to not have attributes combined or priorities superseded when you subsequently unbatch batched items, you can select "Preserve uniqueness" in the batch block and in the corresponding unbatch block. The "Preserve uniqueness" option marks the items in the batch as unique so that an unbatch block can restore all of the items' properties (attributes and priorities). If not selected, the unbatching block will create items with merged properties.

The preserve uniqueness choice uses more memory and slows down the execution of the model. In addition, you must be careful when using any property-setting blocks (attributes or priorities) in the path between the batching and the unbatching blocks. Note that any property modifications you make between the batch and unbatch blocks will be lost once the item is unbatched, since the items are unbatched exactly as they were when they were batched. Also note that 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.

Manufacturing Chapter 10: Resources

Often during a manufacturing process items will require precursors, or resources, before they can proceed to the next step in the process. Resources are entities that provide a service to the items in your model. For example, a palette provides a means for carrying items while a laborer provides a means for assembling parts. This chapter will discuss the various ways to model resources in Extend.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Decision	Decisions	Generic
Generator	Generators	Discrete Event
Get Attribute	Attributes	Discrete Event
Labor	Resources	Manufacturing
Program	Generators	Discrete Event
Queue Resource Pool	Queues	Discrete Event
Resource Pool	Resources	Discrete Event
Release Resource Pool	Resources	Discrete Event
Schedule	Generators	Manufacturing
Station (Attributes)	Activities	Manufacturing
Stock	Resources	Manufacturing

The models discussed in this section can be found in the “Examples \ Manufacturing \ Resources” folder.

Closed and open systems

The blocks that provide a finite resource of items can be part of closed or open systems. How you connect the blocks determines whether the system is considered open or closed. In a closed system, resources are recycled back to the originating block. In an open system, resources are not recycled. Systems can also be partially closed, for example when some of the items are recycled back and some come from an outside source.

Closed systems

In a closed system, items are routed from a resource-type block, where they are used in processing an item. Then the resources are recycled back to the resource-type block's input connector once they are no longer being used elsewhere in the model. For example, assume a technician (the resource) is required to assemble parts of a television. While the technician is assembling the parts, he/she will be busy and will not be available to perform work elsewhere. In a closed system, upon assembling the parts, the technician will return to the technician pool and will become available for other assignments.

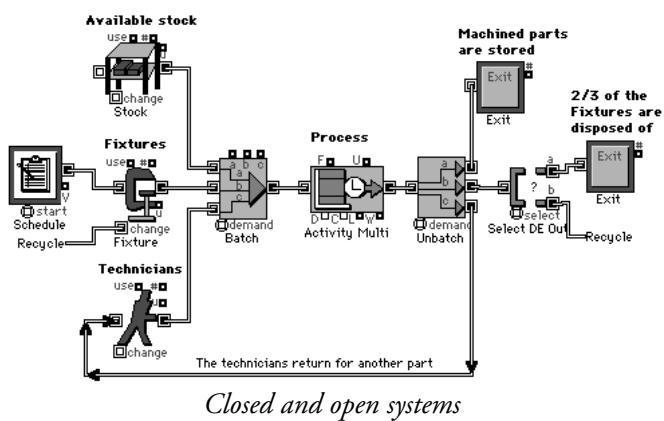
In a partially closed system, only a portion of the resources are returned to the pool for re-use. For example, consider a case where there are different shifts of laborers (the resource). Suppose three laborers are assigned to a task. Upon completion, the shift for one of the laborers is finished and he does not return to the labor pool to be assigned to a new task.

MFG

Open systems

Resource-type blocks are part of *open systems* when the block's items are not recycled. In an open system, items at the end of the line are not passed back to the block's input connectors, they exit the simulation. The most common example of an open system is stock. Normally, stock passes out of the model at the end of the line. Another example of an open system is a consumable resource such as a disposable fixture that makes only one pass through the manufacturing process.

The following model shows an open system (Available stock), a partially closed system (Fixtures), and a closed system (Technicians).



Modeling resources

In Extend, there are two main ways to model resources:

- Batching resources with items, as discussed in Manufacturing Chapter 9: Batching and Unbatching and briefly in this chapter.
- Using the Resource Pool blocks (Resource Pool, Queue Resource Pool, and Release Resource Pool). This method controls the flow of items by keeping track of available resources pool units rather than modeling individual resources as items.

This section will discuss these two methods of modeling resources and their respective strengths and limitations.

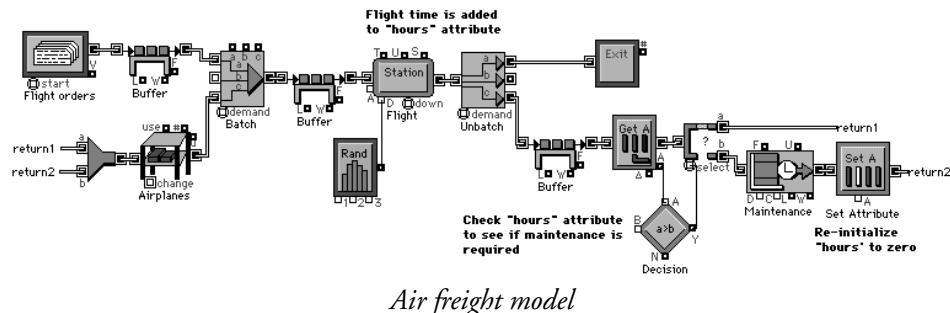
Batching resources with items

When using this method, the resources are represented by items whose purposes are to provide a service for other items in your model. The resources can therefore have attributes, priorities, and values like any other item in your model. For this reason, this method of modeling resources is preferred if you need to track information about resources using attributes (see “Accumulating data using attributes” on page M206).

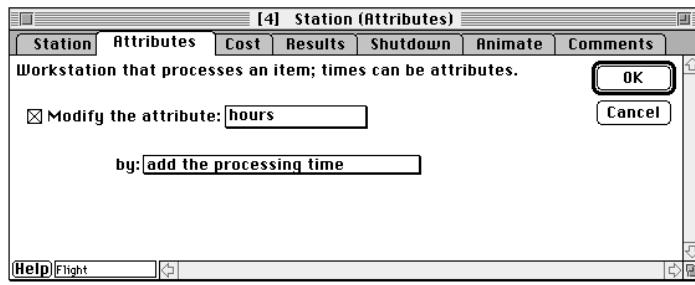
In the dialog of the resource block you specify the number of resources that are initially available. When an item uses a resource, it is batched with the resource (see Manufacturing Chapter 9: Batching and Unbatching). While a resource is batched with an item, it cannot be used elsewhere in the model. 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. Thus 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 be used again.

For example, suppose you are modeling an air freight company. Regulations require that the airplanes must undergo maintenance after every 50 hours of flight time. In this model, you need to track the accumulated processing time (flight time) for each resource (airplane). The model looks like:



As flight orders come in, they are batched with an airplane. If an airplane is not available, the flight order will wait in the queue until one becomes available. When a airplane is batched with a flight order, it will no longer be available to handle other flight orders. The Station (Attributes) block will add the flight time to an attribute called “hours”. The attributes tab of the Station(Attributes) block is:



Adding processing time to “hours” attribute (Macintosh)

MFG

After the flight, the Get Attribute block reads the “hours” attribute. 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 and the “hours” attribute will be re-initialized to zero; if not, the airplane will be returned to the Stock block where it will wait for another flight order.

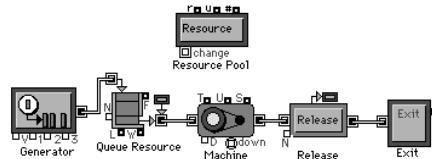
In the above model, it is essential to be able to track information about each resource. Therefore the ability to assign attributes to the resources is critical.

As described in “Preserving uniqueness of attributes and priorities” on page M160, an item returning to a resource-type block after batching may have many attributes that are irrelevant to the returning item. In Extend, the resource-type blocks give you the choice of stripping attributes or keeping them with resources that are recycled; the default is to strip them. If you are tracking information with the resources using attributes (as in the above model), you will not want to strip the attributes. 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 again.

Using the Resource Pool blocks

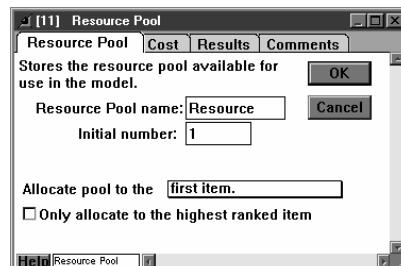
The resource pool blocks (Resource Pool, Queue Resource Pool, and Release Resource Pool) do not use actual items to model the resources. Instead, these blocks cause restraints to be placed on the flow of items in the model based on the availability or lack of resources. The Resource Pool block maintains a count of the number of resources that are currently available for use. When an item enters a Queue Resource Pool block, the block will query the appropriate Resource Pool(s) to determine if the required number of resources are available. If so, the Resource Pool will appropriately decrement the number of resources currently available, and the item will be released from the Queue Resource Pool. If the required number of resources are not available, the item will wait in the Queue Resource Pool until resources become available.

In a closed system, the resources are returned to the Resource Pool block by passing the item through a Release Resource Pool block. A simple model using the resource pool blocks looks like:



Simple model using resource pool blocks

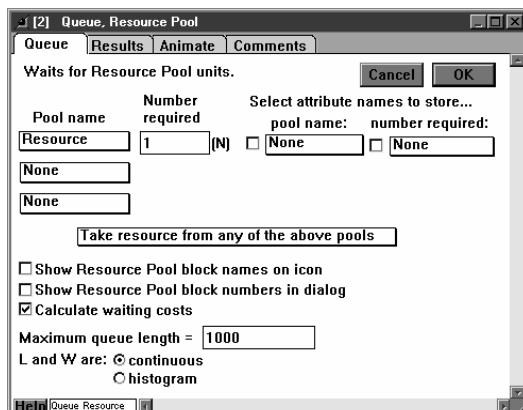
Within the dialog of the Resource Pool block, you must name the resource pool and define the initial number of resource pool units as shown below:



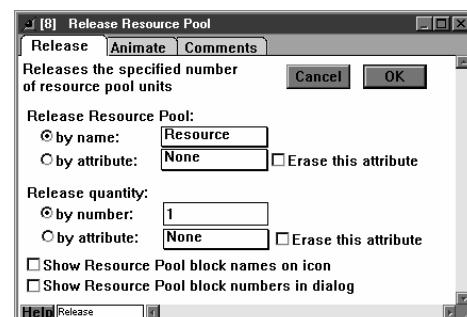
Dialog of Resource Pool block (Windows)

MFG

Within the Queue Resource Pool and Release Resource Pool blocks, you specify the number of resources required (in this case, “1”) and the name of the Resource Pool block from which the resources originate (“Resource”). For the above model, the dialogs of the Queue Resource Pool and Release Resource Pool blocks look like:



Queue Resource Pool



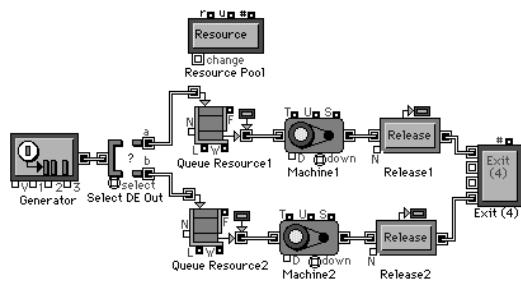
Release Resource Pool

Resource pool block dialogs (Windows)

Notice that the Resource Pool block does not require any connections to the Queue Resource Pool or Release Resource Pool blocks. Because of this, using resource pool blocks to model resources (as opposed to using the batching method described on page M163) is much 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. This 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.

Same resource used in multiple places

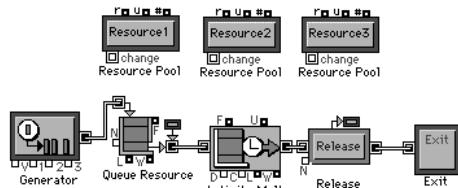
In the following example, items waiting in both Queue Resource1 and Queue Resource2 require a resource from the Resource Pool block. When an item enters one of the queues, a request is sent to the Resource Pool block for a resource. As resources become available, the requests are satisfied in the order in which they were received (or in order of the requesting item's priority as defined in the Resource Pool dialog). Modeling this using the batching method would require complex logic to route the resources to the correct station.



Resources from one Resource Pool block used in multiple places

Resources required from different pools

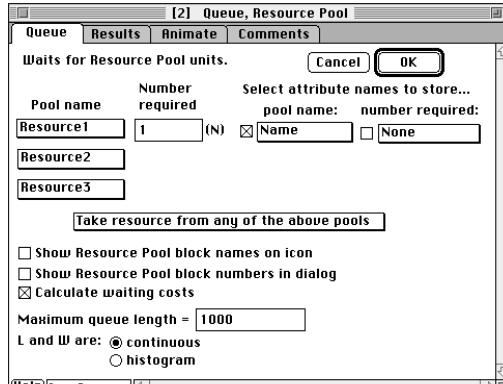
In the following example, items require one resource from *either* Resource1, Resource2, or Resource3.



One resource required from any of the three pools

As shown in the Queue Resource Pool block dialog below, you can instruct the Queue Resource Pool block to take the resource from any one of the pools specified in the dialog. When an item enters the Queue Resource Pool block, the block will query each of the Resource Pool blocks in the order that they are listed (from top to bottom). In other words, if no resources are available in

Resource1, the Queue Resource Pool block will try Resource2, and then Resource3 until the resource requirements are met.



Dialog of Queue Resource Pool block

MFG

Note that you can store the resource pool name and the number of resources used in attributes that attach to the items processed. In this model, the Resource Pool block name is stored in the attribute “Name”. This attribute is used by the Release Resource Pool block to inform the appropriate pool that the resources are no longer required. Again, modeling this using the batching method would require complex logic to correctly route the resources.

A limitation of modeling resources using the resource pool blocks is that this method does not allow you to use attributes to track information about the resources as you did in “Batching resources with items” on page M163.

Scheduling resources

There are two systems available to schedule resources:

- Use the Change connector on individual resource blocks
- Use Shift blocks to change multiple resource and activity blocks

Use of the Shift block is discussed in the section after this.

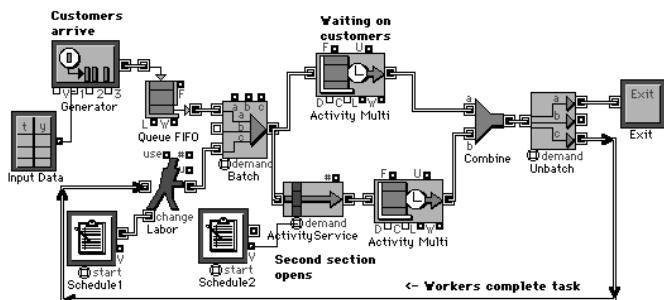
Using change connectors

Blocks that provide resources (whether as items or as constraints on items) have input connectors labeled *change*, in addition to the regular item input connectors. The “change” connector works the same as the regular item input, except that it also takes items with a negative Value (item Values are discussed in the Extend user’s manual).

You typically use the “change” connector to reduce the number of resources available or to recycle items back in a partially closed system. This change can be scheduled, such as when workers take breaks, or unscheduled, such as for loss of tools.

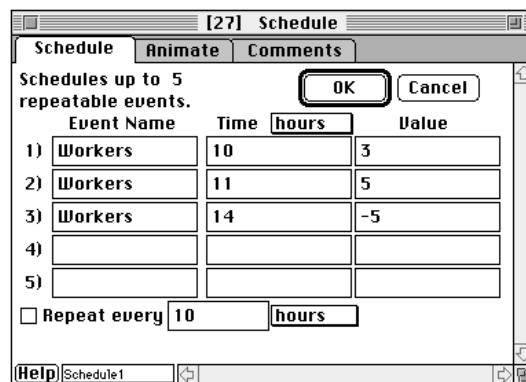
Sending items with a negative Value to the “change” connector reduces the number of items available in the resource by that value, as long as there are sufficient items available. If the block doesn’t have enough items to dispose of, it will retain a negative demand for future items until the demand is met.

It is common to schedule the availability of resources based on some factor in the model, typically time. For example, in the coffee shop model discussed in “Scheduling activities” on page M135, you could schedule workers in the coffee shop depending on the time of day using the Labor and Schedule blocks. The new model is:



Scheduling resources in a coffee shop

Assume there is an initial number of workers (3) available when the coffee shop opens, and 5 additional workers arrive and remain just through the busy period, which is from 11 until 2. You enter the schedule for the day in the dialog of the Schedule block, as shown below:



Arrival times for workers (Macintosh)

Note that at time 14 (2 PM) the number of workers in the dialog is -5. Since the “change” connector of the Labor block accepts negative numbers, the -5 will decrease the available workers from 8 to 3 at exactly 2 PM.

Note that in this model the workers are part of a partially closed system. They are recycled back to the Labor block through the input connector, while additional workers are added to the block through its “change” connector.

Using the Shift block

The Shift block is used to schedule both the magnitude and availability of capacity in other blocks in a model. This is useful for simulating how a system’s resources generally 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.

MFG

The shift block can schedule capacity in a number of blocks in the Discrete Event, Manufacturing and BPR libraries. This includes but is not limited to: Generator, Resource Pool, AGV, Machine, Conveyor Belt, Pallet, Process Preemptive, Labor, Import, and Repository. For example, it can turn on or off an Activity Multiple, or specify a maximum number of activities for that block, depending on how it is set up.

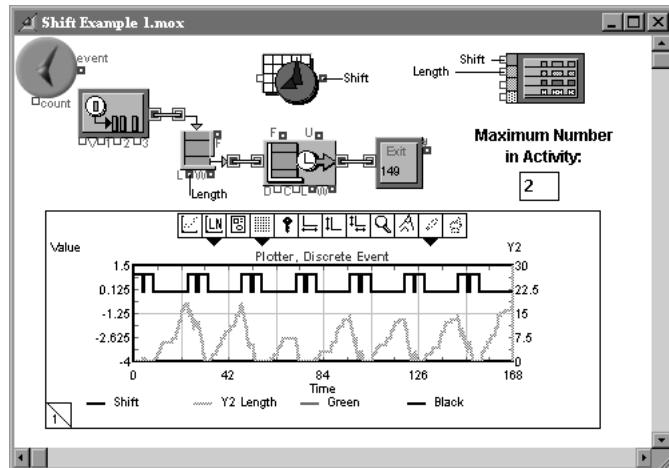
The Shift block controls the capacity of specified blocks in a model based on a schedule that is defined in its dialog table. If a shift schedule is changed, all blocks using that shift will receive the same modified shift pattern. In addition, shifts may be repeated at regular intervals if the “Repeat every” checkbox is selected and a time is entered in the adjacent dialog item. This is useful for modeling repeated shift patterns, e.g., an eight hour work day each day of the week or breaks that occur every four hours.

The block’s dialog has a pop-up menu offering two general types of schedules: On/Off and Numerical. The On/Off shift is a binary switch that turns associated blocks on or off at different points in time. For example, an On/Off shift might be used to shutdown an activity during lunch and evening hours. The other type of schedule, the Numerical shift, explicitly defines the size of a block’s capacity over time. For example, a Numerical shift might set the size of a Resource Pool to three items for the morning shift, zero over lunch, five for the afternoon shift and zero overnight.

The block’s input value connector may be used to override the shift schedule. If the input connector is less than 0.5, the shift is considered off shift and 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 instead. The output connector reports the current shift status (e.g., for ON/OFF shifts, ON = 1 and OFF = 0).

Examples showing the use of the Shift block can be found in the “Examples \ Manufacturing or BPR \ Shift” folder.

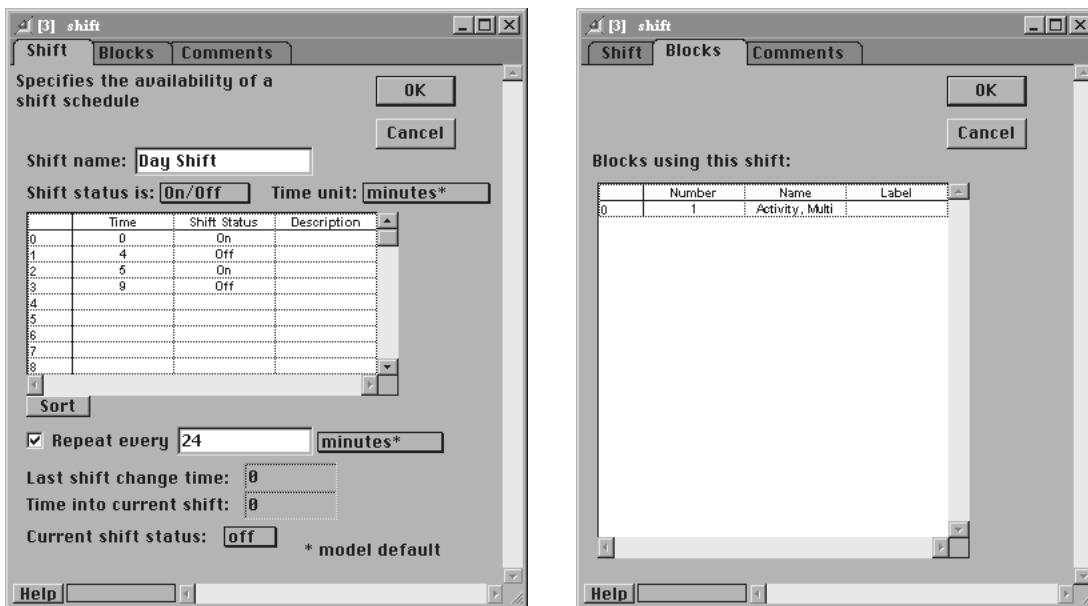
Simple On-Off Shift Example



Shift Example1.mox

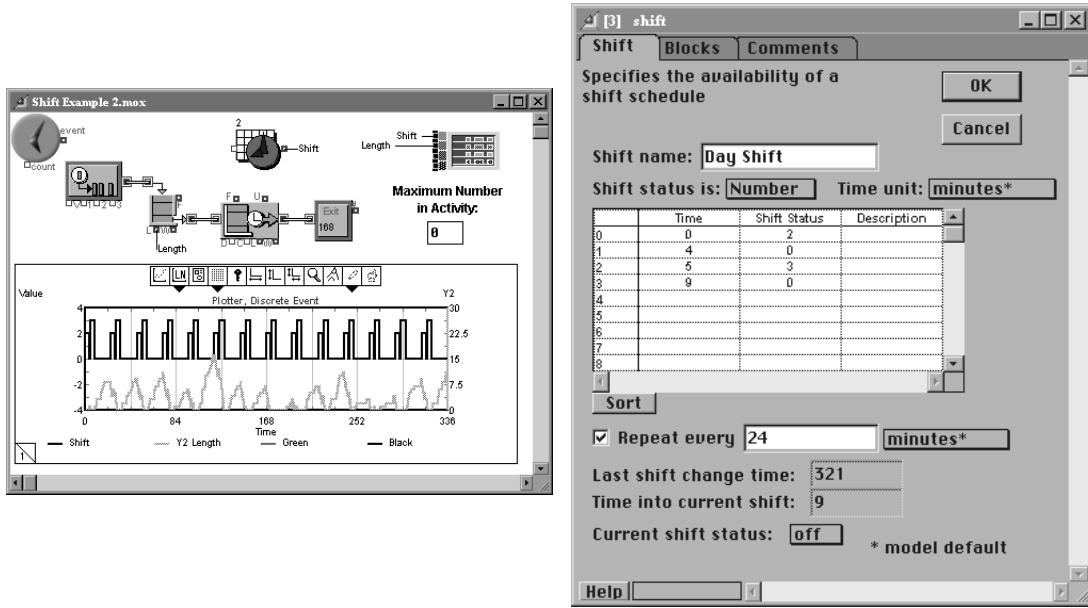
MFG

In this example, *Shift Example1.mox*, an On-Off shift is used to turn the Activity Multiple block on for four hours in the morning, off for one hour at lunch, back on for four hours in the afternoon and off again in the evening until the next morning. The upper black line reflects this 24 hour shift cycle for five days. In addition, observe how the lower gray line reflects the queue length's growth during off shift time periods.



Settings for the Shift and Blocks tabs

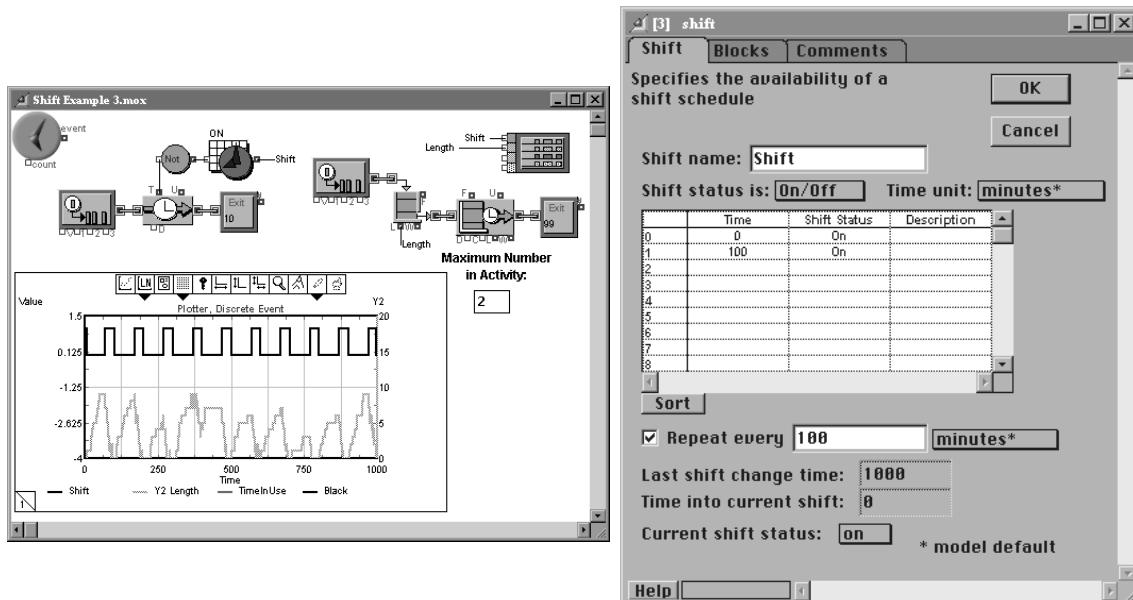
Simple Numerical Shift Example



Shift Example2.mox

In this example, *Shift Example2.mox*, a Numerical shift is used to control the maximum number of items in the Activity Multiple block. The Activity Multiple is limited to two items during the morning shift, zero during lunch, three items during the afternoon shift and zero overnight. This behavior is reflected in the plotted upper black line over the course of five days. Again, the lower grey line reflects the queue length's growth during Activity downtime.

Activity Status Shift Control Example



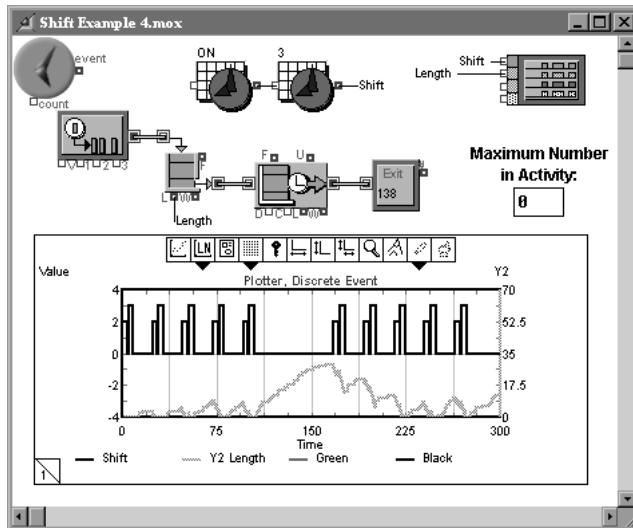
Shift Example3.mox

In the following example, the “T” connector on an Activity Delay block is used to control the shift status of an Activity, Multiple block. The value of the T connector depends on the status of the Activity Delay; is it busy or idle. Given the configuration of the model below, this ultimately impacts the Shift block’s status:

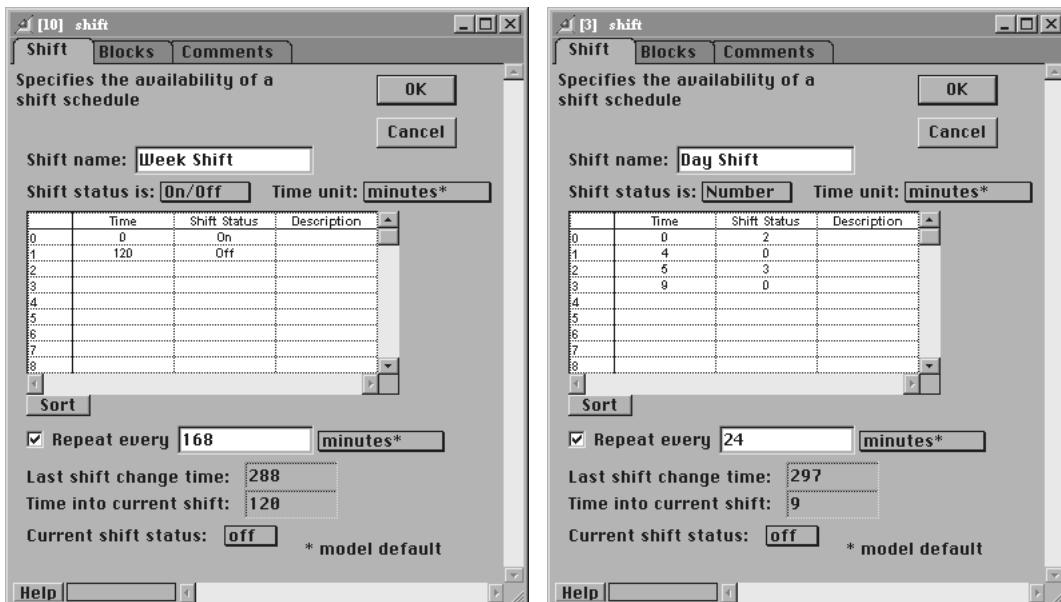
T Connector	Logical Not	Shift Status
1	0	Off
0	1	On

When an item is being processed in the Activity Delay block, its “T” connector value is one and the resulting shift status is Off. Consequently, the Shift block ignores its dialog shift schedule and forces the Activity, Multiple block off line. Conversely, when the Activity delay is idle, the “T” connector value is zero and the Shift block has the opportunity to observe the shift schedule defined in its dialog table. The Shift block’s table has been set to the “On” position 100% of the time so when the Activity Delay is idle, the Activity, Multiple is brought on line. In this example, the Activity Delay is busy with an item 70% of the time, causing the Activity Multiple to be on shift only 30% of the time. This behavior may be noted in the associated plot.

Shift Block in Serial Example



Shift Example4.mox



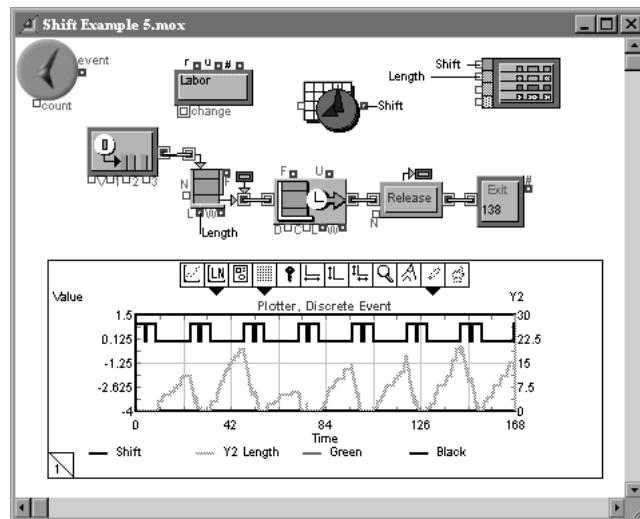
Shift settings for the first and second Shift blocks

Shift blocks may be configured serially to create more complex shift patterns. In the following example, the typical 40 hour work week with two days off during weekends is modeled by feeding a weekly On/Off Shift block into a daily Numerical Shift block. The weekly shift is On for the five weekdays and Off during weekends. Consequently, during weekdays, the weekly shift block sends

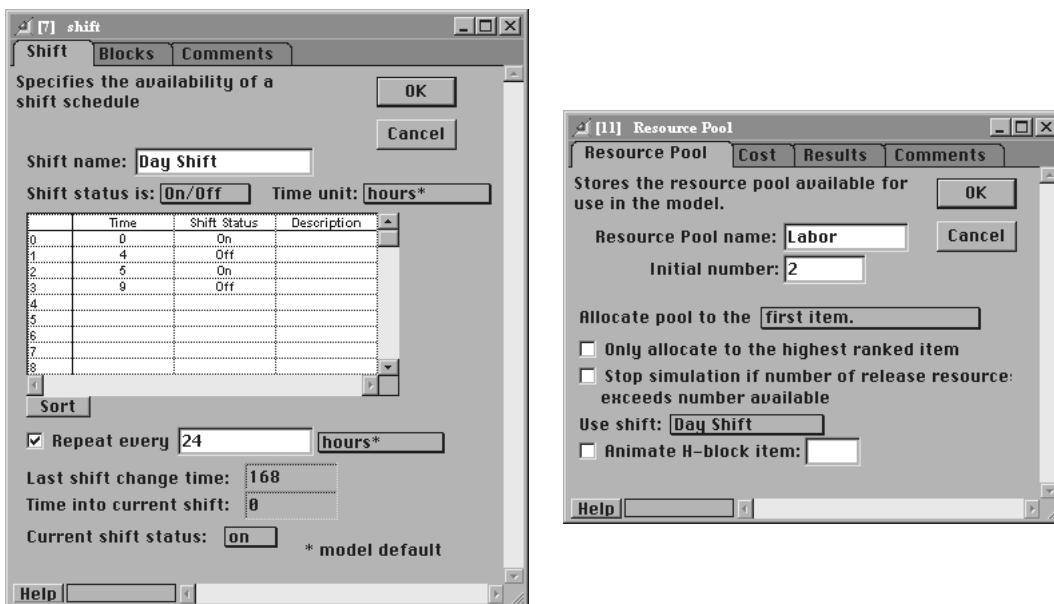
a value of one to the daily shift block, allowing it to observe its own daily schedule. However, during weekends, the weekly shift block overrides the daily shift schedule by sending it a value of zero. The black plotted line shows two work weeks separated by a weekend.

Shift controlling Resource Pool

MFG



Shift Example5.mox



Settings for the Shift and Resource Pool blocks

This example illustrates the Shift controlling a Resource Pool availability so that workers start their shift, work 4 hours, take a lunch break, work four more hours, and then leave for the day.

When workers are not available, the backlog starts building up in the Queue Resource Pool.

Additional Shift block examples can be found in the “Examples \\ Manufacturing or BPR \\ Shift” folder.

An Important Note About Using Shifts

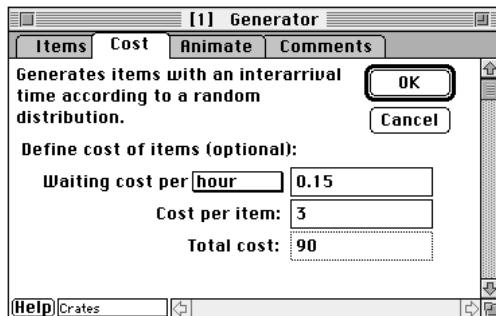
It is quite easy to define a complex shift schedule to a model which may include all breaks, holidays, weekends, etc. However, before adding such complexity to the model, the model builder should carefully consider whether such detail adds to the validity of the model. If for example, nothing at all happens during the weekend, a better solution would be to simply make one week 5 days long 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.



Manufacturing Chapter 11: Activity-based costing

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. You use ABC to identify, quantify, and analyze the various cost drivers (such as labor, materials, administrative overhead, rework, etc.) and to determine which ones are candidates for reduction.

By building your models, you have already identified the discrete outputs of your system as well as the processes and resources that are involved in creating those outputs. To add ABC to your models, you enter costing information into block dialogs. Blocks that generate items or resources and blocks that process items have tabs in their dialogs for specifying costing data. The Cost tabs allow you to enter variable cost rates per time unit and fixed costs per item or use.



Cost tab of Generator block

Once you have specified the cost information, costs will automatically be tracked as the items in your model evolve from system inputs to the finished product.

The first section of this chapter is an overview of how to perform ABC using Extend. The second section takes a more in-depth look at the costing calculations that are automatically performed when Extend tracks costs. The following blocks will be of main focus in this chapter:

Block	Type	Library
Batch	Batching	Discrete Event
Batch(Demand)	Batching	Manufacturing
Batch(Variable)	Batching	Manufacturing
Cost By Item	Statistics	Statistics
Cost Stats	Statistics	Statistics
Generator	Generators	Discrete Event
Labor	Resources	Manufacturing
Program	Generators	Discrete Event
Queue, Resource Pool	Queues	Discrete Event
Resource Pool	Resources	Discrete Event
Release Resource Pool	Resources	Discrete Event
Unbatch	Batching	Discrete Event
UnBatch(Variable)	Batching	Manufacturing

The models discussed in this section can be found in the “Examples \ Manufacturing \ ABC” folder.

Performing activity-based costing

This section will discuss how to define cost rates, how to properly combine cost resources with items, and how to gather and work with the cost information.

Item types

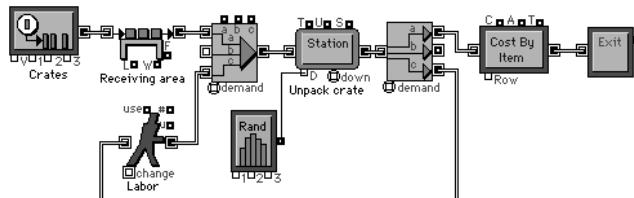
When considering ABC in Extend, every item in your model may be categorized as either a *cost accumulator* or a *resource*. This section will discuss the differences between cost accumulators and resources. It is important to understand that difference, because Extend treats them differently, as you will see in “How Extend tracks costs” on page M188.

Cost accumulators

You perform ABC to determine the costs associated with processing an item. The item being processed is called the *cost accumulator* and will accumulate costs as it waits, gets processed, or uses resources.

For example, assume you want to determine the cost associated with receiving crated inventory at a warehouse. As each shipment arrives, a labor resource is required 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. As the crate item is stored or processed it will accumulate costs. For instance, it might take the laborer approximately 30 minutes to unpack the crate. In the model below, the processing time and the hourly wage of the laborer is used to automatically calculate the cost of that particular step. That cost is then added to the accumulated cost being tracked with

the crate. As the crate progresses through the steps of being received, Extend will add the cost incurred at each step to the accumulated cost.



Receive inventory model

Cost accumulating items can be introduced into a model using some generator-type blocks (Generator and Program blocks) and some resource-type blocks (Resource, Bin, and Stock blocks), as you will see in “Cost accumulator cost rates” on page M180.

Resources

As you saw in Manufacturing Chapter 10: Resources, resources can be modeled using the Resource Pool blocks or by using batching. In the resource pool method, the resource pool units act as constraints on the flow of items throughout the model. In the batching method, resource items are combined, or batched, with other items before the items can proceed to the next station.

Resources are entities that provide a service for the items in your model; they do not accumulate their own costs. However, whenever a cost accumulator uses a resource, the cost rates of that resource (discussed in “Resource cost rates” on page M181) are used to calculate the costs which are added to the total cost of the cost accumulator. For instance, in the example described in “Cost accumulators” above, the laborer is a resource item that is batched with the crate. Although the laborer is modeled as an item, it represents a resource and 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.

Resources are introduced into a model using resource-type blocks (Resource, Resource Pool, AGV, ASR, Bin, Fixture, Labor, Pallet, Stock, and Tool blocks).

Defining costs and cost rates

To include ABC in your model, you simply enter information in the Cost tabs in the dialogs of the appropriate blocks (generators that provide cost accumulators or resources and activity blocks used to process the cost accumulators). 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, Extend will automatically track costs when the simulation is run. There are two types of cost information that you can define:

- The *cost per use/item* or fixed cost.

- The *cost per time unit* or variable cost rate.

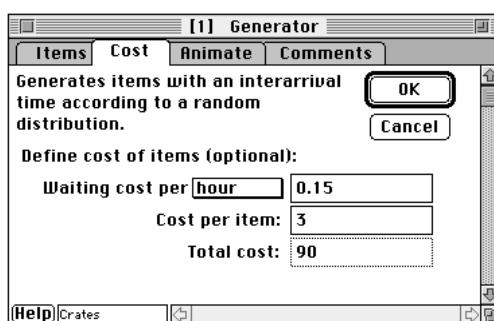
This section will describe how to define costs for cost accumulators, resources, and activities.

Cost accumulator cost rates

Cost accumulators have their own fixed costs and variable cost rates. They acquire additional costs from resource items and activities as they use the resources and are processed.

You specify the cost rates for a cost accumulator from within the Cost tab of the block that originates it. Each cost accumulator can have a fixed cost per item, such as a direct materials cost, and a variable waiting cost rate, causing it to accumulate costs as it is stored or waits for processing.

Cost accumulators are often generated by the Generator block. For example, in the Receive Inventory model described on page M178 the Generator block creates the crates of inventory. Within the Cost tab of the Generator block, define the cost per time unit (otherwise known as waiting or storage cost) and the cost per item, as shown below:

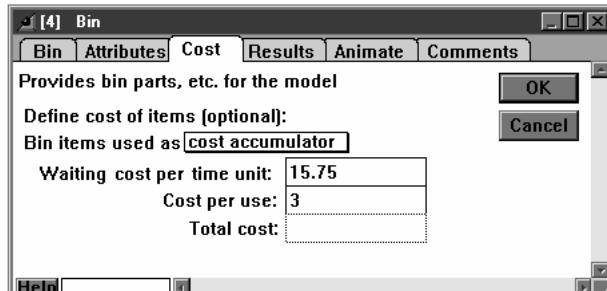


Cost tab of Generator block (Macintosh)

The waiting cost is the rate at which cost is accumulated while the item is waiting in any queue or buffer. The cost per item is a fixed cost. In this case, it costs \$0.15 an hour (Waiting cost per hour) any time the crate waits for processing (such as in the receiving area) and there is a one-time docking fee of \$3.00 (Cost per item) for every shipment that is received.

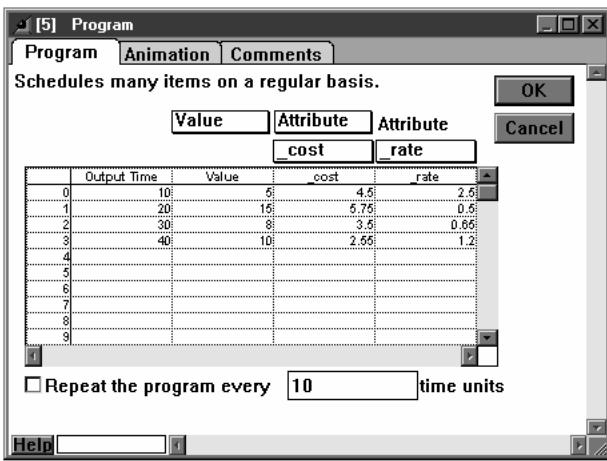
Cost accumulators may also be introduced into a model using certain resource-type blocks (Resource, Bin, and Stock blocks). To do this, you must specify that the items originating in the

block are to be used as cost accumulators. This is done in the Cost tab of the resource-type block, an example of which is shown below:



Bin block providing cost accumulators (Windows)

To define cost rates for cost accumulators that are generated using the Program block, you must explicitly set the “_cost” and “_rate” system attributes (discussed in “Working with cost data” on page M185) for each cost accumulator. 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:



Setting “_cost” and “_rate” in Program block (Windows)

Note The “_rate” attribute, must be defined using the same time unit as the global time unit for the model.

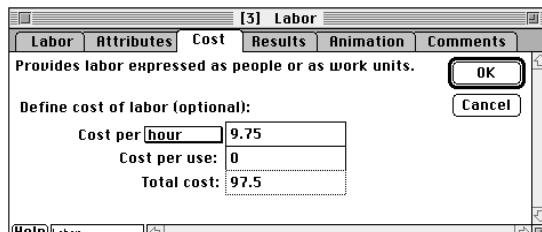
Resource cost rates

Within the Cost tab of resource-type blocks, you can define the cost per time unit and/or the cost per use of the resource. The cost per time unit rate is used to calculate and assign a time-based cost

M182 | Manufacturing Chapter 11: Activity-based costing
Performing activity-based costing

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 on page M178, cost rates for the laborer are defined as below:



Cost tab of Labor block (Macintosh)

MFG

Activity cost rates

You can also define cost information for activities. Within the Cost tab of activity-type blocks, you define a cost per time unit and a cost per use. The costs of using the activity is accumulated by the items that are processed. The cost per time unit is used to calculate the time-based processing cost of each item that passes through the block. The cost per use is a fixed cost added to every item that passes through the block.

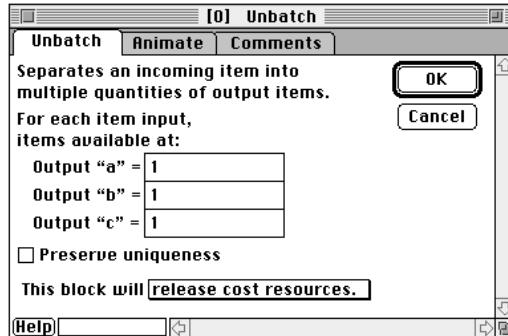
Combining resources with cost accumulators

As discussed in “Modeling resources” on page M162, there are two ways that items can “use” resources. The first method is to batch a resource with the item. While batched, the resource is in use and cannot be used by another item until it is unbatched and returned to the resource-type 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 will discuss how to properly use these two methods when performing ABC.

Batching resources with cost accumulators

When batching a resource with a cost accumulator, the resource’s cost rates are automatically stored with the cost accumulator and used in any subsequent cost calculations.

To unbatch a resource and remove its cost rate information from the cost accumulator, you must specify in the dialog of the Unbatch block to “release cost resources”, as shown below.

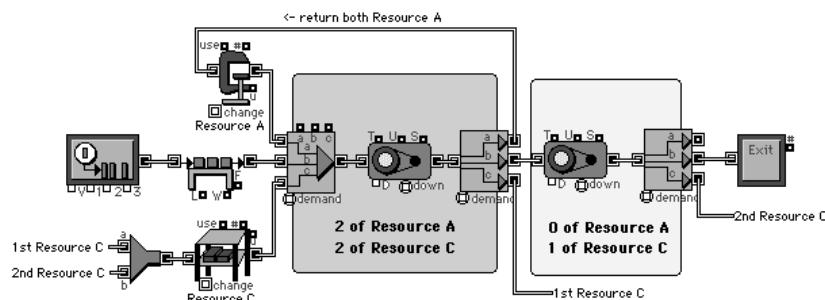


Dialog of Unbatch block (Macintosh)

This choice tells the Unbatch block to modify the information stored with cost accumulator to indicate that the resource has been released.

Note If this popup menu is set to the other choice (“create duplicate items”), the items released by the Unbatch 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.

In the model shown below, 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.



Multiple resources model

(The cost array is briefly described in “Combining resources with cost accumulators” on page M188 and is more thoroughly discussed in the main Extend manual.)

There are three things to remember when batching resource items with cost accumulators:

- The resources will be released from the connector which corresponds to the connector used to originally batch them to the cost accumulator. For example, if the resource entered the Batch block through the “a” connector, it will be released through the “a” connector in the Unbatch block.
- 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 model above, Resource A uses connector “a” and Resource C uses connector “c”.
- When performing ABC, you are limited to having no more than two different types of resource items batched with the 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.

MFG

Cost accumulators and the resource pool blocks

As described in “Using the Resource Pool blocks” on page M164, as cost accumulators pass through a Queue Resource Pool block, resources are allocated to them. When a resource pool unit is allocated to a cost accumulator, its cost rate information is automatically stored with the cost accumulator and used in any subsequent cost calculations.

When a resource is released using the Release Resource Pool block, the information stored with the cost accumulator is modified to indicate that the resource has been released.

The is no limit to the number of different types of resources a cost accumulator can use when using the resource pool blocks, and the two methods of modeling resources (batching and resource pools) may be used in conjunction with each other.

Using the Batch(Variable), Batch(Demand), and Unbatch(Variable) blocks

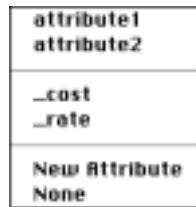
The Batch (Variable) and Batch (Demand) blocks allow you to batch cost accumulators arriving from a single path. These blocks may be used in conjunction with an Unbatch (Variable) block to combine cost accumulators for processing and then to separate them after processing has been completed. If “preserve uniqueness” is selected in both the batch and unbatch blocks, the cost accumulated while the items were batched will be evenly divided among the cost accumulators exiting the Unbatch (Variable) block.

For example, assume you use a Batch (Variable) block to batch three cost accumulators together. While batched, these items accumulate an additional \$9.00 due to processing. If “preserve uniqueness” is selected in both the batch and unbatch blocks, each of the three cost accumulators exiting the Unbatch (Variable) block will have accumulated \$3.00 during the processing.

If “preserve uniqueness” is not selected, the Unbatch (Variable) block will produce three identical items and all the cost accumulated while they were batched (in this case, \$9.00) will be inherited in full by each item. It is unlikely that you would intend to have this happen.

Working with cost data

In order for you to have access to the cost information, Extend creates two attributes (“_cost” and “_rate”) for every item in your model. Since these are automatically created, they are considered *system* attributes, and will appear separately in the attribute popup menus, as shown below:



Attribute popup menu containing system attributes “_cost” and “_rate”

These two attributes will appear in the attribute popup menus only if a cost rate is defined somewhere in model. The information that is stored in these attributes depends on whether the item is a cost accumulator or a resource, as shown in the following table:

Item type	“_cost” attribute	“_rate” attribute
Cost accumulator	The accumulated cost of the item	The waiting cost, or storage cost, of the item (cost per time unit defined using the model’s global time unit)
Resource	The cost per use of the resource	The cost per time unit of the resource (defined using the model’s global time unit)

The attribute handling blocks (Get Attribute, Change Attribute, etc.) can be used to read, set, or manipulate these attributes. In addition, two blocks in the Statistics library can be used to gather the cost data.

Viewing Cost Data

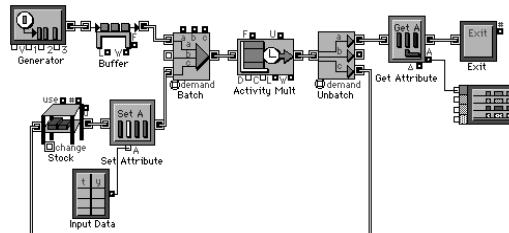
You can use a Get Attribute block to read the “_cost” and “_rate” attributes of any item, then plot the data or use the attribute value to perform additional calculations. For example, you can use a Get Attribute block to read the “_cost” attribute of cost accumulators and connect the “A” connector of the Get Attribute to a Plotter Discrete Event to plot the accumulated cost of each item that passes through the Get Attribute block. This can be seen in the model discussed in the following section.

Changing Cost Data

In most cases, you will simply want to define the cost rates of the various cost drivers in your model and allow Extend to automatically calculate and track costs. However, there may be times when you need to manipulate the cost values generated. The attribute handling blocks can be used to accomplish this.

M186 | Manufacturing Chapter 11: Activity-based costing
Performing activity-based costing

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. You can model this by using the Set Attribute and Input Data blocks to explicitly set the “_rate” attribute of the resource as it exits the resource-type block. This is shown in the model below:

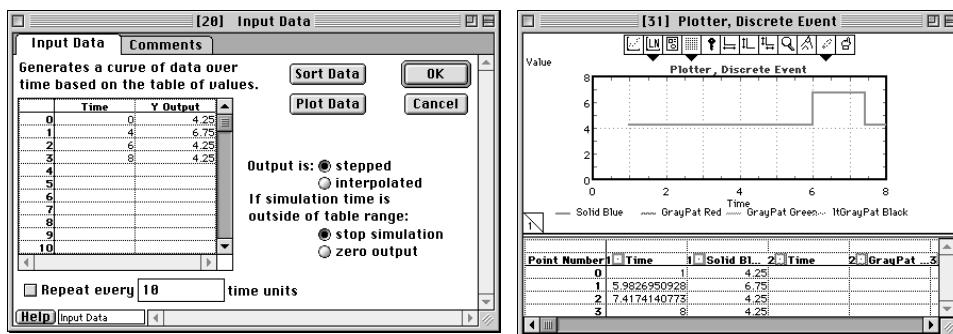


Changing “_rate” attribute of resources

MFG

Within the table of the Input Data block, you specify different rates for different times of the day. Note that a change in the rate will only affect resources as they exit the Stock block. Resources that are currently in use will not be affected unless they are recycled back to the Stock block and through the Set Attribute block.

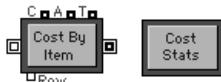
In the above model, the Get Attribute block reads the “_cost” attribute before the items exit the model. The accumulated cost of each cost accumulator is then plotted. In the plot (shown to the right, below), you can see that the cost of the items increases during the period of time that the resources’ cost rates are higher.



Dialog of Input Data and Plotter Discrete Event (Macintosh)

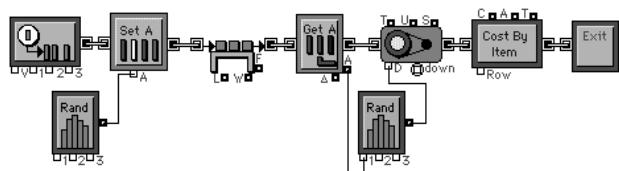
Gathering and Analyzing Cost Data

You use the Cost By Item and the Cost Stats blocks to gather the cost information generated in your models.



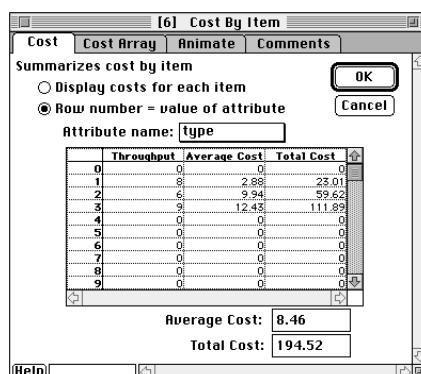
Costing blocks included in the Statistics library

The Cost By Item block is used to read and store the “_cost” attribute of the cost accumulators that pass through it. The block’s dialog will list the accumulated cost of each item, the time the item passed through the block, and the total and average cost of all the items that have passed through the block. This block can also be used to list the cost of the items sorted by type. The Sort By Type model utilizes the Cost By Item block in this manner.



Sort By Type model

In the model, 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. The dialog of the Cost By Item block is shown below:



Dialog of Cost By Item block (Macintosh)

Generator, resource, queue, and activity-type blocks are all capable of generating costs that are tracked with the cost accumulating items. Additionally, each cost-generating block displays the total cost generated by that particular block in its *Total Cost* dialog item. The Cost Stats block is used to collect and display the total cost for each block. You could use this information to determine which blocks are contributing the most to the total cost of the items being processed. See Appendix G: Blocks in the Statistics Library or the help text of the block for a detailed description of how to use the Cost Stats block.

How Extend tracks costs

In the previous section, you learned how to perform ABC in Extend. This section provides a more detailed look at how Extend tracks costs and is included mainly for informational purposes.

MFG

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 M179), Extend 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 “_rate” attribute.

Combining resources with cost accumulators

Whether you use the batching method or the resource pools to model resources, two things happen when a resource is combined with a cost accumulator:

- 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 resources “_cost” attribute. If using the Resource Pools, the resource’s fixed cost comes from the *Cost per use* dialog item from the Cost tab of the appropriate Resource Pool block.
- 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 (see the main Extend manual for a detailed discussion of the cost array). Extend 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 the Resource Pools, the resource’s variable cost rate comes from the *Cost per time unit* dialog item from the Cost tab of the appropriate Resource Pool block.

When a resource is released by an Unbatch or Release Resource Pool 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, generator, activity, queue, and resource-type blocks are all capable of generating costs. As the 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 the *Total Cost* dialog item located in its Cost tab. This dialog item displays the total cost contributed by that particular block only. The following sections briefly discuss how these calculations are performed.

In generator-type blocks

When a cost accumulator is generated, Extend will add the fixed cost or *Cost per use* of the generator to the cost accumulator's “`_cost`” attribute.

For each cost accumulator generated, Extend also will add the fixed cost of the generator to that block's *Total cost* dialog item.

In activity-type blocks

When a cost accumulator enters an activity-type block, Extend will add the fixed cost or *Cost per use* of the activity to the cost accumulator's “`_cost`” attribute. In addition, it will calculate the time-based cost, including the variable cost of the activity as well as the variable cost of any resources currently combined with the cost accumulator, and add it to the “`_cost`” attribute of the cost accumulator.

MFG

For each cost accumulator that passes through the block, Extend 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.

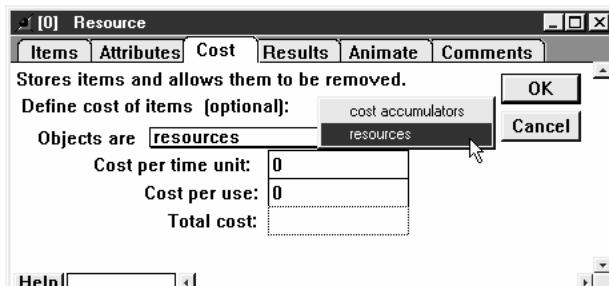
In queue-type blocks

When a cost accumulator enters a queue-type block, Extend will calculate the time-based cost, including the waiting or storage cost of the cost accumulator (calculated from the cost accumulator's “`_rate`” attribute) as well as 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 the block, Extend 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

As shown below, some resource-type blocks are capable of providing either cost accumulators or resources.



Cost tab of Resource block (Windows)

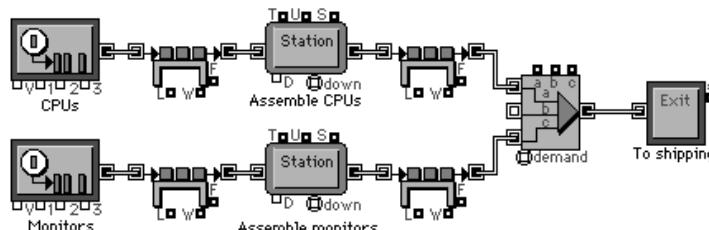
MFG

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 many 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. 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 below:



Multiple cost accumulators

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 resources with cost accumulators” on page M182).



Manufacturing Chapter 12: Statistics and Model Metrics

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 you need for your analysis and the decision-making process. This chapter will provide an overview of important statistical concepts and will discuss ways to extract and document the critical information contained in your models.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Accumulate	Holding	Generic
Change Attribute	Attributes	Discrete Event
Clear Statistics	Statistics	Statistics
Downtime(Unscheduled)	Generators	Manufacturing
Generator	Generators	Discrete Event
Get Attribute	Attributes	Discrete Event
Information	Information	Discrete Event
Input Random Number	Inputs/Outputs	Generic
Readout	Inputs/Outputs	Generic
Timer	Information	Discrete Event

Unless otherwise specified, the models discussed in this chapter can be found in the “Examples \\ Manufacturing \\ Accumulating Data” folder.

Statistical concepts

Although simulation is a quantified subject, there is still room for quick 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, and the sensitivity of the system to the input data.

The information in this section is intended as an overview and guide. For more information, refer to the statistics books listed in “Further reading” on page M7.

Constant values versus random variables

Constant values never change; random variables 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.

- Since all inputs are constant, the results of running a *deterministic* model are “determined” or set based on the input variables. For example, if your model uses constants to represent the time between arrivals of items and how long the operations take, each time you run the model the results will be identical. This is a good technique for early steps in the model-building process, since you can be assured that changes in results will be due to changes you make to the model and not to randomness.

MFG

Notice that each run of a deterministic model will give you the same exact measure of *model* performance but is unlikely to give you an exact measure of *system* or *process* performance.

This is because real-world systems typically contain some element of randomness. Deterministic models do not show the effect of variability, which may be an important aspect of your system or process.

- Adding randomness to a deterministic model changes it to a *stochastic* or *Monte Carlo* model. How long a process takes, equipment reliability, and the time between arrivals of customers are all examples of numbers that are typically random. 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 (discussed below). This means that their range and possibility of values is predictable.

Extend provides several methods for including randomness in your models. As you saw in the Tutorial chapter, and as discussed in the section “Random numbers and probability distributions” below, the Input Random Number, the Downtime (Unscheduled) and the Generator blocks allow you to select a random distribution or enter a table of values which specifies an empirical distribution of probabilities.

Random numbers and probability distributions

As mentioned earlier, the ability to include randomness and show dynamic aspects through time is one of the most valuable characteristics of a simulation experiment. Since most models have randomness, it is important that you understand random numbers and are able to choose appropriate distributions.

Random numbers and seeds

A *random number stream* is a sequence of random numbers; the numbers in the stream are derived based on *seed* values. The *random number generator* is the internal mechanism in Extend which cal-

culates the numbers in the stream. Extend provides independent random number streams and the ability to specify a seed so that sequences of random numbers can be repeated.

The blocks that generate random arrivals or numbers are the Generator, Downtime (Unscheduled), and Input Random Number. A random number is generated based on a seed. Each time a new random number is generated, a new seed is generated. Extend automatically assigns a separate seed to each random number-type block in your model. If you want to specify your own seed, you can enter a seed value in the dialog of those blocks. Any number entered as a seed in a block dialog will result in independent random number streams. 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 Extend is independent.

You can also specify a seed for the model as a whole in the Simulation Setup dialog in the Run menu. If you enter a 0 for the random seed or leave it blank in the Simulation Setup dialog, each random number-type block in the model will use a random seed. If you specify any other number for the random seed, it will cause those blocks to use the same sequence of random numbers every time the model is run. This gives repeatable results, allowing you to determine exactly how your changes affect the model.

MFG

Random 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 *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. When you gather data for a simulation model, it is seldom in a useful form. Stochastic models use distributions as a handy method for converting data into a useful form and inputting it into models. Extend'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. By “filling in the gaps”, they 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.

The most important characteristics of a distribution are its *shape*, the *spread*, and the *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).

In Extend, you usually choose a distribution by selecting it from the popup menu in the dialog of a random number-type block, for example by selecting the normal distribution in the Generator block. The functions that produce the distributions have one or more parameter arguments which

define and control the characteristics (spread, maximum, etc.) of the distribution. In Extend block dialogs, these arguments are usually identified by the labels “(1)”, “(2)”, “(3)”, and “Location”. You specify the characteristics for the selected distribution by the values you enter for (1), (2), or (3).

Using random numbers means choosing the theoretical distribution that best describes the variability of the raw data, describing the data using an *empirical* or user-defined distribution (such as the empirical distribution in the Generator block), or fitting known data to a distribution. As seen below, there are several distributions you can choose in Extend.

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 you find that your 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. As you can see in the Help of the Input Random Number block, there are also software applications which fit data to distributions.

For cases in which you have empirical data which you want to model using random distributions, Extend offers interfaces to a number of distribution fitting packages, such as Stat::Fit from Geer Mountain Software (bundled with Extend Suite), and ExpertFit from Averill Law & Associates. These Companion Products can help you find the statistical distribution that best emulates your real-world data. The Input Random Number block has a Distribution Fitting tab from which you can launch a distribution fitting package, analyze empirical data, and determine the appropriate statistical distribution for a given data set. See “StatFit distribution fitting example” on page M199 for an example of how to use distribution fitting.

Contact Imagine That, Inc. for additional information on distribution fitting packages and other Companion Products.

Distribution guide

When you have sparse or no data, this guide of common uses may help you select a plausible distribution in the Generator, Downtime (Unscheduled), or Input Random Number blocks:

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 and are of a particular type.
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.
Erlang	Frequently used for queueing theory to represent service times for various activities or when modeling telephone traffic.
Exponential	The most common choice in service industries and business processes. 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.
Gamma	Typically used to represent the time required to complete some task. The distribution is shaped like a decaying exponential for <i>shape</i> (2) values between 0 and 1. For shape values greater than one, 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 ((1) prob) 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.
LogLogistic	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.

Distribution	Definition
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.
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.
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 or real)	Describes a 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.

The block descriptions in the appendices of the main Extend manual fully describe the distributions found in the Generator and Input Random Number blocks while Appendix A in this manual describes the distributions found in the Downtime (Unscheduled) block. Extend's distributions are also described in the Help of those three blocks.

For more information about choosing and using distributions, the books *Discrete-Event System Simulation*, *Improve Quality & Productivity with Simulation*, and *Simulation Modeling & Analysis* are excellent resources; they are listed in “Further reading” on page M7.

Distribution fitting

For cases in which you have empirical data which you want to model using random distributions, Extend offers interfaces to a number of distribution fitting packages. These Companion Products can help you find the statistical distribution that best emulates your real-world data. The Input Random Number block (Inputs/Outputs submenu of the Generic library) has a Distribution Fitting tab from which you can launch a distribution fitting package, analyze empirical data, and determine the appropriate statistical distribution for a given data set. See the StatFit example below. Also see the ReadMe file for additional information on distribution fitting packages and other Companion Products.

MFG

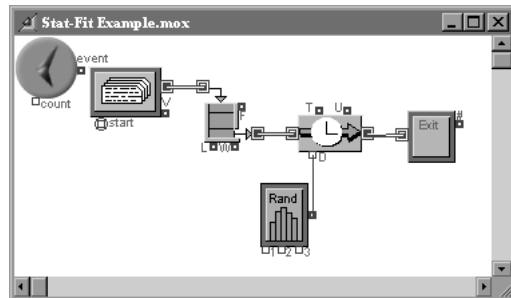
StatFit distribution fitting example

In simulation models, it is often useful to characterize a random input with a probability distribution (e.g., inter-arrival times and demand rates). Typically, this requires historical data documenting the system’s behavior and some analysis to find an appropriate distribution. There are two advantages to using statistical distributions rather than historical data as inputs to a model:

- Values for an input random variables are not limited to what has happened historically.
- For continuous distributions, an infinite pool of data exists. There is seldom enough collected data to support multiple simulation runs.

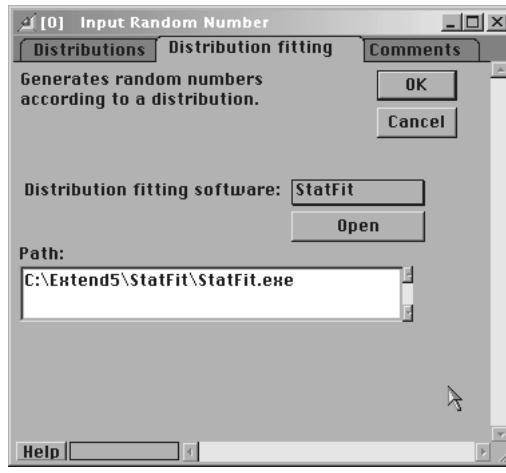
Stat::Fit is a software package from Geer Mountain Software (www.geerms.com) that helps the analyst determine which distributions if any offer a good fit for the underlying random process. An interface has been developed which allows the modeler to easily access the power of Stat Fit from within Extend.

In the simple example below, the distribution used by the Input Random Number block will be chosen using Stat::Fit. The *Statfit Example* model can be found in the “Examples \ Suite” folder.



StatFit Example

- From the Input Random Number block's dialog, go to the Distribution Fitting Tab.
- Under "Distribution fitting software:" select Stat Fit and click the open button.



- Input Random Number dialog showing distribution fitting tab
- The StatFit application should appear on your screen.
- Go to Stat Fit's File menu and select open.
- Choose the file named StatFtEx.sfp.

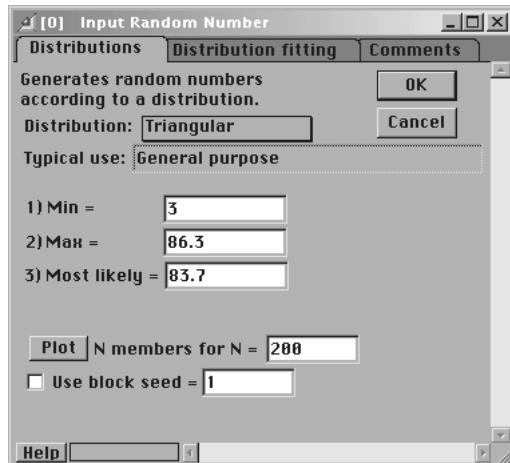
The 32 historical data points appearing in this project will be used by Stat Fit to define which distributions and associated parameters might be appropriate for the Input Random Number block.

- Go to the Fit menu, select autofit and click OK.

A window appears displaying a list of parameterized distributions that have been ranked according to goodness of fit.

- Next, click the Export button.
- Pick Extend under the application window, and then choose the desired distribution.
- Click OK and go back to Extend.

The Input Random Number block's dialog should reflect the choice you made in Stat Fit.



Dialog modified by StatFit results

MFG

Additional StatFit documentation is available in StatFit's Help menu.

Changing parameters dynamically

Another name for the variable or constant number you input in a model 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*, such as based on some model condition. For instance, you might want to show that a clerk who takes 10 minutes to process an order in the morning takes 11 minutes to process an order at midday and 12 minutes later in the afternoon. The most common method for doing this is to use another block which controls how the parameter changes. You can change both constant and random parameters dynamically:

- You can dynamically change a parameter by having it change to one of a series of constants, for example changing the parameter depending on the time of day for the clerk mentioned above. This allows you to predict exactly what the value will be at any point in time. You would commonly use the Input Data block (from the Generic library) to do this, as described in “Scheduled processing time” on page M128.
- You can also cause a random number to change dynamically by changing one of the distribution's arguments. For example, rather than set the mean of an exponential distribution in the Generator block's dialog, you could connect an Input Data block (from the Generic library) to the “1” input of the Generator block, and vary the mean dynamically. You will see how to do this in the section “Generating arrivals with custom intervals” on page M83.

You can also use Extend's control blocks to interactively manipulate values. For example, the Slider control (accessed using the Control command of the Model menu) can be used in the same manner as the Input Data block, discussed above.

Confidence intervals

Confidence interval estimation tells you, with a given level of probability or confidence, how close the average simulation results you obtain are to the model's true (theoretical) 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 $x \pm y$, 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.

The blocks in the Statistics library not only summarize and report statistical information about the model, but calculate confidence intervals given various levels of confidence. For example, the Activity Stats block allows you to select a confidence level and reports confidence intervals for arrivals, departures, and utilization. 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.

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 very important to ensure that the numbers you enter accurately represent the details of the actual system. The significance and relevance of your analysis will depend on how closely your model's inputs correspond with real world data (the phrase "garbage-in/garbage-out" is very appropriate).

If you want more information about confidence intervals, two excellent sources listed in "Further reading" on page M7 are *Simulation Modeling & Analysis* and *Improve Quality & Productivity With Simulation*.

Note If you set a seed in the Simulation Setup dialog, by default each simulation run will produce the same result. If you set seeds in the blocks, you'll get repeatable sequences of random numbers. This invalidates the purpose of using confidence intervals. A way to mitigate this is to use the Random Seed Control block (in the Statistics library). This block controls whether or not the random number seed is reset at the beginning of each simulation run.

Multiple runs and Monte Carlo simulations

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 for Monte Carlo simulations or when performing sensitivity analysis.

- *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 samples. An example of a static model is a spreadsheet which contains sales forecast information; an example of a dynamic

model is a simulation of a bank line which changes over time. Applying Monte Carlo simulation techniques to a dynamic model is also known as *stochastic* modeling, which is discussed in the section titled “Constant values versus random variables” on page M194. You can build both static and dynamic models with Extend.

- 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 the main Extend manual.

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 M76) to determine how many additional simulation runs are required to obtain an accurate estimate of the model’s performance. Some tools in Extend for evaluating multiple simulation runs are:

- Histograms (in the Plotter library) which show the shape and any tendencies in the range of results.
- Error Bar plotters (from the Plotter library) that show how a variable is affected by randomness through the course of the simulation run.
- MultiSim plotters (also in the Plotter library) that can retain data and show the results of up to four simulation runs.
- Confidence intervals (reported by some of the blocks in the Statistics Library) which show the probability that a certain range captures the true mean for a simulation model result.

MFG

Measuring and verifying simulation results

So far, the examples in this manual have focused on processing, routing, and tracking the flow of items through the model. As discussed in “Model verification” and “Model validation” on page M77 you will often need to gather and document other information such as operation utilization, resource use, queue length, or throughput. Extend makes this easy.

Measuring performance and debugging models

Extend facilitates model analysis and provides several methods for reporting simulation results. See the main Extend manual for more details about many of the following techniques:

- Dialog boxes display data pertinent to the specific block and in some cases automatically perform statistical calculations. For instance, the dialog of the Buffer block 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 “Using Notebooks for displaying simulation results” on page M211.

- 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 blocks such as machines and activities outputs utilization values. You can attach any value output to a Plotter Discrete Event block to plot information about model performance, as discussed below. You can also attach value outputs to value inputs on diagnostic-type blocks, such as to the ReadOut block from the Generic library, to display information about that output.
- Plotter blocks, from the Plotter library, conveniently display graphs and tables of data over time. Plotters 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.
- Animation shows the flow of items in a model, levels of values, etc. Extend 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 your model since it can show you 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.
- There are numerous blocks that can be used for debugging your models and verifying results. For example the Stop block from the Generic library stops the simulation and notifies you when its input goes above or below a specified level. The Status block in the Discrete Event 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). The Information block (also from the Discrete Event library) is extremely useful for gathering data and debugging models, as shown in the section “Getting information about items” on page M205. The main Extend manual has an entire section devoted to blocks that are used to debug models and report information.
- 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.
- Running simulations multiple times, such as for Monte Carlo simulations, gives ranges of values indicating the possible outcomes for the model.
- The Report command, discussed in the main Extend manual, instructs Extend to generate a text file of the final model results. You can report on all the blocks in a model, or use menu commands to specify which blocks are included in the report. Reports are especially useful for outputting to other applications, such as statistics packages, for further analysis.
- Some of the blocks in the Statistics library report and statistically evaluate results. For example, the Queue Stats block displays information about every queue-type block in the model and calculates the confidence intervals based on the results. As discussed in “Clearing statistics”, below,

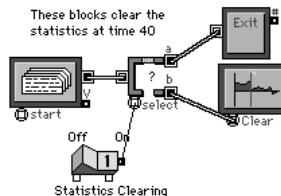
the Clear Statistics block resets statistical accumulators at random intervals or in response to a system event; this is used to eliminate statistical bias during the warm-up period.

- The utilities library contains two blocks which 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 main Extend manual for a detailed discussion on the discrete event messaging system.

Clearing statistics

As discussed in “Non-terminating systems” on page M75, when you start 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. 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 in the Statistics library is used to reset statistical accumulators for the blocks specified in its dialog, eliminating the statistical bias of the warm-up period. In the example below, statistics are cleared after 40 time units:



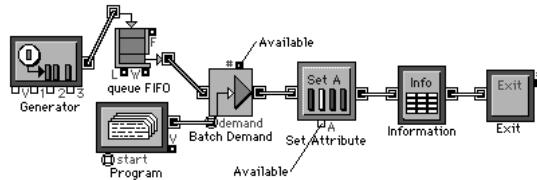
Statistics clearing example

Notice that the Switch control allows you to turn the clearing feature on and off manually, without having to change the model. The above model can be found in the “Examples \ Statistics” folder.

Getting information about items

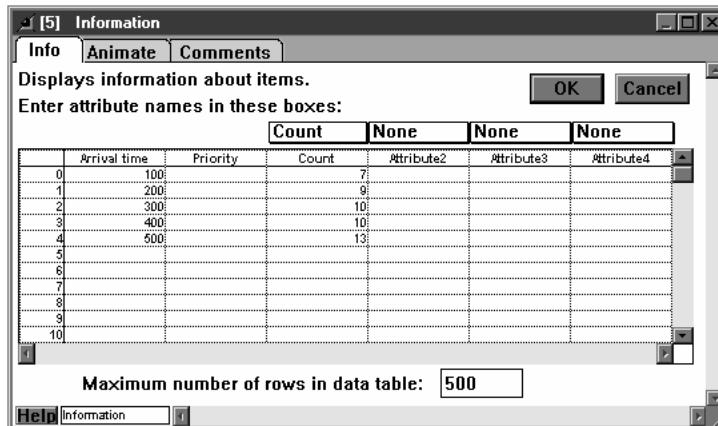
When you are building a model, it is important to start small, verify that the section you have built is working as expected, then enhance that model section. The Information block is particularly useful for verifying model data because it provides important information about each item as the simulation runs. To use the Information block, connect its input to the block of interest and

connect its output to the rest of the model, as shown. The model below can be found in the “Examples \ Manufacturing \ Accumulating Data” folder.



Verifying information using the Information block

The Information block pulls in an item, gets information, then passes the item to its output with no delay. In the Information block dialog, enter the attribute names for any attributes you wish to track, then run the model. The dialog reports the time the item arrived in the block, its priority, and the attribute values for up to four named attributes:



Gathering information about items (Windows)

Since the Information block can use a lot of memory, you should put it in while you are testing, then remove it when you have verified that the section is working as expected.

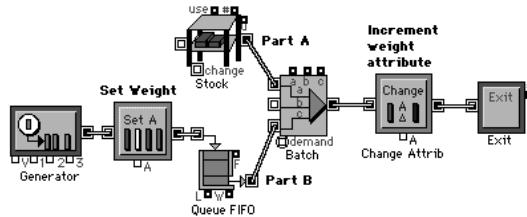
Accumulating data using attributes

Attributes can be used to hold cumulative values, such as the total time that an item has been processed or the total weight of an assembly. When the item reaches the end of the model or model section, you can easily access the attribute value. The models discussed in this section can be found in the “Examples \ Manufacturing \ Accumulating Data” folder.

Accumulating non-processing data

You can accumulate data at any step in the model, even when the item is not being processed. For example, assume you have a system where subassemblies are combined, but the final products

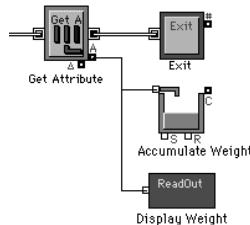
have different mixes of subassemblies. You can have an attribute called “Weight” with a value that gets added each time a subassembly is added. You use a Change Attribute block to increment the “Weight” attribute after the subassemblies are batched:



Non-processing: Accumulating weight data

In the dialog of the Change Attribute block, set the amount of weight you want to add and indicate that this should be added to the value of the “Weight” attribute.

You can monitor these values during the process. For example, at the loading dock, you may want to display the weight of the current item as it leaves, as well as keep track of the total weight of the items. To do this, insert a Get Attribute block before the Exit. Use a ReadOut block to display the current weight and an Accumulate block to keep a running total:

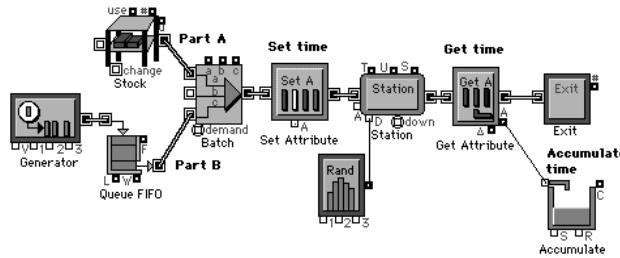


Viewing the weight attribute

Accumulating processing data

If the data you want to accumulate is dependent on processing, you can accumulate values using the attribute manipulating features of the Machine (Attributes) block or Station (Attributes) block. You do this, for instance, when you want to accumulate figures that consider the total amount of processing time each part requires. For example, you can attach a “Time” attribute with an initial value of “0” to a part. Then choose in the dialog of each processing block to modify the Time attribute. You do this by adding the processing time at each process. When the item is

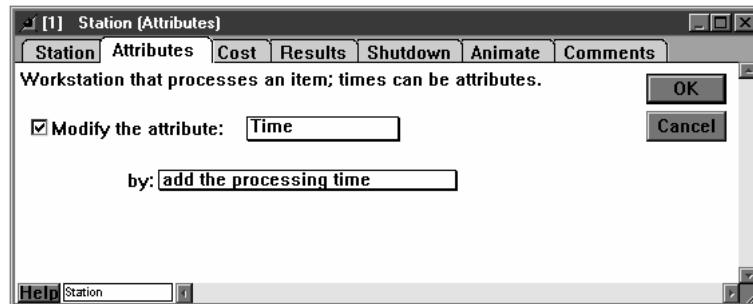
finished processing, you read the attribute value with a Get Attribute block to determine the total processing time of the item. The model is:



Processing: Accumulating processing time data

MFG

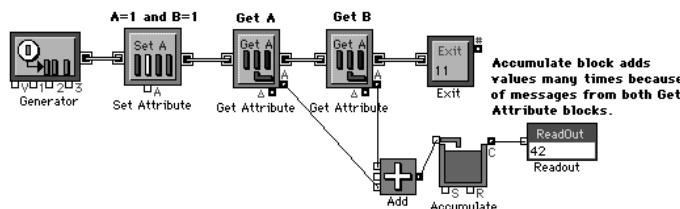
The processing time of the Station (Attributes) block can be set using any of the methods discussed earlier. Be sure to specify in the block that you want to modify the “Time” attribute by “adding the processing time”:



Accumulating processing time (Windows)

Avoiding a conceptual error

It is important that you do not make the error of assuming that you can combine attribute values and then accumulate them, as in the following example:



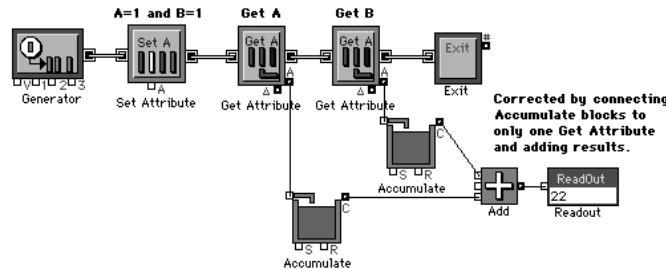
Attributes Error model: Modeling error regarding accumulating attribute values

The problem with this model is clear if you look at the numbers displayed on the Readout and Exit blocks. Since the values of attribute A and attribute B are both 1, the accumulated total dis-

played on the Readout block (from the Generic library) should only be twice the value displayed in the Exit block; clearly this is not the case in the model above.

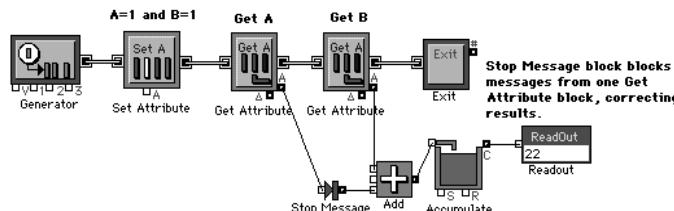
The reason for this has to do with the message passing system in discrete event models. Individual items travel through the Get Attribute blocks sequentially. As an item passes through the first Get attribute block, that block will send a message and the value of the attribute to the Add block (from the Math submenu of the Generic Library). At that point the Add block will recalculate, and send a message and the value to the Accumulate block. When the item moves to the second Get Attribute block, it will send a message to the Add block again. This means the Accumulate block will be getting 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 an Accumulate block (either directly, or indirectly as shown above).

The best solution for this is to have a separate accumulator for each of the Get Attribute blocks. This would cause each Accumulate block to get only one message and value for each item passing through the Get Attribute blocks. The model with this solution is:



Attributes Error model: Solution # 1

An alternate solution for this kind of problem involves using the Stop Message block (from the Utilities library.) This block is designed to solve problems of this nature; it stops messages from being passed through a value connection. As you can see in the model below, connecting the Stop Message block between the first Get Attribute block and the Add block allows the value to be passed but prevents the extra message from being passed. This causes the correct results to occur.



Attributes Error model: Solution # 2

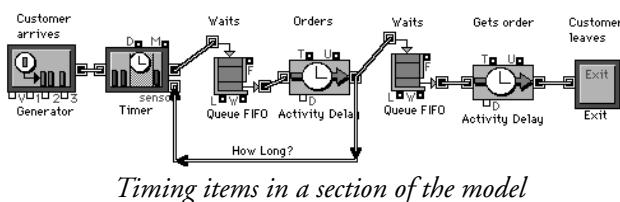
Timing the flow of items in a portion of the model

In addition to performance information that is directly available in a model, you may want to time how long it takes an item to go from one part of the model to another. For example, you may want to determine 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. There are two methods for timing items, as discussed below. The models discussed in this section can be found in the “Examples \ Manufacturing \ Timing Items” folder.

Using the Timer block

The Timer block displays the time that it takes an item to pass between two parts of the model. To use this block in a model, place it at the point where you want to start timing. Connect its regular item output to the rest of the model, and connect its *sensor* input to the output connector at your chosen end point. As the model runs, the Timer places a time tag on each item that passes through it, so the time it travels can be tracked.

For example, assume you want to track how long it takes customers to wait in line and place an order. You can connect a Timer block in the model as shown:



Timing items in a section of the model

The Timer dialog displays the time each item leaves. It also displays the delay time, that is, the time it takes for the item to travel to the end point once it leaves the Timer block. You can plot the delay time by connecting from the “D” output connector to a Plotter Discrete Event; you plot the average delay by connecting from the “M” output.

If you want the delay time to also include the time items wait to be pulled into the next operation, you should follow the Timer block with a Queue FIFO block from the Queues submenu of the Discrete Event library (as was done above). Then, the item will be held in the queue and the wait time counted. You can see this extra time in the following dialog:

	Depart Time	Delay Time	Attrib Value
0	0	0	2
1	1.8209911765	2.1730009234	
2	2.2293137568	3.7706862431	
3	2.4234109177	5.5765890822	
4	2.4598802923	7.5401197076	
5	3.8239622481	8.1760377518	

Timer dialog showing delay plus wait time (Macintosh)

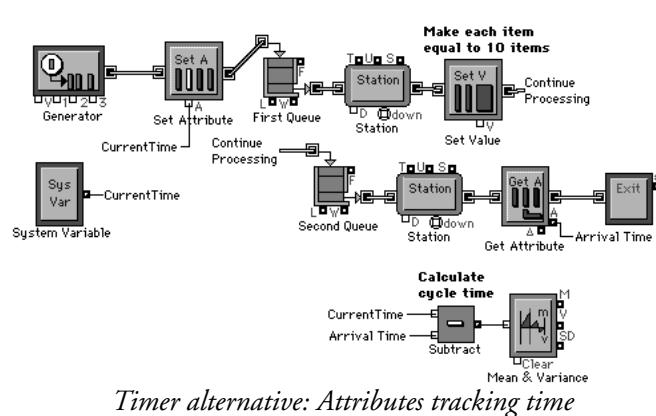
In the model, the ordering process is set to take 2 time units and customers arrive exponentially with a mean of 1. Therefore customers arrive more frequently than orders are processed. New cus-

tomers come into the waiting line but cannot place an order because the customer in front is still being waited on. This causes a delay longer than the time it takes to place an order.

If a name is given for an attribute in the Timer block, then only items with that attribute are measured. Note that the *sensor* connector does not pull items, it only views them. The customers are not removed from the simulation by being “sensed”, they continue on to the next line where they wait to pickup their order.

Using attributes to record times

As an alternative to using a Timer block, you can use an attribute to hold the time that an item arrives in a section of the model. When the item leaves that section, subtract that attribute value from the current time. This is illustrated by the following model:



As each item is generated, it gets an attribute with its value set to the current time. The System Variable block (from the Information submenu of the Generic library) supplies the current simulation time. At the point where the item is finished processing, the attribute value is subtracted from the current time and the cycle time is tallied using a Mean and Variance block.

One advantage of this approach, compared to using a Timer block, is that items can be copied or routed to different points without affecting the timing information. For example, each item arriving in this model represents a batch of items that undergo processing, first as a batch and then as individual items. It is more convenient to use the “attribute” approach to time the flow, since the arrival time is automatically copied when the items are duplicated in the second queue. This method of using attributes also requires less memory per item since the timer “tags” are not allocated if there are no Timer blocks in the model (see the help of the Executive block for information about how memory is allocated).

Using Notebooks for displaying simulation results

Extend's Notebook feature is particularly valuable in models where there are many identical blocks, such as a model with many machines. As described in the Extend manual, you can clone

dialog items from your model into the Notebook, and you may arrange the clones in any way you like.

For instance, in the model discussed in “Simple routing” on page M111, there are three machines with buffers running in parallel. You may want an easy way to track the time that items are waiting for each machine without using plotters. Simply open the model’s Notebook, clone information from the buffers into it and add labels, such as:

	Ave. length	Ave. wait
Machine 1	0.15593043813	0.33176688964
Machine 2	0.02225385852	0.07417952839
Machine 3	0.06705974731	0.30481703322

Notebook items

MFG

Because the notebook can be very large, you can duplicate the items in a different arrangement in another place in the notebook if you want to view the items differently.



Performance modeling for Business Process Reengineering

Extend+BPR User's Guide

Version 5

For Windows or Macintosh

Imagine That![®]

Imagine That, Inc. • 6830 Via Del Oro, Suite 230 • San Jose, CA 95119 USA
Telephone 408-365-0305 • FAX 408-629-1251
Email: extend@imaginethatinc.com • Web Site: <http://www.imaginethatinc.com>

BPR Preface

Organizations are continuously trying to find ways to get new products to market quicker while increasing customer satisfaction with their existing products and services. In their quest, these organizations are faced with questions such as:

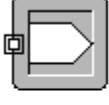
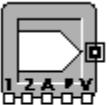
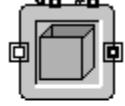
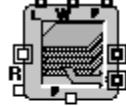
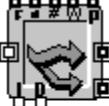
- Should we automate? If we do, when will we see a return on our investment?
- Can we improve the way we do business?
- How much do our processes actually cost?
- How can we reduce time to market?

Considering the rapidly changing global economy and the struggle to remain competitive, the answers to these questions and others like them are more important than ever. One might assume that, given the need to make informed decisions, businesses would be utilizing structured methods to analyze and improve their processes. However, this has not been the typical case.

In the past, there was a tendency to discuss and resolve business problems through Total Quality Management (TQM) and Continuous Improvement (CI); in other words, we talked about them. Total Quality Management and Continuous Improvement are philosophical, and philosophies change constantly. A popular philosophy is Kaizen: it was once Theory Z, and before that, Quality Circles. The answers to the questions asked above cannot be found through traditional methods. If they could, we would have stopped seeing new theories of management long ago.

Extend+BPR was developed to augment the philosophical approaches of TQM and CI by applying systems analysis techniques to process improvement and reengineering efforts. The BPR library is a collection of modeling constructs designed specifically to help business people make informed decisions about the way they do business. Use it to look at business processes from a global view, or use it to get a detailed picture of how processes function. Whether you are in industry or government, using the Extend+BPR package will save you time and money by reducing the risk of counter-productive and non-productive changes while maximizing your reengineering and process improvement efforts.

List of BPR library blocks

				
Decision (2) selects 1 path out of 2 possible paths	Decision (5) selects 1 path out of 5 possible paths	Export removes items from the model	Import provides items for the model	Labor Pool provides workers for the model
				
Measurement measures values in the model	Merge combines streams of items into 1 stream	Operation transforms, batches, and delays items	Operation Reverse unbatches items	Operation Variable transforms, batches, and delays variable numbers of items
				
Repository provides a limited number of items	Stack queues items in specified order	Transaction processes multiple items at a time	Transaction, Preemptive allows items to preempt each other	

BPR

List of Statistics library blocks

			
Activity Stats records statistics on all activities	Clear Statistics clears statistics, used to eliminate warm-up period	Cost By Item records cost statistics on all items that pass through block	Costs Stats records cost statistics on all cost generating blocks
			
Mean & Variance Stats records statistics on all Mean & Variance blocks	Queue Stats records statistics on all queues and buffers	Random Seed Control customizes random streams for multiple runs	Resource Stats records statistics on all resources

BPR Introduction

The goal of every company, government agency, and educational institution should be to develop an extremely strong competitive organization. Cost reduction and quality improvement 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.

Business process reengineering (BPR) is the analysis and redesign of business processes. It requires companies to look at their fundamental processes from a cross-functional perspective and ask “Why?” and “What if?”. Extend+BPR’s graphical interface and dynamic modeling capabilities are designed to help organizations answer questions about how they do work: what they do, why they do it, what it costs, how it can be changed, and what the effects of changes will be.

Whether your industry is health, services, insurance, software, banking/finance, government, manufacturing, or retail/wholesale, Extend+BPR can help reengineer management, legal/regulatory, economic/socioeconomic, and technological systems. Use it for workflow reengineering, policy analysis, product mix evaluation, technology insertion, rightsizing, WIP-reduction, throughput analysis, activity-based costing, and much more.

Background

Most business processes were developed before modern computers and communication systems existed. As the Industrial Age progressed, systems and rules (both documented and undocumented) evolved to deal with the situations encountered. Instead of being designed using a structured, engineered approach, processes mutated.

The current Information Age has not had the expected impact on business operations, since applying technology has typically meant merely automating or speeding up existing processes. The problem is that while processes might have been improved, they were never engineered in the first place. Continuously improving existing processes means that companies are often doing better what should never have been done at all.

The term “business process reengineering” was first coined by Michael Hammer and gained prominence in a Harvard Business Review article he published in 1990. Hammer defined BPR as the “... radical redesign of business processes to achieve dramatic improvements in their perfor-

mance.” When processes are reengineered, they are analyzed and redesigned, rather than just automated.

Business process reengineering

BPR requires the use of tools and techniques, combined with enabling technologies such as automation, to make changes throughout an organization. The objective is to simplify workflow so that the functions within each process are optimized.

Process reengineering goals

Business process reengineering is a structured approach that relies on performance measurement both to determine which processes should be reengineered and to determine if proposed changes will have a productive impact.

The key performance goals, which measure whether processes are optimized, are:

- Decrease cycle time to bring it into alignment with customers' needs
- Decrease costs and increase profits
- Manage throughput based on customer requirements
- Improve efficiency through increased productivity and utilization of resources

BPR

Meeting these performance goals can result in significant changes in a company's financial picture. For example, for a \$1 billion company, a 1% improvement in throughput and resource utilization might result in \$3 million in increased profits, and a 5% reduction in inventory levels could cause a \$10 million decrease in inventory carrying costs.

How processes are changed

To achieve their goals, organizations need to change their existing processes to:

- Eliminate non-essential, non value-adding steps
- Implement and insert technology where appropriate
- Improve workflow to emphasize value-adding functions
- Reduce or eliminate key elements that drive costs
- Provide metrics for meaningful analysis and strategic planning

BPR is a structured, technological methodology, very different from the philosophical approach of Total Quality Management and Continuous Improvement. To achieve world-class status, companies need to use techniques, tools, and technologies – a variation on the phrase “thinking smarter, not harder”.

For example, to decrease time-to-market, some companies are using Extend's engineering systems simulation capabilities to model rather than build engineering prototypes. By going directly from design stage to production, these companies save millions of dollars and get products to market faster than their competition. Organizations that want to reengineer their processes can get that same advantage by using Extend+BPR to dynamically model their processes, rather than going through the expense and delay of creating, testing, and running a pilot project.

Steps in the reengineering process

To reengineer their processes, organizations typically follow these steps:

1. Develop a business vision and objectives: prioritize objectives and evaluate the organization's capacity for change.
2. Understand the existing process: dynamically model and measure the current system to provide a baseline and determine where problems lie.
3. Identify processes to redesign: focus on critical success factors that have the most possibility for change.
4. Identify information technology opportunities: brainstorm and simulate to determine where automation can appropriately be inserted.
5. Design and prototype a new process: redesign and streamline operations so that processes are integrated and workflow is simplified and accelerated. Model the new process to determine its effects on performance measurements.

BPR

Business processes, events, and items

Although businesses have traditionally been discussed in terms of functional units (for example, accounting, manufacturing, or human resources), they are actually composed of processes. Business processes are a series of logically related tasks undertaken to achieve a specified outcome, typically either a product or a service. Each process is a combination of elements (people, procedures, materials, equipment, information, space, and energy) involved in an activity to achieve a goal.

Processes represent how organizations actually work. They are complete cycles such as:

- New product development (concept origination, research, and prototype creation)
- Product flow (purchasing, receiving, and manufacturing)
- Customer acquisition (marketing, customer inquiry, and resolution)
- Product turnover (order entry, delivery, and billing)
- Employee acquisition (hiring, training/promotion, and firing)

In business, processes are often organized around *events*, such as the receipt of an order, a request to purchase equipment, or an idea for a new product. Events occur at random but somewhat predictable intervals. They can be economic or non-economic. Events are what drive the business.

Business processes therefore represent the utilization and interactivity of resources and elements (called *items* in Extend), driven by events. Some examples of business processes, a typical event that might drive them, and the items that flow through or are consumed by the process are:

Process	Event	Item(s)	Activity
Developing a new product	Employee has a new product idea	A prototype	Document the specifications
Providing customer support	Customer calls on telephone	Telephone call	Route call to technical support department
Processing an insurance claim	Claim is received (or accident occurs)	Claim	Review the claim
Planning strategic directions	Plan implementation	Decisions	Planning meetings
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

BPR

Extend+BPR

Extend+BPR is an object-oriented environment for modeling, analyzing, reengineering, and documenting processes. Its iconic building-block paradigm facilitates communication and is designed to allow users to concentrate on the process rather than on any particular methodology.

Using Extend+BPR

Most business systems are composed of real-world elements that interact when specific events occur. Extend+BPR simulates those systems using blocks which mimic business processes and timing that represents the actual occurrence of events. The Extend+BPR blocks directly correspond to the activities, queues, delays, and transformations that comprise business processes. The BPR library also incorporates high-level reengineering concepts such as batching, cycle timing, activity-based costing, and conditional routing.

The goal of Extend+BPR is to de-mystify modeling and simulation and to encourage nontechnical personnel, such as management and the people who do the work, to utilize simulation for the

analysis and redesign of business processes. Therefore, every attempt was made to make the icons used in the library look like the graphical symbols used in flow charts.

Extend+BPR will quickly answer questions about business processes. You use it to state the questions, build a model using familiar flowcharting icons, run the simulation, and analyze the results. Then change aspects of the model and run it again to perform what-if analysis. Extend+BPR is designed to let you model the process as is, then model proposed changes before implementation. This allows you to predict the value, effectiveness, and cost of implementing those changes, and serves as a basis for developing a strategy that will get your organization “from here to there”.

The Extend+BPR libraries

As discussed below, this add-on module contains two libraries, the BPR library and the Statistics library. These libraries are installed into the Libraries folder.

In addition to the BPR and Statistics libraries, your models will use blocks from the Discrete Event, Generic, and Plotter libraries.

The BPR library

Extend's BPR library is an extension of the Discrete Event library that is shipped with the basic Extend package. BPR library blocks directly correspond to the activities, queues, delays, and transformations that comprise business processes. The BPR library also incorporates high-level modeling concepts such as batching, cycle timing, activity-based costing, and conditional routing.

BPR

The Statistics library

The blocks in the Statistics library summarize model statistics, provide increased control over the random numbers, and remove initial bias for statistical analysis. Some Statistics library blocks report statistical information for a particular type of block in the model (activities, mean & variance, queues, resources, or cost generators), others restart statistical calculations or control whether or not the random number seed is reset.

How this manual is organized

Chapter 1 of the Extend+BPR manual shows some example models and typical areas of application. The Tutorial in Chapter 2 shows how BPR and Statistics library blocks are used to build a typical model. Chapter 3 contains a quick overview of the blocks in the BPR and Statistics libraries. Chapters 4 - 12 take detailed looks at the different aspects of modeling with Extend+BPR. The blocks in the BPR library are listed alphabetically and described in detail in Appendix A, while Appendix B does the same for the Statistics library blocks. Appendix C lists other blocks you might use as you build models. And finally, Appendix D has tables giving file names under the Macintosh and Windows systems. Note that the examples discussed in this manual correspond to the model files included with your Extend+BPR package.

Before you proceed with this manual, it is strongly suggested that you read Chapters 1 through 4 in the main Extend manual. Once you are familiar with how Extend works, you will find it easier to understand the examples and concepts in this manual.

Installation and requirements

To install Extend+BPR, follow the Installation Instructions provided with your package. See the Introduction in the Extend manual for system requirements for the Macintosh and Windows platforms. The Extend+BPR package requires additional hard disk space compared to the basic Extend package.

Further reading

Business Process Reengineering: Current Issues and Applications, compiled by the staff of the Institute of Industrial Engineers, Norcross, Industrial Engineering and Management Press, 1993. An excellent overview of BPR, this compilation of articles provides background (including BPR's roots in information technology), case studies, insight, questions, and opinions.

Common Cents: The ABC Performance Breakthrough, Peter B. B. Turney, Oregon, Cost Technology, 1991. An introduction to activity-based costing, demonstrating why it is necessary and how to implement it.

Improve Quality & Productivity with Simulation, Thomas J. Gogg and Jack Robert A. Mott, Palos Verdes, California, JMI Consulting Group, 1995. An excellent "overview of fundamental principles, methodology, procedures, software, mathematics, and benefits associated with simulation modeling." Useful for both simulation users and non-simulation users.

Information Modelling (Practical Guidance), Richard Veryard, Hertfordshire, Prentice Hall International (UK), Ltd. 1992. For practitioners of information systems analysis and design. Shows how to use information modeling as a method of structuring the information needs of an organization.

Measuring Up: Charting Pathways to Manufacturing Excellence, Robert W. Hall, et al., Richard D. Irwin, 1991. Overview of why it is important to measure performance and how it should be accomplished. Applicable to business processes as well as industrial processes.

Quality is Free: The Art of Making Quality Certain, Philip B. Crosby, New York, McGraw-Hill, 1979. One of the originals.

Quality or Else: The Revolution in the World of Business, Lloyd Dobyns and Clare Crawford-Mason, Boston, Houghton Mifflin Company, 1991. A historical perspective on the quality movement with lots of down-to-earth insight into why quality is important.

Reengineering the Corporation: A Manifesto For Business Evolution, Michael Hammer and James Champy, New York, HarperBusiness, 1993. Describes the principles behind a systematic approach to structuring and managing work. Details lessons for business process management and redesign.

Simulation Modeling & Analysis, Averill M. Law and W. David Kelton, New York, McGraw-Hill, 1991. More advanced than the JMI text mentioned previously in this section, this book gives a "...treatment of all the important aspects of a simulation study" ... including modeling, languages, validation, and output data analysis.

Using the Power of Visual Simulation Strategies, Gregory A. Hansen, New Jersey, Prentice-Hall, 1997. Demonstrates how organizations can ascend through process maturity levels using business process reengineering techniques and computer-aided reengineering products.

BPR Chapter 1: Areas of Application

Performance measurement

With Extend+BPR, performance measurement is an integrated part of the model. Unlike process maps or data flow diagrams which only show a pictorial representation, Extend+BPR dynamically models business operations. The metrics obtained by simulating a process provide an objective basis for making strategic improvement and reengineering decisions.

Why measure?

Companies can reap substantial benefits by improving processes. Decreased costs and time for development mean faster time-to-market. Improved quality translates to increased customer satisfaction and repetitive business. Unfortunately, it is estimated that 97% of companies are still functioning at a process maturity level where standard operations are not measured in any meaningful way. Thus, there is no data to measure how effective a process is or to compare it with other processes.

There are three main reasons to measure performance:

- Measurement gives you an objective basis for decision-making
- Systems that get measured are more likely to get improved
- Any measurement is superior to not measuring at all

Measuring operations in a structured manner is the key for companies who want to progress to a higher process maturity level. Typically, industry and government focus solely on productivity and quality measures, a perspective reinforced by practically all TQM and continuous improvement philosophies. However, the number of products produced or the percentage of products considered defective provides only a narrow perspective on the overall dynamics. Other parameters to consider include: time to completion for each task and the overall process, input inventory levels at each step, productivity of workers, and conditions that determine the paths a process follows.

Unless otherwise noted, all of the models and sub-folders discussed in this chapter can be found in the Applications folder within the BPR folder.

Support Services models

Often management's observations of business processes differ from workers' observations. The Support Services models show a method to resolve these differences while avoiding making counter-productive changes. Performance measurements provide objective and verifiable data that is used to reengineer the process.

Overview

A software publisher's support organization provides help to users both on-line (telephone calls) and off-line (written problems submitted by email, mail, or FAX). The manager of the department knows that the department receives only a few telephone calls per hour, and she feels that support personnel could handle an occasional telephone call in addition to performing the off-line support. She also wants to reengineer the process to reduce time-to-completion for both types of problems.

The support personnel feel that they are being overworked. They are also frustrated by being pulled between on-line and off-line support. The three workers in the support department had previously submitted a proposal to management that one person should handle all the telephone calls while the other two would handle off-line support. The request was ignored because management reasoned that if one person was removed to provide on-line support, there would be a 33% reduction in off-line productivity with no increase in on-line productivity.

BPR

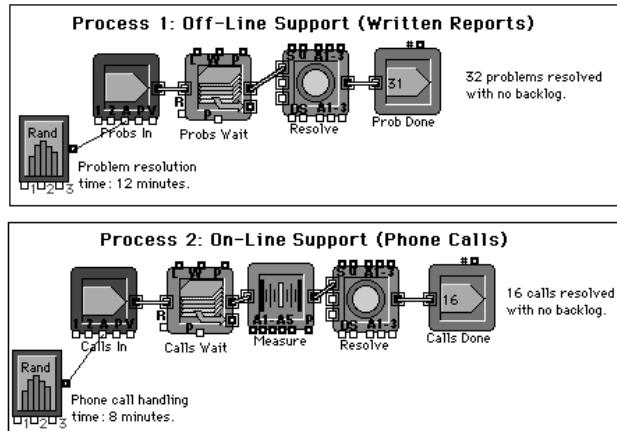
Modeling the process

The facilitator working with the manager collected some data about the support processes and determined that:

- off-line support problems arrive at a rate of about 1 every 5 minutes, and they take an average of 12 minutes to complete
- on-line support calls arrive at a rate of about 1 every 10 minutes, with the majority taking 8 minutes to complete

Based on that information, the facilitator built the first "Support Services" model (in the "Examples \ BPR \ Support Services" folder) to represent the work of one of the three support people. That person was expected to receive 1 off-line problem every 15 minutes and 1 telephone call every 30 minutes, or 1/3 of the problems that the department received. When the model was run for 8 hours (1 day), it showed that the support person completed 31 written problems and 16 tele-

phone calls. Also, as indicated by the results in the two Stack blocks (*Probs Wait* and *Calls Wait*), no work remained uncompleted at the end of the day.

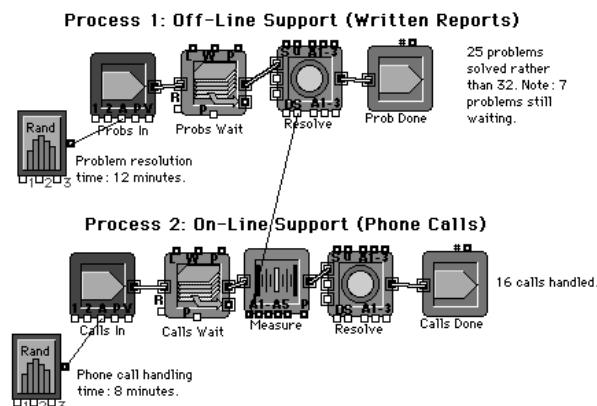


Support Services model - Step 1

BPR

Showing serial processes

Based on this model, the manager felt that her opinion of the work load was justified: support personnel could finish their work in one day and could handle both off-line and on-line support. However, the facilitator pointed out that the model did not fully capture the interaction of the two processes, namely that a person cannot handle off-line and on-line support at the same time. To represent this situation, the facilitator changed the model to show the processes being handled serially instead of in parallel. As seen below (Support Service -2), the changed model gives consideration to the fact that when a phone call occurs, a support person must stop working on the written problem and handle the call. This is modeled by “shutting down” the off-line operation until the on-line operation is complete.



Step 2 of the Support Services model, with shutdown

The on-line process uses an attribute value to specify how long it takes to answer the telephone. That value can also be used to shut down the off-line process for the duration of the call. To do this, the facilitator used the Measurement block to read and report the value of the CallTime attribute. To transfer the call length information, he connected from the A1 output of the Measurement block to the S input of the off-line Operation block (named *Resolve*). He then selected “Use S input as duration” in the Shutdown tab of the Operation dialog to indicate that the operation will shut down for that length of time. The model now shows that each time the support person answers the telephone, the off-line process is interrupted for the amount of time it takes to handle the call.

This second model revealed a quite different result from the first attempt. Instead of resolving 32 off-line problems, the support person completed only 25 problems. Although all telephone calls were handled, at the end of the 8 hour day the worker was left with 7 uncompleted problem reports.

Performance metrics

The modeling process itself provides a structured framework for measuring performance. For instance, the Support Services model makes some assumptions about how often telephone calls and written reports arrive and how long it takes to do the support. When you build your own model, you will gather data, use the actual parameters, and compare model output to the real-world situation.

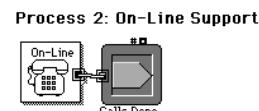
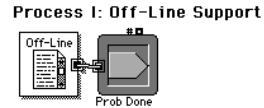
The model also automatically provides additional data that can be used for decision-making. For example, in this model the Stack blocks calculate and display how many telephone calls were received and handled and the average time to process a report. The utilization calculations in the Operation blocks tell how productive the workers are.

When you run this model, notice that the sum of the utilization rates in the two Operation blocks is almost 1. Since 1 means full utilization, this indicates that the worker has hardly any idle time during the day. Also, since the simulation run only represents one day’s work for each support person, by the end of one week each worker could have a backlog of problem reports equivalent to more than one day’s work!

Enhancing the model

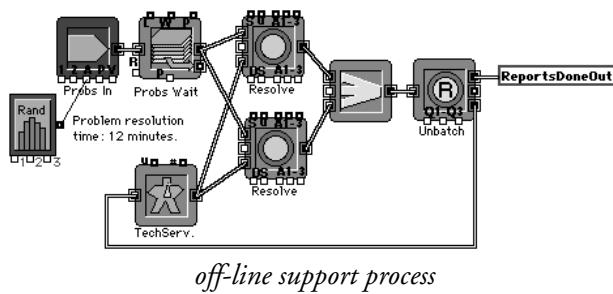
Based on the model information, the facilitator and manager agreed that support department personnel are being fully utilized. To determine if it would be feasible to assign one of the three workers to on-line support and the other two to off-line support, the facilitator revised the model to represent the entire process (Support Service -3). This model, which uses hierarchical blocks,

shows that the workers' recommendation is indeed feasible. The facilitator also recommended that the three workers rotate duties so that each would answer the telephone every third day.



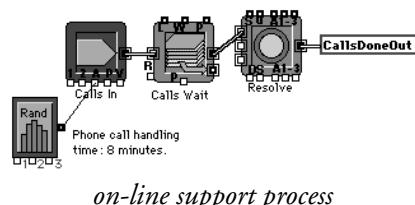
Support Services model- step 3

The hierarchical block for off-line support looks like:



BPR

The hierarchical block for on-line support looks like:



Running the models

The models run for 8 hours. If you run the models with animation on, you can see the activity in the blocks. However, the models will run considerably faster if you turn animation off. Notice that the Export block reports the final number of items, even if animation is not on.

Model highlights

These models illustrate the use of attributes, interrupting operations, and how to specify an empirical distribution. In the models, you see how to set attribute names and values, dynamically set attribute values, and use the values to automatically set processing times. You also see one method for shutting down an operation, based on conditions occurring in another operation. And

finally, the models show how to create a custom distribution based on probabilities using the Input Random Number block's empirical distribution table.

- The length of time for problem resolution, both for telephone calls and for written problems, is set as an attribute with a random value as the item enters the model. The Operation blocks (both called *Resolve*) look at the value of the attribute for each item and process the item for that length of time. As an alternative, if you didn't plan to use the attributes for any other part of the model, you could just connect the Input Random Number block to the *D* input of the Operation block. This would allow you to dynamically change the processing time without using attributes. However in the Support Services - 2 model, notice that the on-line process uses attributes to accomplish two tasks in the model: setting the processing time and serving as the duration of the shutdown time. For that model, using attributes makes a lot of sense.
- In the on-line process, the Measurement block is used to read and report the value of each item's attribute. Since the attribute value is the phone answering time, it is used to shut down the off-line process for the duration of the telephone call. Notice that the model does not use the *A1* output of the on-line Operation block for this, because the *A1* output only reports the new value of an attribute that is being changed by the Operation block. Since you don't want to change the attribute value, but just use it to specify the processing time, you do not name the attribute in the "Set or change..." section of the Attribute tab of the Operation block.

BPR

Cycle time

In order to reengineer or improve business processes, it is important to first model the process "as is" and take measurements of important factors. Once you have measured all important variables, you can redesign the process and see how the new measurements compare to the benchmarks.

One of the most important factors to measure and improve is *cycle time*. Cycle time is a measurement of the time an item remains in a process, either in the entire process or in a specific part of it. Another way to look at cycle time is that it tells how long an item takes to get from point A to point B. For instance, the cycle time for an order in the entire order-turnover process starts when the order is received and ends when the customer pays for the goods. In the computer-entry segment of the process, the cycle time for the order is the time from when the order is received until it is input into the computer.

Cycle time compared to processing time, productivity, and utilization

It is important that you not confuse cycle time with processing time, productivity, or utilization:

- *Processing time* is the time it takes to perform a task or activity; this is also called "task time" or "delay time". For instance, the processing time for an order entry task is the actual time it takes to enter the order into the computer.
- *Productivity* is the ratio of the outputs to the inputs that produce them. For instance, a measurement of productivity could be products produced per employee, dollar revenues per store, cus-

tomers served per payroll dollar, and so forth. In some cases, productivity is based on how many items can be output in a particular segment of time. For example, if 100 orders are entered into the computer in one day of labor, productivity is 100 per labor day.

- *Utilization* is the ratio of the time busy processing compared to the entire amount of time available for processing. *Idle time* is the complement of the utilization amount or 1-utilization. Continuing the preceding example, if it takes 4 minutes to enter each of the 100 orders, the order-entry utilization ratio is 83% based on a 480 minute day. This is calculated as $(4*100)/480$. The idle time percentage would then be 17%.

Note that although a process is composed of one or more tasks, cycle time is not the same as the sum of each task's processing times; in fact, it is almost always greater. In any process or subprocess, cycle time is the sum of:

- the preprocessing times (the sum of the times the item waits until it can be processed by the next task)
- the processing times (the sum of the actual task times)
- the post-processing times (the sum of the times required for the processed item to wait for the next step)

For example, although an order entry clerk can input an order in minutes, the cycle time for each order may be days if orders arrive faster than they are processed or if they are held up in the mail room.

BPR

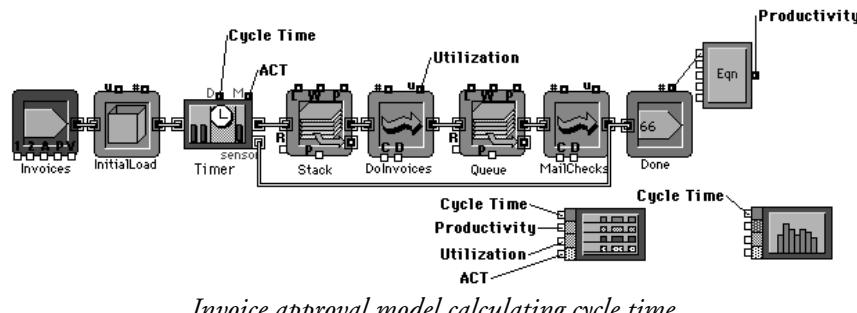
In addition, an improvement in cycle time may not have an impact on productivity. For instance, if workers are already working at full utilization, a decrease in cycle time may not increase the number of items output for the specified time period.

Invoice Approval model

The Invoice Approval model illustrates a two-task process where invoices are approved and checks are generated. The purpose of the model is to calculate cycle time for the process.

- Invoices arrive approximately every 6 minutes and take 7.25 minutes to process
- It takes 15 seconds to print and mail the checks once invoices are approved
- There are 6 invoices awaiting approval when the model starts
- The model runs for one 8 hour day

The model, which uses an Equation block from the Math submenu of the Generic library to calculate productivity per hour, is shown below:



Invoice approval model calculating cycle time

The third block from the left is a Timer block from the Information submenu of the Discrete Event library. The Timer block places a time tag on each item that passes through. Its “sensor” connector notes the time the item gets to an “end point” (the block, in this example named *MailChecks*, whose output is connected back to the sensor input). The Timer block’s dialog displays the time each item leaves and its cycle time, that is, the time it takes for the item to travel to the end point once it leaves the Timer block. In the example above, the end point is the end of the process, after the checks are mailed. You plot the average cycle time by connecting from the *M output* on the Timer block to a Plotter, Discrete Event block (from the Plotter library); you plot the cycle time for each item by connecting from the Timer block’s *D output*. The Histogram block, also from the Plotter library, shows the range of cycle times.

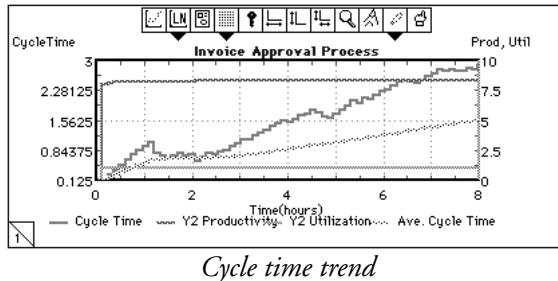
The timing of the cycle starts when the item leaves the Timer block; it does not include any time an item is *blocked* from leaving the Repository block that precedes it. If you want the cycle time to also include the time items wait to be pulled into the first Transaction block (*DoInvoices*), you should follow the Timer block with a Stack block (as was done above) or a Repository block. Then if the item gets blocked in the queue, that time will be counted.

Note that the Timer block’s *sensor* connector does not take items back to the block, it only reads the time tags of the items at the sensing point. Also, if a name is given for an attribute in the Timer block, only items with that attribute name are measured.

The question is, how long is the cycle time? If you guessed, you may have estimated 7 or 8 minutes. If so, you have not considered the cumulative effect of having the processing time be slightly more than the arrival rate.

When you run the model, the plotter shows not only the cycle time for each invoice, but the trend. As seen in this model, the trend is that cycle time continually increases. Although it takes

7.5 minutes to process each invoice, the cycle time increases so that invoices arriving at the end of the day have a cycle time of almost 3 hours:



Cycle time trend

In the graph, the cycle times are plotted on the left y-axis while both the utilization for the approval task and the productivity of the entire process are plotted on the right y-axis. As seen above, by the end of the day average cycle time increases to just over an hour and a half, while productivity is about 8 invoices approved per hour and the utilization is 1, indicating that the approval task is always completely busy.

Model highlights

This model illustrates using the Timer block to measure the cycle time for each invoice. It also shows using the Equation block to calculate productivity based on the number of invoices processed in an hour. Cycle time per item, average cycle time, productivity, and utilization are graphed on the Plotter, Discrete Event from the Plotter library.

BPR

To investigate this model further, try using Sensitivity Analysis to change the amount of time it takes to approve invoices and see what effect that has on the process (Sensitivity Analysis is discussed in the main Extend manual). Or change the maximum number in the “DoInvoices” block to 2 or 3, representing 2 or 3 people processing invoices at the same time.

Technology insertion

As companies bring themselves into alignment with the Information Age, they shift from labor-intensive systems to electronically-enabled systems. This involves the use of computerized technologies such as electronic funds transfer, imaging systems, electronic data interchange, inter-organizational networking, enterprise-wide computing, email, and so on.

Technology insertion, or the substitution of computer technology for manual processes, is a common business process reengineering technique. The promise of technology insertion is that process performance will be enhanced by changing a time-consuming manual process into an instantaneous or nearly instantaneous computer transaction.

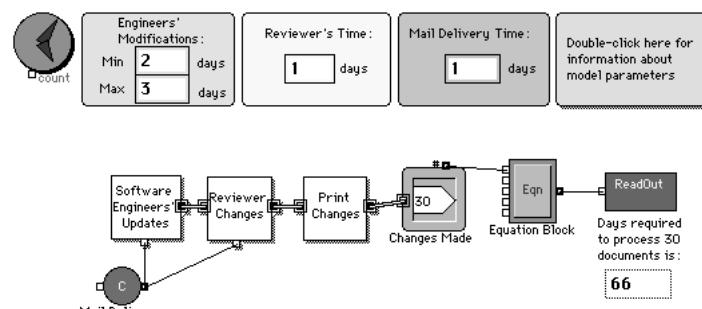
When you consider implementing technology, it is important to realize that any process is part of a larger process, and changes to a portion of a process may not improve the overall performance. Time to completion is regulated by the slowest task in any process, so speeding up a couple of

tasks in a multi-task process may have no effect on the overall process, as seen in the following model.

Documentation Revision model

A major governmental agency has asked you to review the process they use to modify the technical documentation associated with their flight-control software. The agency has determined that the current manual process is too slow and suspects that cycle time can be reduced by applying automation technology to the process.

The Documentation Revision model you prepare for the agency is shown below:



BPR

Notice that this one model represents both the manual and the automated processes (the “as is” and the “to be” scenarios). By simply changing a couple of parameters from the manual process settings, the model will show the effect that automation has on the process as a whole.

The first hierarchical block (counting from the left) represents the software engineers’ modification process. The engineers analyze the changes made to the software and update the associated documentation to reflect those changes.

The second hierarchical block is the reviewer’s task. After review, the modified pages are sent to the printer.

The printer uses a photo-typesetter to print the changes. This task takes 2 days for each document changed. This is shown in the third hierarchical block.

When the model begins, there are 30 software changes that have to be documented. The Equation block on the right is used to calculate the number of days required to review, approve, and print the documentation for all 30 changes.

Manual process

The existing process works as follows:

- For each software change, software engineers manually review and modify the affected pages in the documentation. The 3 software engineers work in parallel, taking from 2-3 days to complete the documentation modifications for each change.
- The packet of “red-lined” pages are passed by inter-office mail to a reviewer who takes 1 day to review, edit, and approve the modified documentation
- The approved changes are forwarded by inter-office mail to the printer who takes 2 days to prepare and print the new documentation
- It takes 1 day to send a package through inter-office mail
- As seen when the model is run, the manual process takes 66 days to document 30 software changes

Automated process

To automate the process, the agency decides to apply information management technology to the first two steps in the process. The installation of an automated document management system will allow the software engineers and the document reviewer to perform their tasks electronically rather than manually. For example:

- The software engineers can automatically find all references to text and graphics, eliminating the need for manual searches
- The engineers can annotate documents electronically and insert both text and graphics in final form
- The reviewer can modify the changes on-line and electronically approve them

BPR

The agency estimates that this application of technology will reduce the software engineers’ task time to 1 day, reduce the review and approval time to a half-day, and eliminate the need for inter-office mail in both steps. When these changes are included in the model, the time for all software documentation changes to finish the first two steps decreases from 34 days to 16, more than a 200% improvement.

Comparing the “as is” to the “to be” process

Based on that information, how much of an improvement would you expect for the entire process? Run the Documentation Revision model with the original settings and then again with the automated process settings (the settings are summarized in the hierarchical block in the upper right-hand corner of the model window) to see if your guess matches the reality.

As you look at the model, consider that the planned application of technology reduced the time to perform tasks early in the process, but did not reduce the time for all the tasks. Since all tasks occurred serially, the printing task dictated the overall timing of the process. In other words, be aware that changes are not made in a vacuum, and when changes are made to pieces of a process,

suboptimization may occur. In this case, the agency should also investigate automating the printing process.

Model highlights

The model uses hierarchical blocks to encapsulate the detail of this somewhat complex model and cloned dialog parameters to provide a user-accessible front end. Having important variables at the top level is handy when your models use hierarchy, saving you from having to drill down into block dialogs each time a change is required. You can copy hierarchical blocks from one part of a model to another or from one model to another. See the main Extend manual for more information about cloning and about creating, using, and saving hierarchical blocks.

A ReadOut block, from the Inputs/Outputs submenu of the Generic library, is used to show the results of the Equation block's calculations. This is sometimes more feasible than plotting the information. In the Readout block's dialog, notice that the "NTicks" choice is set to 0. The NTicks field allows you to set a delay in ticks (60ths of a second) for the block to pause while displaying each number. However, that delay causes the entire simulation to be slowed down. Since you probably don't want to do this, you would usually set the NTicks to 0.

BPR

Notice in the hierarchical block "Software Engineer's Updates" that the same Input Random Number block (from the Inputs/Outputs submenu of the Generic library) is used to specify how long it takes to perform the tasks for all three engineers. Although it is not necessary in this model, it would be more statistically rigorous to use three separate Input Random Number blocks. This would allow each engineer's task time to be generated by an independent random number stream, reducing the chances of statistical correlation.

Workflow

Workflow tools are used to model business processes prior to creating customized applications which will automate them. Before building information systems and databases, it is common to specify types of data or items, show how data and items are routed, and monitor their status.

Routing rules

The inherent discrete-item methodology, coupled with extensive routing capabilities, makes Extend+BPR the ideal tool for workflow modeling. Model items can move sequentially (when one person or operation is done with an item it automatically goes to the next step) or on parallel tracks (items are duplicated and sent on to two or more different but non-contradictory steps). Routing can also be conditional based on:

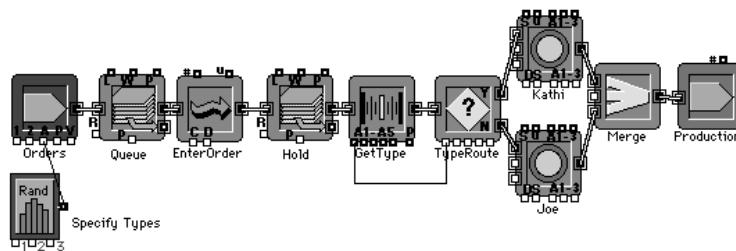
- rules (if the requisition is \$500 or more, route it to A. Otherwise, route it to B)
- events (if A's in-box reaches a specified level, route incoming items to B)
- time (if A hasn't signed the paperwork after 3 days, route it to B)

Order Processing model

The Order Processing model shows the flow of work in an order entry department. Incoming orders are classified by type: either new customer or existing customer. The particulars are:

- Orders arrive about every 8 minutes
- About 20% of the orders are Type 1 (new customer) and the rest are Type 2 (current customer)
- There are 2 order entry clerks working simultaneously. They each take 5 minutes to enter the order into the computer system
- New customer (Type 1) orders are routed to Kathi who takes 20 minutes to review the order, perform a credit check, and add the order to the production schedule
- Current customer (Type 2) orders are routed to Joe who takes 10 minutes to review the order, verify current credit status, and add the order to the production schedule

The model looks like:



Order Processing model

BPR

This model shows both a sequential flow of data and a conditional routing. The routing is accomplished using the Decision (2) block ("TypeRoute"), since there is only one rule with two possible outcomes (if Type 1, route to Kathi, else route to Joe). For more robust routing, for instance where there are multiple rules and outcomes, you would use the Decision (5) block.

Activity-based costing

Extend+BPR can automatically track the accumulated costs of processes and activities, providing management with a valid basis for making decisions about providing services or producing products.

The limitations of traditional cost accounting systems are becoming apparent. World-class manufacturing, TQM, cycle-time reduction, and continuous improvement techniques require that expenditure allocations be realistic and informative. However, current cost accounting systems can provide distorted information to management. Since this information is the basis for strategic decisions regarding capital investment, product pricing, and so forth, inaccurate information directly impacts a company's ability to compete.

Traditional methods

Traditional cost accounting systems assign overhead to products and services based on direct labor hours. However, direct labor is steadily declining as a percentage of product content. Using direct labor as a basis for allocating indirect labor and other overhead expenditures can cause the costs of complex or low-volume products to be understated. The result is that companies produce products which consume resources but provide little or no economic benefit.

ABC method

Activity-based costing (ABC) assigns costs to the final product or service (also called the cost accumulator) based on the use of resources by the processes (or activity centers) that produce the cost accumulator. Resources are the economic elements directed to the performance of the activity, such as workers, parts, or information systems. Resources are consumed or utilized by the activity centers at a rate which is based on variables called activity drivers. These drivers are simply the typical occurrences in any business process, such as the number of insurance claims received, the number of telephone calls in a support center, or how many parts are required for a product. The costs of providing resources to an activity include such items as salaries, office space rents, costs of information systems, outside consulting expenses, and indirect costs such as power and electricity.

BPR

Allocating the cost of resource utilization to a process gives companies the ability to determine the actual cost impact of improving processes. Continuous improvement and reengineering programs can then use ABC as their basis for analysis rather than relying on inaccurate information or faith alone.

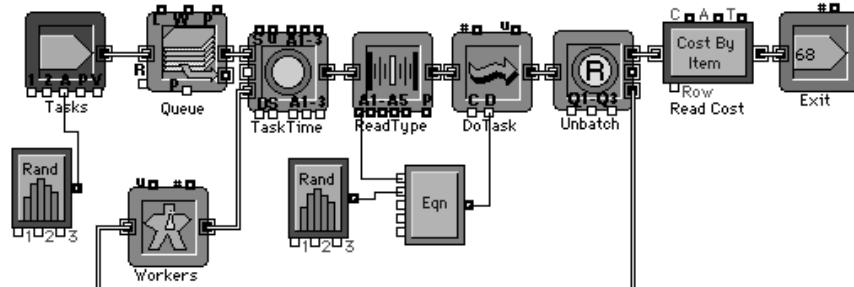
Activity Costing model

Extend+BPR provides the capability to define costs of the resources and activities associated with providing a service or processing an item. As the cost accumulator travels through the model, the cost associated with its processing or use of resources is automatically accumulated and stored as an attribute.

The Activity Costing model calculates the costs of three types of tasks. The particulars are:

- Tasks arrive about every half hour
- The processing times for each task are randomized and are based on task type
- About 30% of the tasks are Type 1 (about 1 hour to process), 50% are Type 2 (about 0.5 hours to process), and the remaining 20% are Type 3 (about 2 hours to process)
- Processing time, which includes computer time and overhead, costs \$50 an hour
- Each task requires one worker; multiple tasks can be performed at the same time
- There are 5 workers available to perform the tasks. Each worker costs \$35 an hour

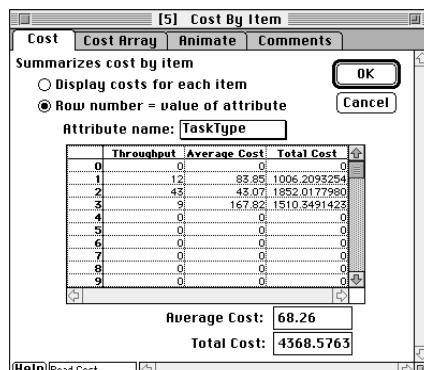
The model is shown here:



Activity Costing model

The Import block is used to set attributes that specify the task type. This allows each type of task to be processed and costed individually. The hourly cost rate of the workers and processing time are defined in the Cost tabs of the Labor Pool and Transaction block respectively. Extend stores the accumulated cost of each item in the “_cost” attribute. The Cost By Item block from the Statistics library reads the “_cost” attribute and displays the costs by task type in a table within the block’s dialog.

BPR



Dialog of Cost By Item block

Notice that Task Type 3 usually has a higher cost than either Type 1 or 2, which appears reasonable since its processing time is longest. See BPR Chapter 11: Activity-based costing for a more detailed discussion on how to perform ABC in Extend.

Implementing strategic plans

Managers and planners need a tool whereby they can interpret their organization's strategic directions, identify tactical areas, and explore alternatives. It is not sufficient merely to specify strategic plans: the plans have to be implemented, the implementation must be consistent with how the

organization works, and the results of the implementation must be weighed against the organization's strategic objectives.

Operational planning

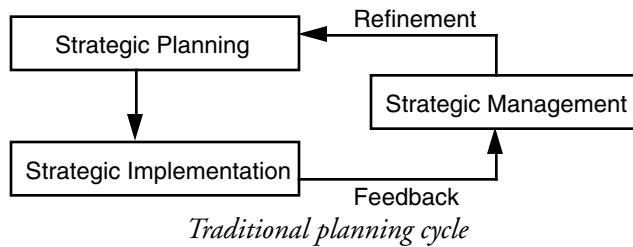
Operational planning is the process of translating top management's focus and strategy into guidelines which are meaningful in the context of the organization's daily operations. For example, operational planning investigates how many people should be added to the sales force this year so that management's goal of increased market share can be realized. Other operational considerations are:

- What should we spend on advertising and marketing?
- How long should this R&D project be funded?
- Where should we build a new plant?
- How do we fund major equipment purchases?

Traditional methods

BPR

Traditional planning methods have not been particularly successful. The continuing jolts of future shocks becoming present reality have made a shambles of many a plan, no matter how coherent and logical these plans were when originally conceived. This is often due to the delays between stages in the traditional planning process, which is shown here:

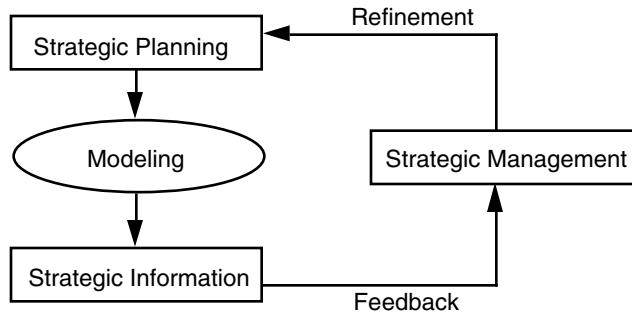


In the traditional method, plans were either implemented directly (faster but dangerous) or implemented in stages through pilot projects (safer but takes longer). Both methods resulted in excessive delays in obtaining feedback and therefore in refining the plan. It was not unusual to have a 3 to 5 year delay from plan creation to plan refinement. Given the current economic and competitive environment, this is totally unacceptable. Furthermore, these delays have caused many managers to abandon formal planning. As a consequence, organizations are often reacting to change rather than anticipating it.

Planning cycle incorporating modeling

In contrast, dynamic modeling allows for the continual evaluation and refinement of how strategic plans are implemented. Instead of implementing the plans directly or in the form of pilot projects, managers use models to aid in strategic thinking and to obtain the information necessary

for plan refinement. This collapses the planning cycle to an acceptable time period and assures that strategic plans are consistent with the operational and tactical workings of the organization.



Planning cycle incorporating dynamic modeling

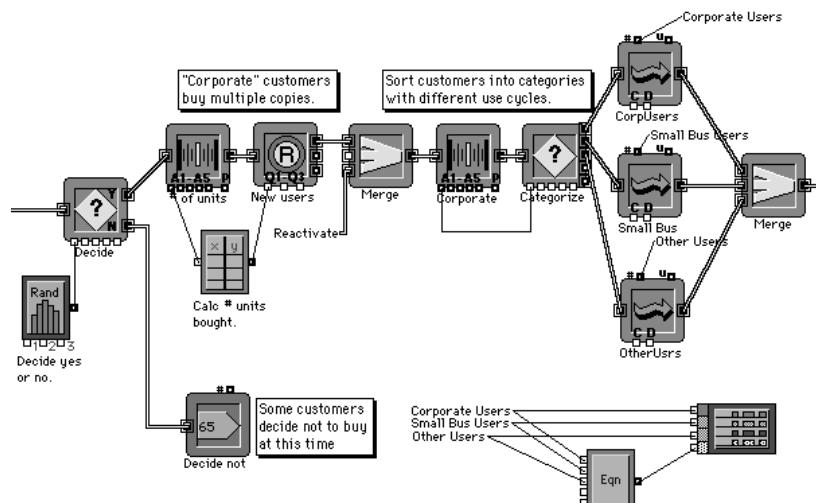
Support Planning model

A software company needs to plan how many customer support representatives to hire over the next two years. In order to forecast hiring requirements, the company first models how it does business. Specifically, the company models how customers evaluate and purchase software and how many users are expected to be active, requiring technical support.

The planning manager determines that there are four types of potential customers, each with their own evaluation, purchasing, and usage patterns:

- Corporate users have a longer evaluation and purchase cycle, but tend to purchase multiple copies after they decide to buy. Because they are in a networked environment, corporate users evangelize products they like, convincing co-workers to make additional purchases. They are usually working against deadlines, are involved in large projects, and require the most technical support.
- Small business users have a shorter evaluation and purchase cycle, tend to purchase only 1 or 2 copies per site, and don't evangelize new users. They need less technical support than corporate users.
- Other users, such as home users, have short evaluation cycles due to their tendency to impulse buying. They purchase single quantities and require the least amount of support because they either experiment on their own or abandon use of the software.
- Finally, there are people who never intend to buy. These are people contacted through direct mail lists who are not interested in the software.

Based on this information, the manager built the Support Planning model, the middle portion of which is shown below:



Portion of the Support Planning model

BPR

The model runs for 2 years and starts with a specified number of contacts per day, where each contact represents 1000 potential customers (this was done to “scale” the model, using less items so that it runs faster). In the first stage, contacts decide whether or not to continue with the evaluation of the software. Each contact has a 20% chance of turning into a sale and an 80% chance of being not interested (just looking). Of those who have decided to evaluate the software, some purchase and some do not. Depending on the type of customer, more than one package can be bought. After a period of time some customers cease using the software, while other customers continue to use the software. Because of their environment and other factors, corporate customers who continue to use the software convince their associates to purchase additional copies.

The model uses attribute names and values to specify customer types: CustomerType 1 (Other), CustomerType 2 (Small Business), CustomerType 3 (Corporate), and CustomerType 4 (Just Looking). The model reads those attributes and calculates data so that each customer type causes specific evaluation, purchasing, and usage results.

In the portion of the model shown above, the customers are sorted by type to determine how much technical support they will require. An Equation block (from the Generic library) is used to calculate the number of support people required based on the installed base of users and their sup-

port requirements. Notice from the plot that while contacts are listed in the thousands, support personnel are not scaled.

ISO9000/EN29000

Extend+BPR is an excellent tool for facilitating many aspects of a company's ISO9000/EN29000 certification process, from documentation of quality procedures to assurance of compliance. (You can also use Extend, and especially its many Add-On modules, for more detailed analyses of specific subprocesses such as engineering product design, calibration, or control systems analysis).

Assessment and certification of a company's quality management system is rapidly becoming a prerequisite for all manufacturing and service industries that trade internationally. The International Organization for Standardization (ISO) and the European Committee for Standardization (ECN) have developed the ISO9000/EN29000 series of quality standards. ISO9000 and EN29000 provide guidelines for implementing a quality management system that will ensure customer satisfaction with the goods and services companies provide. These standards are the benchmarks for quality in any organization.

Certification process

The ISO9000/EN29000 certification process is rigorous and on-going. The operation of the company's quality assurance process must be documented in a "Quality Manual" which addresses what must be done, who is to do it, when it is to be done, and how it is to be done. Then the Quality Management System is assessed for compliance with the ISO9000/EN29000 model (9001, 9002, or 9003) appropriate to the company's scope of operations. After certification, companies must continuously assess and assure compliance through internal and external audits that verify that the actual activities comply with planned arrangements, determine the effectiveness of the quality management system, and provide objective data for review by management and the auditors.

BPR

Extend+BPR provides a rigorous and objective methodology for assessing operations, documenting the quality manual, and demonstrating compliance. You can use Extend+BPR to model and document quality assurance procedures, improve procedures to bring them into compliance, and continuously assess compliance by comparing actual performance to model performance.

Extend+BPR's high ratio of one-to-one conformity with real-world operations facilitates many aspects of the ISO9000/EN29000 process.

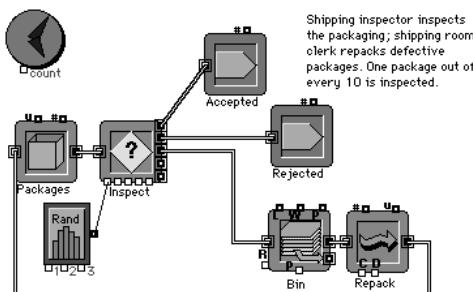
Testing Process model

Whether you manufacture a product or provide a service, quality should be an important component of what you sell. One way to determine and document that your quality assurance process is functioning properly is to simulate it. The Testing Process model shows the final inspection of software packages before shipment. After a period of time, the inspector determines that the packages are acceptable, unacceptable, or require repackaging. The particulars are:

- The model runs for 1 hour

- There are 400 packages to be inspected at the start of the hour
- It takes 15 seconds to inspect a software package
- 85% of the packages pass review, 2% are rejected as being too damaged for shipment, and 13% require repackaging
- The repackaging process takes 1 minute
- After repackaging, the software is returned for inspection

The model looks like:



Testing Process model

BPR

The Input Random Number block provides a convenient table for specifying how many packages pass and how many fail or need repackaging. Notice how easy it is to show decisions or routing in Extend+BPR. Also notice that the process is not only specified but documented.

Resource capacity analysis

Resources are the means by which process activities and operations are performed. Typical resources include equipment, personnel, space, energy, time, and money. Resources can be available in unlimited quantities but most often are limited or constrained.

Utilization and constraints on resources

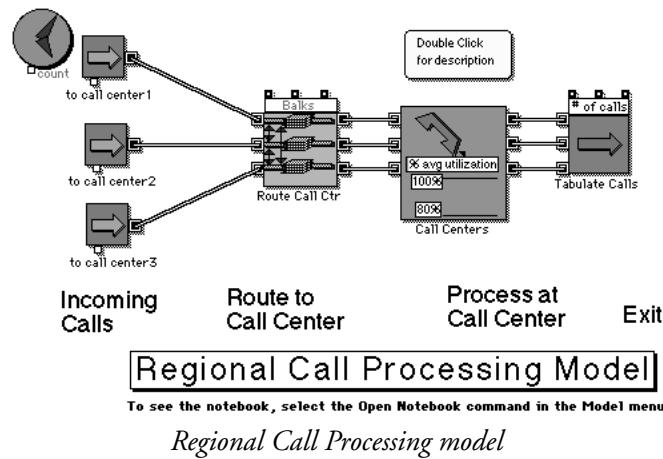
One of the main reasons to model a process is to analyze resource utilization and constraints. This tells how efficiently current resources are being used and what happens when they are not available or when there is a wait for them to become available. For example, a mail order company might want to investigate if it is losing customers because there are not enough sales people for the volume of incoming calls. Companies who are reengineering often model resource capacity to determine how to improve resource utilization without causing overly long waiting lines or how to reduce waiting lines without adding more resources.

Regional Call Processing model

This model is more complex than the models discussed earlier in this manual. It is an excellent illustration of the creative use of hierarchical blocks, customized animation of hierarchical block icons, and the use of the Notebook as a control panel or executive interface. The purpose of the model is to determine how many resources are required to support the desired service level.

Note This model uses more memory than the other examples. If you are warned that you are running low on memory when you run this model, see the Extend manual regarding *low memory problems*.

The Regional Call Processing model represents three regional call centers (this is a common situation for a catalog ordering business or an airline reservation system). Calls are made to the local processing center in random sequence. If no one is immediately available to answer the call, the caller may be placed in a queue, routed to another call center, or told to call back later, depending on the situation. The model is:



About the model

In the model, calls are generated by Import blocks within the three hierarchical blocks at the left. The time between arrivals is exponentially distributed and calculated from the average number of calls observed per hour. Calls are then routed by the “Route to Call Center” hierarchical block. In this block, Decision(5) blocks determine where the call will go next based on various conditions:

- If an agent is available, the call is sent directly to the “Call Centers” hierarchical block.
- If no agents are available and there is space in the call center queue (a resource block) the call is sent to the queue.
- If the queue is full, an attempt will be made to send the call to an alternate call center. To do this, there must be space in the queue of the alternate call center. If there is space, the call will

pass into a Queue Resource Pool block where it waits for a “link” resource to become available. Once a link is available, the call can then be sent to the other center.

- If the queue is full at the current call center and there is no link available to a different call center the call is lost (this is called *balking*).

Call processing is performed at the “Call Centers” hierarchical block using Transaction blocks. The delay time is normally distributed with a standard deviation equal to 20% of the mean (the mean time per call can also be specified from the Notebook).

When calls have completed processing, the “Tabulate Calls” hierarchical block counts the number of calls and (if necessary) restores the links in use to available status.

Model highlights

In this model, important parameters and results are cloned and made accessible in the model Notebook. This makes it easy to see results and change values from one convenient location.

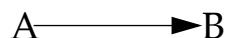
Some of the hierarchical blocks in this model have custom animation on their icon. This was done without programming using the blocks in the Animation library. For example, in the block named “Route Call Ctr”, the Animate Value block is used to display the number of call waiting in each queue on the hierarchical block’s icon. Note that, since there is extensive animation in this model, simulation run time increases significantly when animation is turned on.

Although this model accounts for calls which cannot be accepted and do not enter a queue, it does not model the situation in which a caller who is already in a queue grows tired of the wait and hangs up. This is known as *reneging*. If renege were an important factor in the process you were modeling, you could include it by using the Stack block and selecting the renege choice.

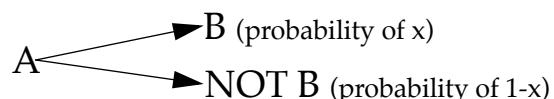
BPR

Causal loops

A common implementation of causal loops implies that the result is a certainty. In other words, the diagram



implies that an increase in A will result in an increase in B. However, when using causal diagrams, it is as likely that the actual case being represented might be more like:

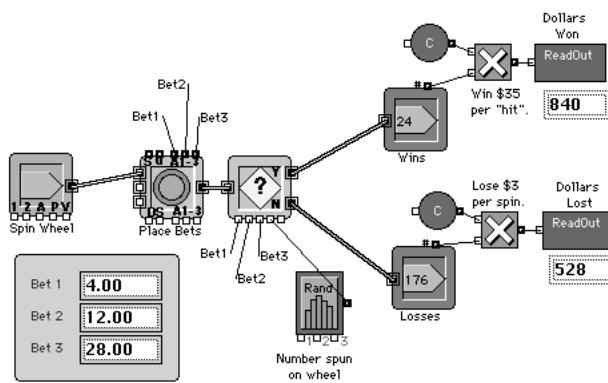


This is the more common case when you are considering options: there is a probability of event A causing event B; there is also a probability of event A causing the opposite of event B (or NOT B). When considering a course of action, it is important to consider all the possible outcomes, the negatives as well as the positives.

For example, government agencies often pass regulations to protect consumers. If they determine that protection mechanisms are not adequate, new laws and regulations are passed which may add to consumer cost. Part of the legislative process might include modeling to determine if the outcome of the new law results in a net actual benefit to the consumer.

Roulette model

The Roulette model represents a game where bets are placed and there are two possible results, a win or a loss. As seen below, the model not only calculates the outcome of A causing B (wins) but the outcome of A causing NOT B (losses).



Roulette model showing wins and losses

BPR

Enter your three favorite numbers in the fields on the worksheet, then run the simulation. (These betting fields are clones of attribute values from the Attribute tab of the block named *Place Bets*.) Each time the wheel is spun, the model will report your wins and losses. The simulation is run for 200 spins; at the end of the simulation, see if you have more money or less than when you started.

In business, managers must weigh the potential positives against the potential negatives. It is important to remember that the risk of fixes that cause counter-productive results must also be included in the equation.

Model highlights

This is a fun model to run, but it also illustrates several important features:

- The Decision (2) block easily routes items to one path or the other based on logic entered in its dialog. This is an automatic “OR” condition, since the item must take one path or the other.
- It is easy to set and change random numbers. The Input Random Number block is used to set the “Number spun on wheel”. This block lets you choose a distribution by clicking on a button. You can then enter the variables for the distribution, such as the mean, right in the dialog.

- The “Dollars Won” and “Dollars Lost” parameters immediately show the amount of winnings and losses. These are “cloned” dialog items. Instead of having to double-click on a block to see or change parameters, Extend allows you to “clone” dialog parameters to the worksheet or to the Notebook. This makes it easy to put important variables where you want to have them. You can change parameters from any location in the model or you can view interim results as the model is running.
- The model uses the Multiply and Constant blocks from the Math submenu of the Generic library to explicitly show mathematical computations. As an alternative, you could use an Equation block (also from the Math submenu of the Generic library), but the calculation would then be hidden.

BPR Chapter 2: Tutorial

Before you begin the tutorial

The BPR and Statistics libraries are used in concert with other Extend libraries, especially the Generic, Discrete Event, and Plotter libraries. The BPR library is based on Extend's Discrete Event library and contains many extensions of that library. To use the BPR library to its fullest, first establish at least an intermediate understanding of Extend and the Discrete Event library. Familiarity with the Extend Generic library is also helpful since there are many cases in which blocks from all three libraries can be used in the same model.

Before you begin this chapter, we suggest that you refer to the main Extend manual, particularly Chapters 1 through 4, for more information on how to use Extend and the Generic and Discrete Event libraries. Chapters 1 through 3 in the main manual provide a brief general tutorial for using Extend. Chapter 4 provides very important information about using the Discrete Event library.

The BPR library is designed to help you build operational models of business processes; the Statistics library reports model-wide metrics for performance analysis. The following tutorial will introduce you to most of the important blocks in these two libraries. Understanding how the blocks in the libraries work will help you quickly create models of any real-world system.

The blocks used in this example are representative of the blocks that you will use in all of your models. Later, the manual discusses the concepts that were used to build this model and gives a complete description of the blocks in the BPR and Statistics libraries.

Rightsizing

Computerized case study models have become fashionable at business schools throughout the world. Teams of students compete to see whose company will have the highest stock prices. Students quickly learn that often the fastest way for the models to show the most profits is to lay off the workers and sell assets. This results in a significant improvement in earnings, which translates into higher stock prices and a “win”.

Computer gaming models have two liabilities: they do not reflect real-world conditions and complexities, and they emphasize short term outcomes at the expense of long term investments. For example, the case study models have no mechanism for determining the long term effect of firing the team that was working on wireless communication, or laying off the support person who answered customers' questions the quickest.

Rather than wholesale downsizing, companies are finding that rightsizing can be an effective method of improving operations by removing poor performers and cancelling redundant projects. The key to rightsizing is to model the process as it exists, then experiment with changes until you achieve the smallest system that will produce optimum results. The objective is to simplify the value-adding activities so that each step in the process adds value and the process becomes a continuous flow.

Description of the problem

The management of a major mortgage company has decided, based on limited information, that the company can save money if it reduces its staff. Before downsizing, management asks you to model the credit application process to provide reassurance that productivity will not be negatively affected by the reduction in staff.

In situations where the organization is new, such as with a newly forming company or after a merger of two companies, you would probably start with a model of the “to be” or proposed process. While you could just model the “to be” credit application process, there are advantages to modeling the “as is” or existing process first. “As is” models provide a common framework for understanding the existing process, help identify problems that can be fixed quickly, and pinpoint other areas for improvement or redesign. In large reengineering projects, companies often find it helpful to model not only the “as is” and the “to be” processes, but also transitional processes which cover the phase of moving from the current system to the redesigned system.

BPR

The mortgage company currently employs three loan agents, two of whom perform the initial review of credit applications and a third who re-reviews applications that fail the initial review. The second review is performed to see whether the deficiencies can be corrected by contacting the originating party and so forth. The process works as follows:

- Approximately four to eight credit applications (and most likely 6) arrive every hour
- Four credit applications await initial review at the start of the simulation
- It takes 12 to 16 minutes to complete the first review
- About 20% of the applications fail the first review
- It takes 25 to 35 minutes to complete the second review
- About 50% of the applications fail the second review

The resulting models are located in the “Examples \ BPR \ Credit Application” folder. Each step in the modeling process is illustrated by a separate model (Credit App -Step1 to Credit App - Step7). As information is added and assumptions are refined, the models become more complex.

Starting the Credit Application model

You need to look at a few salient points before you start your model. By doing this first, you prevent yourself from having to make major changes later in the process. Things to think about include:

- What is the goal of the model
- What will flow through the model
- What kind of delays will happen at each step
- Which data is known exactly and which is estimated

The most important consideration is “What is the desired result?”. Note that this is not the final answer to your question, but what *kind* of answer do you want. For example, are you trying to find out how long something takes, how many people it takes to do it, or how much it costs? All of these take different types of data and give you different results. In this example, the goal is to determine if staff reduction will have a negative impact on the volume of output from the process.

You start by modeling the situation as it currently exists. Since you can compare the model to the existing process, this ensures that you are considering all the important factors. This will also provide benchmark information you can compare to when you change the process.

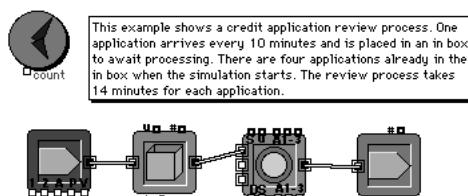
BPR

Define a global time unit

Start by defining a global time unit for the model in the Discrete Event tab of the Simulation Setup dialog from the Run menu. Since you will be modeling an 8-hour day, choose hours as the time unit and set the end time of the simulation to 8.

Add blocks to the model

The following example model shows a basic credit application review process. One application arrives every 10 minutes and is placed in an “In Box” to await processing. There are four applications already in the “In Box” when the simulation starts. Each application takes 14 minutes to review.



Credit Application-Step 1

Executive block



The first block in *every* BPR model is the Executive block from the Discrete Event library. This block must be to the left of any other block in the model, so it places itself at the upper leftmost edge of the model window when you select it from the library.

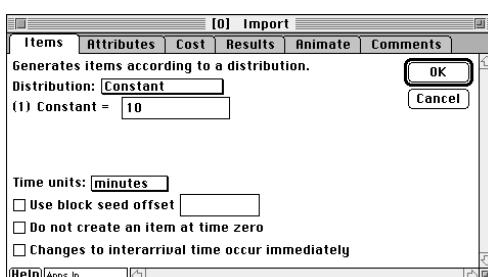
The Executive block controls the timing in BPR models, so that timing is handled automatically, and time advances when events actually occur. Note that, because it is in every BPR model, the Executive block is not always shown in each example in this manual.

Import block



The next block in a BPR model is usually an Import block from the Generators submenu of the BPR library. The Import block (called “Apps In” in the model) represents items entering from outside the scope of the model. When developing the model, you first need to determine what items will flow through it. In this case, the items are credit applications and the Import block represents importing credit applications from outside the model (from the customers).

The next task is to determine how often items will arrive. In the Import dialog, you specify the *time between arrival* (also called the “inter-arrival” time) for each credit application. You can also specify an attribute, a priority, and an item Value (the number of items) to be applied to each item. The dialog for the Import block is:



Import dialog (Macintosh)

As noted in the problem description, “Approximately four to eight credit applications (and most likely 6) arrive every hour”. For simplicity, use a constant distribution with an interarrival time of 10 minutes. To do this, select the Constant in the distribution popup menu, enter 10 into the field labeled “Constant =”, and select minutes in the time units popup menu. This will cause 6 items to arrive per hour.

Note The items output by the Import block occur based on the *time between arrivals* or *inter-arrival rate* of credit applications rather than the number of arrivals per hour.

Notice that we have chosen a “Constant” distribution to represent the time between arrivals. As discussed on page B158, models where all inputs are constant are said to be *deterministic*; models

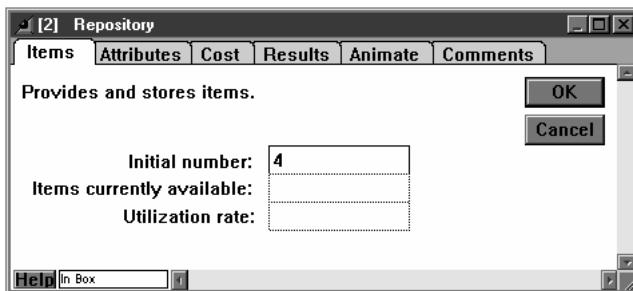
with one or more random input variables are termed *stochastic*. Using constants is a helpful modeling technique because deterministic models always result in the same outcome. With randomness removed as a factor, model validation is easier since any changes in model results would have to be caused by changes to the model itself. Later, you will change the assumptions in the Import block to use a more realistic random distribution.

Repository block



A Repository block from the Resources submenu of the BPR library is like a holding area that can have an initial number of items. In this case, it represents the basket (“In Box”) into which the credit applications are piled until the person processing them can take one. In the dialog, you can specify how many items the block has at the beginning of the simulation. You can also add an attribute and a priority to the items as they leave.

When an Import block is used to generate items in the model, you must follow it with a Repository or a Stack block because the Import block always pushes items out of the block regardless of whether or not the block following it is ready to accept them. Repository and Stack blocks can always accept items unless they have reached their limit. The Repository block’s dialog is:



BPR

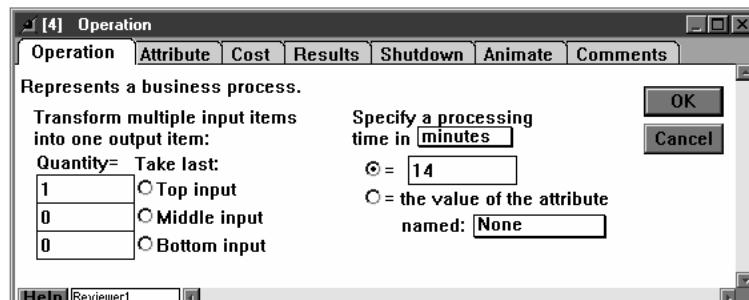
Dialog of Repository (Windows)

The Repository has an initial value of 4. This value comes from the information that “Four credit applications await initial review at the start of the simulation”.

Operation block



The Operation block (Reviewer 1) from the Activities submenu of the BPR library has many uses including transforming items, batching, and processing. Here, it is just used to delay or process the item (the credit application) for an amount of time. The dialog for the Operation block is:



Dialog of Operation (Windows)

Set the time unit for the processing time to minutes and enter 14 into the field directly below the time unit popup menu. This is an average derived form the information that “It takes 12 to 16 minutes to complete the first review”. In this case, so that you will get reproducible results, the processing time variable is a static constant (later you will see how to change this to a dynamically changing random value).

BPR

Export block



The Export block from the Routing submenu of the BPR library, labeled “Apps Done”, is used to pass the items out of the model. This block represents the place where the items leave the realm of the current simulation. In this case, it is where the credit applications go after they are processed, namely back to the applicants with a letter of acceptance or rejection. Most models have at least one Export block so that the items are removed when they are no longer needed.

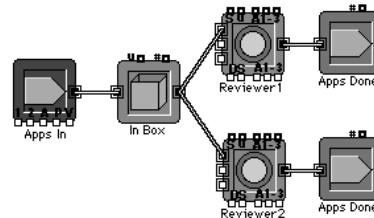
Add a second reviewer

The first model illustrated a common method in modeling, namely making simple assumptions first, then expanding those assumptions later. Now, because there are two people reviewing the credit applications, you need to expand the model by duplicating the Operation block and the Export block. In this case, it is easy to double the number of reviewers:

- ⇒ Select the Operation and Export blocks.
- ⇒ Move them up to make room for the second reviewer.
- ⇒ Choose the Copy command.
- ⇒ Click below the original two blocks

- ⇒ Choose the Paste command.
- ⇒ Attach the input on the new Operation block to the output on the Repository block.

This is shown below:



Credit Application -Step2

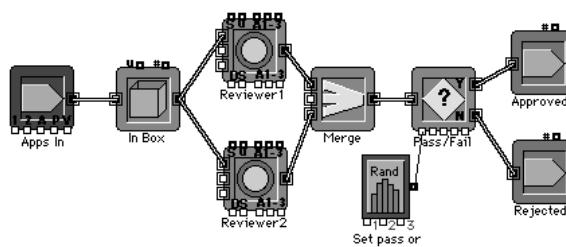
When you add processes in parallel to existing processes, it is important to decide exactly where you want the two processes to begin. In this case, they both begin immediately after the Repository block because each reviewer will be taking applications out of the “In Box”.

Determine which applications pass on the first review

Instead of exporting the applications after they are processed, 80% of them must go to one stream (those that pass the review), and the remaining 20% go to a second stream (those that fail). This corresponds to the original information that about 20% fail the first review. Thus, the items from the two reviewers are merged, a decision about each item in the merged stream is made, and two different streams (based on the pass/fail decision) are generated.

BPR

If you follow the steps in this section the model will look like:



Credit Application -Step3

Note that the two streams that are connected to the Export blocks now represent approved and rejected applications, not the results of each individual reviewer.

Merge block

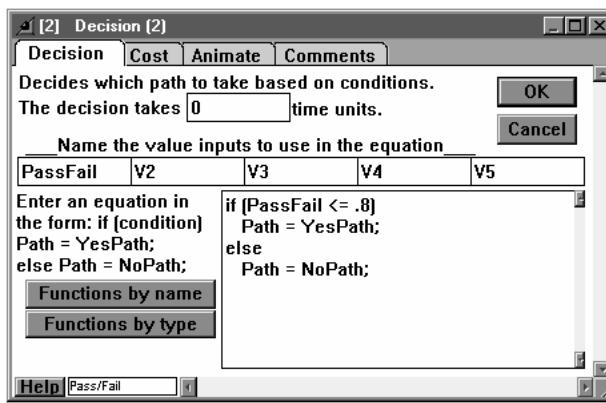


To combine up to three streams of items into one, use a Merge block from the Routing submenu of the BPR library. This block does no processing on the items and does not hold them, it simply combines them into a single stream.

Decision (2) block



The Decision (2) block from the Routing submenu of the BPR library evaluates an equation based on the values at its input connectors; the result of that equation determines whether the item input goes to either the *Y* or *N* connector. The block's dialog has an area for you to enter the equation:



Decision (2) dialog (Windows)

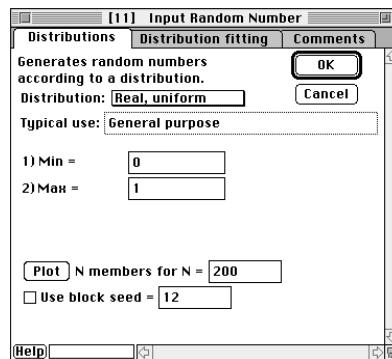
BPR

The Decision (2) block gets a random value from 0 to 1 from the Input Random Number block, discussed below. In this case, the Decision (2) block sends an item to the top exit if the value at the leftmost connector (called *PassFail*) is less than or equal to 0.8. This indicates that 80% of the applications pass review. Because the model uses random numbers to determine which stream the items go into, the actual results may vary slightly each time you run the model.

Input Random Number block



The Input Random Number block from the Inputs/Outputs submenu of the Generic library generates many types of random distributions. Its dialog is:



Input Random Number dialog (Macintosh)

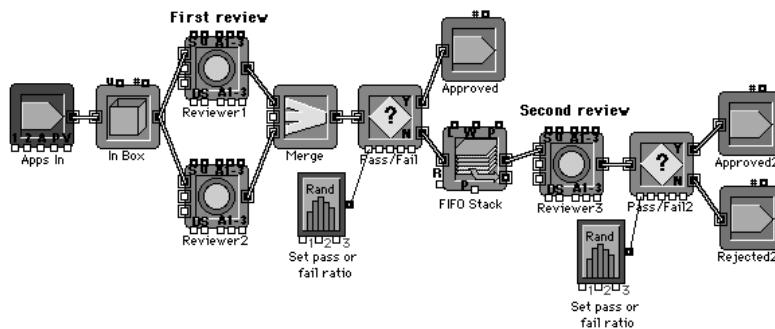
Here, the block generates a real number between 0 and 1 for each item. This number is read by the Decision (2) block which determines if the value is less than or equal to 0.8 (the application passes) or greater than 0.8 (the application is rejected).

BPR

So that results are reproducible every time the model is run, a *random seed* is set in the Random Numbers tab of the Simulation Setup dialog. As discussed in “Random numbers and seeds” on page B159, this causes the same random number stream to be generated with each simulation run.

Add a second review process

The next step is to add the review process for the applications that fail the first test. Because this process takes time, another Operation block is added. When you are finished with this section the model will look like:



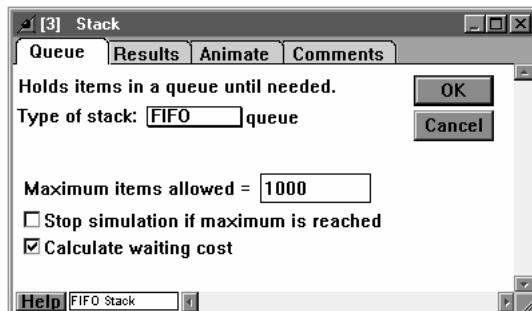
Credit Application -Step4

A second Decision (2) block will also be added with an Input Random Number block connected to it that outputs a real number from 0 to 1. It passes half of the items to the top output and half to the bottom, corresponding to the original information that only 50% of the applications pass the second review.

Stack block



The Stack block from the Queues submenu of the BPR library is similar to the Repository block. However, you can specify the order in which items are taken from the stack (in the Repository block, they always leave in the order they entered), and you can specify the maximum number of items allowed in the block at a time. Your choices are FIFO (first in, first out), LIFO (last in, first out), Priority (items with the *lowest* priority number go first) and Reneging (items leave if the wait is too long). The dialog is:



Stack dialog (Windows)

BPR

Add randomness

Up to now the model has used constant values to represent the time between arrivals of credit applications and the time it takes for both review processes. The next step is to have these parameters more closely approximate the real-world conditions. You do this using random distributions which cause the values to change dynamically.

Import block

The first step is to change the distribution used to specify how frequently credit applications arrive.

- ⇒ In the dialog of the Import block (labelled “Apps In”), change the distribution for the arrival of credit applications to “Triangular”. Notice that the triangular distribution allows you to specify the minimum, maximum, and most likely time between arrivals.
- ⇒ Next, calculate the time between arrivals. At the beginning of this Tutorial, the information was that 4 to 8 credit applications arrived every hour, with a mostly likely occurrence of 6 per hour. To calculate the time between arrivals, divide the time period (using the Import block’s local time units) by the number of occurrences during that period. For example, when 4

credit applications arrive every hour, and the Import block's time unit is in minutes, then the time between arrivals is 15 (60 divided by 4). Likewise, for 8 applications every hour, the time between arrivals is 7.5; for 6 the time is 10. With these values, the Import block now looks like:

Distribution:	Triangular
(1) Min =	7.5
(2) Max =	15
(3) Most likely=	10

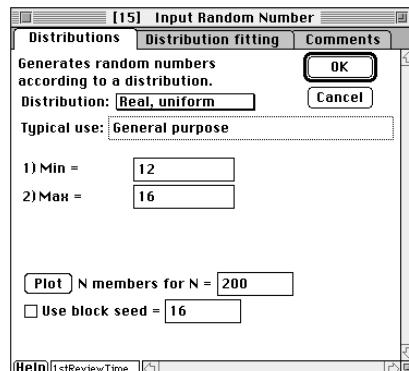
New distribution and arrival rate in Import block

Input Random Number block

The next step is to change the time it takes for the first review process to a random number. To do this:

- ⇒ Add an Input Random Number block near the first review process.
- ⇒ Connect the output of the Input Random Number block to the “D” inputs of both Operation blocks (labelled “Reviewer 1” and “Reviewer 2”).
- ⇒ In the dialog of the Input Random Number block, select the “Real, uniform distribution”. Set the “Min=” parameter to 12 and the “Max=” parameter to 16. This corresponds with the information that there is 12-16 minutes spent on the first review process.

BPR



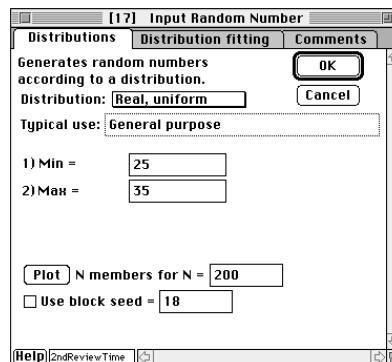
Input Random Number block for first review process time

As discussed in “Random distributions” on page B159, the uniform distribution is appropriate for this purpose since no other information, other than the range of values, is known.

Finally, you need to have the second reviewer's time be random:

B48 | BPR Chapter 2: Tutorial
Test for reducing the number of reviewers

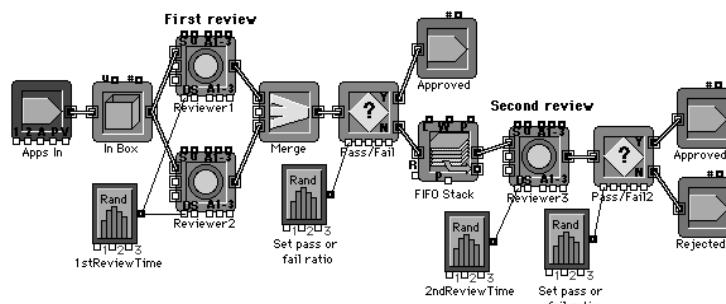
- ⇒ Add another Input Random Number block near the second reviewer and connect its output to the Operation block representing the second review process, as was done above for the first review process. Select the “Real, uniform distribution” for this activity, as was done above.
- ⇒ Set the “Min=” parameter to 25 and the “Max=” parameter to 35. This corresponds with the information that there is 25-35 minutes spent on the second review.



Input Random Number block for second review process time

BPR

The model, which represents the present situation in the mortgage company, now looks like:



Credit Application -Step5

Test for reducing the number of reviewers

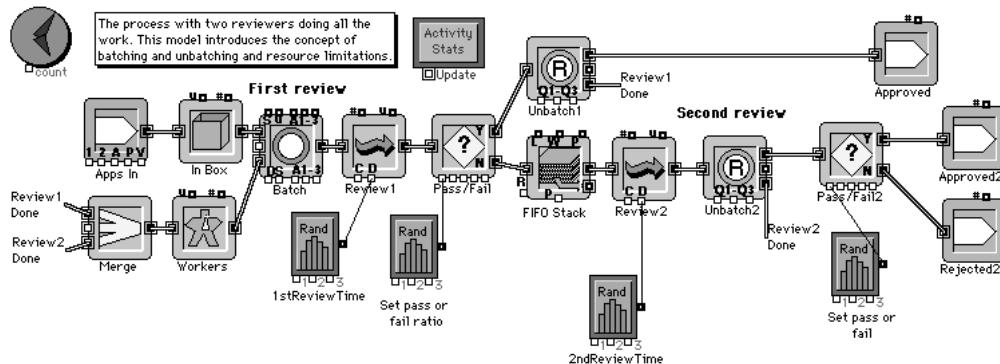
You can now look at the result of removing one worker. Your first thought might be “simply remove Reviewer2 from the model and run the simulation again”. This would be incorrect because then Reviewer1 and Reviewer3 would be working serially, meaning that many applications will get backed up behind Reviewer1 while Reviewer3 sits idle. Management wants both steps of the reviewing to be done by the two remaining reviewers. To do this you need to show the workers working in parallel.

There are two methods you can use to model situations where workers work in parallel on multiple operations:

- You can model each worker's workload as a series of operations. For instance, in this credit application department you could show each reviewer's workload as two Operation blocks, one representing initial review and one representing the second review. This is easy and practical for modeling a small or medium sized operation, but can get quite confusing when modeling situations where there are many workers working in parallel.
- You can use a Transaction block to represent multiple operations occurring at the same time, where the number of operations that can occur at any time is limited by the number of available workers. This method has the advantage of allowing you to change the number of workers as conditions change, without having to change the model. This is the method you will use here.

Up to now, your models have not directly accounted for the workers in the credit approval department. Since each operation in the process required only one worker and each worker only performed one task, it was simple to model the credit department using Operation blocks to represent both the process (either initial or secondary review) and the person performing the process.

To model several processes with a variable number of workers performing the processes in parallel, you need to change the model as shown below. The new model uses one Transaction block to represent the initial review and another to represent the second review. It uses the concepts of batching and unbatching (discussed in detail in Manufacturing Chapter 9: Batching and Unbatching) to put together two items at either step of the process: a credit application and one of the two reviewers. Thus, either reviewer might be temporarily “batched” with the application. If the application passes in the first step, that reviewer is freed up to take another application. If the application fails during the first step, the same reviewer takes it through the second step. After the determination is made in the second step, the reviewer is free to take another application.



Changing the model

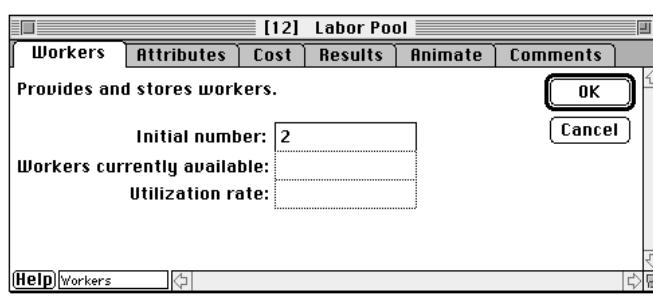
In order to make these changes, you will need to delete some blocks and change some connections:

- ⇒ Delete the Operation blocks labeled “Reviewer1” and Reviewer3”.
- ⇒ Because this batching operation takes no time, delete the connection between the remaining Operation block (labeled “Reviewer2”) and the Input Random Number block (labeled “1stReviewTime”) and specify a processing time of “0” in the Operation block’s dialog.
- ⇒ Cut the Merge block from between the Operation and Decision (2) blocks and paste it to the left of the model. Notice that “cutting” the block automatically removes its connections to other blocks.
- ⇒ Now add the new blocks as indicated below.

Labor Pool block



The first addition is at the lower left of the model. Models in which people move through the simulation as items should use Labor Pool blocks from the Resources submenu of the BPR library to represent the people so as to differentiate them from the other items that are used. In the Labor Pool block, you can specify an initial number of people, as well as a default attribute and priority for the people. Its dialog is:



Labor Pool dialog (Macintosh)

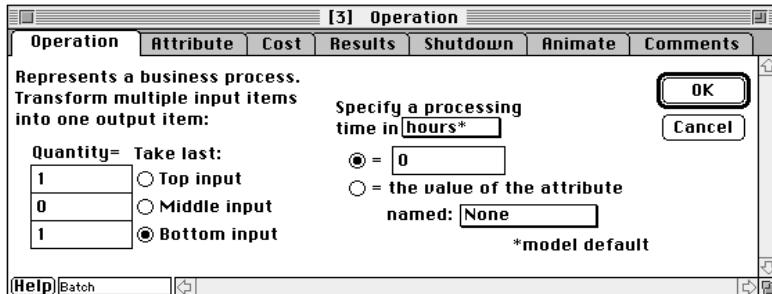
The Labor Pool block starts with two workers, representing the two reviewers left in the department. The Merge block to the left of the Labor block combines the two streams of workers (those finished processing successful applications from the first step and those finished processing all applications from the second step) and returns them to the pool.

Operation block



Because you want to show two workers working in parallel on multiple processes, the Operation blocks are no longer used to specify the processing time; instead, one of them is used to batch one application with one reviewer.

The Operation dialog choices in this situation are:



Operation dialog choices for batching (Macintosh)

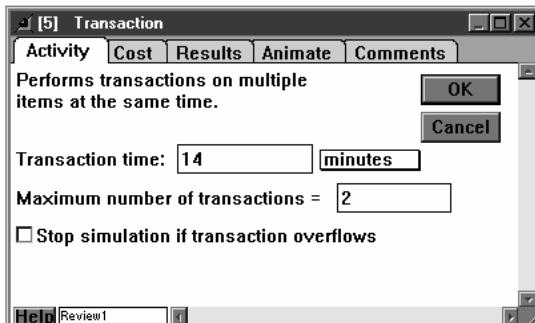
Two items, an application and a reviewer, come into the Operation block, but only one, an application/reviewer batch, leaves. Enter a quantity of 1 for both the top and bottom outputs. Select the Bottom input radio button in the Take last: column. Since the worker (at the bottom input) is taken last, the block will not pull a worker from the Labor Pool until an application is waiting for review. In addition, if one of the two incoming items is not available (such as if an application is waiting to be processed but there is no reviewer ready), the Operation block doesn't release the batched item until both inputs come into the block.

BPR

Transaction block



The Transaction block from the Activities submenu of the BPR library can be used to represent several processes that occur in parallel. The maximum number of transactions that can occur at the same time are specified in the dialog which looks like:



Transaction dialog (Windows)

In the two Transaction blocks in this model, set the maximum number of transactions allowed to 2, indicating that up to two reviewers might be working on that particular step at any given time. If you had instead used the Operation block to represent the review process, only one reviewer could be reviewing at a time, which is clearly not the case in this situation. (Note that, although

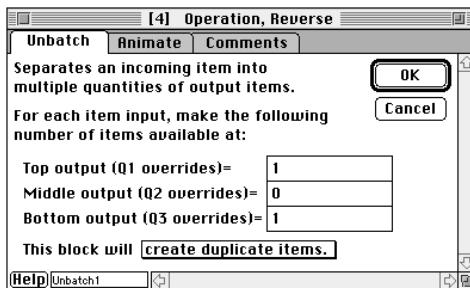
each of the two Transaction blocks allows up to two reviewers, there can only be a maximum of two reviewers in the model due to the limit set in the Labor Pool block.)

Because the Transaction blocks represent the two review processes, connect the Input Random Number blocks to the “D” connectors of the Transaction blocks.

Operation, Reverse block



After you have temporarily batched items together using the Operation block, you want to later unbatch them. The Operation, Reverse block from the Batching submenu of the BPR library does this easily. It takes a single item (the batch) and clones it into many items, each symbolizing the individual, unbatched items. Its dialog is:



Operation, Reverse dialog (Macintosh)

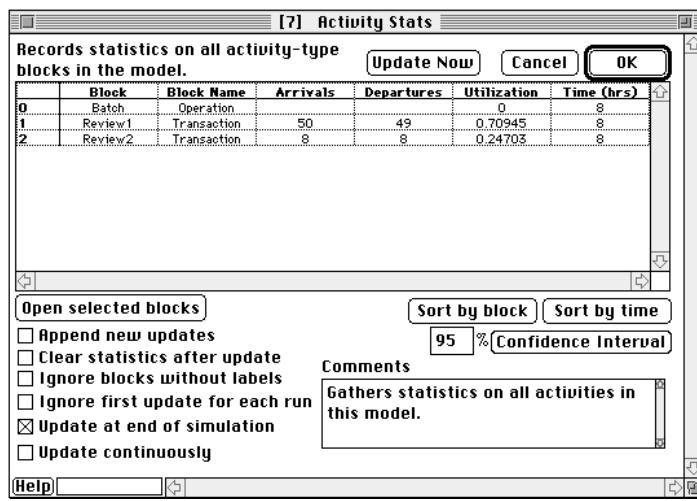
BPR

In this model, after a credit application review is finished, the Operation, Reverse block unbatches the application and the reviewer. In this case, the top output from the Operation, Reverse block is the application, the bottom output is the reviewer. The reviewer returns to the Labor Pool block through the *named connection*, and the application leaves the model through the Export block.

Activity Stats block



The Activity Stats block (from the Statistics library) provides a convenient method for accumulating model data from any blocks that have a delay or processing time. While not necessary in this model, due to its small size, the statistical blocks are invaluable for gathering and reporting metrics in large models. The dialog of the Activity Stats is:



Activity Stats dialog for Step 6 (Macintosh)

BPR

In the dialog you can specify how you want the block to perform. For example, you can have the block report statistics as the simulation runs, or just at the end of the simulation. You can also copy the results and paste them into any word processing program, or *clone* them to the model Notebook.

Customize animation

Turn on animation by selecting the Show Animation command in the Run Menu. When you run the simulation, Extend's default animation picture, a green circle, will be animated along the connection lines between blocks.

In the Animate tab of blocks that generate items, you can choose the animation picture that will represent your items. In this model, the Import block generates credit applications. In the Animate tab of the Import block, choose the paper picture to represent the applications as shown below:

Items will appear as:



Setting the animation icon in the Import block

As the credit applications generated by the Import block flow through the model, they will be represented by the paper picture.

Some of the credit applications originate in the Repository block (there is an initial number of 4 credit applications in the Repository block when the model starts). In the Animate tab of the Repository block, choose the paper picture to represent the applications.

The Labor Pool block provides workers for the review processes. In the Animate tab of the Labor Pool block, choose the person picture to represent the workers. Because of how attributes are merged in the batching process, the workers will inherit the animation picture of the credit applications (paper picture) after being unbatched from the credit applications. In the Labor Pool block, you can change the animation picture of the recycled workers back to the person picture by checking the “All items passing through this block will animate the above picture” option in the Animate tab.

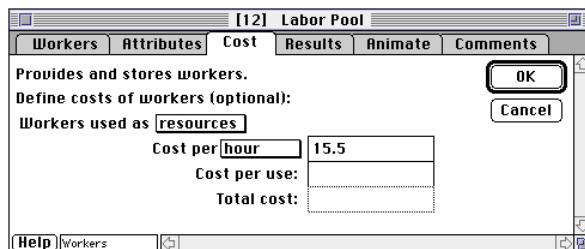
Calculate cost of review process

Extend's activity-based costing feature can help you calculate the cost of reviewing the credit applications (see Chapter 10, “Activity-based costing” for a detailed discussion). The two things that contribute to the cost of reviewing a credit application are:

- The reviewers are paid an hourly wage of \$15.50.
- As part of the first review process, a credit report is obtained for each applicant. The cost of a credit report is \$25.

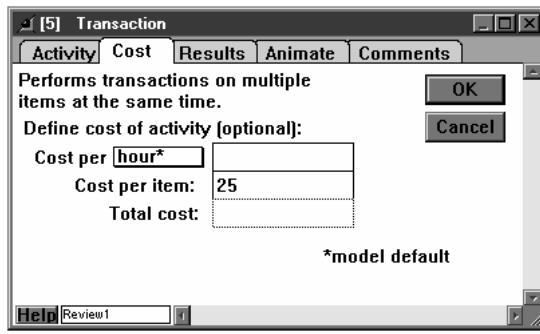
BPR

In the Cost tab of the Labor Pool block, fill in the Cost per hour for the reviewers and select “resources” in the “Workers used as:” popup menu, as shown below:



Cost tab of Labor Pool block (Macintosh)

In the Cost tab of the Transaction block representing the first review, fill in the cost of the credit report (Cost per item) as shown below:

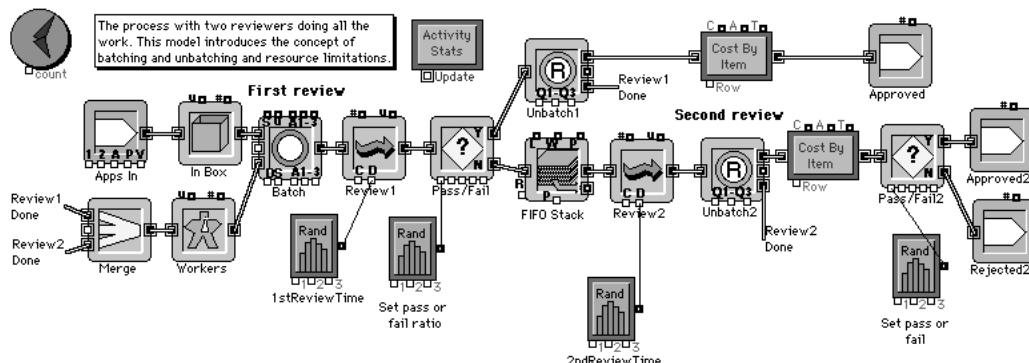


Cost tab of Transaction block (Windows)

Note In the dialog of both Operation Reverse blocks, the block must be set to “release cost resources”.

When the simulation is run, the costs are tracked with each credit application. To collect the cost data, add a Cost Stats block anywhere in the model and Cost By Item blocks after the Operation Reverse blocks as shown below:

BPR



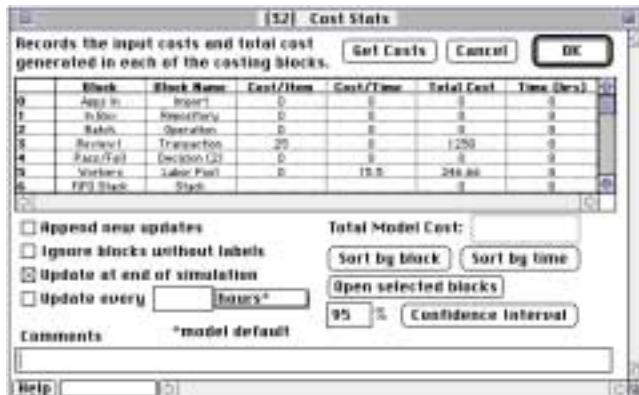
Credit Application - Step 7

Cost Stats block



Place this block (from the Statistics library) anywhere in the model and it will report the following statistics for all costing blocks in the model: Block Number (or label, if a label is entered in the block), Block Name, Cost Per Item, Cost Per Time Unit, and Total Cost.

After the simulation is run, the cost generated in each activity, queue, resource, and generator-type block is collected and displayed in the Cost Stats block as shown below:



Dialog of Cost Stats block (Macintosh)

Cost By Item block

BPR

 This block views and displays the cost of the items 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.

The cost of each item that passes through the Cost By Item block will be displayed in the block's dialog as shown below:



Dialog of Cost By Item block (Macintosh)

Model highlights

Most of your effort so far has been in building the models and learning about the blocks. Now it is time to look at the results of running the models to see what conclusions can be reached.

In Step 5 of the model, you can see that a total of 41 credit applications have been approved. In addition, the utilization of each of the first two reviewers is about 0.75 and the utilization of the

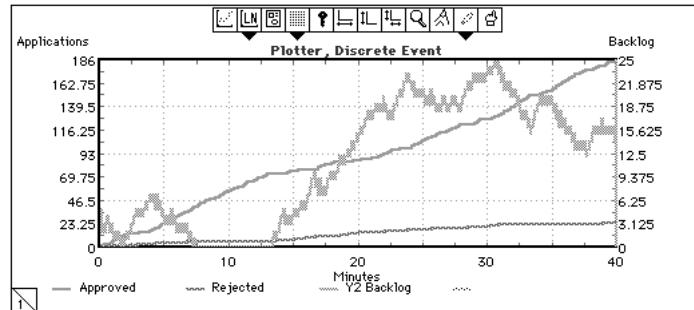
third reviewer is about 0.50. Since full utilization is “1”, this finding appears to support management’s contention that it is at least possible to reduce staffing without adversely affecting productivity.

In Steps 6 and 7, which have one less reviewer, 41 credit applications are still approved. Furthermore, as seen in the dialog of the Labor Pool block, the utilization of the workers is now almost “1”, indicating full utilization. It would appear that productivity is not affected, and management can safely reduce the size of the department to 2 workers.

Since the two workers in the Step 6 and Step 7 models are fully utilized, how can you tell if there were credit applications which were not reviewed because the workers were too busy? Remember that the Stack and Repository blocks serve as holding areas for items waiting to be processed. If you check their dialogs, you will see that there are no applications waiting for processing. However, simulation shows you changes over time, not just the end results. To see the iterative process, you need to plot the information.

The model for Step 8 is the same as the Step 7 model except that it is run for 5 days (40 hours) and has a Plotter, Discrete Event (from the Plotter library) which reports the number of applications approved and rejected and the backlog of applications waiting for first review. When you run this model you see:

BPR



Longer-term effect of decrease in workers as seen in Step 8

The level of credit applications waiting for first review (the “Backlog”) is plotted as a grey line on the Y2 axis. In the graph, you can see that the backlog level frequently goes above “0” and that sometimes there are up to 25 applications awaiting processing. This means that some applicants must wait longer for approval. As part of your analysis of this process you should determine if the reduction in staffing outweighs the potential customer dissatisfaction caused by the periodic backlog.

Things to consider

When you use random numbers the results will probably differ slightly each time you run the model. This randomness is appropriate since it reflects the random happenings of real-world

events. However, while you are building and testing models it is a good idea to use constant values and/or a random seed so that the model uses the same sequence of random numbers every time it is run. This gives repeatable results, allowing you to determine exactly how your changes affect the model. The random seed for the model is set in the Simulation Setup command's dialog.

To investigate the cycle time for this process, you could use the Timer block (from the Discrete Event library). This is shown in “Cycle time” on page B18.

While you were building this model, you probably thought of several changes you could make to correspond to real-world processes. However, remember that your goal is not to model every variable, but to obtain sufficient information on which to base a decision.

BPR Chapter 3: Overview of blocks in the BPR and Statistics libraries

The BPR library is a tool kit for modeling business processes. You use the blocks in this library, in conjunction with other Extend blocks, to create models that show how your business works or how you want it to work. You use the Statistics library blocks to accumulate and report important statistics about how your process operates. In order to use these libraries and Extend to the fullest, you first need to be familiar with some modeling concepts.

The main Extend manual describes events, items, and other concepts used in discrete event modeling. You should read and be familiar with those terms before you read this chapter.

Chapters 4 - 12 in this manual discuss additional modeling concepts you should be aware of as you build models. They also show how the concepts are applied to solve real-world problems. Remember that you build a simulation model and study its behavior in order to draw conclusions about the real-world system. How you build the model, what data and assumptions you use, directly affects the outputs you obtain and your analyses of those results.

Throughout this manual, references to blocks in libraries other than the BPR library will be identified by library:

- Animation library
- Discrete Event library
- Generic library
- Plotter library
- Statistics library

In addition to the following information, page B4 of the Preface has a picture and brief description of the blocks in the BPR and Statistics libraries. Appendices A and B in this manual also provide detailed descriptions of each block in these libraries, and Appendix C gives a list of commonly used blocks from other libraries.

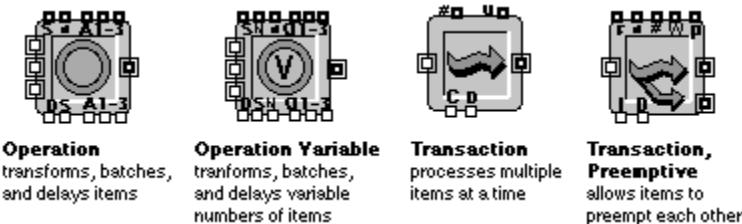
BPR library blocks

The BPR library is based on activities, attributes, batching, generators, queues, resources, and routing.

Activities

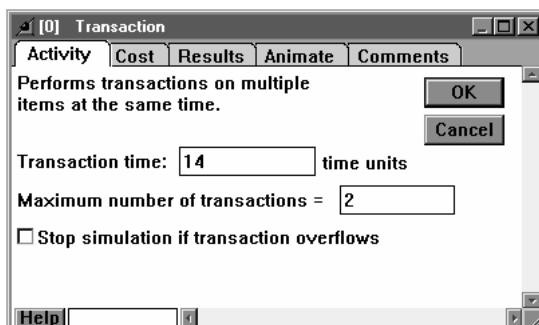
Activities are events that happen to resources and items, such as matching invoices with receiving documents, waiting on customers, or shutting down one task to perform another. Activities can involve a transformation, a delay, a batching, an interruption, or any combination of these.

The activity blocks in the BPR library (Operation, Operation Variable, Transaction, and Transaction Preemptive) regulate the flow of items through the process. Each activity block automatically holds the item until the next block is ready for it. The activity-type blocks are:



BPR

The Operation and Operation Variable blocks delay one item (or one batched item) at a time and also have the ability to batch, or join, items together to form one unit. The Transaction blocks do not transform, batch, or interrupt, but they can process multiple items at a time. The dialog of the Transaction block looks like:



Transaction dialog showing processing time (Windows)

While they are not normally used as activities, the decision blocks (discussed below) can also delay items for a period of time. You would choose this method if the decision process takes time and you want that level of detail in your model.

Attributes

The one attribute-type block in the BPR library is the Measurement block. This block allows you to read up to 5 attributes and the priority of each item that passes through it. The Measurement block is discussed in “Reading attributes” on page B79.



Batching

As mentioned earlier, the Operation and Operation Variable blocks are capable of batching, or joining, items together to form one unit. The Operation Reverse block corresponds to disassembling one unit into its original components or into duplicates of itself. For more information, see BPR Chapter 9: Batching and Unbatching.



BPR

Generators

Generators introduce items into the model. The Import block can generate items at fixed or random intervals. Refer to “Item generation” on page B72 for more information.



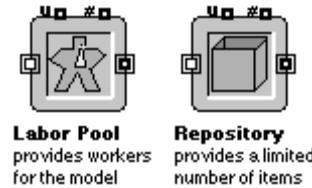
Queues

Queues hold items and release them in a specified order. The Stack block is capable of acting as a FIFO, LIFO, priority, or reneging queue. For detailed information, see BPR Chapter 6: Queueing.



Resources

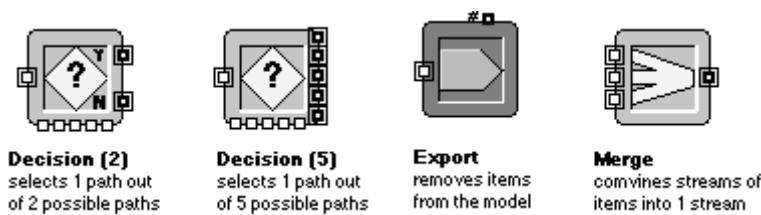
A *resource* is an entity, such as tools, customers, or labor, that is involved in operations with items and/or provides a service to items. The resource blocks in the BPR library are the Labor Pool and Repository block, as seen below:



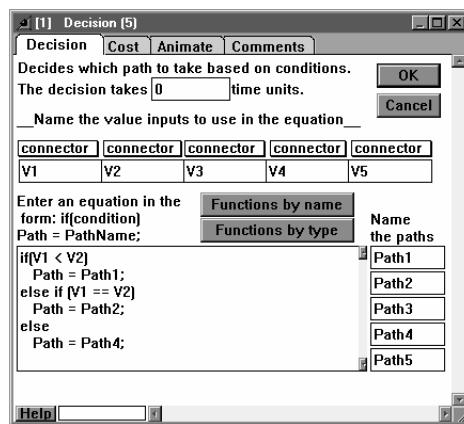
As you will see in BPR Chapter 10: Resources, you can model resources two ways: by batching a resource item with another item or by using resource pool blocks to track available resources.

Routing

Routing blocks are blocks that direct the flow of items. The Decision (2) and Decision (5) blocks let you enter an equation or logical expression that combines and compares the value inputs to choose the output the item will take. The 4 routing blocks are:



As you will see in BPR Chapter 7: Routing, you can specify that the decision process takes some time or that it takes no time:



Decision (5) dialog (Windows)

Statistics library blocks

The blocks in the Statistics library are divided into three groups: blocks which report statistical information for particular blocks by type, blocks that report statistical information for items that pass through the block, and blocks which aid in refining statistical data collection.

The “Stats” blocks

The “stats” blocks accumulate data and calculate statistics for particular blocks in the model. In addition to the block label, block name, and the time the information was observed, each of these blocks displays metrics that are specific to their block type, such as utilization or average wait time.

These blocks record their information in tables in their dialogs. One row of data is recorded for each block in the model each time an observation is made. Observations may overwrite previous observations (restart at the top of the table) or be appended to existing values.

To use these blocks, place them anywhere in the model. You can choose in their dialogs to update the statistics continuously (which significantly slows simulation time) or only at the end of the simulation run. You can also force a periodic or scheduled update of statistical information by connecting a block (such as a Schedule block) to the “Update” input connector. 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.

The “stats” blocks are Activity Stats, Cost Stats, Mean & Variance Stats, Queue Stats, and Resource Stats:

BPR



The Cost By Item block

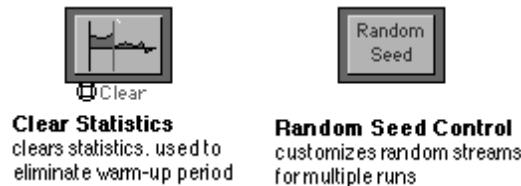
The Cost By Item block reports on the cost information of every item that passes through the block. The data is recorded in the table in the block’s dialog. The block can be instructed to record one row of data for every item that passes through it or to organize the data according to an attribute value. There is also a tab which displays the contents of the cost array for each item. This

tab is included primarily for debugging purposes. See the main Extend manual for a description of the cost array.



Other Statistics library blocks

The blocks in this group are Clear Statistics and Random Seed Control, as shown below:



BPR

- The Clear Statistics block clears and resets statistical accumulators in specified blocks at periodic intervals or in response to a system event. When you start a simulation run, the model often starts out empty. For example, queues have no items and operations have nothing to process. In this situation, gathering statistical data from when the simulation first starts could skew the results. The *warm-up period* is the amount of time a model needs to run before statistical data collection is valid. The Clear Statistics block is used to eliminate the statistical bias caused by the warm-up period.
- The Random Seed Control block controls when and if the seeds used throughout a model are re-initialized. As discussed in “Confidence intervals” on page B165, this is particularly important when analyzing model results using the “stats” blocks.

BPR Chapter 4: General Modeling Concepts

This chapter discusses general concepts that apply to the model as a whole. Time units, the length and number of simulation runs, model verification, and model validation will all be considered.

Time units

As described in Chapter 7 of the main Extend manual, there are two ways to deal with time units in Extend:

- Use a generic time unit in the model and in each block.
- Establish a global time unit for the model and use any time unit to define time-based parameters in the individual blocks (local time units).

This section assumes that a global time unit (a unit of time other than “generic”) has been established. You do this by choosing a global time unit in the Simulation Setup dialog from the Run menu.

The default time unit

Once you select a global time unit, all time-based parameters in the blocks in your model will be set to the *default time unit*. The default time unit will always be equal to the global time unit for that model. For example, if you change the global time unit from “hours” to “minutes”, all time units set to the default will also change from “hours” to “minutes”. The default time unit is denoted by an asterisk (*) next to its name in a block’s dialog, as shown below:

Delay: **hours***

Time-based parameter set to the default time unit

All time-based parameters for any new blocks added to the model will also be initially set to the default time unit.

The default time unit provides consistency when adding new blocks to the model, since every new block will initially be using the same time unit. It also allows users to easily convert a model built

using generic time units to a model using a global time unit, without affecting the simulation results. You do this by selecting a global time unit in the Simulation Setup dialog.

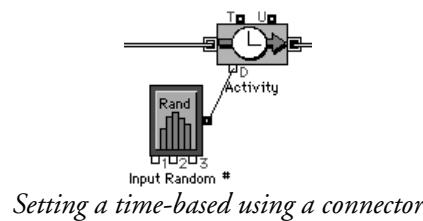
However, in most cases you will not want to keep all parameters in default mode. Explicitly selecting a time unit (even the same unit of time as the global time unit) for each local parameter in the model is a safer choice. Then, if you subsequently decide to change the global time unit, you will not accidentally affect the simulation results.

For example, assume you initially set your model's global time unit to "hours" and that you specify 2 as the delay time for an Activity, Delay block. However, instead of explicitly selecting "hours" as the local time unit for the activity block, you leave it in default mode (in this case, "hours*"). If you later change the global time unit from hours to days, the delay time for the Activity, Delay will become 2 *days*, adversely affecting simulation results. By explicitly selecting a local time unit (a time unit without an * after it) for all time-based parameters, you can prevent this type of error.

Setting time-based parameters using connectors

Some time-based parameters can be set using a connector value. For example, you can set the delay time of an Activity, Delay block by connecting an Input Random Number block to its "D" connector, as shown below:

BPR



Setting a time-based using a connector

In these situations, the value sent to the input connector must be defined in the time unit specified in the receiving block. For instance, assume that you want the delay in the Activity, Delay block to be approximately 30 minutes. If the local unit of time for the delay is minutes, you would set the Input Random Number block to generate numbers with a mean of 30. However, if the Activity, Delay block used hours as its local time unit, the Input Random Number block should be set to generate numbers with a mean of 0.5.

It is possible that you may be in a situation where you set numbers in one column of a block based on the time unit for that block, and set numbers in another column based on the time unit for a second block. An example of this is described in "Generating arrivals with custom intervals" on page B74.

The length and number of simulation runs

An important consideration when building a model is *how long and how often should the simulation be run?* Your entries for the "end time" and "number of 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. For more information, including how to calculate the number of runs and the run length, refer to *Simulation Modeling & Analysis* and *Improve Quality & Productivity with Simulation*; these books are listed in the section “Further reading” on page B10.

Terminating systems

Some systems obviously lend themselves to a natural determination of end time. For instance, most service operations and job shop projects 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 which 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.

BPR

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 B165, 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 exam-

ple, 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 2 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 your model simulates processes operating at their normal or steady-state level. In your 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 using blocks in the Statistics library (as discussed in “Clearing statistics” on page B168), or run the simulation for a very 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 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, you would typically model the 8 hours that a bank was open 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) for a total of 8 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 the 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.

There are several excellent sources for determining the length and number of simulation runs. For more information, refer to the simulation and statistics texts mentioned in “Further reading” on page B10.

Model verification

The process of debugging a model to ensure that every portion operates as expected is called model verification. In the Tutorial, you performed part of this verification process by building the model in stages and with minimal detail, then running it at each stage to observe the results. A common verification technique could be termed *reductio-ad-absurdum* (reducing to the absurd) since you reduce a complex 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 which interact to see how they run on their own

BPR

Other methods for verifying models include making sure that you can account for all the items in a model, animating the model or portions of the model, or using diagnostic blocks from Extend’s libraries (there are several debugging-type blocks described in the “Debugging hints” section of Chapter 7 in the main Extend manual.) For more information, see “Measuring and verifying simulation results” on page B167.

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 reason-

able. While you would normally compare critical performance measures, it is also sometimes helpful to compare nonessential results which 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 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.

For more information, see “Measuring and verifying simulation results” on page B167.

BPR Chapter 5: Items and Informational Values

As discussed in the Introduction to this manual, items are what flow through the model. Informational values provide information about items and model conditions.

An *item* is a process element or object that is being tracked or used in a model. For example, items can be parts, customers, workers, telephone calls, data packets, and so on. Items are individual entities and can have unique properties that are specified by their attributes and priorities (discussed later). An item can only be in one place at a time. Items change state (physically move, are transformed, or are delayed) when events occur, such as a part being assembled, a customer arriving, and so on. In discrete event models, items arrive, are delayed and transformed, then exit the model.

Values provide information about items and about model conditions. Values tell you the number of customers waiting in a line, how many applications have been approved, and how frequently data packets are sent. They also report delay time, operation utilization, and costs of processes. When you use a plotter in a discrete event model you are plotting information about items, not the items themselves. For example, when you connect from the “#” connector on an Exit block to the input of a plotter, you are displaying the time that each item left the model and how many items have exited.

Note Do not confuse informational values with *item Values* such as the “Value of item (V)” set in the dialog of the Generator block from the Generators submenu of the Discrete Event library. Each item can be cloned into duplicates of itself, for example to represent a group of items arriving at the same time. This is known as the number of items or *item Value*. See “Choosing an item’s Value” on page B73 for more information about item Values.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Change Attribute	Attributes	Discrete Event
Generator	Generators	Discrete Event
Get Attribute	Attributes	Discrete Event
Input Data	Inputs/Outputs	Generic
Input Function	Inputs/Outputs	Generic
Input Random Number	Inputs/Outputs	Generic
Import	Generators	BPR
Labor Pool	Resources	BPR
Measurement	Attributes	BPR
Program	Generators	Discrete Event
Repository	Resources	BPR
Set Attribute	Attributes	Discrete Event
Set Attribute (5)	Attributes	Discrete Event
Set Priority	Attributes	Discrete Event

BPR

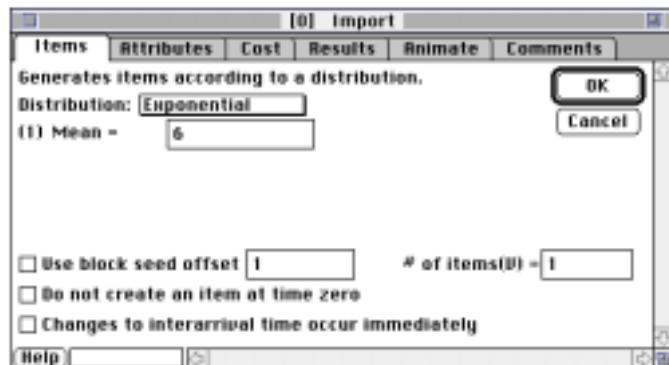
The models discussed in this chapter can be found in the “Examples \ BPR \ Items” folder.

Item generation

You can specify how often items arrive using the Import, Input Data, Generator, and Program blocks. The interval between item arrivals determines how often events occur. This interval is known as the *inter-arrival time*.

Generating arrivals at random intervals

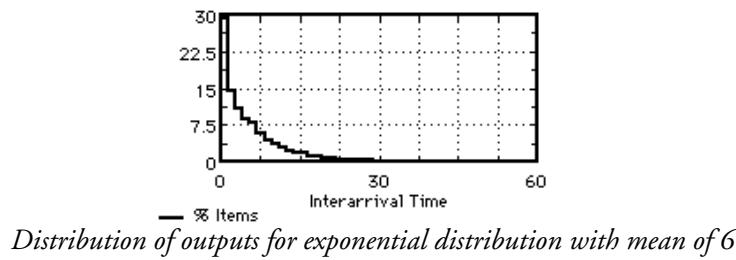
The Import block is the most common method for generating random arrivals; its dialog contains a popup menu with several distribution choices.



Import dialog showing exponential distribution with a mean of 6 (Macintosh)

Each of the distributions is described fully in the Import block's Help. In the Import block's dialog, the parameter arguments for each distribution are labeled (1) for the first argument, (2) for the second argument, and (3) for the third argument, if used. For instance, with the exponential distribution selected above, the first argument is the "mean". Because the second and third arguments are not used, the input boxes for those parameters do not show up in the dialog.

When you choose a distribution in the dialog of the Import block, you are specifying both the interval between item arrivals and the characteristics of the rate of arrival. For example, when you choose an exponential distribution with a mean of 6, items will arrive approximately every 6 time units for the duration of the simulation.



Choosing an item's Value

BPR

You can choose that one item is input at each arrival event – this is the most common case and the Import defaults to this option. Alternately, you can specify that a number of items are released at each event. You do this by using the "# of Items (V)" option in the dialog, changing the item's Value to a number greater than 1.

For example, assume the simulation time units are in minutes and you want to show that one customer arrives approximately every 6 minutes. Select the exponential distribution, enter a 6 in the entry box labeled "(1) Mean", and leave the "# of Items (V)" set to 1, as seen in the picture above. To show that 3 customer/items arrive every 6 minutes, keep the settings at exponential with a mean of 6, but change the "# of Items (V)" entry to a 3. When you do this, the block will output one item with an Item Value of 3 approximately every 6 minutes.

As discussed in the main Extend manual, the blocks that follow the Import block determine how the item with a Value greater than 1 is treated. For instance, if an item with a Value of 3 goes directly into a queue, or resource-type block, it will be split into three items each with an item Value of 1. However, if the item goes directly into an activity-type block, it will be treated as a single item with a Value of 3. In most cases, you will want to follow the Import block with a queue or buffer, which will decompose the item into three separate items. For more details, see the main Extend manual.

Note In most cases, you probably will not want to generate more than one item at each event. For example, rather than inputting 3 items every 6 minutes as discussed above, you would probably want to generate 1 item every 2 minutes. This is because it is not common to see three

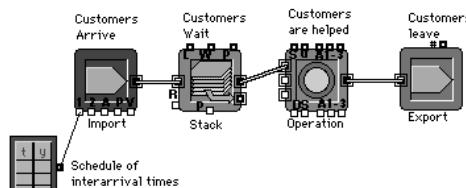
customers arrive at exactly the same time; customers are more likely to arrive at slightly different times.

When building models, it is common to use a constant or uniform (integer or real) distribution in the early stages 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 your real-world processes.

Generating arrivals with custom intervals

You might want to show items arriving randomly where you customize some aspect of the distribution. You can do this, using an Input Data block to modify one of a distribution's arguments.

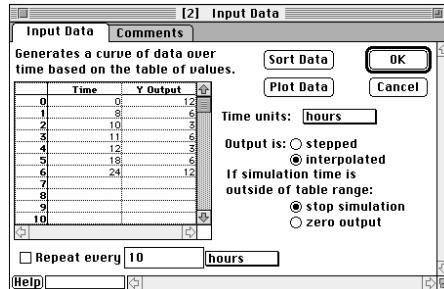
In the simplest case, you use an Input Data block to specify one or more of the parameter arguments for the distribution selected in the Import block. This is an example of dynamically changing a value, as discussed in “Changing parameters dynamically” on page B164. For example, you can use the Input Data block to specify the mean of an exponential distribution so that it changes based on the time of day:



Custom intervals model

In the dialog of the Import block, choose the exponential distribution so that items (in this case customers) arrive exponentially. Notice that the mean of the distribution is specified by the parameter labeled “(1) Mean =”. When you connect the output of the Input Data block to the “1” input connector of the Import block, you change the mean of the distribution, overriding the dialog parameter. This allows you to dynamically change the average time between arrivals (the inter-

arrival time). Notice that a smaller mean value indicates that there is less time between customer arrivals, so they arrive more frequently. The Input Data block dialog is:



Scheduling the inter-arrival time (Macintosh)

The table in the Input Data block is used to define the output value (“Y Output” column) at a given simulation time (“Time” column). The time units popup menu represents the time unit used to define the values in the “Time” column and is set to hours, therefore this block is set up to output values from time 0 to 24 hours. This corresponds to the simulation start and end times specified in the Simulation Setup dialog.

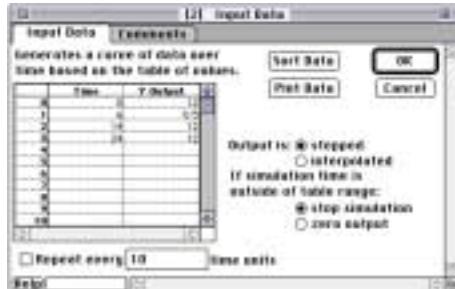
In this model, the “Y Output” values are sent to the Import block and are used to set average time between arrivals. As noted in “Setting time-based parameters using connectors” on page B66, when a value is passed to a block’s input connector to be used to set a time parameter (as in this case), the value sent must be defined in the time unit that the receiving block expects. Since the Import block is using minutes as its local time unit, the “Y Output” values of the Input Data block must also be stated in minutes. For instance, the value of 6 represents a mean or an average of 6 minutes between arrivals. The values in the Input Data dialog cause parts to arrive more frequently from time 10 until time 12, since the mean is smaller for that period.

BPR

Note Do not set the interarrival time to “0”. The Import block will warn you if you make this modeling error.

To avoid unexpected results, it is important to understand what happens in the Import block when you vary the mean of the arrival intervals over time. The Import’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 Import 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 Import block will not react to changes in the input parameters. This can cause unexpected results if the inputs change drastically, as in the following example.

If you connect an Input Data block to the “1” input connector of an Import block, the interarrival mean (average time between arrivals of items) will vary according to the table:



Dialog of Input Data block

Both the “Time” and “Output” columns 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. In this example, it is possible for an item to be generated at time 0 with an arrival time of 14. The Import will therefore ignore the fact that the mean has changed to 0.5 between hours 6 and 14, and the number of arrivals will be much less than expected.

BPR

Within the dialog of the Import block, there is a check box labeled “Changes to interarrival time occur immediately”. When checked, it will cause the Import block to immediately respond to changes to any input parameter. If you do this in the above example, the Import will recognize that the mean has changed from 12 to 0.5 at time 6. A new arrival time will be calculated by generating a new random number using the new input values and interpolating from the original arrival time.

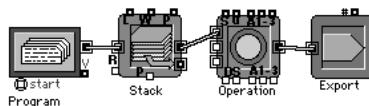
Scheduled arrivals

You can easily schedule item arrivals to occur at fixed intervals using the Program block from the Discrete Event library. For example, assume you receive 500 orders a week, but that almost all of your orders are received on Wednesday. You can enter the arrival times (“Output Time”) and the number of orders arriving (“Value”) in the dialog of the Program block, a portion of which is below:

Row	Output Time	Value	Priority	Attrib. value
0	1	50		
1	2	50		
2	3	300		
3	4	50		
4	5	50		
5				
6				
7				
8				
9				

Scheduling the arrival of orders

When you use the Program block, you enter the actual arrival times rather than entering the time between arrivals as was done with the Import block; the Program block automatically calculates the time between arrivals based on your entries. In this example there are five arrival events (one event each on days 1 through 5), each with an item value of either 50 or 300. The table indicates that on the third day (Wednesday) you receive 300 orders while you only receive 50 orders on each of the other days. Remember that, since the Program block always pushes items, you would follow it with a Stack or Repository to avoid losing items, as shown in the model:



Scheduling fixed interval model

Like the Import block, the Program block outputs items with values greater than 1 as a batch of items all arriving at the same time. When the batch item goes to a Stack or Repository block, it becomes multiple copies of itself. For example, as each order item is input from the Program block to the Stack block, it will become either 50 or 300 orders, depending on its Value.

The “start” connector

The Program block has a *start* connector that you can use to control the output times of the data entered in their dialogs. The relative timing of the schedule depends on whether or not the “start” connector is connected.

BPR

- If the “start” connector is not connected, the timing of the schedule entered in the dialog is absolute simulation time compared to the time entered in the Simulation Setup dialog. For example, start time 2 in the Program dialog would occur at simulation time 2, as long as simulation time 2 happens. If the starting time entered in the Simulation Setup dialog is 0, the Program would begin 2 time units after the simulation began. However, if the starting time entered in the Simulation Setup dialog is 4, the program at time 2 will never occur.
- If the “start” connector is connected, such as to a Decision block from the Decision submenu of the Generic library, the time entered in the dialog is relative to when the “start” connector is activated. For instance, assume simulation starting time is 0 and the Program block is scheduled to start at time 11. If the “start” connector gets activated at simulation time 5, the Schedule block would start its program at time 16 (5 plus 11).

The “start” connector is activated whenever it gets an item or sees a true value (a value >0.5). Once “start” is activated, it causes the entire schedule to happen. If “start” is activated while the schedule is happening, the new starting message is ignored.

Attributes

Attributes play a very important role in business process simulations. As discussed in the Extend manual, an attribute is a quality or characteristic of an item that stays with it as it moves through

the model. Each attribute consists of a name and a numeric value. The name identifies some characteristic of the item (such as “CallLength”) while the value gives the dimension of the characteristic (such as “15”).

You can attach many attributes to any item flowing through a simulation. You use the Import, Operation, Set Attribute, Set Attribute (5), Change Attribute, blocks to set and modify attributes. The Measurement and Get Attribute blocks can be used to check the values of the attributes on the items flowing through it. You use the Queue, Attribute and Queue, Matching blocks (from the Queues submenu of the Discrete Event library) to release items from queues based on attributes. Most of the resource-type blocks let you specify default attributes for items in their dialogs.

In addition to the attributes that you can set yourself, Extend has two system attributes, “_cost” and “_rate”. These attributes will only appear in the attribute popup menu if one or more cost rates has been defined in the model. Since they only apply to costing, these two attributes are discussed in “Working with cost data” on page B150.

Setting attributes

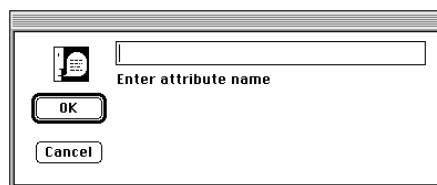
BPR

You can attach many attributes to any item flowing through a simulation. There is a limit of 300 attribute names for the entire model. If you try to add a new attribute that causes the number of attributes to exceed the limit, Extend will warn you.

Attributes are meant to be unique; if you attempt to add a new attribute with exactly the same name as an existing attribute, Extend warns you that the name already exists. Attribute names are not case sensitive (“Type” is equal to “type”). Attributes can be up to 15 characters in length.

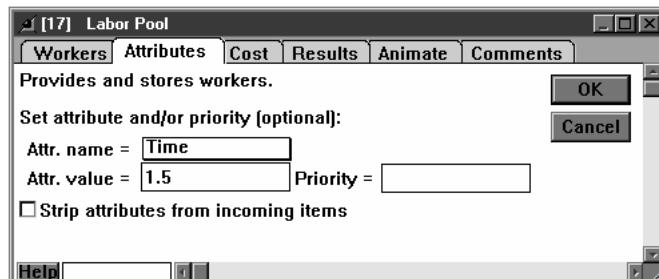
The easiest method for setting item attributes is to use the Import, Operation, Repository, or Labor Pool block. In the Attribute tab of these blocks, you can select an attribute from a popup menu and type in the attribute’s value.

All the attributes within the model are accessible from popup menus in the block dialogs. To create a new attribute, select “New attribute” from the attribute popup menu and in the New Attribute dialog, type the name of the new attribute in the New Attribute dialog.



New attribute dialog (Macintosh)

To set an attribute, select the attribute in the popup menu (or create a new attribute), then enter the value of the attribute into the box labeled “Attr. value =”, as shown below. The attribute information indicated in the dialog will be assigned to that item created in the block.



Setting the attribute of an item in the Labor Pool dialog (Windows)

To set attribute values dynamically, use the value input connectors on the Operation block to override the attribute value set in the dialog. You can modify the values for up to 3 attributes using the “A1-A3” value input connectors on the Operation block. This is often done using an Input Random Number block or Input Data block.

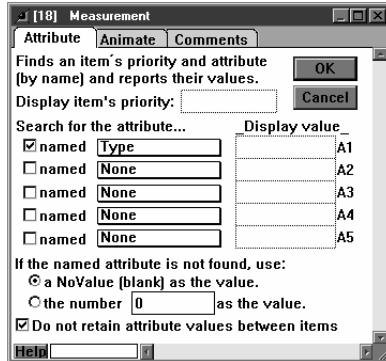
Note The attribute value for any attribute set in the Operation Variable block will automatically be the same as the quantity of items required for transformation; you do not have the ability to choose the attribute value in the Operation Variable block.

BPR

Reading attributes

You read an item’s attribute name in order to manipulate the item based on the attribute value, usually to route it or to process it. The most common method for reading attributes is to select the attribute by name from the list in the popup menu in the Measurement block. The Measurement block reads the attributes value as items pass through it. The attribute value found is output at the corresponding *A* output connector, and can be used by other blocks to determine the disposition of that item. If the attribute is not used by an item, you can specify in the Measurement dialog

that it output a NoValue (blank) or some other number entered in the dialog. The dialog of the Measurement block is:



Dialog of Measurement block (Windows)

Using attributes

BPR

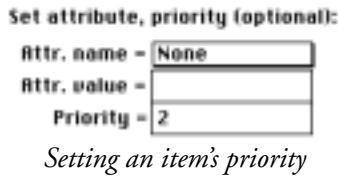
Attributes are commonly used to specify amount of work time required, routing instructions, operation times, item type, or part quality in statistical process control. They are also used to reflect progress in the amount of work done, cumulative values, or a change in status or quality. The Operation and Operation Variable blocks can use attribute values to determine the amount of operation time needed on an item. They can also modify existing attribute values.

Attributes are useful for assigning properties to items, then using those properties to specify a processing time or to route the items for processing. For example, you can use attributes to specify things such as “needs final check” or “ship unchecked”. There are two ways to set attributes for this:

- Set an attribute value to the amount of time it takes to process the item, where 0 means that the item doesn't need any further processing.
- Set an attribute value so that it dictates the route. Set an attribute value to a yes-or-no (1 or 0) value indicating whether or not a particular process is applied to the item.

Priorities

Priorities allow you to specify the importance of an item. The top priority has the lowest value, including negative values. You can assign priorities to items and manipulate them based on those priorities. For example, you can use the Repository block to assign priorities to items:



Setting an item's priority

You usually assign priorities as items enter the model from the Import block, then have all items enter a queue or buffer such as the Stack block. Priorities are particularly useful when you want to examine a population of waiting items and determine their order for processing. For example, you might have a step in a process where a worker examines the pending job orders and chooses the one that is the most urgent.

In Extend, items can only have one priority. (If you need multiple levels of priorities, you should use attribute values instead.) When a new priority is added to an item that already has a priority, the new priority is used. When items are batched, the highest priority of the items prevails in the resulting batched item.

BPR

Blocks in the BPR and Discrete Event libraries pass items' priorities with the items. Most of the resource blocks let you specify default priorities for items. The Repository block assigns a priority to an item as it passes through, while the Import block assigns the priority when the item is created. You can view an item's priority with the Measurement block. If "Priority queue" is selected in its dialog, the Stack block will pass out highest priority items first.

Note 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 Stack block if they have to wait there with other items.

BPR Chapter 6: Queueing

Queues provide buffers, or lines, for items awaiting further processing. The Stack block in the BPR library provides different types of queues or waiting lines. This chapter discusses the use of these queues.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Decision (2)	Routing	BPR
Import	Generators	BPR
Set Priority	Attributes	Discrete Event
Stack	Queues	BPR

Scheduling algorithms

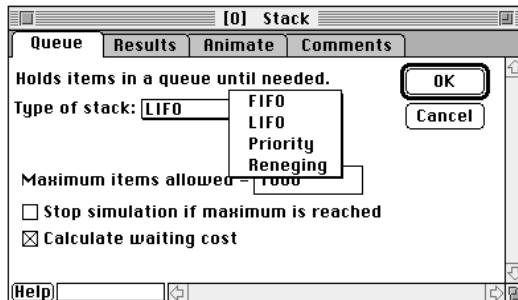
There are several predefined scheduling algorithms, also known as *queueing disciplines*, in the Stack block. The FIFO option in the Stack block represents a first-in, first-out or first-come, first-served queue. The LIFO option represents a last-in, first-out queue. The Priority option reads item's priorities and passes the items out highest priority (lowest number) first. The Reneging option allows you to specify how long an item will wait in the queue before it reneges, or prematurely leaves. In the Discrete Event library, the Queue Attribute block and Queue Matching blocks allow you to define custom scheduling algorithms based on item attributes.

Queueing considerations

Once items are generated for the model, it is common that they will be held in a queue or buffer block. The models discussed in this chapter can be found in the “Examples \ BPR \ Queues” folder.

Choosing a queue

As seen below, the dialog of the Stack block provides 4 choices of queues:



Stack dialog (Macintosh)

It is important to remember that, except for the FIFO choice, there must be other items in the queue at the same time for the queueing disciplines to have an effect. For example, if you choose the priority queue, and there is never more than one item in the queue at a time, the effect of choosing based on priority is negated.

BPR

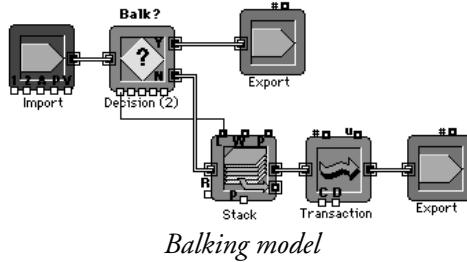
Blocking

Blocking occurs when an item is finished processing, but is prevented from leaving the block because the next process or operation is not ready to pick it up, or the next resource block is full. Blocking is most common in serial operations such as when there are several activities in a row performing tasks. In that case, each activity has the potential for blocking arriving items. Blocking also occurs when activities are not preceded by queues, 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 section “Processing in series” on page B103 illustrate potential blocking situations.

Balking

Sometimes customers enter a facility, look at the long line, and immediately leave. This is an example of *balking*. Balking is typically represented by having a Decision (2) block 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 block routes the item out of the model before it enters the queue.



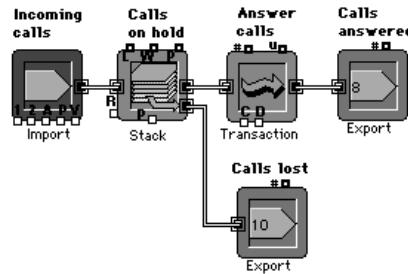
In the above example, customers will automatically exit if the length of the queue is 5 or greater. The Decision (2) block monitors the length of the queue. Within the dialog of the Decision block, an equation checks if the length of the queue is 5 or greater. If this is true, the item will pass through the top right output connector of the Decision block and pass to the Export block. If the queue length is less than 5, the item will exit the bottom right connector of the Decision block and enter the Stack block.

Reneging

Reneging occurs when an item, having already entered a queue, leaves before it reaches the output. For example, telephone callers who are put on hold will hang up without getting help if they feel they have waited too long for assistance.

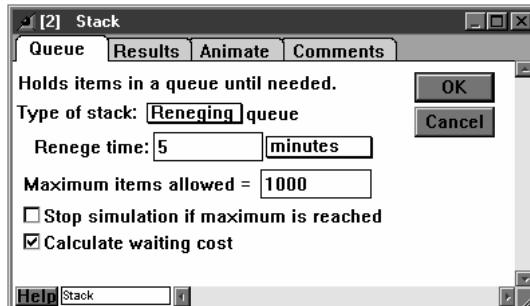
BPR

You can simulate this by selecting “reneging” in the “Type of stack” popup menu of the Stack block. In the following example, incoming calls wait on hold until a representative is available to assist them. If too much time passes before a caller is assisted, the caller will hang up and exit the model:



Reneging model showing callers hanging up if not assisted promptly

The dialog of the Stack specifies how long each caller will wait before they will decide to hang up:



Stack block in Reneging mode with “Reneging time” set to 5 (Macintosh)

To set the reneging time dynamically or based on model conditions, connect to the block's "R" input connector.

As seen in the Results tab, the Stack block automatically counts and reports how many items have reneged. Items that renege leave through the output connector on the lower right. Reneging items can be routed back to the original line, routed elsewhere in the model, or they can exit the model (as in the above example).

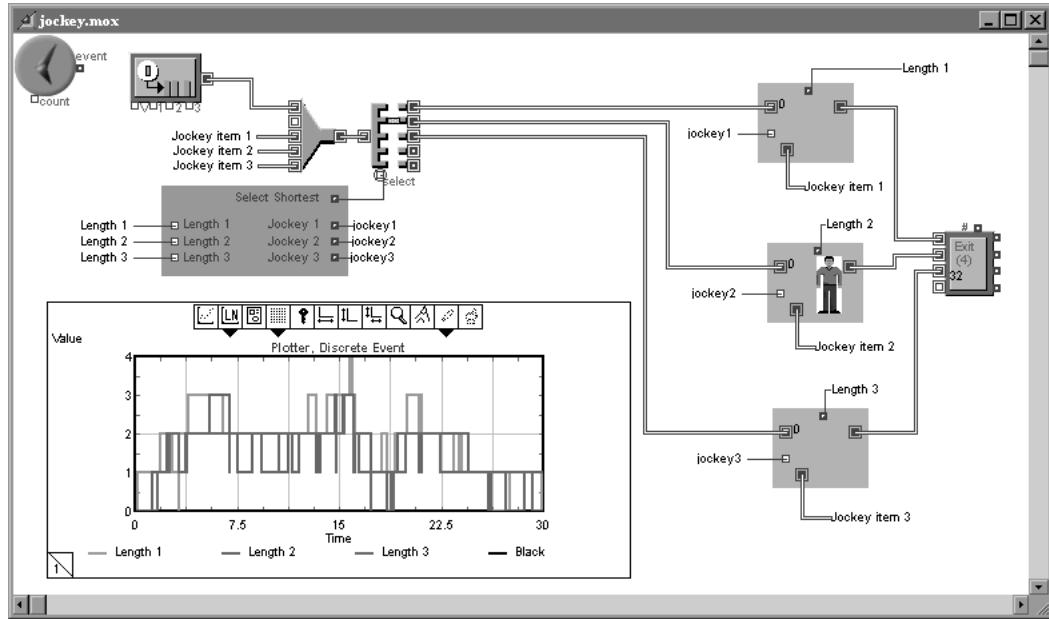
BPR

Jockeying

You can see a good example of jockeying if you go to the supermarket. Watch as people leave the end of a slow moving cashier's line to try and get onto a faster line.

The Stack block set to Reneging is useful in building a model of this type of behavior. Normally, items renege if they have spent too much time in the queue, but the Stack block has a connector

that can force reneging of the last item in the queue. The *Jockey BPR.mox* model is a good example of this:



BPR

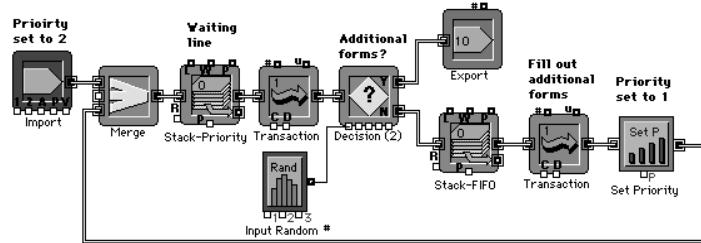
Jockey BPR.mox model showing hierarchical blocks that calculate the shortest queue

The leftmost hierarchical block forces the longest queue to renege its last item and then routes that item to the shortest queue.

Priority queues

When the block downstream can accept an item, a priority queue will search through the contents of the queue and release the item with the highest priority. The Stack block can be set to operate as a priority queue by selecting “priority” in the “Type of stack” popup menu. In the following example, customers enter the model and immediately have their priority set to 2 by the Import block. Customers wait in line (Stack block set as a priority queue) to be assisted by a clerk. After being assisted, some customers will need to fill out additional forms. After filling out the additional forms, these customers are allowed to jump to the front of the line to finish their transaction when the next clerk becomes available. This is modeled by setting the customer’s priority to 1 before re-routing the costumer back to the waiting line. When a clerk is available to assist a customer, the Stack block will release the item with the highest priority. In this case, any customer waiting to be re-assisted will be released first. 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 first Stack block.



Priority model - top priority given to items waiting to be re-machined

BPR Chapter 7: Routing

As you build models, you will frequently encounter situations where you want to manipulate items coming from several sources or send items out on various paths. Depending on your purpose, there are several methods for accomplishing this. Remember, an item can only be in one place at a time.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Activity, Service	Routing	Discrete Event
Catch	Routing	Discrete Event
Decision	Decisions	Generic
Decision (2)	Routing	BPR
Decision (5)	Routing	BPR
Input Random Number	Inputs/Outputs	Generic
Max & Min	Math	Generic
Measurement	Attributes	BPR
Prioritizer	Routing	Discrete Event
Stack	Queues	BPR
Throw	Routing	Discrete Event

The models discussed in this chapter can be found in the “Examples \ BPR \ Routing” folder.

Items from several sources

Depending on your modeling needs, you may want to merge different streams of items into one stream of individual items or join separate items into a single item.

Overview of merging and joining multiple input streams

- To *merge* items from several sources into one stream, where each item remains separate and retains its unique identity, use the Merge block from the BPR library or the Throw and Catch blocks from the Discrete Event library. You then typically direct the resulting single stream into one queue. For example, you can use this technique to represent telephone and mail orders being directed to the order entry department or workers returning to a Labor Pool from differ-

ent parts of the process. The model described in “Simple routing” on page B92 provides an example of using the Merge block. “Using the Throw and Catch blocks” below illustrates using a Catch block to merge multiple streams of parts into one.

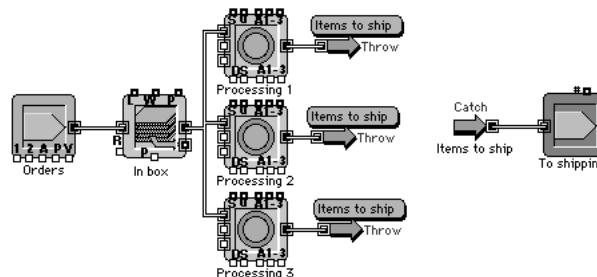
- To join items from various sources and process them as one unit, use blocks that *batch* such as the Operation block described in “Batching” on page B119. This is common when modeling shipping processes or packaging operations where subassemblies are joined together. It is also used when two or more 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 Merge block, where each item remains separate and is separately processed.

Using the Throw and Catch blocks

Most of the examples in this chapter will discuss routing items using connections to blocks which are nearby and at the same level of hierarchy. Sometimes, especially in large models, this is not convenient. The Throw and Catch blocks (from the Discrete Event library) 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 the Throw and Catch blocks are used as an adjunct to routing, not a replacement for the methods described in this chapter.

BPR

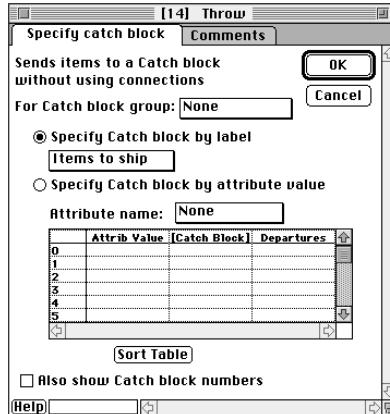
Throw and Catch blocks pass items between blocks without using connections and can even be used deep within nested hierarchical blocks to send items to other hierarchical blocks. For that reason, they are often more convenient to use than the Merge block. In the following example, three Throw blocks route items to a Catch block which sends the items “To shipping”:



Model using Throw and Catch blocks to route items

Within the dialog of the Throw block, you have the option of routing all items to a single Catch block specified in the popup menu (as shown below) or routing items to different Catch blocks

dependent on the value of a specified attribute (see “Using the Throw and Catch blocks” on page B97).



Throw block dialog showing Catch block selected (Macintosh)

Within the popup menus of the Throw block’s dialog, the block labels of different Catch blocks will appear, allowing you to select the appropriate Catch block. Therefore as you add Catch blocks to your model, you must add a label to the lower left hand corner of the Catch block’s dialog. Only after a label has been added will it appear within the Throw block’s popup menus.

BPR

Items going to several paths

Items do not usually just flow from a Stack block to an Operation block to an Export block. Real world situations call for routing items for processing, checking, approval, and so forth. Your models can cause items to be routed based on a logical decision, a characteristic of the item, the time of day, and so on. You can even cause a new system to start operating based on model factors, for example if there are too many items for the current systems to handle. Some ways items can be conditionally routed are given in “Routing rules” on page B24.

Overview of parallel processing, unbatching, and selecting multiple output streams

- Taking a stream of items and routing them to a group of processors or operations is called *parallel processing*. In parallel processing, each item is handed off to one (and only one) of several operations. The logic that determines which operation the item is routed to can be simple (call is answered by the first available clerk) or it can be complex (requisitions over \$1000 are approved by Supervisor A). The models in “Simple routing” on page B92, “Explicit parallel processing” on page B94, and “Parallel processing based on an item attribute or priority” on page B95 provide examples of routing to parallel processes.
- For situations where one item is separated into its component items, each of which goes on a separate path, use the Operation Reverse block described in “Unbatching” on page B121. For example, you might receive a shipment of furniture consisting of 8 desks, 20 chairs, and 7 type-

writer returns, or a mail cart with 1000 pieces of mail. You use the Operation Reverse block to disassemble that batched item into its individual components, then route the items to appropriate destinations, as described later.

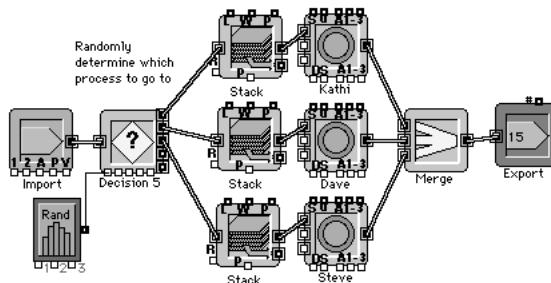
- The Decision (2) and Decision (5) blocks are useful for routing a stream of items to several paths based on some decision or logic. For instance, you can send all the packages that need rework to a repackaging station, and ship the remaining units. Or you can direct patients requiring further examination to the re-examination room. This chapter contains numerous examples of using the Decision (2) and Decision (5) blocks.

Simple routing

The simplest routing example is where a percentage or specified number of items are routed to one section of the model, while the rest are routed to another. An example is a formal processional line, where the first person in line goes to the left row of seats and the next person goes to the right.

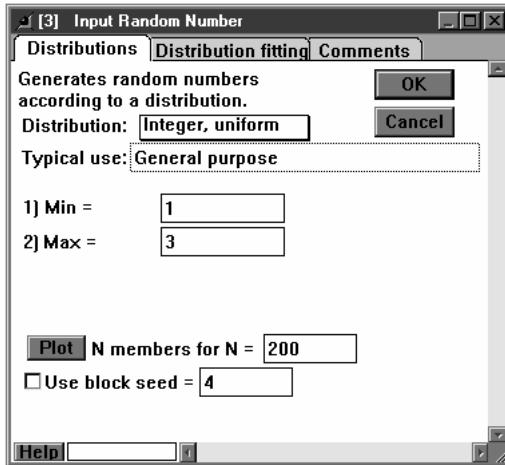
The decision blocks are most appropriate for routing a number of items into one path or another. You choose which output connector an input item should be routed to based on logic in the Decision (2) or Decision (5) block's dialog and model information provided by the value inputs.

To hand items to operations such that each operation has an equal chance of processing an item regardless of whether another operation is free, use the Input Random Number block (from the Generic library) and the Decision (5) block. An example of equalized selection among three Operation blocks is:



Parallel processing in equalized order

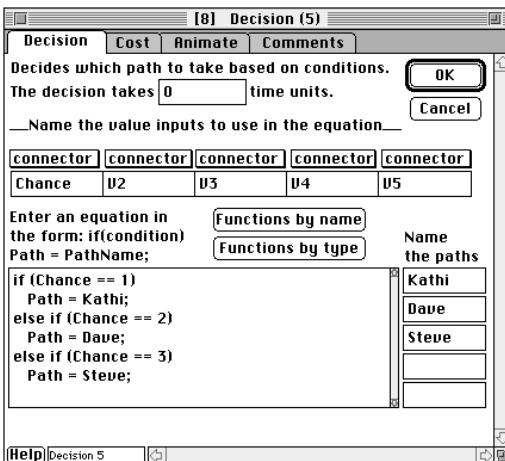
The Input Random Number block outputs the values 1, 2, or 3 with an equal probability of occurrence, as seen in its dialog:



Input Random Number dialog (Windows)

The Decision (5) block routes incoming items to the Operation blocks at an equal rate, where each Operation is selected based on the value the Decision (5) block sees. For example, each time the Decision (5) block sees a value of 1, it sends the item out its top output to the Operation block named "Kathi". The dialog of the Decision (5) is as follows:

BPR



Decision (5) dialog showing ordering (Macintosh)

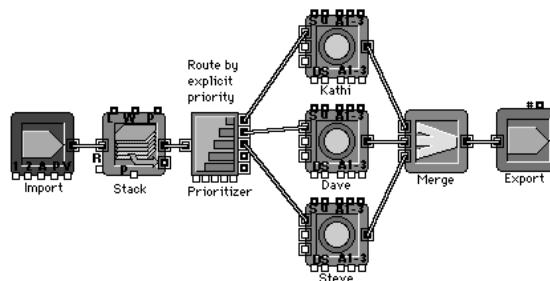
In the model, notice that there are queues in front of each of the operations. If you omitted the queues, there is a possibility that the items would be blocked, as discussed in "Blocking" on page B84. For example, if the top operation is due to receive an item, but is still processing, the

other operations will have to wait for items until the top operation finishes processing and pulls in the item.

Explicit parallel processing

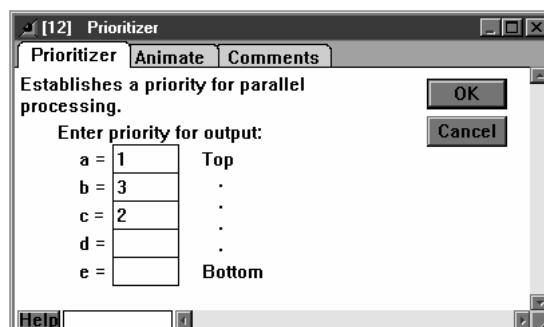
If you have several operations and you prefer certain ones to be active more than others, you can explicitly state which operations have a higher priority for items. This is common when you want to specify that one path is preferred or to avoid an uneconomical use of a resource, such as having a supervisor wait on customers.

The Prioritizer block from the Discrete Event library allows you to specify the priority of each output. Note that this is different from assigning a priority to an item since the Prioritizer block essentially prioritizes the path, not the item. For example, assume that you want Kathi and Steve to get most of the items for processing and Dave to only get items if both Kathi and Steve are busy. The model is:



Parallel processing in explicit order

In the dialog of the Prioritizer block, assign the highest priorities (which are the lowest numbers) to the first (top) and third outputs, and the next lowest priority to the second output (Dave):



Prioritizer block dialog showing output priorities (Windows)

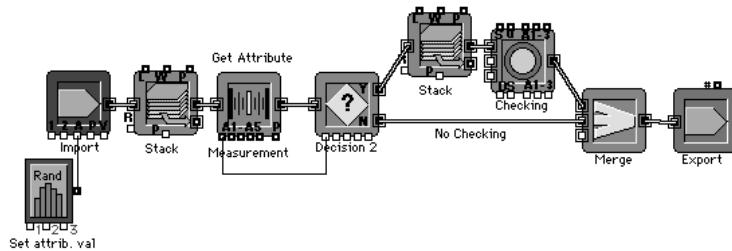
When you do this, Kathi has first priority on items. If Kathi is busy, Steve will get the item. Only if Kathi and Steve are busy will Dave get an item.

You can assign the same priority to multiple outputs. If you do this, however, the highest priority will go to the topmost output that is free. This is rarely what you want, since the outputs would not always have an equal chance of being chosen. A better method for assigning equal priority to two or more outputs is to connect an Input Random Number block to the value inputs of the Prioritizer block. Set the distribution of the Input Random Number block to be an Integer, Uniform distribution with a minimum of 1 and a maximum of 10. With this method, each output would then have an equal chance of being chosen.

Routing decisions based on attributes

You may want to route items based on some characteristic of the item, such as its size, quality, age, or requirements. To do this, assign an attribute to the item and read that attribute value to route the item.

For instance, to specify whether or not an item must have a process performed on it, set the item's attribute to a yes-or-no value using the Input Random Number block, as shown below:



BPR

Routing based on attributes

The empirical distribution in the Input Random Number block specifies that 25% of the items do not require checking (0 for attribute value) and 75% do (1 for attribute value). The Measurement block reads and outputs that value to the Decision (2) block. That block uses the 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 Merge block is used to combine both lines into one stream, exiting the simulation.

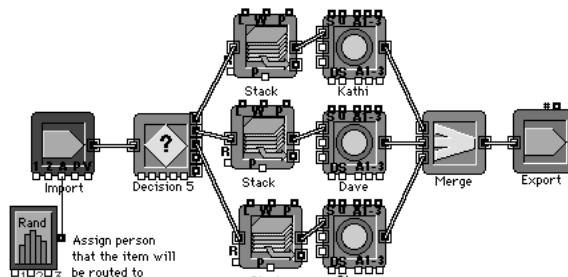
Notice that there is a Stack block between the Operation block (labeled "Checking") and the Decision (2) block. The Decision (2) block passes one item out at a time and the checking process takes some time. You need to put a queue or buffer in front of the Operation block so that items that do not require checking will not be blocked by an item waiting for processing.

Parallel processing based on an item attribute or priority

The same technique can be applied to choosing an activity based on an attribute or priority value. It is most common to use the decision blocks (Decision (2) and Decision (5)) or the Throw and Catch blocks to route items to parallel processes based on an attribute value.

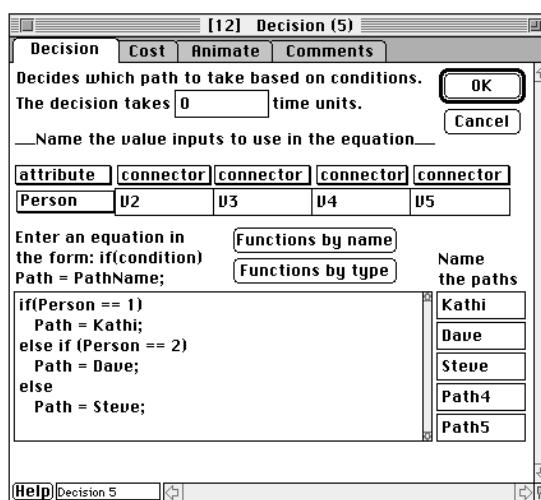
Using the decision blocks

The Decision (2) and Decision (5) blocks are useful for choosing which activity will process an item based on the item's attribute or priority value. For instance, assume you have an attribute value that gives the number of the activity that an item is to be processed by. The Decision (5) block can be used to read the attribute and send the item to the appropriate activity. The model is shown below:



Parallel processes selected by attribute value

In the model, the combination of the Import and Input Random Number blocks assigns an attribute value of 1, 2, or 3 to the items passing through, corresponding to the person who will process that item (1 is Kathi, 2 is Dave, 3 is Steve). Within the dialog of the Decision (5) block (shown below), you name the inputs to use in the equation.



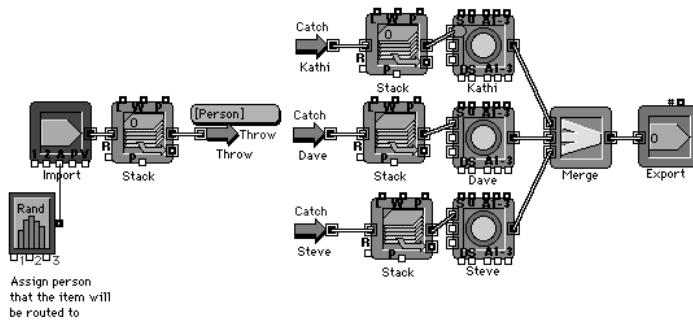
Choosing an output based on the select connector value (Macintosh)

The Decision (5) allows you to use connector values or attribute values as input to the equation. In this case, the only input is the attribute named "Person". As seen in the dialog, the top popup menu for the first input is set to "attribute" indicating that the input value will come from an attribute. The popup menu below it is used to indicate which attribute the value will come from.

The equation indicates that the top output is selected if the value of “Person” is 1, the second output is selected by a value of 2, and so forth.

Using the Throw and Catch blocks

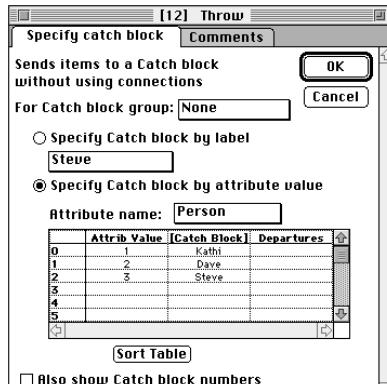
As described in “Using the Throw and Catch blocks” on page B90, the Throw block can be used to route items to a specific Catch block. The Throw block can also be used to route items to different Catch blocks depending on the value of an attribute. The previous example, built using Throw and Catch blocks as opposed to the Decision (5) block, is shown below:



Parallel processes selected by attribute value using the Throw and Catch blocks

In this example, the Throw block reads the attribute “Person” and routes the items to the appropriate Catch block according to the table in the Throw block’s dialog, shown below.

BPR



Dialog of Throw block (Macintosh)

To associate an attribute value with a specific Catch block, type the value into the “Attribute Value” column and select the appropriate Catch block using the popup menu in the “[Catch Block]” column.

Conditional routing

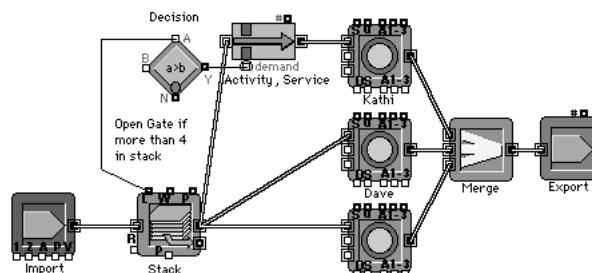
Sometimes you will want to route items based on the current conditions of the model. For example, you may want to monitor queue lengths to determine whether or not a machine will be brought on-line or to balance the use of parallel lines. Some typical conditional routing situations are discussed in “Routing rules” on page B24.

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 or based on some other factor such as the backlog of work. For example, you might have an operation where most of the processing is done by two clerks, but excess work is handled by a third clerk.

Extend can simulate this easily using the Decision block from the Generic library (note that this is not the same as the decision blocks in the BPR library) and the Activity Service block. The “L” output of the Stack block reports the number of items waiting to be processed. If this value is greater than a certain threshold, you can route some of the work to another operation. An example of this is:

BPR



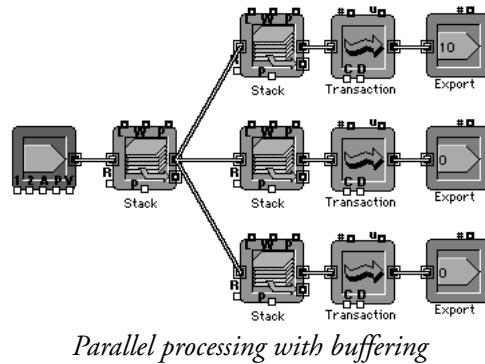
Routing based on model conditions

The dialog of the Decision block specifies that the “Y” output connector outputs a true value (1) when the value at the A input is greater than or equal to 5. This activates the “demand” connector so that the Activity Service lets items through to the top operation (until then, it will not accept items). When the Stack block holds 4 or fewer items, the Activity Service closes.

You could 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. For example, you did this in the “Support Services models” on page B14. However, one different result is that items were trapped in the shut down operation block until operations resumed.

Balancing multiple output lines

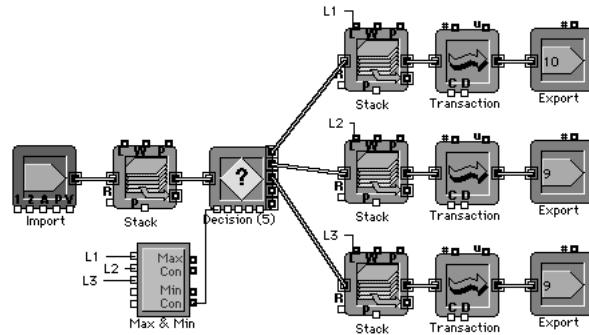
Parallel activities are often preceded by buffers before each activity, such as a staging area for each activity. This would look like:



Parallel processing with buffering

In this case, the first queue connected receives all the items until that queue reaches its maximum, then the next queue starts to fill (unless the first Transaction block kept up with the flow of items, in which case the next queue will never receive any items). This is rarely what you want.

To even out the use of the activities, you can check the length of each queue and give the next item to the queue that is shortest. The Max & Min block is excellent for this:



Line balancing - choosing the shortest queue

On the Max & Min block, the bottom *Con* output connector tells which of the inputs has the lowest value, indicating the shortest queue. The equation in the dialog of the Decision (5) block

decides which queue to route the next item to based on the value received from the Max & Min block (“line”).

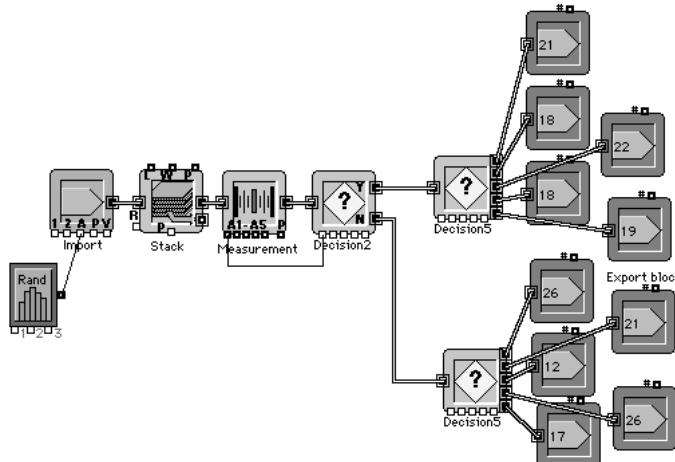
```
if(line == 1)
    Path = Path1;
else if (line == 2)
    Path = Path2;
else
    Path = Path3;
```

Equation from Decision(5) block

Extended routing

In some instances, you may want to route items to more than five destinations. For example, patients in an emergency room may require one of several different treatments: laboratory work, x-ray, cast room, burn treatment, poison center, operating room, and so on. To do this, you use a Decision (2) block to route to two or more Decision (5) blocks in two or more stages.

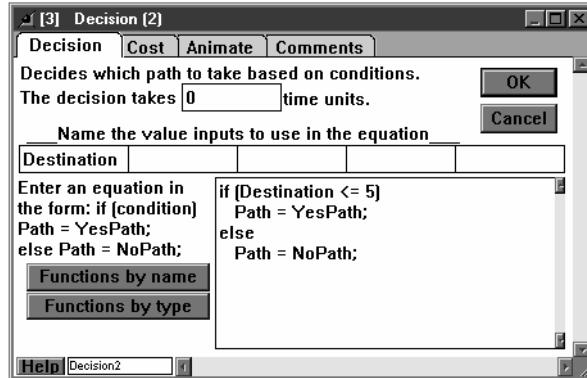
BPR



Routing to 10 destinations

The model uses item attributes to specify a destination for routing, where each item gets an attribute value from 1 to 10. The first Decision (2) block gets the attribute value from the Mea-

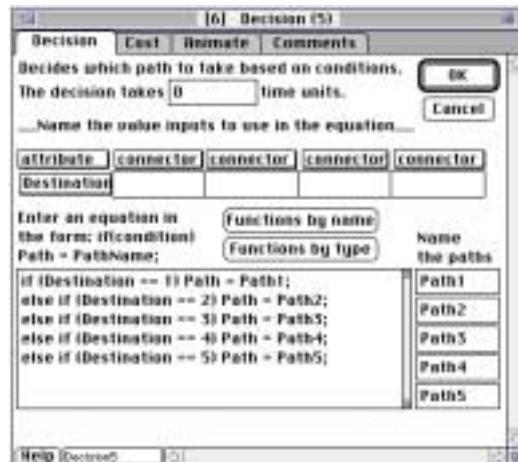
surement block and routes items with attribute values from 1 to 5 to the top Decision (5) block and items with attribute values from 6 to 10 to the bottom Decision (5) block:



Decision (2) block separating items into two paths (Windows)

The Decision (5) block has the additional ability to directly read attribute values from items that pass through it. Note how the attribute named "Destination" is read by the Decision (5) block and is used in the equation to route the items to the appropriate path:

BPR



First Decision (5) block dialog (Macintosh)

BPR Chapter 8: Processing

This chapter will discuss different ways to connect multiple activity blocks and how to control processing time and the availability of items and resources.

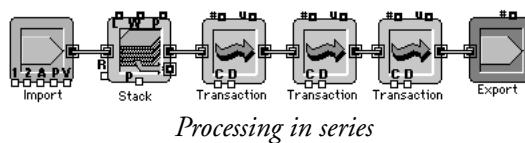
The following blocks will be the main focus of this chapter:

Block	Type	Library
Activity Service	Routing	Discrete Event
Decision	Decisions	Generic
Input Data	Inputs/Outputs	Generic
Input Random Number	Inputs/Outputs	Generic
Import	Generators	BPR
Measurement	Attributes	BPR
Operation	Activities	BPR
Operation Variable	Activities	BPR
Program	Generators	Discrete Event
Transaction	Activities	BPR
Transaction Preemptive	Activities	BPR

The models (and any sub-folders) discussed in the chapter can be found in the “Examples \ BPR \ Processing” folder.

Processing in series

Serial processing occurs when items flow in series from one activity to another. In this situation, each activity performs one required task, out of a series of required tasks, on the item. This is most common in processes that are heavily dependent on paper flow (such as order processing) or service-intensive situations (such as governmental processes). A simple example of serial processing is a loan approval process, where several tasks are performed before the loan is approved.



Since you have multiple activities in series, it is likely that items (in this case loan applications) will not be able to leave one activity because the next activity will still be busy (this is known as *blocking* and is described on page B84). Serial processes can cause the entire process to be slowed down to the speed of the slowest activity. If this doesn't accurately reflect your process, you can put a Stack block in front of each Transaction block to represent an in-box or some other form of holding area. Also, see "Processing in parallel" below.

Processing in parallel

It is common in business processes for there to be multiple activities working in parallel, each of which represents performing the same task. For example, you might have a 5-station order entry department, where each station can process any order that is received. With the BPR library, there are many ways to hand items off to parallel activities.

Remember that, unless items are purposefully duplicated, they can only follow one path at a time. Extend+BPR automatically represents the real-world situation that an item cannot be in two places at the same time.

Parallel processing using the Transaction block

BPR

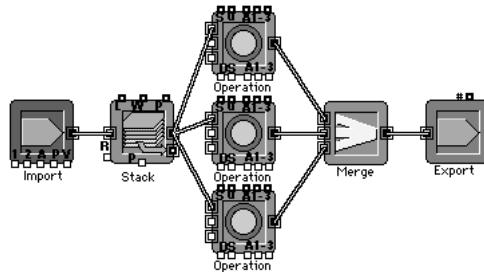
You can use the Transaction block to represent several processes that occur in parallel. This is handy in situations where you do not need to show each process as a separate block. For example, in BPR Chapter 2: Tutorial, the Transaction block represented two reviewers processing credit applications at the same time.

The Transaction block takes in items (up to a specified maximum) and processes them for a specified time, starting from when they arrive. The item with the earliest completion time is passed out first. For example, you would use this 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

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 Extend to pass items to the first available activity. However, if more than one activity is free when an item is

ready, it is unpredictable which block will get the item. For instance, a model where a Stack block holds items for three activities would look like this:



Simple parallel processing

The first free Operation block will get the item. If two Operation blocks are free when an item comes out of the queue, it is not clear which block will receive it.

Note Unless it is unimportant in your model, you should explicitly state the ordering for parallel processes. See BPR Chapter 7: Routing for examples of how to control the flow of items to parallel processes.

BPR

Setting the processing time

Activities, such as the Operation and Transaction blocks, involve a processing time or delay that represents the amount of time it takes to perform a task. The Decision (2) and Decision (5) blocks also allow you to specify a time that it takes to make the decision. The models discussed in this section show the many ways you can specify a processing time in Extend; they can be found in the “Examples \ BPR \ Time” folder.

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 can set the delay in the dialog of the activity block if the delay doesn’t change and you know how long it is.

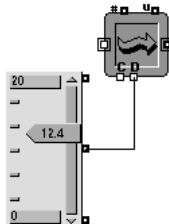
For example, if the task of entering an invoice always takes 5 minutes, you enter the value 5 as the processing time in the activity block’s dialog. If you do not know the exact amount of processing time, or if the delay varies based on some factor in the model, you can cause the delay to change dynamically using the “D” connector of activity-type blocks, or you can use activity blocks that interact with item attributes (such as the Operation or Operation Variable) and have the attribute value be the delay.

Fixed processing time

You can set the delay in the dialog of the activity-type block if the delay doesn’t change and you know how long it is. For example, if Transaction A always takes 5 minutes to process parts, you

enter the value 5 as the processing time in the dialog. This is most common when you are building the early stages of a model and are using constant parameters so you will get repeatable results.

Another method for having a fixed processing time is to connect a Slider control to the “D” input of an activity block, such as for the Transaction block below:



Slider control used to set processing time of Transaction block

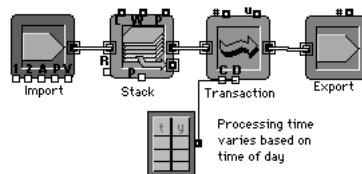
You can manipulate the Slider control with each simulation run, or within a simulation run, to see the effect of various processing times.

Scheduled processing time

BPR

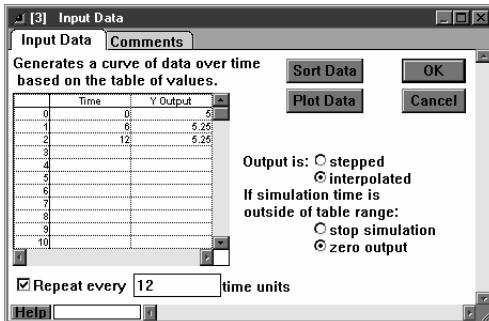
If you know that an activity takes a specific amount of time under most conditions, but takes another amount of time if the conditions are different, you can dynamically schedule the processing time as discussed in the section “Changing parameters dynamically” on page B164. This is common when simulating worker performance, where output could be a factor of time.

For example, assume that a clerk 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, you can connect an Input Data block to the “D” connector of a Transaction block, as shown:



Scheduled processing time

The time required to perform the task is entered in the “Y Output” column of the Input Data block’s dialog, as shown below:

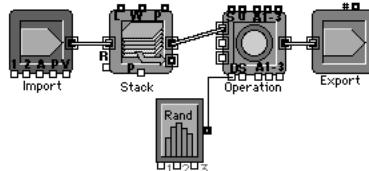


Entering the time to perform the task (Windows)

Notice that the time units for the Time column are in hours and that the time the worker takes to perform the task changes at 6 hours. Also notice that since the output of the Input Data block is connected to the “D” connector of the Station block, the Y Output column should be defined in the same time units that are set for the delay parameter in the Station block (this is, in minutes). See “Setting time-based parameters using connectors” on page B66 for a discussion on time unit consistency.

Random processing time

A common requirement for activities is to set a random processing or delay time. This is easily accomplished by connecting an Input Random Number block to the “D” connector of the Operation block:



Random processing time

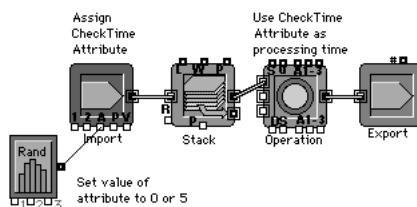
In the dialog of the Input Random Number block, select a distribution from the distribution popup menu, for example “normal”, and specify the value of the parameters, such as a mean of 2 and a standard deviation of 0.2. The processing time will then be normally distributed and the Operation block will process each item for approximately 2 time units.

Custom processing time

Attributes can be used to specify how long a specific item will be processed by a particular activity. The Operation and Operation Variable blocks can use attribute values as the processing time. You

can set the attribute in the dialog of the Operation block, then use that time for processing. Or you can specify that the processing time be based on some other, pre-existing attribute value.

In the simplest case, you set an item's attribute value to the desired amount of processing time, then use the Operation block to read the attribute value and process the item for that period of time. The model is:



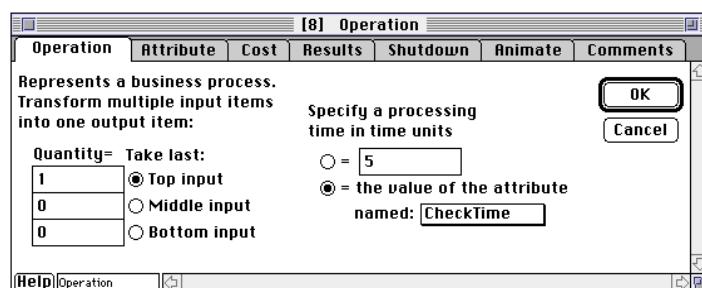
Custom processing time

For example, you can set the value of an attribute called "CheckTime" to the amount of time it takes to review an item. Items that need a final review have an attribute value of 5, for instance, and items that proceed without review have an attribute value of 0. In the dialog of the Import block, enter "CheckTime" as the name of the attribute that gets its value from the A input. That value (0 or 5) is provided by the Input Random Number block using a Empirical distribution where 25% of the items have a value of 0 and 75% have a value of 5, representing a 5-minute review process. The Empirical table from the Input Random Number block dialog looks like:

	Value	Probability
0	0	0.25
1	5	0.75
2		
3		
4		

Setting a custom processing time (Windows)

All items then go through the review step, represented by the Operation block. In its dialog, this block indicates that the processing time is "the value of the attribute named CheckTime":



Processing time based on attribute value (Macintosh)

As the simulation runs, the Operation block will display in its dialog the actual processing time it gets.

Of course, different items may have different values for their CheckTime attribute. The Operation will delay each item by whatever value you have assigned to that item.

Bringing an activity on-line

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 \ BPR \ Bringing On-Line” folder.

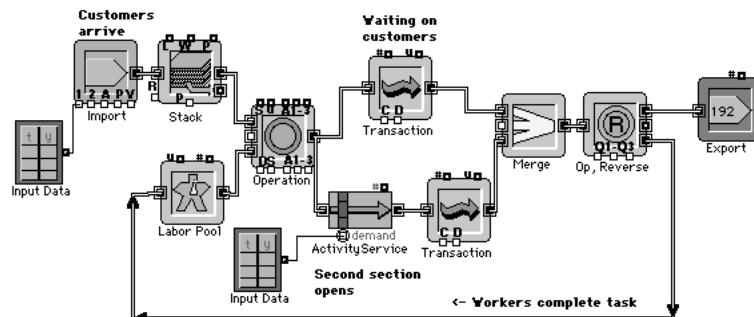
Please also see “Using the Shift block” on page B136 for examples of activity and resource allocation that are tied together and scheduled as “Shifts.”

Scheduling activities

Activities can be scheduled. This is most common when you bring an activity on-line based on the time of day. The following example shows you how to schedule the availability of a portion of an operation. For this example, assume that you are modeling a coffee shop that opens at 10 AM and closes at 6 PM. Customers arrive exponentially throughout the day, with most customers arriving during the lunch period, that is from 11 until 2. You have 8 service people and 2 dining sections available. The simulation time is in hours, and the simulation is run for 8 hours, from time 10 (10 AM) until time 18 (6 PM).

BPR

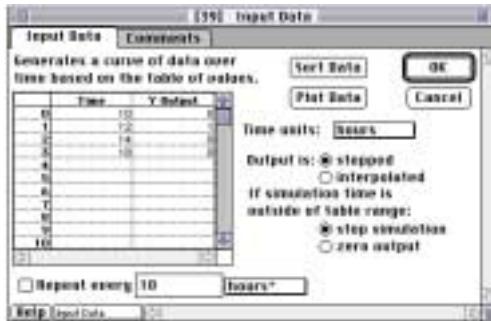
Assume that you have two service counters, but only open the second counter from 11 until 2 PM. You connect a Program or Input Data block to an Activity Service block, allowing customers access to the second counter only during certain hours. The model is:



Scheduling activities 1

The Activity Service block only allows items through when its “demand” connector is activated. You often do this by connecting a value connector, such as the output connector on the Input

Data block, to “demand”. As long as the value connector is true (outputs 1), the Activity Service stays open; when the value is 0, for false, it closes. The dialog of the Input Data block is:



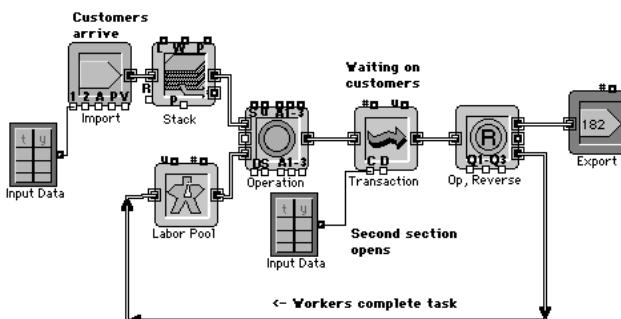
Opening and closing the counter (Macintosh)

From hour 11 to hour 14 the Input Data block will output a value of 1, telling the Activity Service to allow items to go to the second service counter.

BPR

Notice that the “demand” connector is a universal connector. The Activity Service’s universal connector treats item and value connectors differently. Value inputs cause “demand” to be activated as long as the value is true (1) and close when the value is false (0). Item inputs cause “demand” to accumulate a demand for a number of items.

In the above example, you could have alternatively used the “C” (change) connector on the Transaction block to control the capacity of the block to simulate the opening of the second counter. The resulting model would look like:



Scheduling activities 2: Modeling both counters using a Transaction block

In this model each counter has the capacity to serve 5 customers at once. By doubling the capacity of one Transaction block during the period between 11 am and 2 pm, you can model both

counters being open using just the single Transaction block. The portion of the Schedule block that controls the capacity of the Transaction is:

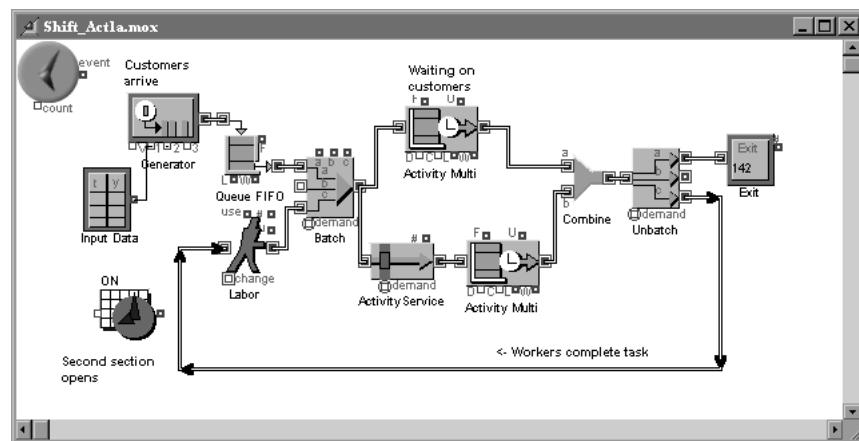
Time	Y Output
0	10
1	12
2	14
3	16
4	
5	
6	
7	
8	
9	
10	

Scheduling Transaction capacity

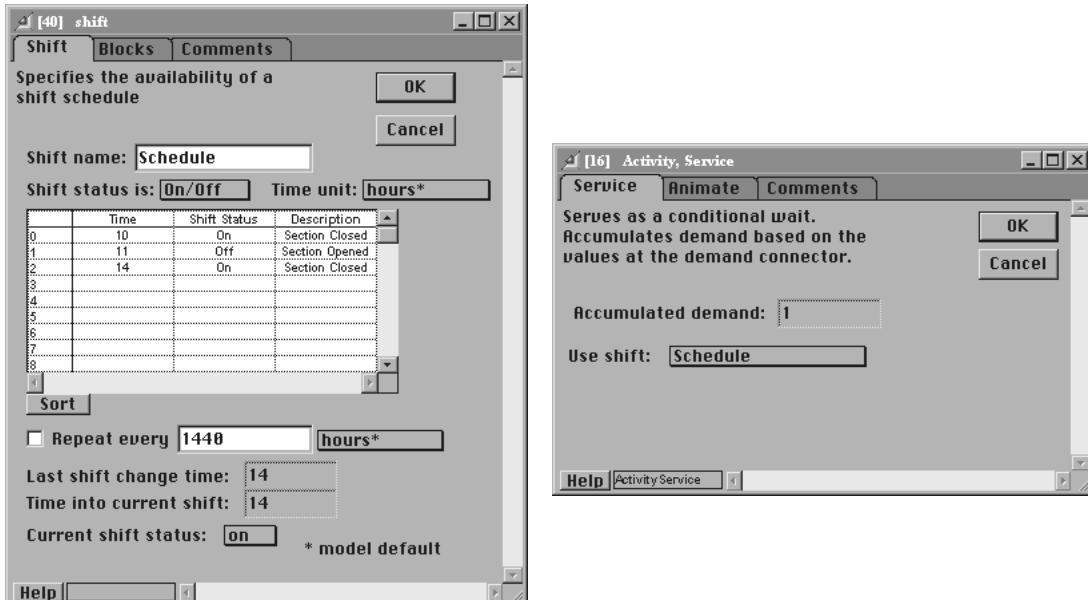
Shift block used to schedule

Yet another approach to this would be to use a Shift block to control an Activity Service. The Shift block would contain the same information as the schedule block.

Examples of use of the Shift block can be found in the “Examples \ Manufacturing or BPR \ Shift” folder.



Scheduling using the Shift block



Dialogs of the Shift block and Activity Service block showing shift use

BPR

The “schedule” shift is selected in the Activity Service block. As long as the shift is on, the Activity Service block will allow items through, when the shift closes down, items will no longer be permitted to pass through the Activity Service block.

Controlling the flow of items to an activity

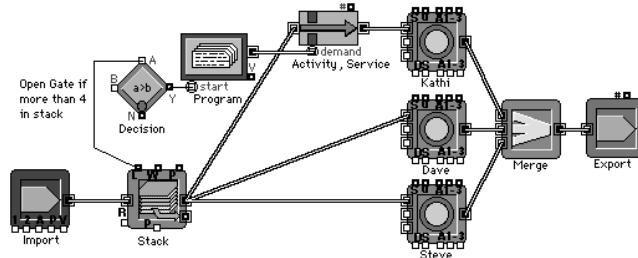
As discussed in “Bringing a system on-line” on page B98, you can use logic constructs in Extend to route items to a portion of the model that will then start operating, and you can bring a system on-line based on the time of day. However, the method shown in those examples may cause the activity being brought on-line to cycle on and off frequently. You probably do not want this to occur, as it typically not an efficient way to use resources. 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, you there are two main ways to control the flow of items to the activity: by specifying the number of items which will be processed or by specifying the amount of time the activity will be on-line.

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 an activity on to process a particular number of items, even if the waiting line for the other activity is below the threshold which originally activated it. This reduces the number of times the activity turns on and off. To do this, use the model described in “Conditional

routing” on page B98 (in the “Examples \ BPR \ Routing” folder), and add a Program block before the Activity Service block. The resulting model would look like:



Bringing a system on-line for a fixed number of items

When activated, the Program block puts out a single item with a value of 10. This causes the Activity Service block to stay open until 10 items pass through, before shutting off again. The table in the Program’s dialog looks like:

	Output Time	Value	Priority	Attribute
0	0	10		
1	5	0		
2				
3				
4				
5				
6				
7				
8				
9				

Activating the demand connector with an item

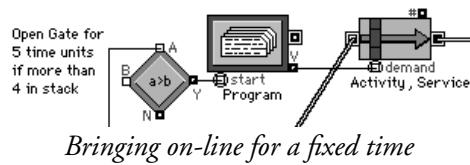
BPR

There are two items of special note in this model: how the “start” and “demand” connectors are activated.

- Since the *start* connector is connected, the Program block runs in relative simulation time (begins its program relative to when start is activated), as explained in the section “The “start” connector” on page B77. Once “start” is activated (gets an item or sees a value greater than 0.5) it causes the entire program to happen; subsequent activations are ignored until the program is complete. The second line of the program means that the Program block will ignore any start messages for 5 time units. This allows the machine to process items (and hopefully reduce the buffer length) before the sequence is activated again. If the program did not include this pause, the Activity Service block could be activated constantly.
- The *demand* connector is activated when it gets an item, causing the Activity Service to open and allow items through. The number of items allowed through before the Activity Service closes is determined by the value of the item at “demand”. For instance, each item with a value of 10 creates a demand for 10 items before the gate is shut.

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. This is also easy. To do this, use the Program block to output values to the “demand” connector on the Activity Service block. Connect from the Program block’s value (*V*) connector instead of its item connector:



In the table in the Program’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 time units, you enter:

	Output Time	Value	Priority	Attribute
0	0	1		
1	5	0		
2				
3				
4				
5				
6				
7				
8				
9				

Activating the demand connector 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”, it will stay activated as long as the value it receives is greater than 0, which in this example is for 5 time units.

Interrupting activities and preempting items

In some instances, business processes need to be interrupted or temporarily shutdown. This most often occurs when a worker is performing multiple tasks and must interrupt one task to perform another, typically higher-priority task. As a general rule in Extend, items are preempted and processes are interrupted:

- *Preemption* occurs when an item being processed is superseded by another item. An example of this is when one job is superseded by another, higher priority job. The preempted item (in this case, the current job) could be finished by another piece of equipment, finished later by the original piece of equipment, or never finished. The Transaction Preemptive block allows you to specify which item will be preempted: the item that has been processed for the longest time, the item that has taken the least amount of time, or the item with the lowest priority. Items that are preempted can be routed back to the original Transaction Preemptive block for later processing or routed elsewhere in the model.

- When an activity is *interrupted*, it temporarily stops processing. A process that is interrupted can resume when the interruption ends. The “S” input on the Operation and Operation Variable blocks is used to interrupt or shut down processes.

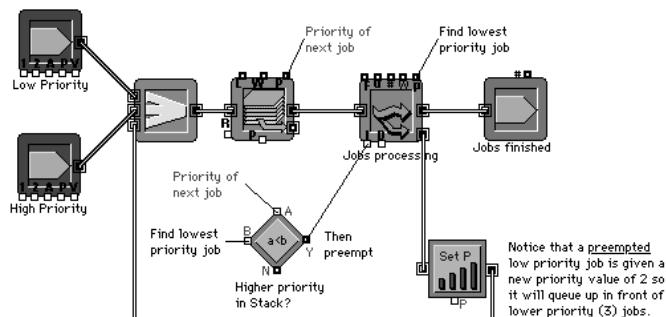
Preemption

The Transaction, Preemptive block allows you to model situations where one item preempts (prematurely supersedes) another item. You can preempt the item that has been in the block the longest time, the shortest time, or which has the lowest priority (the highest value is the lowest priority). You can specify that preemption occur only if the block is already processing the maximum number of items and you can store the preempted item's remaining processing time as an attribute for subsequent processing. Items in the Transaction, Preemptive block exit one at a time from either of the two item outputs: the top output on the right side under normal conditions; the lower right output if the item is preempted.

You use the Transaction, Preemptive block in conjunction with other blocks which give the priority of waiting items, compare that priority to the items being processed, and trigger the preemption. Items are only preempted if the “I” input of the Transaction, Preemptive block gets a True value (a value greater than 0.5).

For instance, to preempt based on priority there must be one block which reports the incoming item's priority value and another block which compares the priority of that item to items already in the Transaction, Preemptive block. The following model causes an item to be preempted if the Transaction, Preemptive block is full and a higher priority item is next in line to come in:

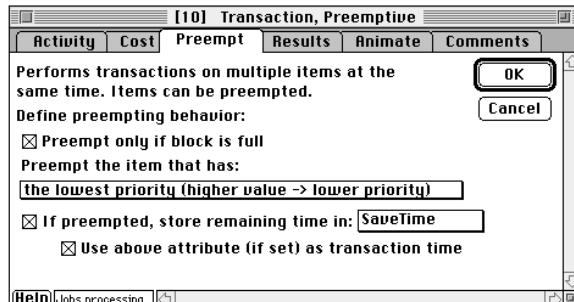
BPR



Preempting model

The Decision block from the Generic library reads the priority of the first item waiting in the Stack block (which is set as a priority queue) and compares it to the priority of the items in the Transaction, Preemptive block. If the priority of the waiting item is higher (has a lower value), the Decision block sends a value of “1” to the “I” input of the Transaction, Preemptive block. This causes the item with the lowest priority in the Transaction block to be sent out its lower right output connector.

The settings in the Preempt tab of the Transaction Preemptive block determine how the preemption will occur:



Transaction, Preemptive dialog (Macintosh)

As the dialog indicates, if the item waiting in the Stack block has a higher priority (a lower priority value) than one of the items already being processed, *and* if the Transaction, Preemptive block is at full capacity, the item being processed will be preempted. In addition, the amount of processing time that remains is attached to the preempted item as an attribute named “SaveTime”. Since “Use above attribute (if set) as transaction time” is also checked, when the preempted item returns to the Transaction, Preemptive block it will process only for the time indicated by “SaveTime”.

Notice that the model also uses a Set Priority block (from the Discrete Event library) to change the priority of the preempted item so that it will take precedence over lower priority items.

Shutting down activities

You can shut down activities at a scheduled time, such as for national holidays, or it can be a random occurrence, such as for power failures or ice storms. You can also shut down activities based on some factor in the model, such as when a clerk stops one task to perform another. The duration of the downtime can either be a constant or a random value.

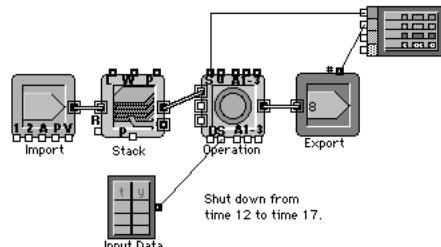
The *S* input connector on the Operation and Operation Variable blocks is used to shut activities down for a period of time. The *S* output connector on those blocks shows when the activity shuts down and the duration it is down. It does this by outputting a value of 1 while an activity is shut down and a 0 while it is not shutdown.

Activating the *S* input connector causes the block to be shut down. You can specify in the dialog that the value at the *S* input should be interpreted as the duration of the downtime, or that the activity is shut down until the *S* input connector value returns to 0. After a shutdown, the activity ignores all further shutdown messages until it is in activity again.

Scheduled shutdown

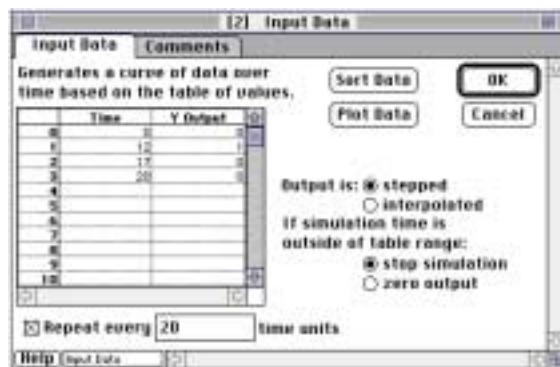
The Input Data block from the Inputs/Outputs submenu of the Generic library is often used to schedule an activity to shut down. You can also use the Shift block from the Resources submenu

of the Discrete Event library to shut down many blocks according to shifts. For example, to shut an activity down from time 12 to time 17, connect the output from a Input Data block to the *S* input connector of the Operation block:



Shutting down an activity on schedule

In the Shutdown tab of the Operation block, choose “Shut down if *S* input is >0.5”. The dialog of the Input Data block is:



Dialog of Input Data block (Macintosh)

BPR

As seen in the dialog, the Operation will be shut down from time 12, when it receives a value of 1 at the *S* input connector, until time 17, when the *S* input sees a value of 0. From time 17 until time 20 the block will output a 0. Then, since “Repeat every 20 time units” has been selected, the schedule will repeat until the end of the simulation run. In this case, there would be a downtime from time 12 until time 17 and from time 32 until time 37, and so forth.

Alternatively, to be sure that the entries in the dialog did not cause the model to run out of data before the simulation ends, you could select the “zero output” dialog option. This would output a zero after time 17, causing the Operation to stay open until the end of the simulation run.

Random shutdown

You can also shut down an activity based on some occurrence in the model. For example, the Support Services-2 model shows how one task is randomly interrupted while the worker performs another task. In this instance, the Operation block’s dialog is set to “use the *S* input as duration”.

- Note** Do not connect from the S input to a Generic library block which is standing alone. In other words, do not use a Generic library block for this unless it is (at least at some other point) connected to a discrete event block, such as one from the BPR or Discrete Event libraries. This option only works if the value at S comes (at least at some point) from a BPR or Discrete Event block, such as a Measurement or the attribute output of an Operation.
- Note** Please also see “Shift block used to schedule” on page B111 and “Using the Shift block” on page B136 for examples of activity and resource allocation that are tied together and scheduled as “Shifts.”

BPR Chapter 9: Batching and Unbatching

In many office processes, paperwork passes through the office and work is performed on it. At the beginning, precursors of the final products come into the process, usually as invoices, requests, memos, orders, and so on. These are made available to the process at times and in quantities that are different for each office. These precursors do not exist in a vacuum, however. During the process, they are often temporarily batched with other precursors such as outside approvals, or they may require additional resources such as workers for processing. These batched items move through the process together. For instance, in an office that handles real estate acquisitions the requests, bids, and real estate analyst come together in an early step and travel as a single unit through the rest of the process. This chapter will discuss how to join, or batch, items together. See BPR Chapter 10: Resources for a discussion on modeling resources.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Operation	Activities	BPR
Operation Variable	Activities	BPR
Operation Reverse	Batching	BPR

Batching

Batching allows multiple items from several sources to be joined as one item for simulation purposes (processing, routing, and so on). The Operation and Operation Variable blocks are used for batching. The operation blocks accumulate items from each source to a specified level, then release a single item that represents the batch.

Represents a business process.
Transform multiple input items
into one output item:

Quantity = Take last:

1	<input type="radio"/> Top input
	<input type="radio"/> Middle input
	<input type="radio"/> Bottom input

Batching portion of Operation dialog

Batching is used to accomplish two slightly different tasks, *kitting* and *binding*. Both kitting and binding are merely modeling concepts used to describe specific types of batching. How you construct your model (whether the batched items remain joined or not) determines whether the items are considered kitted or bound.

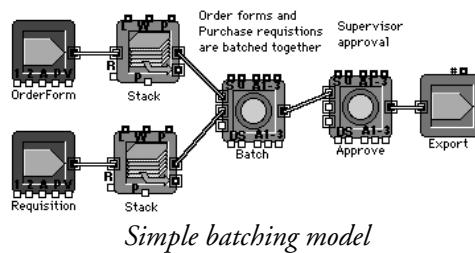
- Kitting occurs when a specified number of items are physically joined together, then released as a single item. This is most common when simulating the assembly of different kinds of paperwork into a single package. You may or may not unbatch the item at some other point in the model; most often you would not. For example, you can batch an original order, its internal representation, and a salesperson's memo together as the order's kit, which is then stored in a single place after the shipment is made. If the order is for many items, you would put them in a separate kit.
- Binding items is used to reflect a situation where one item is required to be temporarily associated with one or more other items as they progress through a portion of the model. Items that are batched for this purpose are usually separated from each other later in the simulation, although they do not have to be separated if you do not care to track the disposition of one or more of the original items. This type of batching is common when simulating labor movement or material handling, and items are most often bound with one or more resources. For example, when a customer arrives wanting service, he is bound with a customer service representative and possibly some paperwork. You batch these together, then direct them to an operation block representing the service process. After this, you *unbatch* the customer, the customer service representative, and the paperwork, and send them on separate paths in the model.

BPR

In business processes, it is more common that you will temporarily bind items rather than permanently join them. Both kitting and binding are merely modeling concepts used to perform specific tasks. How you construct your model (whether or not the batched items remain joined) determines whether the items are considered kitted or bound. The models discussed in this chapter can be found in the “Examples \ BPR \ Batching” folder.

Simple batching

The batching example below joins a company's purchase requisition with a vendor's order form; the assembly then travels as one item to a supervisor for approval and signature:



The order form and purchase requisition are joined by the Operation block according to the dialog, which is shown below. The Operation block will not release an assembled packet until it has received the required item from each of the inputs.

Represents a business process. Transform multiple input items into one output item:	
Quantity= Take last:	
1	<input type="radio"/> Top input
1	<input type="radio"/> Middle input
0	<input type="radio"/> Bottom input

Portion of the Operation block dialog

Merging Attributes

By default, when items are batched, all of the input items' attributes are combined in the attribute set for the output item. In addition, the highest priority (lowest priority value) of any of the input items is transferred to the output item. In cases where there are multiple input items that are using the same attribute, the attribute value from the first item to enter the block will be used.

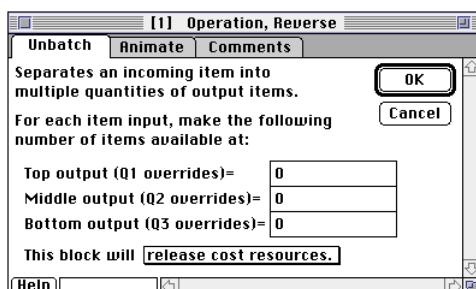
“Take last” option

When you batch, you can specify that items at one input will not be brought into the batching block until one or more of the other inputs has its requirements filled. You do this when you have a limited resource, such as a technician, that you don't want to have restricted while waiting for other required items to arrive. The input for which “Take last” is selected in the dialog will have its items wait outside the block until all required items for the unselected inputs are in the block. For example, the Operation block on page B123 has the “Bottom input” option selected. In this case, items at the top and middle inputs are pulled in until the quantities required at each are filled, then the bottom item is pulled in.

BPR

Unbatching

Unbatching is usually used to separate items that were temporarily batched. It can also be used to duplicate or clone items, even items that have never been batched. The Operation Reverse block is used for unbatching:

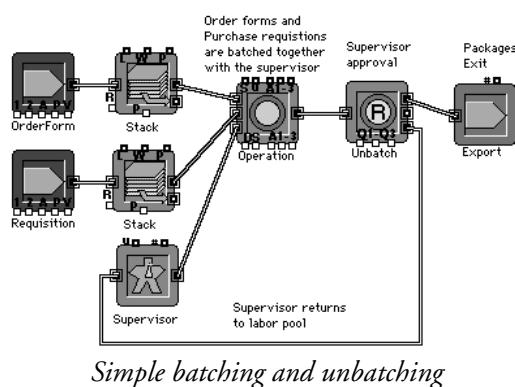


Operation Reverse dialog (Macintosh)

- There are cases when you want to batch items only temporarily; this is called binding, as discussed earlier in this chapter. Generally, items that are bound are subsequently unbatched. It is common to have resources (especially labor) that are temporarily batched with other resources, delayed, then returned to the resource block. For example, you can have a worker, a mail order, and the ordered product that need to come together be delayed as the worker boxes the product. After packaging, the product gets mailed while the worker goes back to get the next order and product. To do this you first batch the items using the Operation or Operation Variable blocks, specifying in the dialog how many of each item is required from each input. You can then use an Operation Reverse block to separate the items, routing them appropriately into the rest of the model. When you unbatch items in this situation, you specify the same number for each output in the Operation Reverse block's dialog as you specified for the inputs in the Operation dialog.
- Unbatching can also be used to duplicate an item into many *clones* of itself, even if the item has not been previously batched. For instance, you might want to simulate order processing procedures where the order is received, entered into the computer, and printed. You can unbatch each resulting invoice package into three duplicate items that represent a packing slip, an invoice, and a shipping copy. As another example, you can have one item represent a group of items throughout the modeling process, then unbatch the group at the end of the model to determine how many individual items were actually processed. The numbers specified in the dialog of the Operation Reverse block determine how many clones or duplicates of the item you want at each output.

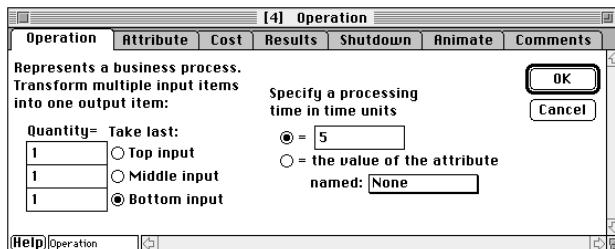
BPR

As discussed earlier, you use the binding concept of batching to model when one item temporarily requires the presence of one or more other items for processing, then is separated and routed as needed. The following model is an extension of the “Simple batching model” discussed earlier, with the addition of binding and unbatching:



Before the purchase can be made, the packet must be approved by a supervisor, who is represented by a Labor Pool block. Although not actually joined with the paperwork, the supervisor is bound

temporarily with it by the Operation block. The Operation dialog is modified to require an additional item at the bottom input (that is, one supervisor):

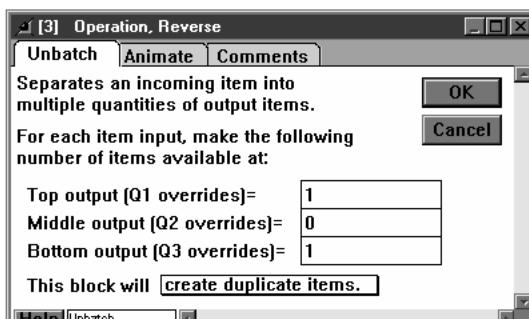


Binding a supervisor (Macintosh)

In the dialog, select “Take last” for the supervisor, indicating that the supervisor will not be pulled into the block until the other required items are present. (This is especially useful for conserving resources, such as when you model workers being needed at more than one location in the model).

In this model, the supervisor examines and approves the packet and, when finished, returns to the Labor Pool block ready to perform the task on another unit. This is modeled by the Operation Reverse block, which takes a single item (the output from the Operation block) and creates two items. One item represents the supervisor who is sent back to the labor pool and the other represents the approved purchase requisition packet that is sent to purchasing:

BPR



Unbatching 1 packet and 1 supervisor (Windows)

The supervisor is batched from the bottom input of the Operation block and exits at the bottom output of the Operation Reverse block. In this case it does not matter which output is used for unbatching a particular item. What is important is that the correct quantity be output and that it be correctly identified in the model. For example, if it is more convenient, you could route the supervisor out the top output on the Operation Reverse block. However, if this were an activity-based costing model, you would need to use the correct output to release the supervisor (see BPR Chapter 11: Activity-based costing for a complete discussion on modeling resources in activity-based costing models).

BPR Chapter 10: Resources

Often during a business process, items will require precursors, or resources, before they can proceed to the next step in the process. For example, a customer service representative provides a means for assisting customers. This chapter will discuss the various ways to model resources in Extend.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Activity Delay (Attributes)	Activities	Discrete Event
Activity Service	Routing	Discrete Event
Decision	Decisions	Generic
Decision (2)	Routing	BPR
Decision (5)	Routing	BPR
Equation	Math	Generic
Get Attribute	Attributes	Discrete Event
Input Data	Inputs/Outputs	Generic
Labor Pool	Resources	BPR
Program	Generators	Discrete Event
Queue Resource Pool	Queues	Discrete Event
Repository	Resources	BPR
Resource Pool	Resources	Discrete Event
Release Resource Pool	Resources	Discrete Event
Stack	Queues	BPR

The models discussed in this chapter can be found in the “Examples \ BPR \ Resources” folder.

Overview

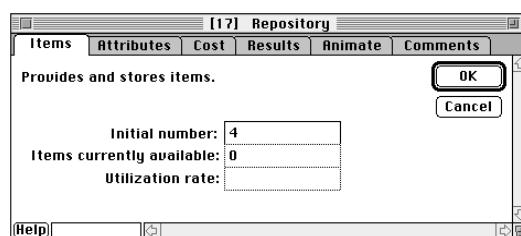
Resources are entities that provide a service to the items in your model. In Extend, resources can be modeled either explicitly from a resource block or implicitly using an activity block which represents both the resource and the activity the resource is performing. For example, in BPR Chapter 2: Tutorial you first modeled the credit application reviewers using an Operation block to

represent both the reviewer and the review process. Later you used a Labor Pool block as a source of reviewers for the model.

How you choose to model resources depends on how much detail is important in the model and your own personal preference. This chapter focuses on the explicit modeling of resources using resource blocks such as the Labor Pool block (which provides resource items) or the Resource Pool block (which controls the flow of items based on resource availability).

Resources can be unlimited: you can provide a steady stream of resources to the process for the duration of the simulation. Resources can also be limited: you can specify how many of a particular resource will be available to the process at any one time or even during the entire simulation run. Resources can be reused if you wish. This is most common with labor, where the worker is working on a task with other resources for some time, then returns to the pool of available labor.

For instance, you can attach a Generator block (from the Generators submenu of the Discrete Event library) to a Repository block to simulate parts being delivered to a stockroom at random intervals. If the items for your operations are delivered at regular intervals, you can connect a Program block (also from the Generators submenu of the Discrete Event library) to the Repository block. To specify a passive, limited source, you just use a Repository block with the initial number entered in its dialog, such as:



Setting the initial number in a Repository block (Macintosh)

In addition to modeling resources as individual items as was done in the above example, you can use the resource pool blocks. As discussed later in this chapter, resource pools do not use resource items; instead, they constrain the flow of items in the model based on the availability of resources.

Closed and open systems

The blocks that provide a finite resource of item resources (the Labor Pool and Repository blocks) can be part of closed or open systems depending on whether or not the resources are recycled back to the originating block. In an open system, resources are not recycled. Systems can also be partially closed, for example when some of the items are recycled back and some come from an outside source.

Closed systems

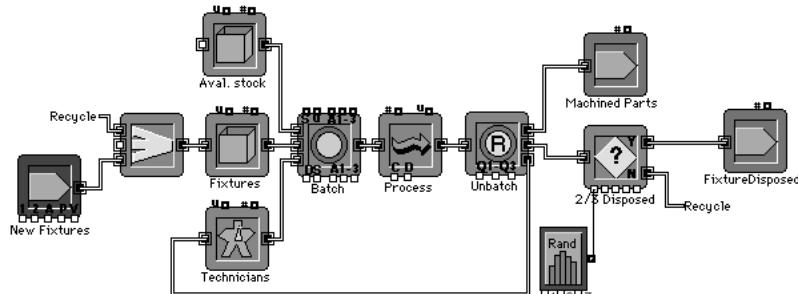
In a closed system, items are routed from a resource-type block, where they are used in processing an item. The resources are then recycled back to the resource-type block's input connector once they are no longer being used elsewhere in the model. Once they are returned to the resource-type block, they become available for use with other items. For example, assume a technician (the resource) is required to assemble parts of a television. While the technician is assembling the parts, he/she will be busy and will not be available to perform work elsewhere. In a closed system, upon assembling the parts, the technician will return to the technician pool and will become available for other assignments.

In a partially closed system, only a portion of the resources are returned to the pool for re-use. For example, consider a case where there are different shifts of laborers (the resource). Suppose three laborers are assigned to a task. Upon completion, one of the laborers' shifts is up and he therefore does not return to the labor pool to be assigned to a new task.

Open systems

Resource-type blocks are part of *open systems* when the block's items are not recycled. In an open system, items at the end of the line are not passed back to the block's input connectors, they exit the simulation. The most common example of an open system is stock. Normally, stock passes out of the model at the end of the line. Another example of an open system is a consumable resource that makes only one pass through the business process.

The following model shows an open system (Available stock), a partially closed system (Fixtures), and a closed system (Technicians).



Closed and open systems

Modeling resources

In Extend, there are two main ways to model resources:

- Batching resources with items, as discussed briefly in this chapter and more fully in BPR Chapter 9: Batching and Unbatching.

- Using the Resource Pool blocks (Resource Pool , Queue Resource Pool, and Release Resource Pool). This method controls the flow of items by keeping track of available resources pool units rather than modeling individual resources as items.

This section will discuss these two methods of modeling resources and their respective strengths and limitations.

Batching resources with items

When using this method, the resources are represented by items whose purposes are to provide a service for other items in your model. The resources can therefore have attributes, priorities, and values like any other item in your model. For this reason, this method of modeling resources is preferred if you need to track information about resources using attributes.

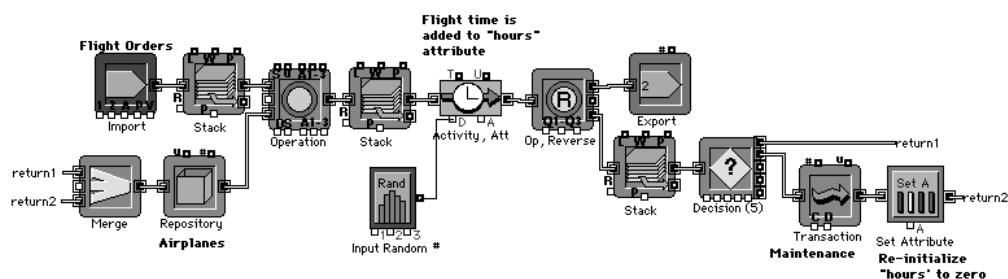
In the dialog of the resource block, you specify the number of resources that are initially available. When an item uses a resource, it is batched with the resource, as discussed in BPR Chapter 9: Batching and Unbatching. While a resource is batched with an item, it cannot be used elsewhere in the model. 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. This limitation restrains the movement of items in the model.

BPR

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 be used again.

Assigning attributes to resources

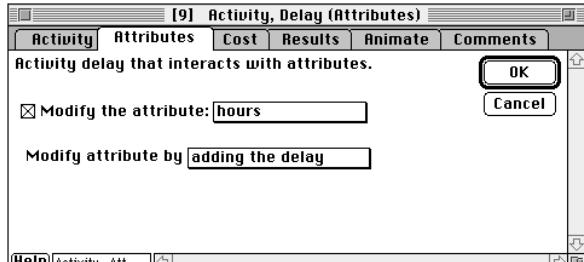
Suppose you are modeling an air freight company. Regulations require that the airplanes must undergo maintenance after every 50 hours of flight time. In this model, you need to track the accumulated processing time or flight time for each airplane resource. The model looks like:



Air freight model

As flight orders come in, they are batched with an airplane. If an airplane is not available, the flight order will wait in the queue until one becomes available. When an airplane is batched with a flight order, it will no longer be available to handle other flight orders. The Activity Delay

(Attributes) block will add the flight time to an attribute called “hours”. The attributes tab of the Activity Delay (Attributes) block is:



Adding processing time to “hours” attribute (Macintosh)

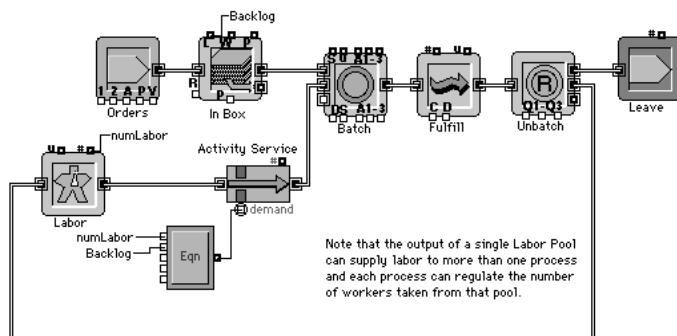
After the flight, the Decision (5) block reads the “hours” attribute and determines if the accumulated flight time is greater than 50 hours. If it is, the airplane will be routed to the maintenance group and the “hours” attribute will be re-initialized to zero; if not, the airplane will be returned to the Repository block where it will wait for another flight order.

In the above model, it is essential to be able to track information about each resource. Therefore the ability to assign attributes to the resources is critical.

BPR

Allocating Resources

Another advantage of using the batching method to model resources is the flexibility in allocating the resources to a particular operation. By using an Activity Service, Decision (2), or Decision (5) block, you can control when and where resources are allocated. For example, the model below shows workers in an “on-call” situation, where the number of workers called to work on a process is regulated by the backlog.

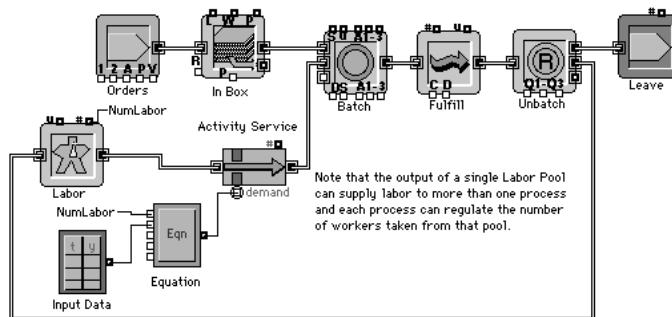


Labor varied by backlog

The Activity Service block is used as a switch that is controlled by the backlog (queue length) from the Stack. The Equation block outputs a “true” value if there are no workers processing or the

backlog exceeds 3. This “opens” the Activity Service block, adding an additional worker to the process.

You can also regulate the amount of labor depending on the time of day. In the model below, the Activity Service block is used as a switch that is controlled by the Input Data block over time.



Labor varied by time

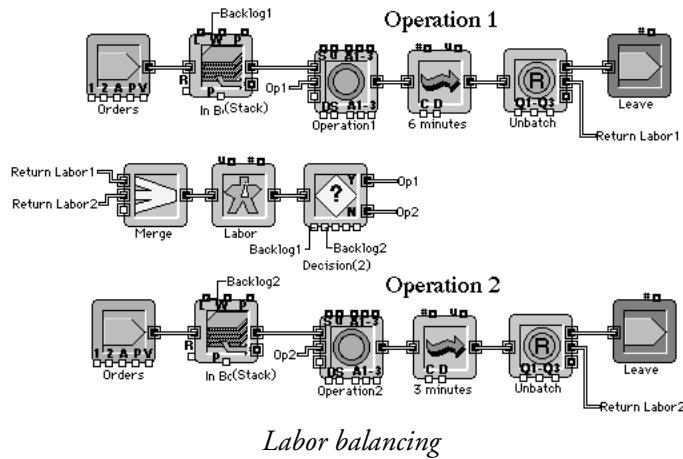
In the Input Data block, you can specify the desired number of workers at different times during the simulation, as shown below:

Time	Y Output
0	1
1	2
2	3
3	2
4	1
5	1000
6	
7	
8	
9	
10	

Table in Input Data block

The Equation block outputs a “true” value if the number of workers in the process is less than the desired number. This “opens” the Activity Service block, allowing an additional worker into the process.

You can balance the backlog of parallel operations by routing labor to the operation with the largest backlog, as shown in the model below:



The Decision (2) block is used to compare the backlog (queue length) of the two Stacks. If Backlog1 is greater, labor is routed to Operation1. If Backlog2 is greater, labor is routed to Operation2.

BPR

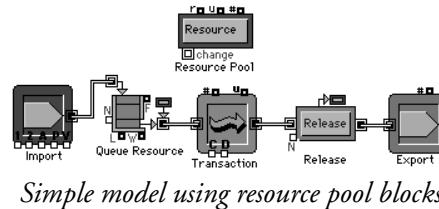
```
if (Backlog1 > Backlog2)
    Path = YesPath;
else
    Path = NoPath;
```

Equation in Decision(2) block

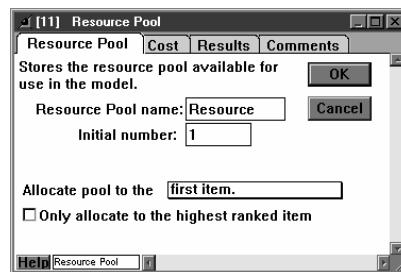
Using the Resource Pool blocks

The resource pool blocks in the Discrete Event library (Resource Pool, Queue Resource Pool, and Release Resource Pool) do not use actual items to model the resources. Instead, these blocks 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 Resource Pool block, the block will query the appropriate Resource Pool(s) to determine if the required number of resources are available. If so, the Resource Pool will appropriately decrement the number of resources currently available, and the item will be released from the Queue Resource Pool. If the required number of resources are not available, the item will wait in the Queue Resource Pool until resources become available.

In a closed system, the resources are returned to the Resource Pool block by passing the item through a Release Resource Pool block. A simple model using the resource pool blocks looks like:

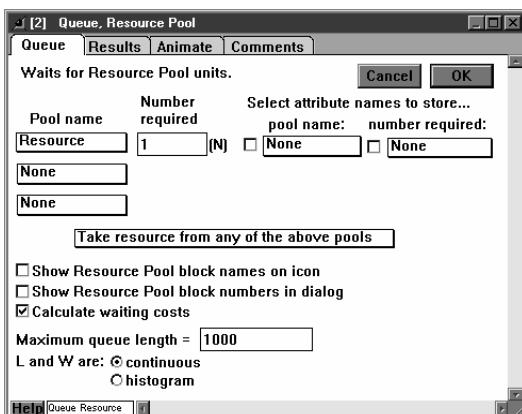


Within the dialog of the Resource Pool block, you must name the resource pool and define the initial number of resource pool units, as shown below:

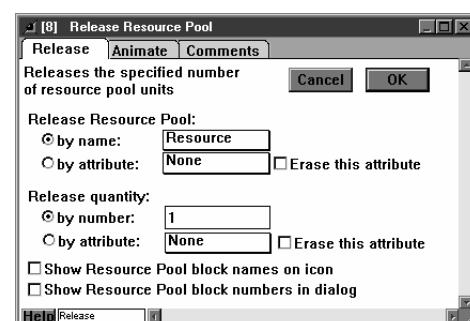


Dialog of Resource Pool block (Windows)

Within the Queue Resource Pool and Release Resource Pool blocks, you specify the number of resources required (in this case “1”) and the name of the Resource Pool block from which the resources originate (“Resource”). For the above model, the dialogs of the Queue Resource Pool and Release Resource Pool blocks look like:



Queue Resource Pool



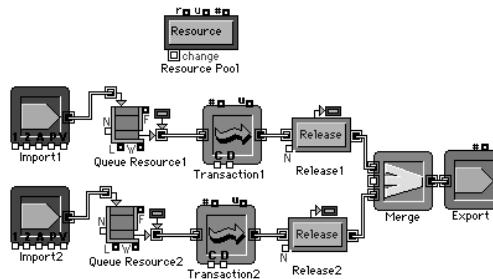
Release Resource Pool

Resource pool block dialogs (Windows)

Notice that the Resource Pool block does not require any connections to the Queue Resource Pool or Release Resource Pool blocks. Because of this, using resource pool blocks to model resources is much more flexible than using the batching method (as described on page B128) when the same resource is used in many different places, or when an item can use any one of a group of resources. This 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.

Same resource used in multiple places

In the following example, items waiting in both Queue Resource1 and Queue Resource2 require a resource from the Resource Pool block. When an item enters one of the queues, a request is sent to the Resource Pool block for a resource. As resources become available, the requests are satisfied in the order in which they were received (or in order of the requesting item's priority as defined in the Resource Pool dialog). Modeling this using the batching method would require complex logic to route the resources to the correct station.

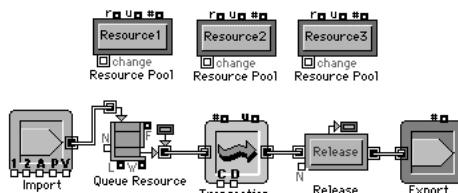


BPR

Resources from one Resource Pool block used in multiple places

Resources required from different pools

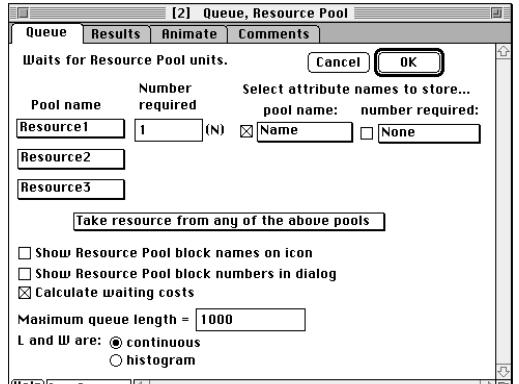
In the following example, items require one resource from *either* Resource1, Resource2, or Resource3.



One resource required from any of the three pools

As shown in the Queue Resource Pool block dialog below, you can instruct the Queue Resource Pool block to take the resource from any one of the pools specified in the dialog. When an item enters the Queue Resource Pool block, the block will query each of the Resource Pool blocks in the order that they are listed (from top to bottom). In other words, if no resources are available in

Resource1, the Queue Resource Pool block will try Resource2, and then Resource3 until the resource requirements are met.



Dialog of Queue Resource Pool block (Macintosh)

Note that the resource pool name and the number of resources used can be stored in attributes assigned to the item that uses the resources. These attributes are later used by the Release Resource Pool block to inform the appropriate pool that the resources are no longer being used by the item. In this model, the Resource Pool block name is stored in the attribute “Name”. When the item passes through the Release Resource Pool block, the block will read the “Name” attribute and determine to which resource pool the resources should be returned. Again, modeling this using the batching method would require complex logic to correctly route the resources.

BPR

Note that modeling resources using the resource pool blocks does not allow you to use attributes to track information about the resources as you did in “Batching resources with items” on page B128.

Scheduling resources

There are two systems available to schedule resources:

- Use the Change connector on individual resource blocks
- Use Shift blocks to change multiple resource and activity blocks

Use of the Shift block is discussed in the section after this.

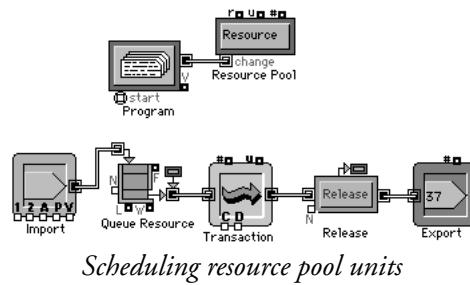
Using change connectors

The Resource Pool block has an input connector labeled *change*. The “change” connector works the same as the regular item input connector, except that it also takes items with a negative Value (item Values are discussed in the Extend user’s manual).

You typically use the “change” connector to add to or subtract from the number of available resource pool units in the Resource Pool block. This change can be scheduled, such as when workers take breaks, or unscheduled, such as for loss of tools.

Sending items with a negative Value to the “change” connector reduces the number of items available in the resource by that Value, as long as there are sufficient items available. If the block doesn’t have enough items to dispose of, it will retain a negative demand for future items until the demand is met.

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 “Using the Resource Pool blocks” on page B131, you could schedule the resources depending on the time of day using the Resource Pool and Program blocks. The new model is:



BPR

Assume there is an initial number of workers (3) available when the run starts at time 0, and 5 additional workers arrive and remain just through the busy period, which is from 4 until 6. You enter the schedule for the day in the dialog of the Program block, as shown below:

	Output Time	Value	Priority	Attribute
0	0	8		
1	4	5		
2	6	-5		
3				
4				
5				
6				
7				
8				
9				

Table in Program block (Macintosh)

Note that at time 6 the number of workers in the dialog is -5. Since the “change” connector of the Resource Pool block accepts negative numbers, the -5 will decrease the available workers from 8 to 3 at time 6. If all of the workers are busy, the change will occur as soon as the workers become available.

Note that in this model the workers are part of a partially closed system. They are recycled back to the Resource Pool block when the items are passed through the Release Resource Pool block,

while additional workers are added to or subtracted from the block through its “change” connector.

Using the Shift block

The Shift block is used to schedule both the magnitude and availability of capacity in other blocks in a model. This is useful for simulating how a system’s resources generally 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.

The shift block can schedule capacity in a number of blocks in the Discrete Event, Manufacturing and BPR libraries. This includes but is not limited to: Generator, Resource Pool, AGV, Machine, Conveyor Belt, Pallet, Process Preemptive, Labor, Import, and Repository. For example, it can turn on or off an Activity Multiple, or specify a maximum number of activities for that block, depending on how it is set up.

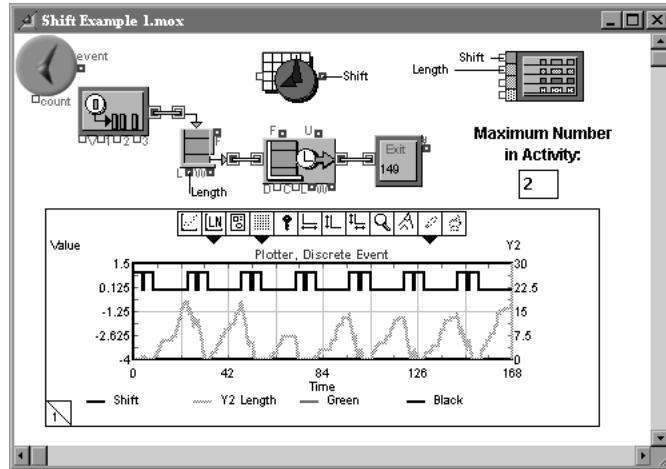
The Shift block controls the capacity of specified blocks in a model based on a schedule that is defined in its dialog table. If a shift schedule is changed, all blocks using that shift will receive the same modified shift pattern. In addition, shifts may be repeated at regular intervals if the “Repeat every” checkbox is selected and a time is entered in the adjacent dialog item. This is useful for modeling repeated shift patterns, e.g., an eight hour work day each day of the week or breaks that occur every four hours.

The block’s dialog has a pop-up menu offering two general types of schedules: On/Off and Numerical. The On/Off shift is a binary switch that turns associated blocks on or off at different points in time. For example, an On/Off shift might be used to shutdown an activity during lunch and evening hours. The other type of schedule, the Numerical shift, explicitly defines the size of a block’s capacity over time. For example, a Numerical shift might set the size of a Resource Pool to three items for the morning shift, zero over lunch, five for the afternoon shift and zero overnight.

The block’s input value connector may be used to override the shift schedule. If the input connector is less than 0.5, the shift is considered off shift and 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 instead. The output connector reports the current shift status (e.g., for ON/OFF shifts, ON = 1 and OFF = 0).

Examples showing the use of the Shift block can be found in the “Examples \ Manufacturing or BPR \ Shift” folder.

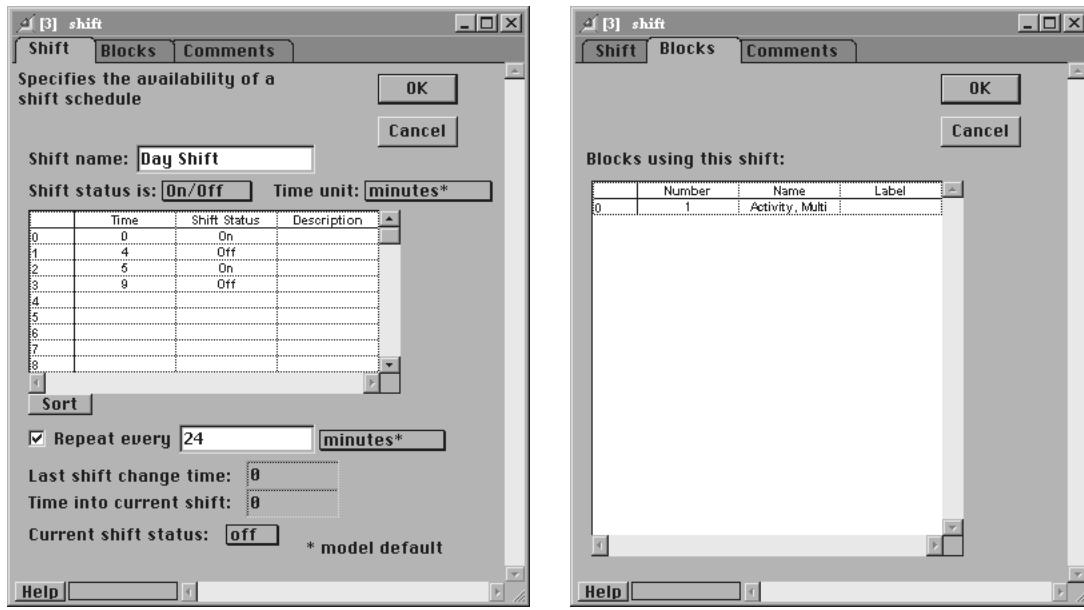
Simple On-Off Shift Example



Shift Example1.mox

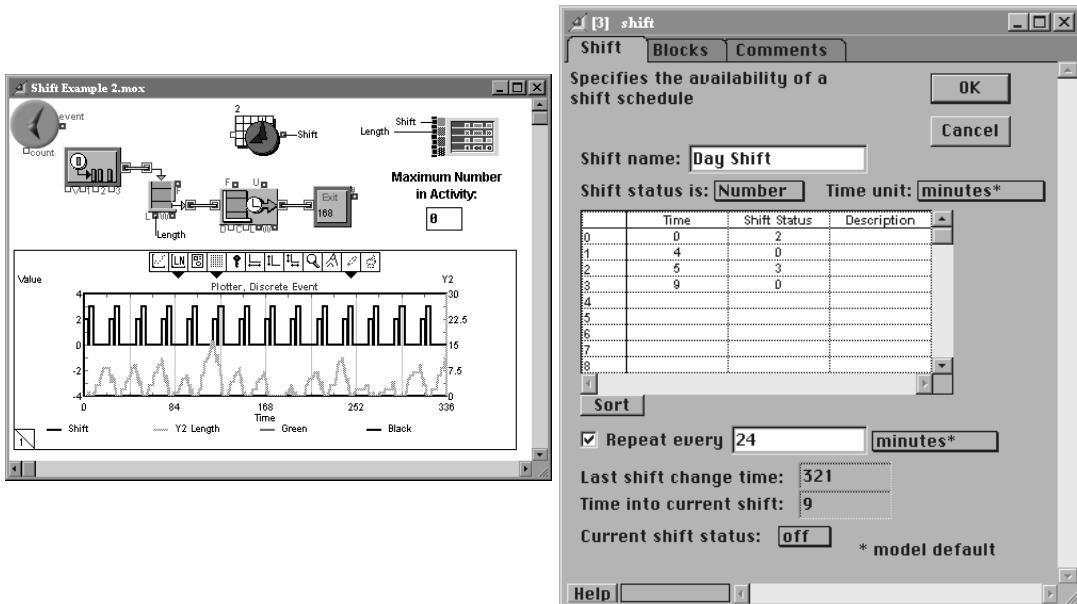
In this example, *Shift Example1.mox*, an On-Off shift is used to turn the Activity Multiple block on for four hours in the morning, off for one hour at lunch, back on for four hours in the afternoon and off again in the evening until the next morning. The upper black line reflects this 24 hour shift cycle for five days. In addition, observe how the lower gray line reflects the queue length's growth during off shift time periods.

BPR



Settings for the Shift and Blocks tabs

Simple Numerical Shift Example

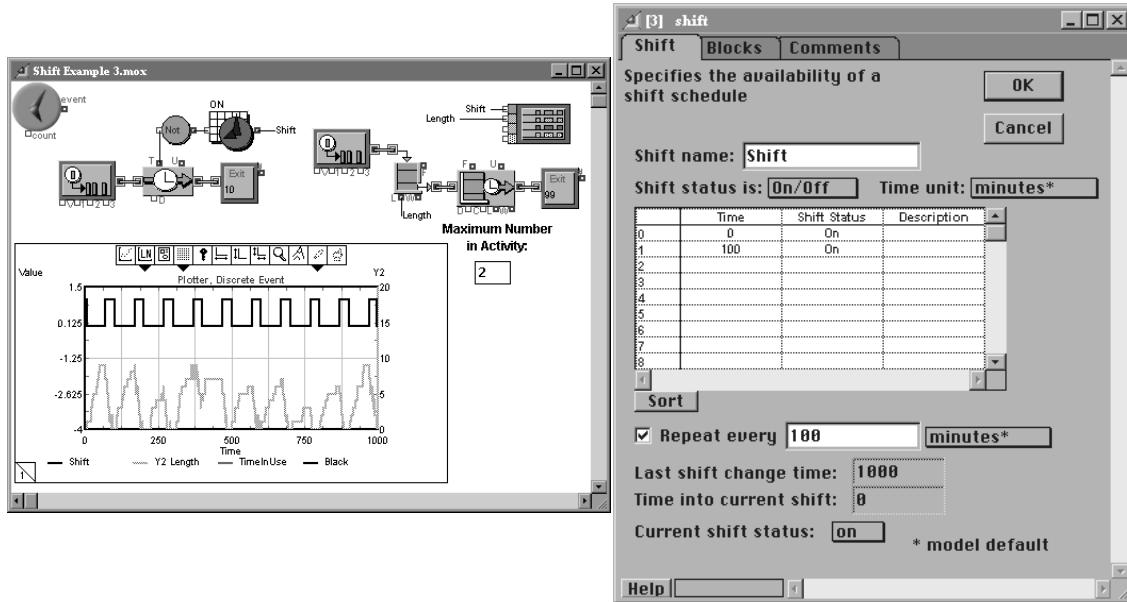


Shift Example2.mox

BPR

In this example, *Shift Example2.mox*, a Numerical shift is used to control the maximum number of items in the Activity Multiple block. The Activity Multiple is limited to two items during the morning shift, zero during lunch, three items during the afternoon shift and zero overnight. This behavior is reflected in the plotted upper black line over the course of five days. Again, the lower grey line reflects the queue length's growth during Activity downtime.

Activity Status Shift Control Example



Shift Example3.mox

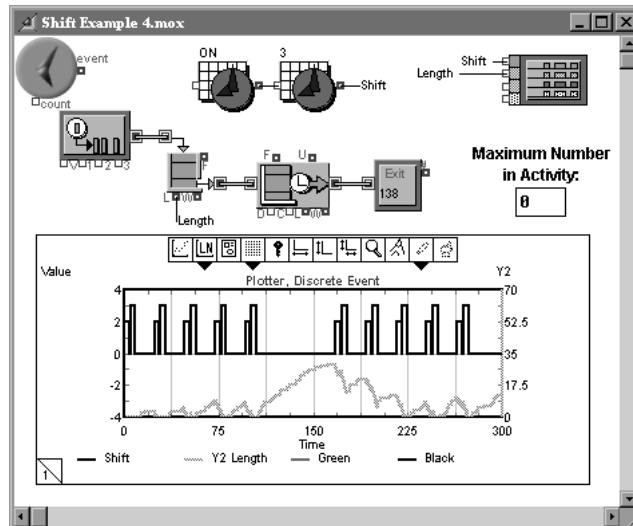
BPR

In the following example, the “T” connector on an Activity Delay block is used to control the shift status of an Activity, Multiple block. The value of the T connector depends on the status of the Activity Delay; is it busy or idle. Given the configuration of the model below, this ultimately impacts the Shift block’s status:

T Connector	Logical Not	Shift Status
1	0	Off
0	1	On

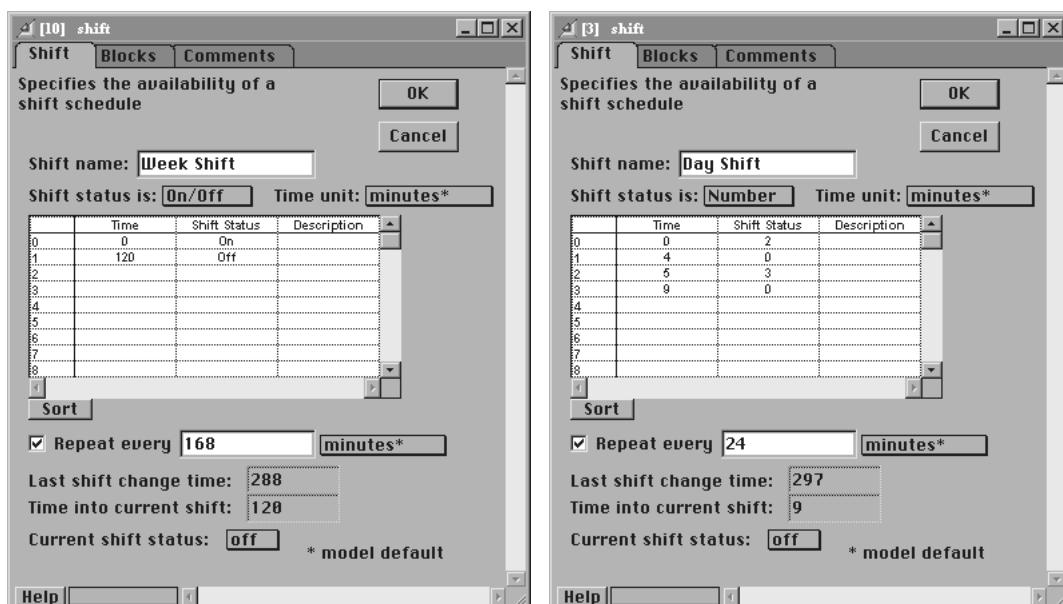
When an item is being processed in the Activity Delay block, its “T” connector value is one and the resulting shift status is Off. Consequently, the Shift block ignores its dialog shift schedule and forces the Activity, Multiple block off line. Conversely, when the Activity delay is idle, the “T” connector value is zero and the Shift block has the opportunity to observe the shift schedule defined in its dialog table. The Shift block’s table has been set to the “On” position 100% of the time so when the Activity Delay is idle, the Activity, Multiple is brought on line. In this example, the Activity Delay is busy with an item 70% of the time, causing the Activity Multiple to be on shift only 30% of the time. This behavior may be noted in the associated plot.

Shift Block in Serial Example



Shift Example4.mox

BPR

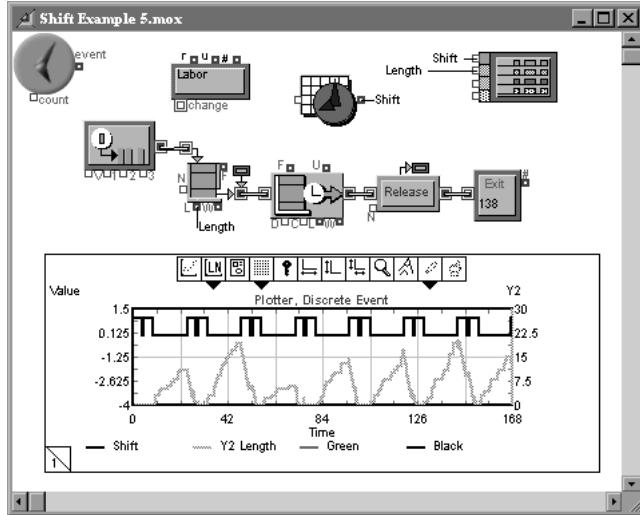


Shift settings for the first and second Shift blocks

Shift blocks may be configured serially to create more complex shift patterns. In the following example, the typical 40 hour work week with two days off during weekends is modeled by feeding a weekly On/Off Shift block into a daily Numerical Shift block. The weekly shift is On for the five weekdays and Off during weekends. Consequently, during weekdays, the weekly shift block sends

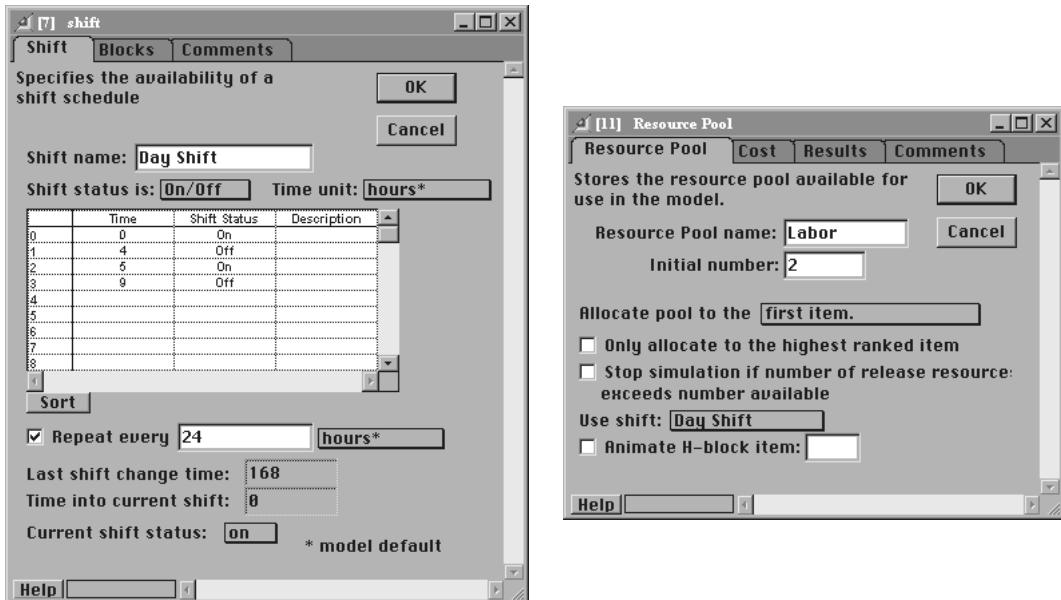
a value of one to the daily shift block, allowing it to observe its own daily schedule. However, during weekends, the weekly shift block overrides the daily shift schedule by sending it a value of zero. The black plotted line shows two work weeks separated by a weekend.

Shift controlling Resource Pool



Shift Example5.mox

BPR



Settings for the Shift and Resource Pool blocks

This example illustrates the Shift controlling a Resource Pool availability so that workers start their shift, work 4 hours, take a lunch break, work four more hours, and then leave for the day.

When workers are not available, the backlog starts building up in the Queue Resource Pool.

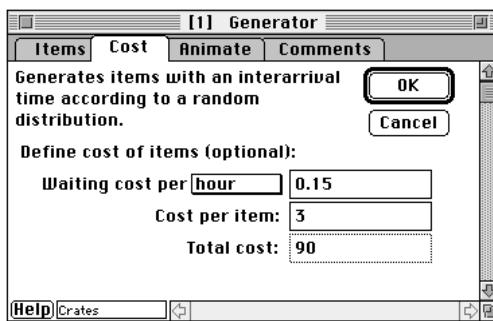
An Important Note About Using Shifts

It is quite easy to define a complex shift schedule to a model which may include all breaks, holidays, weekends, etc. However, before adding such complexity to the model, the model builder should 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 make one week 5 days long 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.

BPR Chapter 11: Activity-based costing

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 its total as precisely as possible based on the fixed and variable costs of the inputs. You use ABC to identify, quantify, and analyze the various cost drivers (such as labor, materials, administrative overhead, rework, etc.) and to determine which ones are candidates for reduction.

By building your models, you have already identified the discrete outputs of your system as well as the processes and resources that are involved in creating those outputs. To add ABC to your models, you enter costing information into block dialogs. Blocks that generate items or resources and blocks that process items have tabs in their dialogs for specifying costing data. The Cost tabs allow you to enter variable cost rates per time unit and fixed costs per item or use.



Cost tab of Generator block

Once you have specified the cost information, costs will automatically be tracked as the items in your model evolve from system inputs to the finished product.

The first section of this chapter is an overview of how to perform ABC using Extend. The second section takes a more in-depth look at the costing calculations that are automatically performed when Extend tracks costs. The following blocks will be of main focus in this chapter:

Block	Type	Library
Cost By Item	Statistics	Statistics
Cost Stats	Statistics	Statistics
Import	Generators	BPR
Input Data	Inputs/Outputs	Generic
Labor Pool	Resources	BPR
Operation	Activities	BPR
Operation Reverse	Batching	BPR
Program	Generators	Discrete Event
Queue, Resource Pool	Queues	Discrete Event
Resource Pool	Resources	Discrete Event
Release Resource Pool	Resources	Discrete Event
Repository	Resources	BPR
Set Attribute	Attributes	Discrete Event
Stack	Queues	BPR
Transaction	Activities	BPR

BPR

Unless otherwise specified, the models discussed in this chapter can be found in the “Examples \\ BPR \\ ABC” folder.

Performing activity-based costing

This section will discuss how to define cost rates, how to properly combine cost resources with items, and how to gather and work with the cost information.

Item types

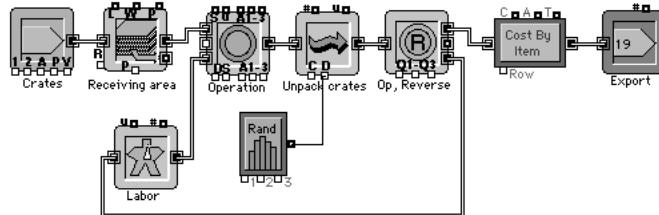
When considering ABC in Extend, every item in your model may be categorized as either a *cost accumulator* or a *resource*. This section will discuss the differences between cost accumulators and resources. It is important to understand that difference, because Extend treats them differently, as you will see in “How Extend tracks costs” on page B153.

Cost accumulators

You perform ABC to determine the costs associated with processing an item. The item being processed is called the *cost accumulator* and will accumulate costs as it waits, gets processed, or uses resources.

For example, assume you want to determine the cost associated with receiving crated inventory at a warehouse. As each shipment arrives, a labor resource is required 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. As the crate item is stored or processed it will accumulate costs. For instance, it might take the laborer approximately 30 minutes to unpack the crate. In the model below, the processing time and the hourly wage of the laborer is used to automatically calculate

the cost of that particular step. That cost is then added to the accumulated cost being tracked with the crate. As the crate progresses through the steps of being received, Extend will add the cost incurred at each step to the accumulated cost.



Receive inventory model

Cost accumulating items can be introduced into a model using some generator-type blocks (Import, Generator, and Program blocks) and resource-type blocks (Labor Pool, Repository, and Resource blocks), as you will see in “Cost accumulator cost rates” on page B146.

BPR

Resources

As you saw in BPR Chapter 10: Resources, resources can be modeled using the Resource Pool blocks or by using batching. In the resource pool method, the resource pool units act as constraints on the flow of items throughout the model. In the batching method, resource items are combined, or batched, with other items before the items can proceed to the next station.

Resources are entities that provide a service for the items in your model; they do not accumulate their own costs. However, whenever a cost accumulator uses a resource, the cost rates of that resource (discussed in “Resource cost rates” on page B147) are used to calculate the costs which are added to the total cost of the cost accumulator. For instance, in the example described in “Cost accumulators” above, the laborer is a resource item that is batched with the crate. Although the laborer is modeled as an item, it represents a resource and 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.

Resources are introduced into a model using resource-type blocks (Labor Pool , Repository, and Resource blocks).

Defining costs and cost rates

To include ABC in your model, you simply enter information in the Cost tabs in the dialogs of the appropriate blocks (generators that provide cost accumulators or resources and activity blocks used to process the cost accumulators). 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, Extend will automatically track costs when the simulation is run. There are two types of cost information that you can define:

- The *cost per use/item* or fixed cost.
- The *cost per time unit* or variable cost rate.

This section will describe how to define costs for cost accumulators, resources, and activities.

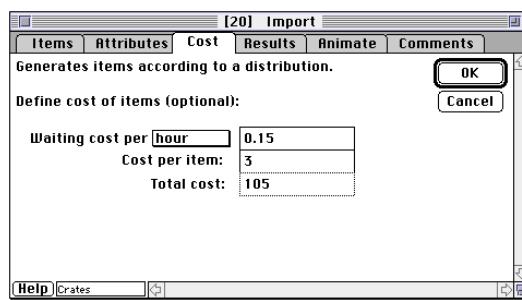
Cost accumulator cost rates

Cost accumulators have their own fixed costs and variable cost rates. They acquire additional costs from resource items and activities as they use the resources and are processed.

You specify the cost rates for a cost accumulator from within the Cost tab of the block that originates it. Each cost accumulator can have a fixed cost per item, such as a direct materials cost, and a variable waiting cost rate, causing it to accumulate costs as it is stored or waits for processing.

Cost accumulators are often generated by the Import block. For example, in the Receive Inventory model described on page B144 the Import block creates the crates of inventory. Within the Cost tab of the Import block, define the cost per time unit (otherwise known as waiting or storage cost) and the cost per item, as shown below:

BPR

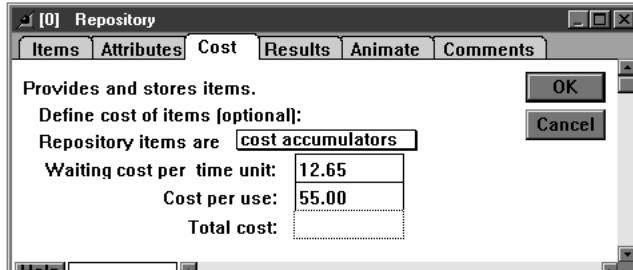


Cost tab of Import block (Macintosh)

The waiting cost is the rate at which cost is accumulated while the item is waiting in any queue. The cost per item is a fixed cost. In this case, it costs \$0.15 an hour (Waiting cost per hour) any time the crate waits for processing (such as in the receiving area) and there is a one-time docking fee of \$3.00 (Cost per item) for every shipment that is received.

Cost accumulators may also be introduced into a model using resource-type blocks (Labor Pool, Repository, and Resource blocks). To do this, you must specify that the items originating in the

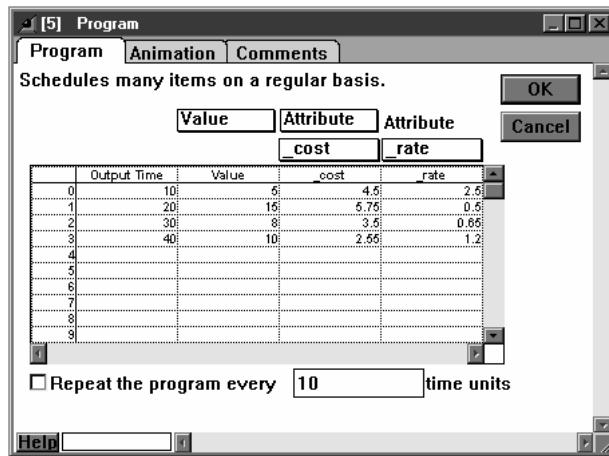
block are to be used as cost accumulators. This is done in the Cost tab of the resource-type block, an example of which is shown below:



Repository block providing cost accumulators (Windows)

To define cost rates for cost accumulators that are generated using the Program block, you must explicitly set the “*_cost*” and “*_rate*” system attributes (discussed in “Working with cost data” on page B150) for each cost accumulator. 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:

BPR



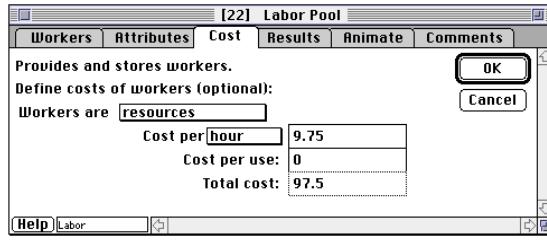
*Setting “*_cost*” and “*_rate*” in Program block (Windows)*

Note The “*_rate*” attribute, must be defined using the same time unit as the global time unit for the model.

Resource cost rates

Within the Cost tab of resource-type blocks, you can define the cost per time unit and/or the cost per use of the resource. The cost per time unit rate is 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 on page B144, cost rates for the laborer are defined as below:



Cost tab of Labor Pool block (Macintosh)

Activity cost rates

You can also define cost information for activities. Within the Cost tab of activity-type blocks, you define a cost per time unit and a cost per use. The cost of using the activity is accumulated by the items that are processed: the cost per time unit is used to calculate the time-based processing cost of each item that passes through the block; the cost per use is a fixed cost added to every item that passes through the block.

BPR

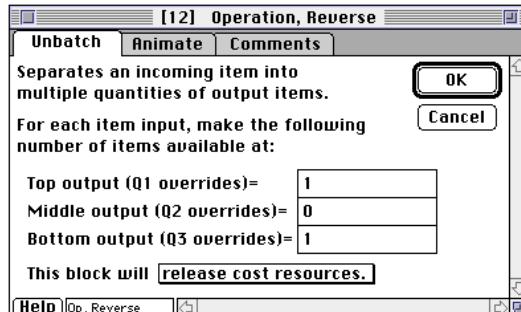
Combining resources with cost accumulators

As discussed in “Modeling resources” on page B127, there are two ways that items can “use” resources. The first method is to batch a resource with the item. While batched, the resource is in use and cannot be used by another item until it is unbatched and returned to the resource-type 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 will discuss how to properly use these two methods when performing ABC.

Batching resources with cost accumulators

When batching a resource with a cost accumulator, the resource’s cost rates are automatically stored with the cost accumulator and used in any subsequent cost calculations.

To unbatch a resource and remove its cost rate information from the cost accumulator, you must specify in the dialog of the Operation Reverse block to “release cost resources”, as shown below.



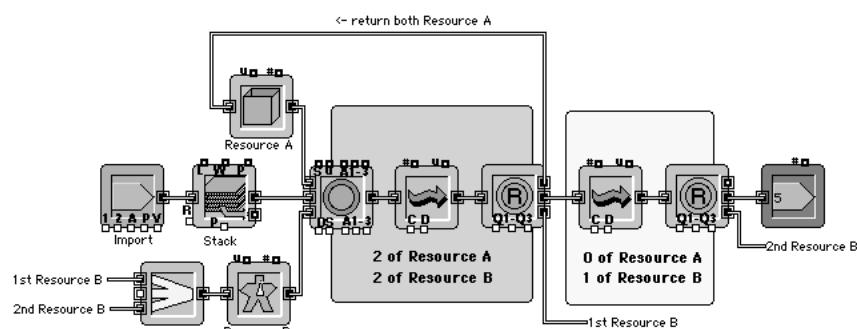
Dialog of Operation Reverse block (Macintosh)

This choice tells the Operation Reverse block to modify the information stored with cost accumulator to indicate that the resource has been released.

Note If this popup menu is set to the other choice (“create duplicate items”), the items released by the Operation Reverse 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.

BPR

In the model shown below, the cost accumulator is initially batched with 2 of Resource A and 2 of Resource B. 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 B), 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.



Multiple resources model

(The cost array is briefly described in “Resources combined with cost accumulators” on page B154 and is more thoroughly discussed in the main Extend manual.)

There are three things to remember when batching resource items with cost accumulators:

- The resources will be released from the connector which corresponds to the connector used to originally batch them to the cost accumulator. For example, if the resource entered the Operation block through the top connector, it will be released through the top connector in the Operation Reverse block.
- 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 model above, Resource A uses the top connector and Resource B uses the bottom connector.
- When performing ABC, you are limited to having no more than two different types of resource items batched with the 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 “Using the Resource Pool blocks” on page B131, as cost accumulators pass through a Queue, Resource Pool block, resources are allocated to them. When a resource pool unit is allocated to a cost accumulator, its cost rate information is automatically stored with the cost accumulator and used in any subsequent cost calculations.

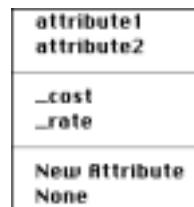
BPR

When a resource is released using the Release Resource Pool block, the information stored with the cost accumulator is modified to indicate that the resource has been released.

The is no limit to the number of different types of resources a cost accumulator can use when using the resource pool blocks, and the two methods of modeling resources (batching and resource pools) may be used in conjunction with each other.

Working with cost data

In order for you to have access to the cost information, Extend creates two attributes (“_cost” and “_rate”) for every item in your model. Since these are automatically created, they are considered *system* attributes, and will appear separately in the attribute popup menus, as shown below:



Attribute popup menu containing system attributes “_cost” and “_rate”

These two attributes will appear in the attribute popup menus only if a cost rate is defined somewhere in model. The information that is stored in these attributes depends on whether the item is a cost accumulator or a resource, as shown in the following table:

Item type	“_cost” attribute	“_rate” attribute
Cost accumulator	The accumulated cost of the item	The waiting cost, or storage cost, of the item (cost per time unit defined using the model’s global time unit)
Resource	The cost per use of the resource	The cost per time unit of the resource (defined using the model’s global time unit)

The attribute handling blocks (Measurement, Get Attribute, etc.) can be used to read, set, or manipulate these attributes. In addition, two blocks in the Statistics library can be used to gather the cost data.

Viewing cost data

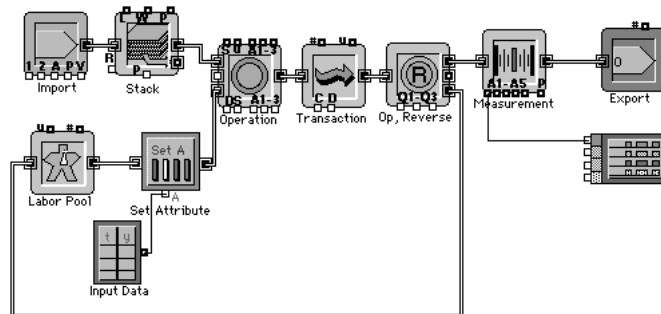
You can use a Measurement block to read the “_cost” and “_rate” attributes of any item, then plot the data or use the attribute value to perform additional calculations. For example, you can use a Measurement block to read the “_cost” attribute of cost accumulators and connect the “A1” connector of the Measurement to a Plotter Discrete Event to plot the accumulated cost of each item that passes through the Measurement block. This can be seen in the model discussed in the following section.

BPR

Changing cost data

In most cases, you will simply want to define the cost rates of the various cost drivers in your model and allow Extend to automatically calculate and track costs. However, there may be times when you need to manipulate the cost values generated. The attribute handling blocks can be used to accomplish this.

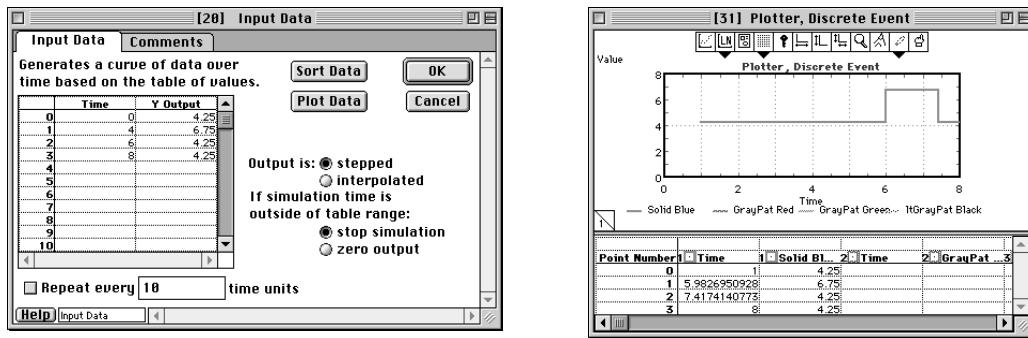
For example, suppose the cost rates of a resource vary throughout the day. During peak times the demand for the resource is high and the cost per time unit increases. You can model this by using the Set Attribute and Input Data blocks to explicitly set the “_rate” attribute of the resource as it exits the resource-type block. This is shown in the model below:



Changing “_rate” attribute of resources

Within the table of the Input Data block, you specify different rates for different times of the day. Note that a change in the rate will only affect resources as they exit the Stock block. Resources that are currently in use will not be affected unless they are recycled back to the Stock block and through the Set Attribute block.

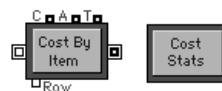
In the above model, the Get Attribute block reads the “_cost” attribute before the items exit the model. The accumulated cost of each cost accumulator is then plotted. In the plot (shown to the right, below), you can see that the cost of the items increases during the period of time that the resources’ cost rates are higher.



Dialog of Input Data and Plotter Discrete Event (Macintosh)

Gathering and analyzing cost data

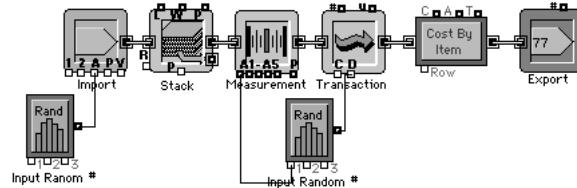
You use the Cost By Item and the Cost Stats blocks to gather the cost information generated in your models.



Costing blocks included in the Statistics library

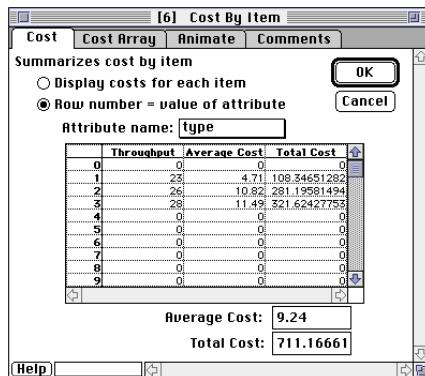
The Cost By Item block is used to read and store the “_cost” attribute of the cost accumulators that pass through it. The block’s dialog will list the accumulated cost of each item, the time the item passed through the block, and the total and average cost of all the items that have passed

through the block. This block can also be used to list the cost of the items sorted by type. The Sort By Type model utilizes the Cost By Item block in this manner.



Sort By Type model

In the model, three different item types are generated by randomly assigning a “type” attribute of 1, 2, or 3. Each transaction costs \$5.00 per hour. The transaction’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. The dialog of the Cost By Item block is shown below:



Dialog of Cost By Item block (Macintosh)

BPR

Generator, resource, queue, and activity-type blocks are all capable of generating costs that are tracked with the cost accumulating items. Additionally, each cost-generating block displays the total cost generated by that particular block in its *Total Cost* dialog item. The Cost Stats block is used to collect and display the total cost for each block. You could use this information to determine which blocks are contributing the most to the total cost of the items being processed. See Appendix G: Blocks in the Statistics Library or the help text of the block for a detailed description of how to use the Cost Stats block.

How Extend tracks costs

In the previous section, you learned how to perform ABC in Extend. This section provides a more detailed look at how Extend 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 B145), Extend 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 “_rate” attribute.

Resources combined with cost accumulators

Whether you use the batching method or the resource pools to model resources, two things happen when a resource is combined with a cost accumulator:

- 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 resources “_cost” attribute. If using the Resource Pools, the resource’s fixed cost comes from the *Cost per use* dialog item from the Cost tab of the appropriate Resource Pool block.
- 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 (see the main Extend manual for a detailed discussion of the cost array). Extend 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 the Resource Pools, the resource’s variable cost rate comes from the *Cost per time unit* dialog item from the Cost tab of the appropriate Resource Pool block.

BPR

When a resource is released by an Operation Reverse or Release Resource Pool 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, generator, activity, queue, and resource-type blocks are all capable of generating costs. As the 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 the *Total Cost* dialog item located in its Cost tab. This dialog item displays the total cost contributed by that particular block only. The following sections briefly discuss how these calculations are performed.

In generator-type blocks

When a cost accumulator is generated, Extend will add the fixed cost or *Cost per use* of the generator to the cost accumulator’s “_cost” attribute.

For each cost accumulator generated, Extend also will add the fixed cost of the generator to that block’s *Total cost* dialog item.

In activity-type blocks

When a cost accumulator enters an activity-type block, Extend will add the fixed cost or *Cost per use* of the activity to the cost accumulator's “_cost” attribute. In addition, it will calculate the time-based cost, including the variable cost of the activity as well as the variable 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, Extend 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

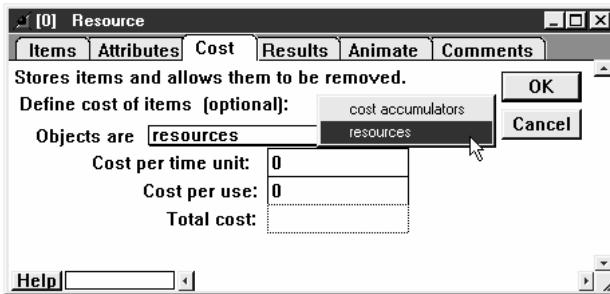
When a cost accumulator enters a queue-type block, Extend will calculate the time-based cost, including the waiting cost of the cost accumulator (calculated from the cost accumulator's “_rate” attribute) as well as the variable cost of any resources currently combined with cost accumulator. The time-based cost is added to the “_cost” attribute of the cost accumulator.

For each cost accumulator that passes through the block, Extend 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.

BPR

In resource-type blocks

As shown below, some resource-type blocks are capable of providing either cost accumulators or resources.



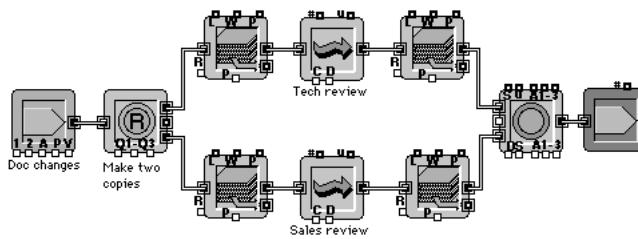
Cost tab of Resource block (Windows)

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 many business processes, different portions of the process 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. For example, when a computer manufacturer makes changes to their sales brochures, a copy of the proposed changes is sent to the technical department for a technical review, while another copy is sent to the marketing department for an additional review. The initial document change can be split into two copies using the Operation Reverse block. The two reviews are worked on in parallel, then combined using an Operation block, as shown below:



Multiple cost accumulators

BPR

When the two cost accumulators, the Tech review and the Sales review, 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 resources with cost accumulators” on page B148).

BPR Chapter 12: Statistics and Model Metrics

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 you need for your analysis and the decision-making process. This chapter will provide an overview of important statistical concepts and will discuss ways to extract and document the critical information contained in your models.

The following blocks will be of main focus in this chapter:

Block	Type	Library
Accumulate	Holding	Generic
Change Attribute	Attributes	Discrete Event
Clear Statistics	Statistics	Statistics
Downtime (Unscheduled)	Generator	Manufacturing
Generator	Generators	Discrete Event
Get Attribute	Attributes	Discrete Event
Import	Generator	BPR
Information	Information	Discrete Event
Input Random Number	Inputs/Outputs	Generic
Measurement	Attributes	BPR
Readout	Inputs/Outputs	Generic
Timer	Information	Discrete Event

Unless otherwise specified, the models discussed in this chapter can be found in the “Examples \\ BPR \\ Accumulating Data” folder.

Statistical concepts

Although simulation is a quantified subject, there is still room for quick 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, and the sensitivity of the system to the input data.

The information in this section is intended as an overview and guide. For more information, refer to the statistics books listed in “Further reading” on page B10.

Constant values versus random variables

Constant values never change; random variables 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.

- Since all inputs are constant, the results of running a *deterministic* model is “determined” or set based on the input variables. For example, if your model uses constants to represent the time between arrivals of items and how long the operations take, each time you run the model the results will be identical. This is a good technique for early steps in the model-building process, since you can be assured that changes in results will be due to changes you make to the model and not to randomness.

BPR

Notice that each run of a deterministic model will give you the same exact measure of *model* performance but is unlikely to give you an exact measure of *system* or *process* performance. This is because real-world systems typically contain some element of randomness. Deterministic models do not show the effect of variability, which may be an important aspect of your system or process.

- Adding randomness to a deterministic model changes it to a *stochastic* or *Monte Carlo* model. How long a process takes, equipment reliability, and the time between arrivals of customers are all examples of numbers that are typically random. 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 (discussed below). This means that their range and possibility of values is predictable.

Extend provides several methods for including randomness in your models. As you saw in the Tutorial chapter, and as discussed in the section “Random numbers and probability distributions” below, the Import, Input Random Number, Downtime (Unscheduled), and Generator blocks allow you to select a random distribution or enter a table of values specifying an empirical distribution.

Random numbers and probability distributions

As mentioned earlier, the ability to include randomness and show dynamic aspects through time is one of the most valuable characteristics of a simulation experiment. Since most models have randomness, it is important that you understand random numbers and be able to choose appropriate distributions.

Random numbers and 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 Extend which calculates the numbers in the stream. Extend provides independent random number streams and the ability to specify a seed so that sequences of random numbers can be repeated.

The blocks that generate random arrivals or numbers are the Downtime (Unscheduled), Generator, Input Random Number, and Import. A random number is generated based on a seed. Each time a new random number is generated, a new seed is generated. Extend automatically assigns a separate seed to each random number-type block in your model. If you want to specify your own seed, you can enter a seed value in the dialog of those blocks. Any number entered as a seed in a block dialog will result in independent random number streams. 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 Extend is independent.

You can also specify a seed for the model as a whole in the Simulation Setup dialog. If you enter a 0 for the random seed or leave it blank in the Simulation Setup dialog, each random-number type block in the model will use a random seed. If you specify any other number for the random seed, it will cause those blocks to use the same sequence of random numbers every time the model is run. This gives repeatable results, allowing you to determine exactly how your changes affect the model.

BPR

Random 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 *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. When you gather data for a simulation model, it is seldom in a useful form. Stochastic models use distributions as a handy method for converting data into useful form and inputting it into models. Extend'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. By “filling in the gaps”, they 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.

The most important characteristics of a distribution are its *shape*, the *spread*, and the *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).

In Extend, you usually choose a distribution by selecting it from the popup menu in the dialog of a random-number type block, for example by selecting the normal distribution in the Import block. The functions that produce the distributions have one or more parameter arguments which define and control the characteristics (spread, maximum, etc.) of the distribution. In Extend block dialogs, these arguments are usually identified by the labels “(1)”, “(2)”, or “(3)” and “Location”. You specify the characteristics for the selected distribution by the values you enter for these arguments.

Using random numbers means either choosing the theoretical distribution that best describes the variability of the raw data, describing the data using an *empirical* or user-defined distribution (such as the empirical distribution in the Generator block), or fitting known data to a distribution. As seen below, there are several distributions you can choose in Extend.

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 you find that your 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.

BPR

There are also software applications which fit data to distributions. You would use these tools in situations where you have empirical data you want to model using random distributions, but the Extend distributions do not fit. These Companion Products can help you find the statistical distribution that best emulates your real-world data. The Input Random Number block has a Distribution Fitting tab from which you can launch a distribution fitting package, analyze empirical data, and determine the appropriate statistical distribution for a given data set. See the Help of the Input Random Number block for additional information.

When you have sparse or no data, this guide of common uses may help you select a plausible distribution in the Generator, Import, Downtime (Unscheduled), or Input Random Number blocks:

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.
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.

Distribution	Definition
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.
Empirical	Used to generate a customized or user-defined distribution with a special shape when the probability of occurrence is known.
Gamma	Typically used to represent the time required to complete some task. The distribution is shaped like a decaying exponential for <i>shape</i> (2) values between 0 and 1. For shape values greater than one, 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.
LogLogistic	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.
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.

Distribution	Definition
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.
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 or real)	Describes a 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.

BPR

The block descriptions in the appendices of the main Extend manual fully describe the distributions found in the Generator and Input Random Number blocks while Appendix A in this manual describes the distributions found in the Import block. Extend's distributions are also described in the Help of those three blocks.

For more information about choosing and using distributions, the books *Discrete-Event System Simulation*, *Improve Quality & Productivity with Simulation*, and *Simulation Modeling & Analysis* are excellent resources; they are listed in “Further reading” on page B10.

Distribution fitting

For cases in which you have empirical data which you want to model using random distributions, Extend offers interfaces to a number of distribution fitting packages. These Companion Products can help you find the statistical distribution that best emulates your real-world data. The Input Random Number block (Inputs/Outputs submenu of the Generic library) has a Distribution Fitting tab from which you can launch a distribution fitting package, analyze empirical data, and determine the appropriate statistical distribution for a given data set. See the Read Me file for additional information on distribution fitting packages and other Companion Products.

StatFit distribution fitting example

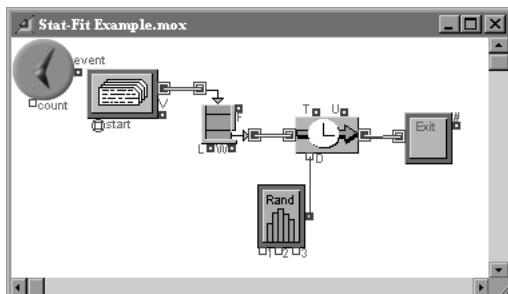
In simulation models, it is often useful to characterize a random input with a probability distribution (e.g., inter-arrival times and demand rates). Typically, this requires historical data documenting the system's behavior and some analysis to find an appropriate distribution. There are two advantages to using statistical distributions rather than historical data as inputs to a model:

- Values for an input random variables are not limited to what has happened historically.

- For continuous distributions, an infinite pool of data exists. There is seldom enough collected data to support multiple simulation runs.

StatFit 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 random process. An interface has been developed which allows the modeler to easily access the power of Stat Fit from within Extend.

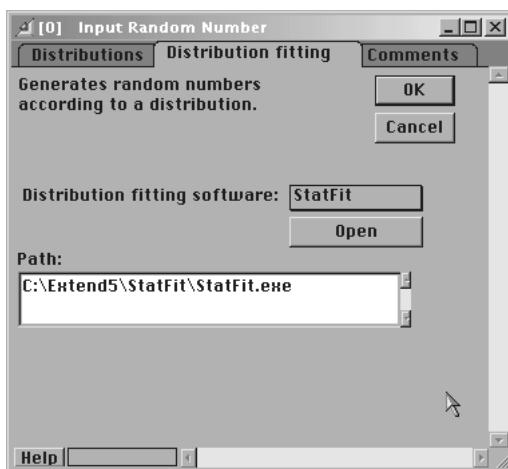
In the simple example below, the distribution used by the Input Random Number block will be chosen using StatFit. The *Statfit Example* model can be found in the “Examples \ Suite” folder.



StatFit Example

BPR

- From the Input Random Number block’s dialog, go to the Distribution Fitting Tab.
- Under “Distribution fitting software:” select Stat Fit and click the open button.



StatFit pathname

- Input Random Number dialog showing distribution fitting tab

- The StatFit application should appear on your screen.
- Go to Stat Fit's File menu and select open.
- Choose the file named StatFtEx.sfp.

The 32 historical data points appearing in this project will be used by Stat Fit to define which distributions and associated parameters might be appropriate for the Input Random Number block.

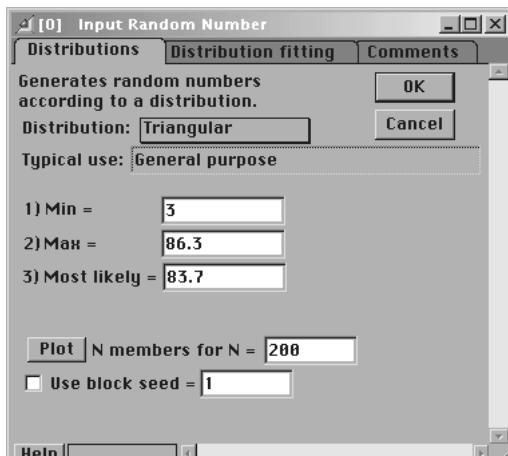
- Go to the Fit menu, select autofit and click OK.

A window appears displaying a list of parameterized distributions that have been ranked according to goodness of fit.

- Next, click the Export button.
- Pick Extend under the application window, and then choose the desired distribution.
- Click OK and go back to Extend.

The Input Random Number block's dialog should reflect the choice you made in Stat Fit.

BPR



Dialog modified by StatFit results

Additional StatFit documentation is available in StatFit's Help menu.

Changing parameters dynamically

Another name for the variable or constant number you input in a model 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*, such as based on some model condition. For instance, you might want to show that a clerk who takes 10 minutes to process an order in the

morning takes 11 minutes to process an order at midday and 12 minutes later in the afternoon. The most common method for doing this is to use another block which controls how the parameter changes. You can change both constant and random parameters dynamically:

- You can dynamically change a parameter by having it change to one of a series of constants, for example changing the parameter depending on the time of day for the clerk mentioned above. This allows you to predict exactly what the value will be at any point in time. You would commonly use the Input Data block (from the Generic library) to do this, as described in “Scheduled processing time” on page B106.
- You can also cause a random number to change dynamically by changing one of the distribution’s arguments. For example, rather than set the mean of an exponential distribution in the Import block’s dialog, you could connect an Input Data block (from the Generic library) to the “1” input of the Import block, and vary the mean dynamically. You will see how to do this in the section “Generating arrivals with custom intervals” on page B74.

You can also use Extend’s control blocks to interactively manipulate values. For example, the Slider control (accessed using the Control command of the Model menu) can be used in the same manner as the Input Data block, discussed above.

BPR

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 (theoretical) 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 $x \pm y$, 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.

The blocks in the Statistics library not only summarize and report statistical information about the model, but calculate confidence intervals given various levels of confidence. For example, the Activity Stats block allows you to select a confidence level and reports confidence intervals for arrivals, departures, and utilization. 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.

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 very important to ensure that the numbers you enter accurately represent the details of the actual system. The significance and relevance of your analysis will depend on how closely your model’s inputs correspond with real world data (the phrase “garbage-in/garbage-out” is very appropriate).

If you want more information about confidence intervals, two excellent sources listed in “Further reading” on page B10 are *Simulation Modeling & Analysis* and *Improve Quality & Productivity With Simulation*.

Note If you set a seed in the Simulation Setup dialog, by default each simulation run will produce the same result. If you set seeds in the blocks, you'll get repeatable sequences of random numbers. This invalidates the purpose of using confidence intervals. A way to mitigate this is to use the Random Seed Control block (in the Statistics library). The “Do not resend any seeds” option in this block controls whether or not the random number seed is reset at the beginning of each simulation run.

Multiple runs and Monte Carlo simulations

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 for Monte Carlo simulations or when performing sensitivity analysis.

- *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 samples. An example of a static model is a spreadsheet which contains sales forecast information; an example of a dynamic model is a simulation of a bank line which changes over time. Applying Monte Carlo simulation techniques to a dynamic model is also known as stochastic modeling, which is discussed in the section titled “Constant values versus random variables” on page B158. You can build both static and dynamic models with Extend.
- 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 the main Extend manual.

Whenever a model contains random variables, you should plan on running a simulation at least 3-5 times to estimate the variability in output. Then you can use statistical sample-size calculations (discussed in “Determining the length and number of runs” on page B68) to determine how many additional simulation runs are required to obtain an accurate estimate of the model's performance. Some tools in Extend for evaluating multiple simulation runs are:

- Histograms (in the Plotter library) which show the shape and any tendencies in the range of results.
- Error Bar plotters (from the Plotter library) that show how a variable is affected by randomness through the course of the simulation run.
- MultiSim plotters (also in the Plotter library) that can retain data and show the results of up to four simulation runs.
- Confidence intervals (reported by some of the blocks in the Statistics Library) which show the probability that a certain range captures the true mean for a simulation model result.

Measuring and verifying simulation results

As discussed in “Model verification” on page B69 and “Model validation” on page B69, you will often need to gather and document other information such as operation utilization, resource use, queue length, or throughput. Extend makes this easy.

Measuring performance and debugging models

Extend facilitates model analysis and provides several methods for reporting simulation results. See the main Extend manual for more details about many of the following techniques:

- Dialog boxes display data pertinent to the specific block and in some cases automatically perform statistical calculations. For instance, the dialog of the Buffer block 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 at “Using Notebooks for displaying simulation results” on page B173.
- 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 blocks such as operations and activities outputs utilization values. You can attach any value output to a Plotter, Discrete Event block to plot information about model performance, as discussed below. You can also attach value outputs to value inputs on diagnostic-type blocks, such as to the ReadOut block from the Inputs/Outputs submenu of the Generic library, to display information about that output.
- Plotter blocks, from the Plotter library, conveniently display graphs and tables of data over time. Plotters 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.
- Animation shows the flow of items in a model, levels of values, etc. Extend blocks have built-in customizable animation; you can also add custom animation through the Animate tab in block dialogs or by using blocks from the Animation library. Animation is especially useful for verifying your model since it can show you 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.
- There are numerous blocks that can be used for debugging your models and verifying results. For example the Stop block from the Generic library stops the simulation and notifies you when its input goes above or below a specified level. The Status block in the Discrete Event 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). The Information block (also from the Discrete Event library) is extremely useful for gathering data and debugging models, as shown in the section “Getting information about items” on page B169.

The main Extend manual has an entire section devoted to blocks that are used to debug models and report information.

- 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.
- Running simulations multiple times, such as for Monte Carlo simulations, gives ranges of values indicating the possible outcomes for the model.
- The Report command, discussed in the main Extend manual, instructs Extend to generate a text file of the final model results. You can report on all the blocks in a model, or use menu commands to specify which blocks are included in the report. Reports are especially useful for outputting to other applications, such as statistics packages, for further analysis.
- Some of the blocks in the Statistics library report and statistically evaluate results. For example, the Queue, Stats block displays information about every queue-type block in the model and calculates the confidence intervals based on the results. As discussed in “Clearing statistics”, below, the Clear Statistics block 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 which 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 main Extend manual for a detailed discussion on the discrete event messaging system.

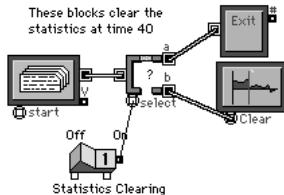
BPR

Clearing statistics

As discussed in “Non-terminating systems” on page B67, when you start 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. 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 in the Statistics library is used to reset statistical accumulators for the blocks specified in its dialog, eliminating the statistical bias of the warm-up period. You can con-

trol when the statistics are cleared by setting the “Clear statistics at time:” option within the dialog of the Clear Statistics block or by passing an item into the “Clear” connector as shown below:



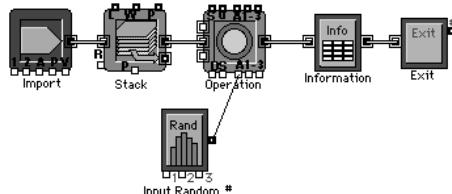
Statistics clearing example

In the example above, statistics are cleared after 40 time units. Notice that the Switch control allows you to turn the clearing feature on and off manually, without having to change the model. The above model can be found in the “Statistics” folder.

Getting information about items

When you are 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 Information block is particularly useful for verifying model data because it provides important information about each item as the simulation runs. To use the Information block, connect its input to the block of interest and connect its output to the rest of the model, as shown. The models below can be found in the “Accumulating Data” folder.

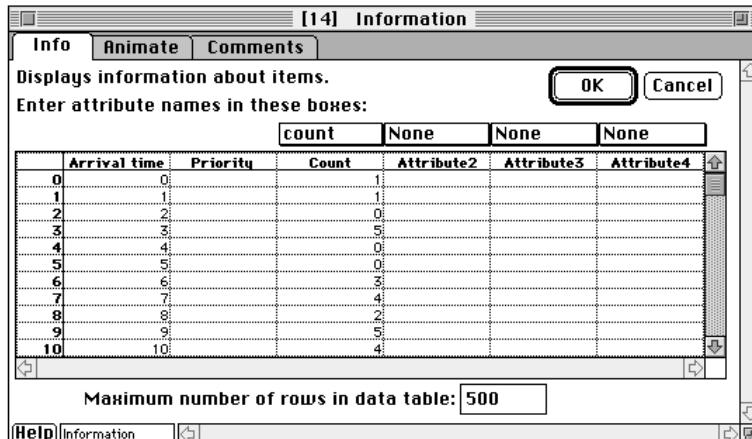
BPR



Verifying information using the Information block

The Information block pulls in an item, gets information, then passes the item to its output with no delay. In the Information block dialog, enter the attribute names for any attributes you wish to

track, then run the model. The dialog reports the time the item arrived in the block, its priority, and the attribute values for up to four named attributes:



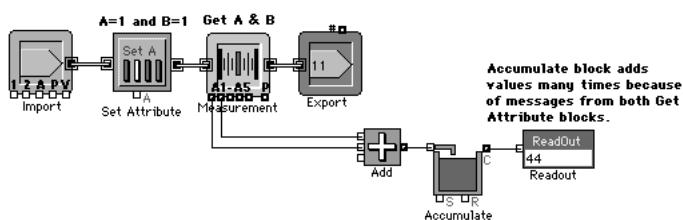
Gathering information about items (Macintosh)

BPR

Since the Information block can use a lot of memory, you should put it in while you are testing, then remove it when you have verified that the section is working as expected.

Avoiding a conceptual error

It is important that you do not make the error of assuming that you can combine attribute values and then accumulate them, as in the following example:



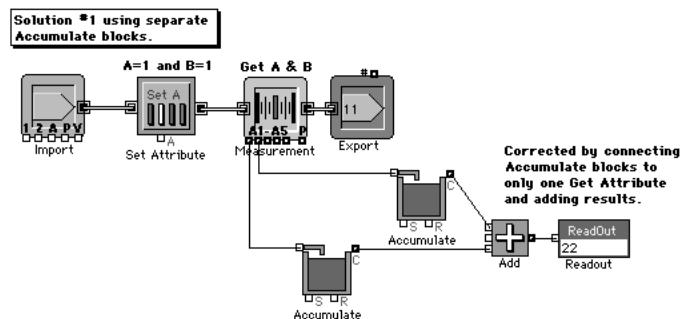
Attributes Error model: Modeling error regarding accumulating attribute values

The problem with this model is clear if you look at the numbers displayed on the Readout and Exit blocks. Since the values of attribute A and attribute B are both 1, the accumulated total displayed on the Readout block (from the Generic library) should only be twice the value displayed in the Exit block; clearly this is not the case in the model above.

The reason for this has to do with the message passing system in discrete event models. Individual items travel through the Get Attribute blocks sequentially. As an item passes through the first Get attribute block, that block will send a message and the value of the attribute to the Add block (from the Math submenu of the Generic Library). At that point the Add block will recalculate,

and send a message and the value to the Accumulate block. When the item moves to the second Get Attribute block, it will send a message to the Add block again. This means the Accumulate block will be getting 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 an Accumulate block (either directly, or indirectly as shown above).

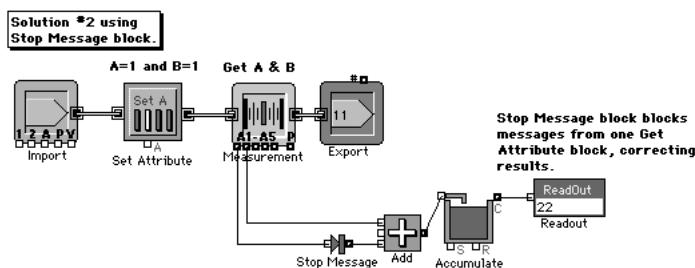
The best solution for this is to have a separate accumulator for each of the Get Attribute blocks. This would cause each Accumulate block to get only one message and value for each item passing through the Get Attribute blocks. The model with this solution is:



Attributes Error model: solution # 1

BPR

An alternate solution for this kind of problem involves using the Stop Message block in the Utilities library. This block is designed to solve problems of this nature; it stops messages from being passed through a value connection. As you can see in the model below, connecting the Stop Message block between the first Get Attribute block and the Add block allows the value to be passed but prevents the extra message from being passed. This causes the correct results to occur.



Attributes Error model: solution # 2

Timing the flow of items in a portion of the model

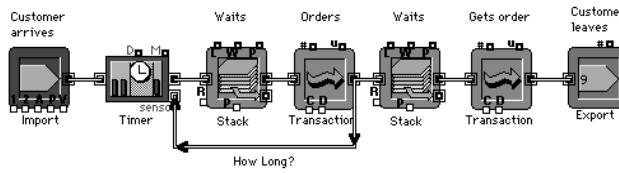
In addition to performance information that is directly available in a model, you may want to time how long it takes an item to go from one part of the model to another. For example, you may want to determine how long a customer waits in line to place an order, or how long it takes that

customer to get served once the order is placed. There are two methods for timing items, as discussed below. The models discussed in this section can be found in the “Timing Items” folder.

Using the Timer block

The Timer block displays the time that it takes an item to pass between two parts of the model. To use this block in a model, place it at the point where you want to start timing. Connect its regular item output to the rest of the model, and connect its *sensor* input to the output connector at your chosen end point. As the model runs, the Timer places a time tag on each item that passes through it, so the time it travels can be tracked.

For example, assume you want to track how long it takes customers to wait in line and place an order. You can connect a Timer block in the model as shown:



Timing items in a section of the model

BPR

The Timer dialog displays the time each item leaves. It also displays the delay time, that is, the time it takes for the item to travel to the end point once it leaves the Timer block. You can plot the actual cycle time (the delay) by connecting from the ‘D’ output connector to a Plotter Discrete Event; you plot the average delay by connecting from the “M” output.

If you want the delay time to also include the time items wait to be pulled into the next operation, you should follow the Timer block with a Stack block (as was done above). Then, the item will be held in the queue and the wait time counted. You can see this extra time in the following dialog:

	Depart Time	Delay Time	Attrib Value	Up
0	0	0	2	
1	0.5086561444	3.4913438555		
2	1.8296264654	4.1703735345		
3	2.4579643190	5.5420356819		
4	4.2375906609	3.7624093390		
5	5.0594330122	6.9405669877		

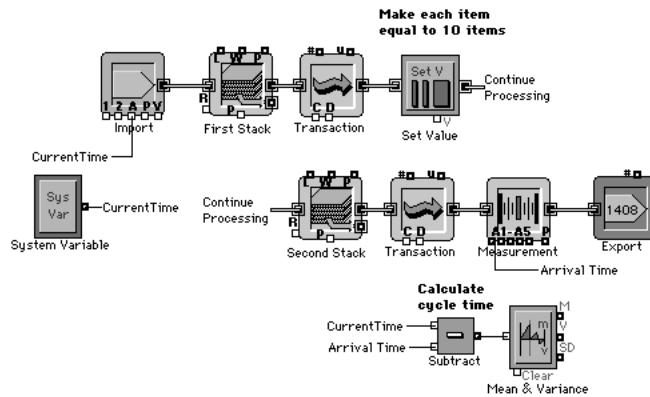
Timer dialog showing delay plus wait time (Macintosh)

In the model, the ordering process is set to take 2 time units and customers arrive exponentially with a mean of 1. Therefore customers arrive more frequently than orders are processed. New customers come into the waiting line but cannot place an order because the customer in front is still being waited on. This causes a delay longer than the time it takes to place an order.

If a name is given for an attribute in the Timer block, then only items with that attribute are measured. Note that the *sensor* connector does not pull items, it only views them. The customers are not removed from the simulation by being “sensed”, they continue on to the next line where they wait to pickup their order.

Using attributes to record times

As an alternative to using a Timer block, you can use an attribute to hold the time that an item arrives in a section of the model. When the item leaves that section, subtract that attribute value from the current time. This is illustrated by the following model:



Timer alternative: Attributes tracking time

BPR

As each item is generated, it gets an attribute with its value set to the current time. The System Variable block (from the Information submenu of the Generic library) supplies the current simulation time. At the point where the item is finished processing, the attribute value is subtracted from the current time and the cycle time is tallied using a Mean and Variance block.

One advantage of this approach, compared to using a Timer block, is that items can be copied or routed to different points without affecting the timing information. For example, each item arriving in this model represents a batch of items that undergo processing, first as a batch and then as individual items. It is more convenient to use the “attribute” approach to time the flow, since the arrival time is automatically copied when the items are duplicated in the second queue. This method of using attributes also requires less memory per item since the timer “tags” are not allocated if there are no Timer blocks in the model (see the help of the Executive block for information about how memory is allocated).

Using Notebooks for displaying simulation results

Extend's Notebook feature is particularly valuable in models where there are many identical blocks, such as a model with many machines. As described in the Extend manual, you can clone dialog items from your model into the Notebook, and you may arrange the clones in any way you like.

For instance, in the model discussed in “Simple routing” on page B92, there are three operations with queues running in parallel. You may want an easy way to track the time that items are wait-

ing for each operation without using plotters. Simply open the model's Notebook, clone information from the buffers into it, and add labels, such as:

	Ave. length	Ave. wait
Operation 1	0.0045739	0.0182957
Operation 2	1.5393738	2.2670555
Operation 3	0.3999151	1.1426146

Notebook items

Because the notebook can be very large, you can duplicate the items in a different arrangement in another place in the notebook if you want to view the items differently.



*SDI Industry*TM *Version 5.0*

Extend+Industry User's Guide

powered by Extend™

Extend is a trademark of Imagine That, Inc.

Simulation Dynamics, Inc.
416 High Street Maryville TN 37804-5836
Phone 865-982-7046 FAX 865/982-2813
<http://www.SimulationDynamics.com>

Copyright © 2000. Simulation Dynamics, Inc.
All rights reserved. Printed in the United States of America.

You may not copy, transmit, or translate this manual or any part of this manual in any form or by any means, electronic or mechanical, photocopying, recording, or information storage and retrieval systems, for any purpose other than purchaser's personal use without the express written permission of Simulation Dynamics, Inc.

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.

Extend+Industry

Welcome to *SDI Industry*, an Extend module that will enable your company to build powerful simulations.

SDI Industry includes:

- **The SDI Database**

A complete system to easily manage all your model data, configure tables for reports and experiments, use Database-aware attributes (including string values), leverage dates, times, and other formats such as currency, and use database-aware blocks to build powerful model constructs.

- **Discrete Rate Flow Blocks**

A system for modeling high-speed packaging, continuous processing, or any business process where flow is better treated as a rate.

- **Fully Compatible with Extend**

Fully integrated with the most popular simulation product in the world.

This tutorial manual will provide a background in the development and application of *SDI Industry*, and will show you how to use the SDI Database as a tool for developing a model.

What is the SDI Database?

The SDI Database is the foundation of *SDI Industry*. The SDI Database system is specially designed from the ground up as a tool for simulation modelers. It is completely integrated with Extend, so that a Database is embedded in a model.

The database management system provided by SDI Industry includes data manipulation features in both Extend and Excel. Databases can be built and managed entirely from within Extend. For users who prefer using Excel, the provided Add-In transforms Excel into a model interface, supporting data transfer to and from the model.

SDI Industry includes database-aware blocks such as DB Lookup, DB Matrix Lookup, DB 3D Lookup, DB Write, DB Matrix Write, DB Extract, DB Inject, and DB Specs. The database management system is also comprised of table creation wizards, built-in table editors, and an Excel Add-In which allows the creation of DB workbooks. DB Workbooks serve as an Excel front end, or data interface.

Benefits

SDI Industry allows easy manipulation of all your model data. SDI Industry's data management system is designed specifically for simulation projects. By separating model data from the model itself, the system enables better project and experiment management, as well as more powerful model constructs.

The SDI Database provides unique capabilities for handling data. The SDI Database also allows for flexibility in creating, viewing and manipulating the data through the centralized management of data. For example, a machine's rate in Extend is typically accessed by 'double-clicking' on a block that represents that machine. If there are 10 machines in the model, each machine's rate must be entered in each block's dialog. The database provides a means by which the rates of all machines may be managed and viewed in one table.

The Embedded Database

A database is created in Extend by placing a Database Manager block onto an Extend worksheet. The Database Manager houses an embedded Active-X control, which stores multiple tables of data. These tables are accessed throughout the model by any database-aware block. The database is "embedded" because the data within the database is saved with the model. When the model is opened, the database becomes active.

Since Extend "knows" about the structure of all tables, you can configure database-aware blocks in the model to reference your data by using pop-up menus to select data from the database. Every database-aware block uses a programming interface (SDI Database API) to communicate with the model database. Refer to the Extend Programmer's Reference for more details of the SDI Database API.

Industry

Design Criteria for the Embedded Database

The embedded database is not meant to be a general-purpose database transaction and reporting tool. Instead, it is designed from the ground up as a flexible tool for simulation modelers. The following criteria have guided the SDI Database development:

- Access speed and write speed
- Low processing overhead (as compared to a transaction tool such as Oracle)
- Simplicity
- Granular Data Support (built-in randoms)
- Portability (easy import/export)

Importing Data from outside of Extend

SDI Industry provides an Excel Add-In for managing data in Excel workbooks and then importing the data into the SDI database. Using Excel is optional - data may be imported from any text

editing application that can create a text file provided that the data conforms to the SDI Database text format. An ODBC Import feature is also provided.

Once the data is imported into the model it is independent from the outside data source. When you save the model, all of the data present in SDI Database is stored with the model. You can transmit the model to others so that they can run the model, and even change the table data from within Extend.

Database Manager Block

The Database Manager block is found in the library SDI Tools.lix and is the main user interface for the SDI Database. Its functions include:

- Importing a database text file into the model
- Exporting an SDI Database to an outside source such as Excel, or any database text file.
- Creating the database from within Extend
- Displaying the database
- Editing the database
- Configuring the model's start date and time
- Deleting the database

SDI Database-Aware Blocks

Table data is accessible for reading or writing by database-aware blocks in the model. The general purpose database-aware blocks are the DB Lookup, DB Matrix Lookup, DB 3D Lookup, DB Write, DB Matrix Write, DB Extract, DB Inject, and DB Specs blocks. The DB Lookup family of blocks provide data retrieval while the DB Write family of blocks provide reporting capabilities. In addition, the Equipment Controller, Value Schedule, Join 5 (Percent), and Split 5 (Percent) blocks are database-aware. Several Item blocks including the Generator, Get Attribute, Information, Program, Set Attribute, and Shift blocks are also database-aware and may be found in the Items (DB) Library.

Industry

SDI Database Wizards and other Database Tools

SDI Industry uses several wizards to help make creating and changing the database simple and quick. These include the New Table wizard in the DB Lookup blocks, Equipment Controller block, Variable Schedule block, and the Join Percent or Split Percent blocks. The DB Inject and DB Extract blocks in the SDI Tools.lix can be used to make a non-database-aware block database-aware. These blocks work by using the clone tool to drag and drop dialog parameters to and from various blocks. The DB Extract block can put results from any block into a database table for analysis or you may find them useful for putting Resources into a database as shown in the DB Extract-Inject Demo Model. See the online help for specifics on how to use these blocks.

Learning to use SDI Industry

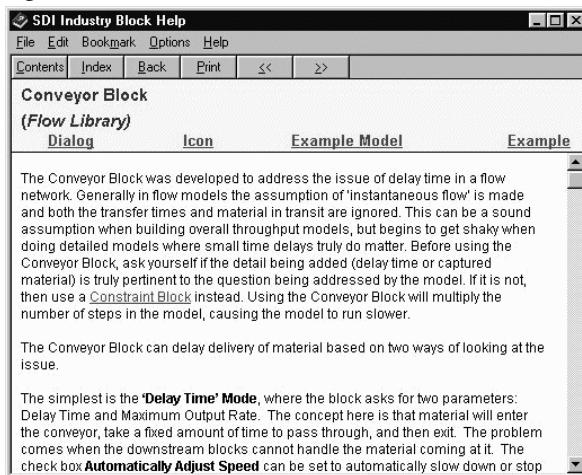
This manual contains several SDI Database tutorials which will walk you through setting up a database from within Extend and from within Excel. It will show you how to import data into and export data out of the model. The set up of many DB aware blocks is also covered in the tutorials. The use of Flow blocks is illustrated in several Flow tutorials. These tutorials will clarify the differences between Flow and Item blocks and show the proper use of Flow blocks.

Online Help

Extensive online help is available from within Extend by choosing Extend's *Help* menu. Most topics are covered in greater detail in the on-line help. The on-line block help may also be accessed by clicking on the 'Help' button on the bottom left corner of any block dialog. This will take you directly to that particular block's help file. The on line block help gives a description of the blocks function, describes each dialog along with all the options available, and shows the block's icon with a description of each connector. When a block's help is opened it brings you to the block description. To get to the dialog descriptions click on the 'dialog' jump. When in the dialog click on a box, checkbox, or window as if in Extend and a description of that function will pop up. Clicking on a dialog tab will move you to that dialog. The block help is divided by library, i.e Flow Library, SDI Tools Library, and Items (DB) Library.

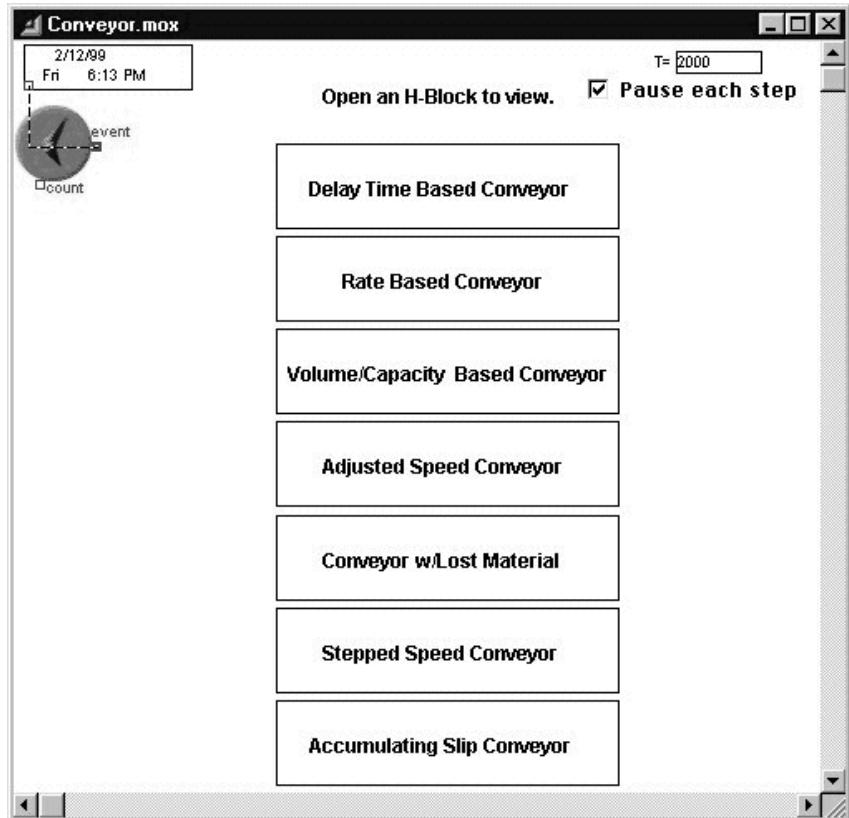
Block Models

The on-line block help includes a demo model for each block which demonstrates the uses of the block. A block model can be opened from within the on-line block help by clicking on 'Example Model' within each blocks help, or by opening Extend5\Extensions\SDIBlock Models\name of desired block.mox. when in Extend. For example, if you open the Conveyor Block help you will see the following dialog.



SDI Industry Block Help Conveyor Block Dialog

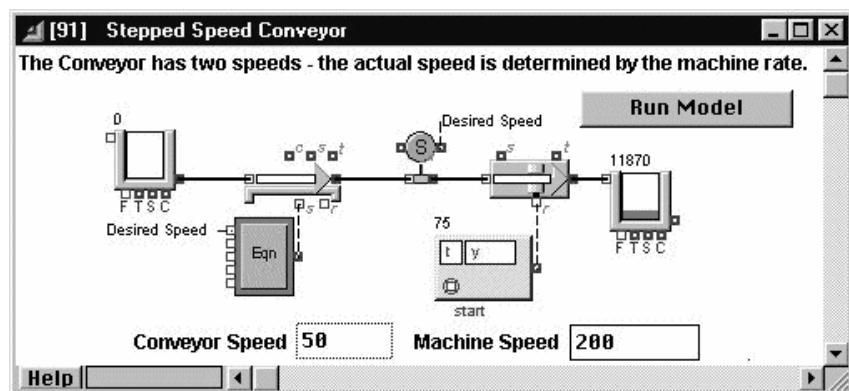
By clicking on 'Example Model' the following model will open up in Extend:



Conveyor Block Model Worksheet

Industry

By clicking on the 'Stepped Speed Conveyor' H-block the following Block Model is opened.



Stepped Speed Conveyor Model

SDI Database Tutorial 1

This tutorial will teach you to create a database from inside of Extend. You will see how easy it is to set up attributes and randoms using the database. We will build a model of a job shop which makes a variety of products. In this first tutorial you will learn the following:

- How to Build a Database from within Extend
- How to Make a Database Cell Random
- How to set the Model Start Date and Time
- How to set up a Database-Aware Attribute
- How to set up DB Specs and DB Write blocks

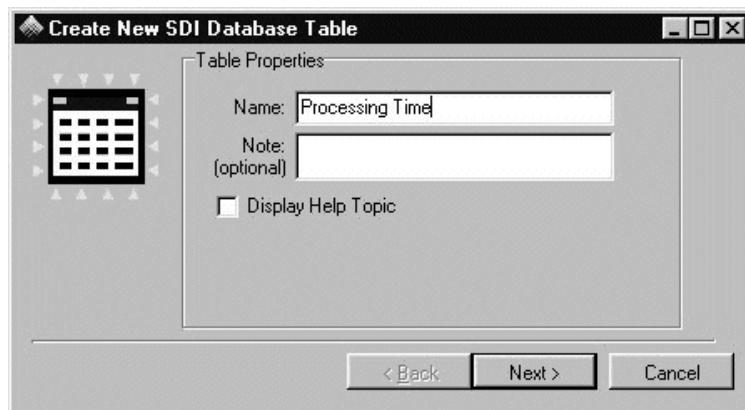
Building a Database from inside of Extend

We will begin the tutorial by identifying the data requirements for the model. The tutorial model will have three types of products (small, medium, large widgets). The products will be completed through a single step process with variable times for the step, and will be introduced into the model randomly. All the parameters needed for the model to run will be contained in the SDI Database.

Open Extend. On a new model worksheet place a Database Manager block from the SDI Tools library and an Executive block. Open the dialog of the Database Manager block and click on the 'Database' button.

Creating a Database Table

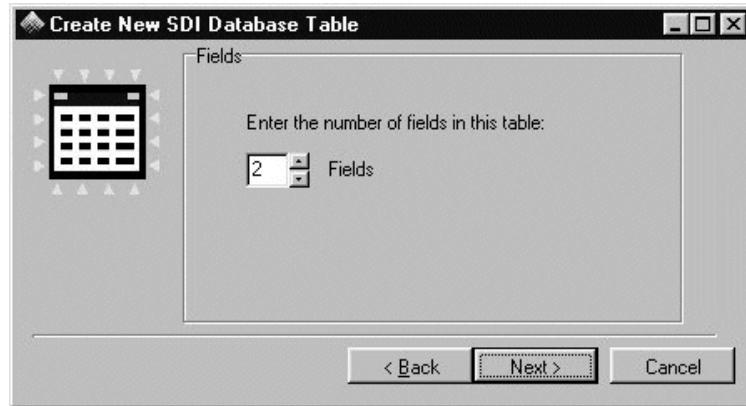
From the menu select *Insert:Table*. Enter 'Processing Time' for the Table name.



New Table Wizard Dialog

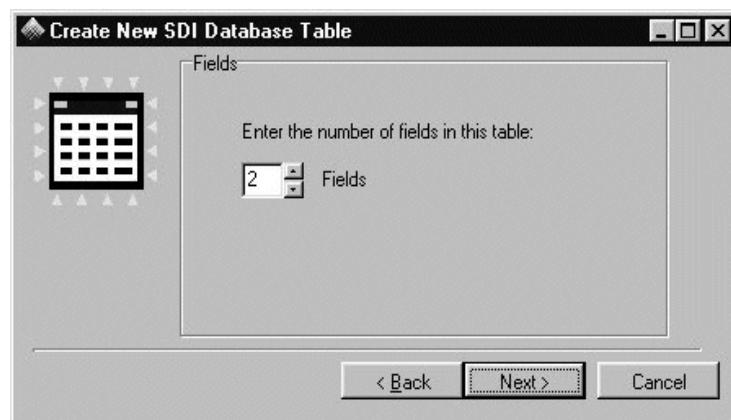
Defining Table Fields

Click on the 'Next' button. Enter '2' as the number of fields needed.



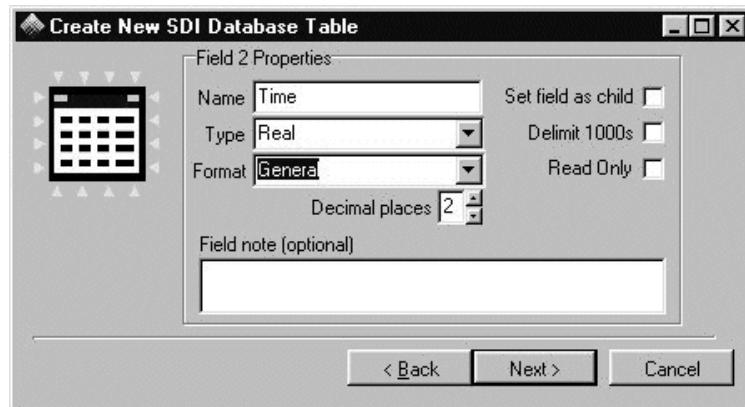
New Table Wizard Field Number Dialog

Click on the 'Next' button. For the first field click on the 'Name' field to highlight it and name the field 'Product'. Highlight the 'Type' field and choose 'String'. Highlight the 'Format' field and choose 'General'.



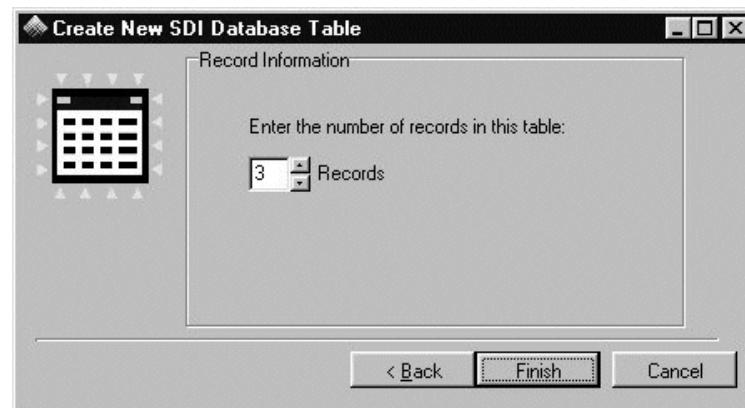
SDI New Table Wizard Field Edit Dialog

Click on the ‘Next’ button and name the second field ‘Time’, type: real; Format type: General.



New Table Wizard Field Name and Type Edit Dialog

Click on the ‘Next’ button and set the number of records at ‘3’.

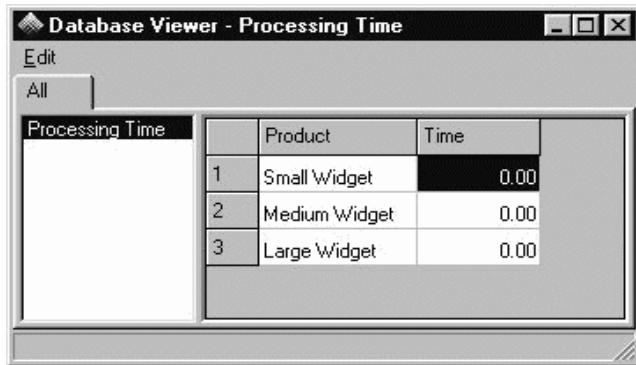


New Table Wizard Record Number Dialog

Click the ‘Finish’ button. The ‘Processing Time’ table is now in the database.

Entering Table Data

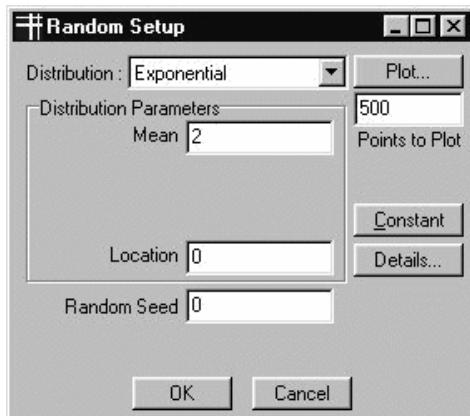
Click on the Database button and choose the ‘Processing Time’ table. In the cells under the ‘Product’ field enter the following strings: ‘Small Widget’; ‘Medium Widget’; ‘Large Widget’, respectively.



SDI Database Viewer Table Edit dialog

Making a Database Cell Random

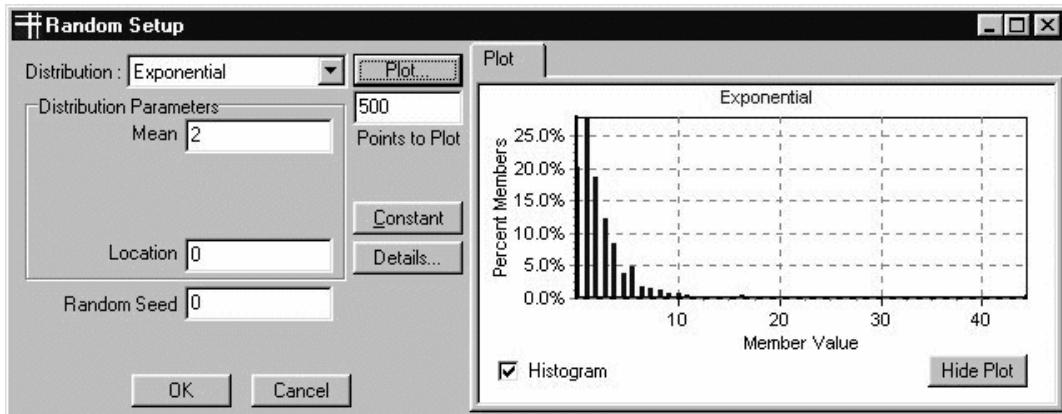
For each record in the ‘Time’ field make the time random by double clicking on the cell and then clicking on the ‘Random’ button.



SDI Database Random Setup field

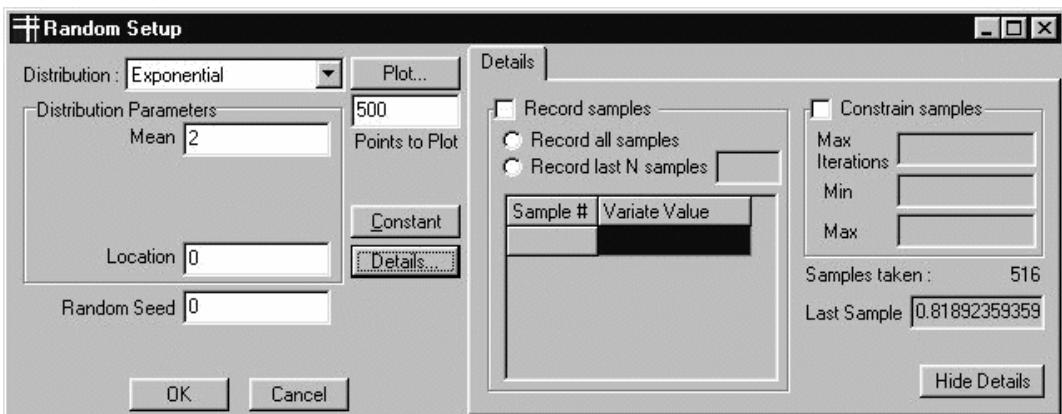
In each Random Setup dialog box choose ‘Exponential’ distribution from the pop up menu. Set the mean for records 1,2 and 3 as ‘2’, ‘3’, ‘4’, respectively. Click ‘Ok’ when finished with each records random dialog box. The data in the ‘Time’ field will turn blue indicating that these are now Random cells. This means that whenever this cell is read by a block such as a DB Lookup block, a random value is returned.

To view a plot of the distribution, click on the ‘plot’ button and this dialog will appear.



Random Setup Plot distribution graph

By clicking on the ‘Details...’ button you can set up the additional parameters.



Random Setup Details display

Repeat the new table steps to set up 2 more tables with the following parameters:

Table name: **Process Variables** Number of Fields **2** Number of records **1**

Field Name	Data Type	Format
Process	string	general
Distribution	real	general

In Field 1, cell 1 enter: ‘Time between Widget arrival’. In Field 2, cell 1 make the cell random with a Distribution = Exponential, Mean = 4.

Table name: **Model Output** Number of Fields **2** Number of Records **1**

Fields Name	Data Type	Format
Description	string	general
Parameter	real	general

In Field 1, Record 1 enter: Total Widgets Processed. Leave Field 2, Record 1 blank.

You should now have three tables similar to these:

The image shows three separate windows or tabs from a software application, likely a database manager, displaying different types of data structures. The first window, titled 'Product', contains a table with columns 'Step 1 Time' and 'Description'. It lists three rows: 'Small Widget' with a value of 3.946618, 'Medium Widget' with a value of 3.946618, and 'Large Widget' with a value of 3.946618. The second window, titled 'Model Output', contains a table with columns 'Description' and 'Parameter'. It has one row with the value 'Total Widgets Pro' and a parameter value of 0. The third window, titled 'Process Variables', contains a table with columns 'Process' and 'Distribution'. It has one row with the value 'Time Between Wi' and a distribution value of 3.946618. On the far right of the image, there is a vertical black bar with the word 'Industry' written vertically in white.

Step 1 Time	Description
3.946618	Small Widget
3.946618	Medium Widget
3.946618	Large Widget

Description	Parameter
Total Widgets Pro	0

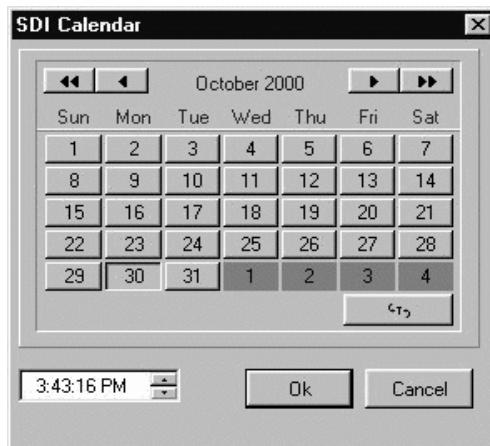
Process	Distribution
Time Between Wi	3.946618

Tutorial 1 Database Tables

Setting up the Date and Time

In the Database Manager block dialog choose the Date & Time tab and click on the dialog item which displays the date. A calendar will pop up which will allow you to pick the model Start Date

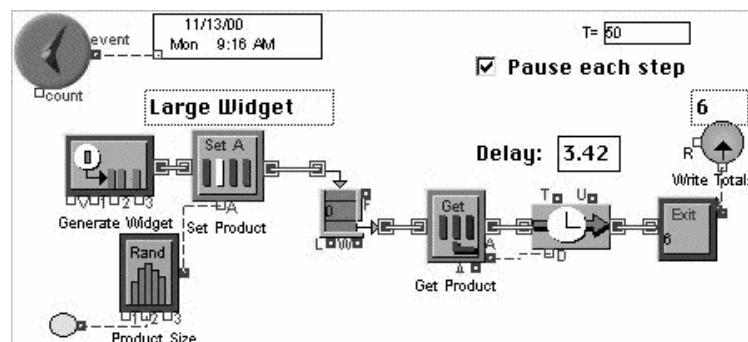
by selecting the date on the calendar and clicking on it. Set the model Start Time in the bottom left of the popup dialog. Click on the ‘OK’ button to accept the date and time.



SDI Database Manager Calendar Date Setup Field

Setting up the Model Worksheet

Now it is time to set up the model worksheet to look like the following.



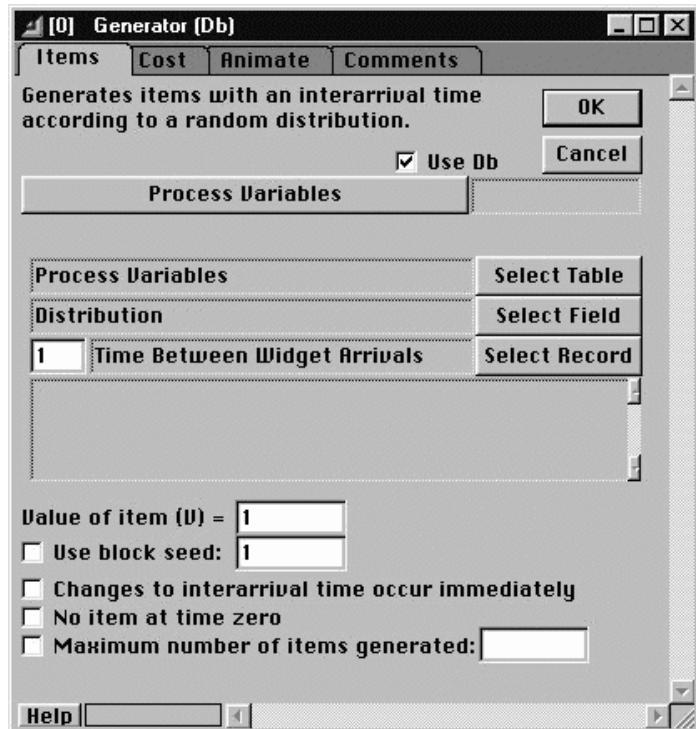
Database Tutorial 1 worksheet

The Database Manager, Executive, and DB Specs, and the DB Write blocks are in the SDI Tools library. Use the Generator (DB), Set Attribute (DB) and the Get Attribute (DB) from the Items (DB) library. Use the Queue (FIFO) and the Activity Delay blocks from the DE Library.

In this model, we are producing widgets in our job shop. In a simulation, these widgets generically are referred to as items. A Generator (DB) block creates the items to be processed in the simulation. These items are generated according to the random output variable from the database table which is referenced by the Generator (DB) block.

Setting up the Item Generator Block

From the Items (DB) library drag a Generator (DB) block onto the model worksheet. This block is database-aware, meaning that it will refer to the “Process Variables” table just created to determine how often items will be introduced into the model.



Generator Dialog setup

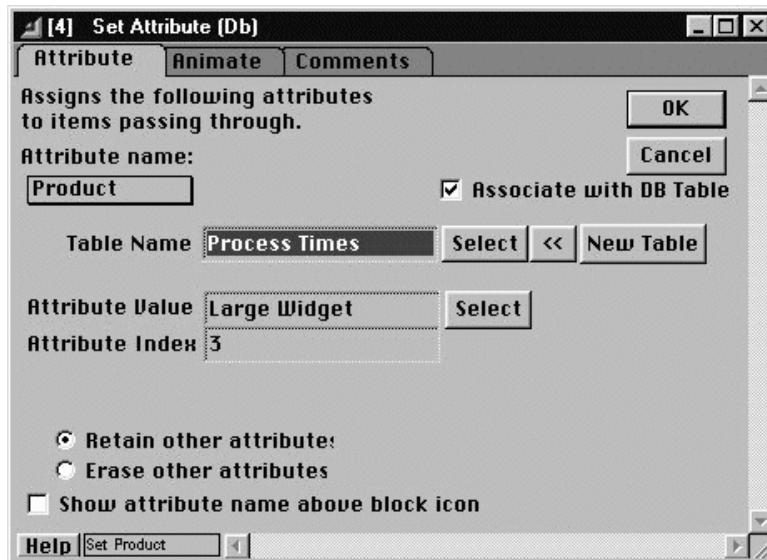
Industry

To set up the block, click “Select Table” and from the drop-down menu of available tables in the database, select the “Process Variables” table. Click “Select Field” and select “Distribution”. Click “Select Record” and select “Time Between Widget Arrivals”. The record number will then be displayed. Now, every time the Generator (DB) block generates an item, it will access the “Process Variables” table, look at the “Distribution” field of the “Time Between Widget Arrivals” record and use the value for that cell to determine when to generate the next item. Because the cell is a Random Cell, a random value will be returned each time.

Creating Database-Aware Attributes

Now we will describe the items generated by defining attributes for each item. From the Items (DB) library, drag a Set Attribute (DB) block onto the model worksheet and draw a connection

between the output connector of the Generator (DB) block to the input connected of the Set Attribute (DB) block.

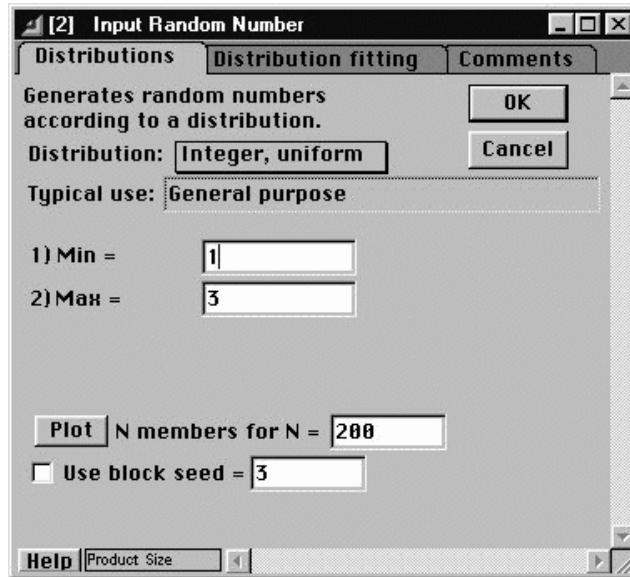


Set Attribute (DB) dialog setup

First, create an attribute named “Product”. To do this, click on “none” under Attribute Name to display a drop-down menu of existing attributes (we have none so far). Select “New Attribute” and enter “Product” for the name.

Next, check the box labeled “Associate with DB Table”. For the Table Name, select the “Process Times” table. The Set Attribute (DB) block will look up an attribute value from the first field of the selected table. In this case, the Attribute value will be “Large Widget”, “Medium Widget”, or “Small Widget”, the values we created in the “Product” field of the “Process Times” table. You could select one of these three values by clicking Attribute Value, Select, but instead, we are going to connect an Input Random Number block to the Set Attribute (DB) block’s “A” connector. (The Input Random Number block is located in the Generic Extend library). Open the Input Random Number block dialog and set up the block as shown below.

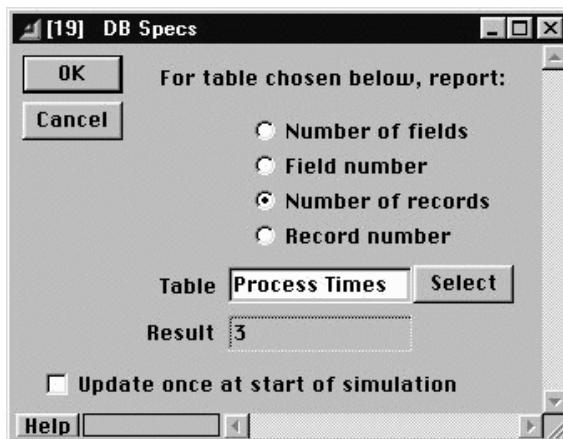
The input value from the Input Random Number block is used as the record number of the associated database table for looking up the attribute value.



Input Random Number Dialog

Attach the output connector of the Set Attribute (DB) block to the input connector of a Queue (FIFO) block. Use the default settings of the Queue (FIFO) block.

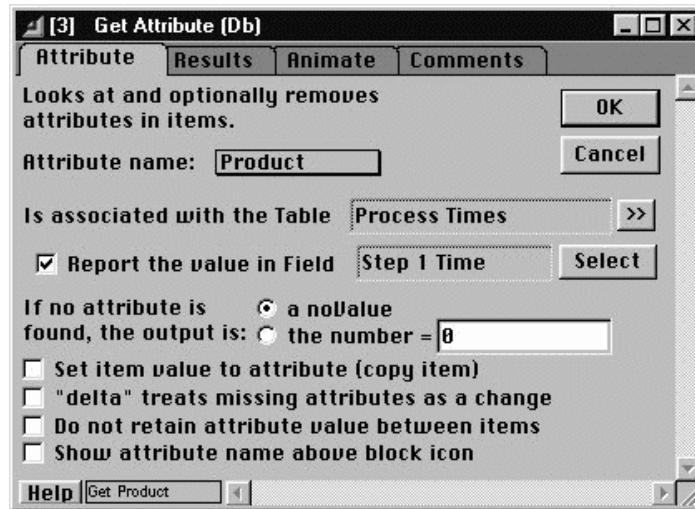
Setting up the DB Specs block



DB Specs dialog setup

The DB Specs block is looking up the number of records in the Process Times table. Attach its output connector to the #2 input connector of the Input Random Number block.

Once an item is associated with a particular attribute value using a Set Attribute (DB) block, a Get Attribute (DB) block enables us to access other data related to that attribute value, i.e., values in other fields in that related table.



Get Attribute (DB) Dialog setup

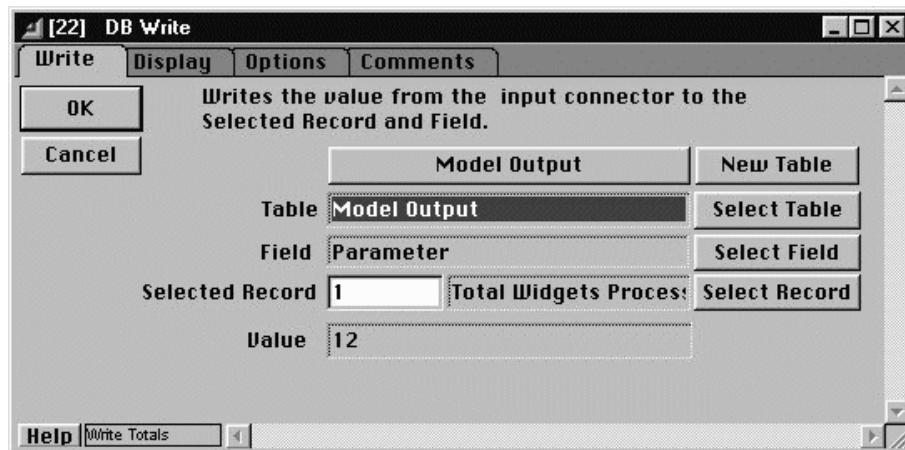
When an attribute is associated with the database, the Get Attribute (DB) block shows the table that the attribute is associated with. Select the field which contains the data you need by checking the ‘Report the value in field’ checkbox and using the ‘Select’ button to choose the correct field.

The “Process Times” table has a record for each Product. The product name is the first field. “Step 1 Time” is the name of the second field and each record in this field contains a Random Cell. To retrieve a “Step 1 Time” value for an item, drag a Get Attribute (DB) block onto the model worksheet and draw a connection from the output connector of the Queue (FIFO) block to the input connector of this block. In the dialog box, click “Attribute Name: None” and select “Product”. Check the “Report the value in Field” checkbox and select “Step 1 Time” for the Field to report.

This block will send a random value generated using the Random Cell distribution in the “Step 1 Time” field through its output connector for the product record associated with an item’s product attribute. Attach an Activity (Delay) block to the output connector of the Get Attribute (DB) block. Use the default values for the Activity (Delay) block. Draw a connection between the ‘A’ output of the Get Attribute (DB) block to the ‘D’ connector of the Activity (Delay) block.

Attach an Exit block to the output connector of the Activity delay block. Attach a DB Write block to the output connector of the Exit block.

Setting up a DB Write block



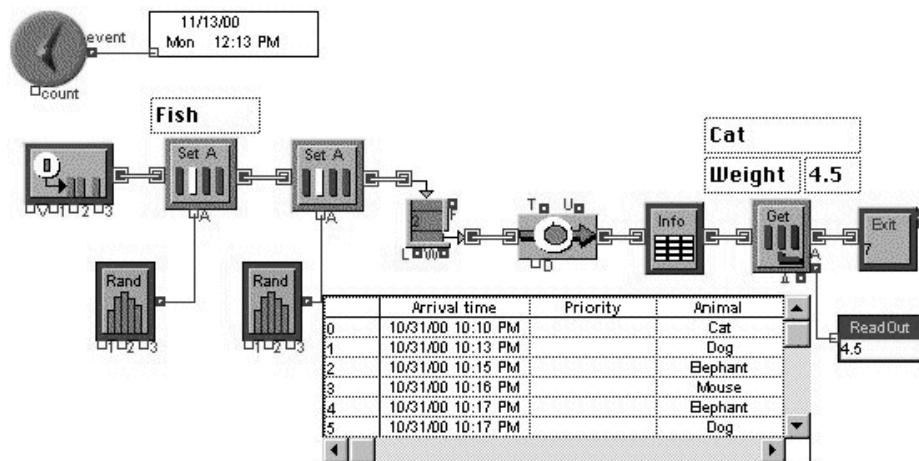
DB Write dialog setup

Set the DB Write block to write to the table “Model Output” in which the model data will be stored. The table can be set to acquire data from multiple runs in the options dialog of the DB Write block.

Use the Clone Tool to clone out the dialog parameters to your worksheet. Save the model, then run it.

Industry

Example: Using the Information (DB) block to verify a Model



Example: Animals

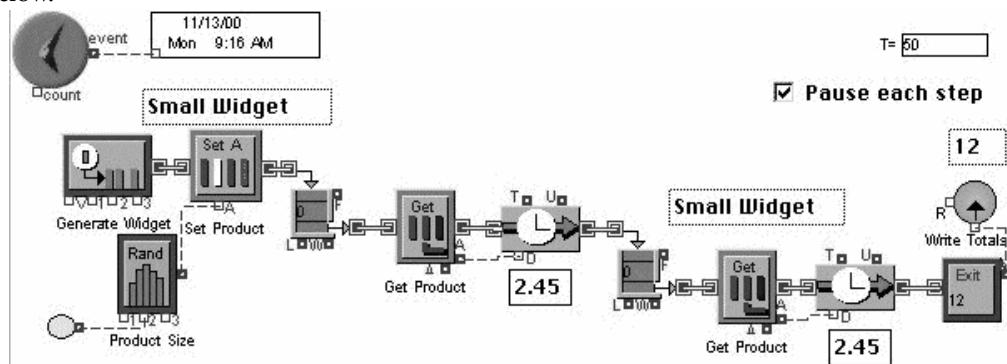
The above example model illustrates the function of the Information (DB) block as well as the Set Attribute (DB) and Get Attribute (DB) blocks. The table on the model worksheet is cloned out of the Information (DB) block and shows the Arrival time in the date and time format. These dates are relative to the Start Date you set in the Database Manager. The attributes are also shown as their string name and not the attribute value index number. The boxes cloned out above the Set Attribute (DB) and Get Attribute (DB) blocks are cloned out of the Results dialog of each block. This illustrates how the string attributes make identification of what is happening in a model much clearer.

SDI Database Tutorial 2

In this tutorial you will add to the model from Tutorial 1. You will learn the following:

- How to export a database from Extend to an Excel DB workbook
- How to edit tables and set up randoms in an Excel DB workbook
- How to import a database from Excel to Extend

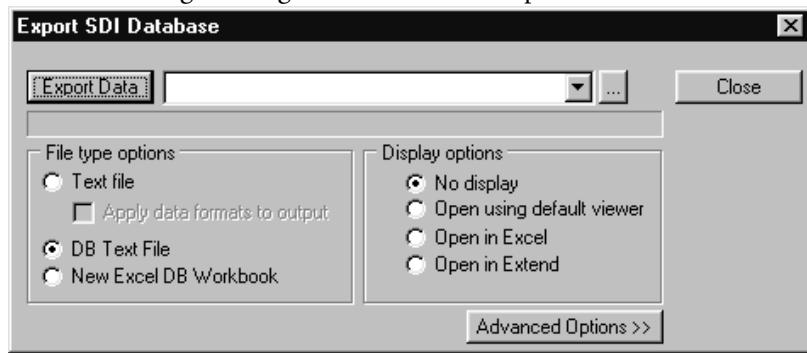
Begin Tutorial 2 by modifying the Tutorial 1 model. Duplicate the Queue (FIFO), Get Attribute (DB), and the Activity Delay blocks. Insert them on to the model and connect them as shown below.



Tutorial 2 model worksheet

Exporting an SDI Database to an Excel DB Workbook

Open the Database Manager dialog and click on the 'Export SDI Database...' button.



Export SDI Database dialog

Under 'File type options' select the 'New Excel DB Workbook' radio button. Under 'Display options' choose the 'Open in Excel' radio button. Type the name of the file you want to save the database to or click the browse button to select an existing file. Click on the 'Export Data' button to open up Excel with the SDI Database workbook. Later on, you may want to export one or more tables manually to an existing Excel DB workbook.

Steps to manually export the database:

1. Open the Database Manager block and select 'Export...'
2. Under 'File type options' choose 'DB Text File'
3. Under 'Display options' select 'No display'
4. Choose the destination file
5. Click on the 'Export Data' button
6. Open Excel (with the DB Add-In installed)
7. Select Menu *DB>New DB Worksheet*
8. Select Menu *DB:Import Text*
9. Select source file from the 'Open File...' button
10. Click the 'Yes' button

The data can now be saved and manipulated in Excel.

Editing Tables and Randoms in Excel

Open Excel and select the file your database is saved in. The ‘Process Time’ table should look like this:

Process Times		
Table	Notes	Fields
		string real
Product	Step 1 Time	
1 Small Widget	[Exponential,2,...]	
2 Medium Widget	[Exponential,3,...]	
3 Large Widget	[Exponential,4,...]	

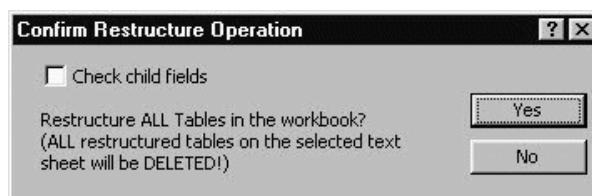
Tutorial 1 Process Times Schedules

We are going to add a third field to this table named ‘Step 2 Time’ with a string field type. To expand the table highlight all of the data in ‘Step 1 Time’ field and copy it to the adjacent column ‘e’. Change the name of the field to ‘Step 2 Time’. Change the mean of the randoms to ‘1.5’, ‘3.5’, and ‘4.5’ in records 1,2, and 3, respectively, by double-clicking on each cell. If the random dialog box does not appear check that menu *DB:DB Edit Enable* is checked. The resulting table should look like this:

Process Times		
Table	Notes	Fields
		string real real
Product	Step 1 Time	Step 2 Time
1 Small Widget	[Exponential,2,...]	[Exponential,1.5,...]
2 Medium Widget	[Exponential,3,...]	[Exponential,3.5,...]
3 Large Widget	[Exponential,5,...]	[Exponential,4.5,...]

Tutorial 2 Process Times Schedules

When your table is complete from the menu, choose *DB:Restructure ALL Tables*. The dialog shown below will appear. Click on the ‘Yes’ button to perform the restructuring.



Confirm Database Restructure Operation display box

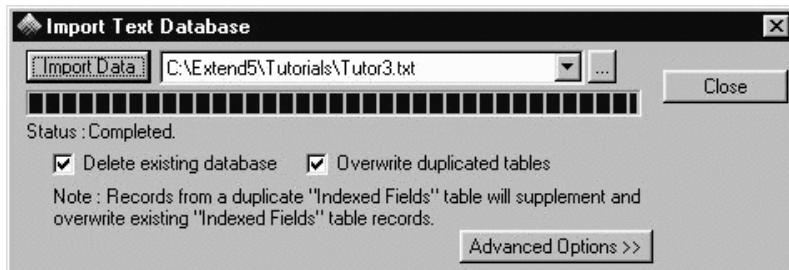
A small dialog box telling you that the restructure is complete will appear when it has finished.

Creating a DB Text File from Excel

While still in the Excel DB workbook select the menu item *DB:Text Output*. Click on the ‘OK’ button and choose the directory and file where you want to save the model. You may rename the AllTables.txt if you like. Choose ‘OK’ to create a DB Text file. This file can then be imported to an Extend model using the SDI Database.

Importing the DB Text File into Extend

Open the model from previous examples in Extend and open the Database Manager dialog. Click on the 'Import SDI Database...' button. The following dialog will appear:



DB Manager Read Database display box

Select the DB Text File just created by clicking the browse button. The status bar will reflect the progress of the import. If there are any errors detected in the DB Text File return to Excel, correct the problem, restructure the table(s), and import the file again.

Change the second Get Attribute (DB) block to select the Step 2 Time field. Run the model, then save it.

SDI Database Tutorial 3 (Advanced Concepts)

The purpose of this tutorial is to demonstrate how databases can be created and manipulated in Excel using the DB Add-In (included with SDI Industry), then imported by a model in Extend. The advanced indexing demonstrated in this tutorial may also be accomplished entirely within Extend using the SDI Database Viewer. We are going to build a job shop model by adding two more machines to the model and running them all in parallel to each other. The product will be routed to the machines based on a processing sequence table associated with each product. The step the product is on will determine which machine it is routed to. Each product will each go through the machines in a unique order. This will require an expansion of the database including the use of parent-child relationships (indexing).

Industry

In this tutorial you will learn the following:

- How to create a new Excel DB workbook
- How to set up indexing (parent-child relationships) in Excel
- How to create new tables in the DB workbook
- How to use the 3D Lookup block

Creating an Excel DB Workbook

Open Excel and from the menu choose *DB>New DB Workbook*. Change the tab name from 'New Tab' to 'Products' by double-clicking on the tab to highlight the text and typing in the new name.

Select the menu *DB:Insert User Table*. Click ‘yes’ on the next dialog box. Change the table name to ‘Products’ and give it three records. Enter the following data into the table:

Field Name	Data Type	Format
Product	general	string
Routing Table	general	integer

In Field 1, cells 1,2, and 3 enter ‘SKU 1’, ‘SKU 2’, ‘SKU 3’, respectively. In Field 2, cells 1,2, and 3, enter ‘Routing SKU 1’, ‘Routing SKU 2’, ‘Routing SKU 3’, respectively.

You should now have a table that looks like the following:

Products			
Table	Notes		
		Product	Routing Table
1	SKU 1		Routing SKU 1
2	SKU 2		Routing SKU 2
3	SKU 3		Routing SKU 3

Products table

Next, insert another user table called ‘Operations Table’ to set the route where each machine is located. Note: Insert a row by high-lighting the last entire row beginning from column A to the column containing the last field in the table and dragging the selected row down to the number of rows needed. Set up the table as follows:

Field Name	Data Type	Format
Operations Table	general	string
Route	general	integer

In Field 1, cells 1,2,3,4 and 5 enter ‘Milling’, ‘Tapping’, ‘Drilling’, ‘Cutting’, ‘Shipping’, respectively. In Field 2, cells 1,2,3,4, and 5 enter 1,2,3,4,5, respectively.

You should now have a table that looks like the following.

Operations Table		
Table	Note for Field 1	Note for Field 2
Fields	string	integer
	Operation	Route
1	Milling	1
2	Tapping	2
3	Drilling	3
4	Cutting	4
5	Shipping	5

Operations Table table

Repeat the steps for creating new table and create a table named ‘Routing Tables Directory’ with one field and three records as shown below.

Field Name	Data Type	Format
Operations Table	general	string

In Field 1, cells 1,2, and 3 enter ‘Routing SKU 1’, ‘Routing SKU 2’,‘Routing SKU 3’, respectively. You should now have a table that looks like the following.

Table	Routing tables Directory
Notes	Note For Field 1
Fields	Routing Table
1	Routing SKU 1
2	Routing SKU 2
3	Routing SKU 3

Routing Tables Directory table

Copy and paste the ‘Process Variables’ and ‘Model Output’ tables from the Tutorial 2 onto the worksheet.

Creating the Routing Tables (on a separate worksheet)

Before we create the Routing tables we are going to create a new worksheet (and tab) called ‘Routing’. Click on ‘Move Tables’ in the DB menu. Click on the ‘New Sheet’ button. Click on the ‘Rename Sheet’ and rename it ‘Routing’. Exit the ‘Move Table’ dialog and you will have a new worksheet called ‘Routing’. Click on the ‘Routing’ tab to move onto that worksheet.

Create another table called ‘Routing SKU 1’ with three fields and five records.

Field Name	Data Type	Format
Step	general	string
Operation	general	string
Time Estimate	general	real

In Field 1, cells 1,2,3,4, and 5 enter 1,2,3,4,5, respectively. In Field 2, cells 1,2,3,4, and 5 enter ‘Milling’, ‘Tapping’, ‘Drilling’, ‘Cutting’, ‘Shipping’, respectively. In field 3 make all the cells ran-

dom with an Exponential distribution. Set the means for cells 1-5 as follows: 0.8, 2.2, 3.8, 1.7, 0.5, respectively. You should have a table similar to this:

Routing SKU 1			
Notes		Note for Field 2	Note for Field 2
Fields	integer	string	real
Step		Operation	Time Estimate
1	1	Cutting	[Exponential,0.8....]
2	2	Drilling	[Exponential,2.2....]
3	3	Milling	[Exponential,3.8....]
4	4	Tapping	[Exponential,1.7....]
5	5	Shipping	[Exponential,0.5....]

Routing SKU 1 table

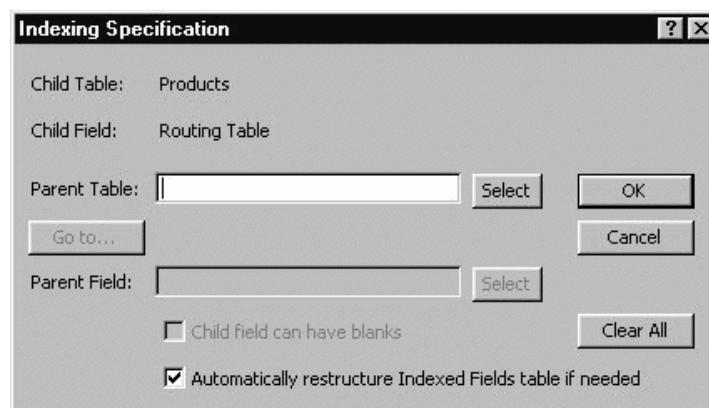
Now duplicate this table twice and name them ‘Routing SKU 2’ and ‘Routing SKU 3’. For each table change the order of the operations, the number of steps, and the time for each step. Be sure to have ‘Shipping’ once (and last) in each table.

Indexing Tables (Creating Parent-Child relationships)

Now that all the tables are created, there is still one more thing that must be done before restructuring and importing the database to Extend. You have probably observed that certain tables reference other tables in the database, e.g. the field ‘Routing Table’ in the table ‘Products’ is identical to the one in the ‘Routing Tables Directory’ table. When this is the case we must set up a parent-child relationship.

Double click on the ‘Routing Table’ field name in the ‘Products’ table to select it. The Indexing Specification dialog will appear. Select the ‘Automatically restructure the Indexed Fields table if needed’ checkbox. Click on the ‘Select’ button. Choose the ‘Routing Tables Directory’ as the Parent Table and then the ‘Routing Table’ field as the Parent Field.

Industry



Indexing Specification dialog box

Click ‘Ok’. A dialog will tell you that the changes are OK, or it will give you an error message (often for spelling). You will notice that the field name ‘Routing Table’ in the ‘Products’ table is now italicized. This indicates that it is a child field.

Table	Products	
Notes		
Fields	Type	
	string	integer
	Product	<i>Routing Table</i>
1	SKU 1	Routing SKU 1
2	SKU 2	Routing SKU 2
3	SKU 3	Routing SKU 3

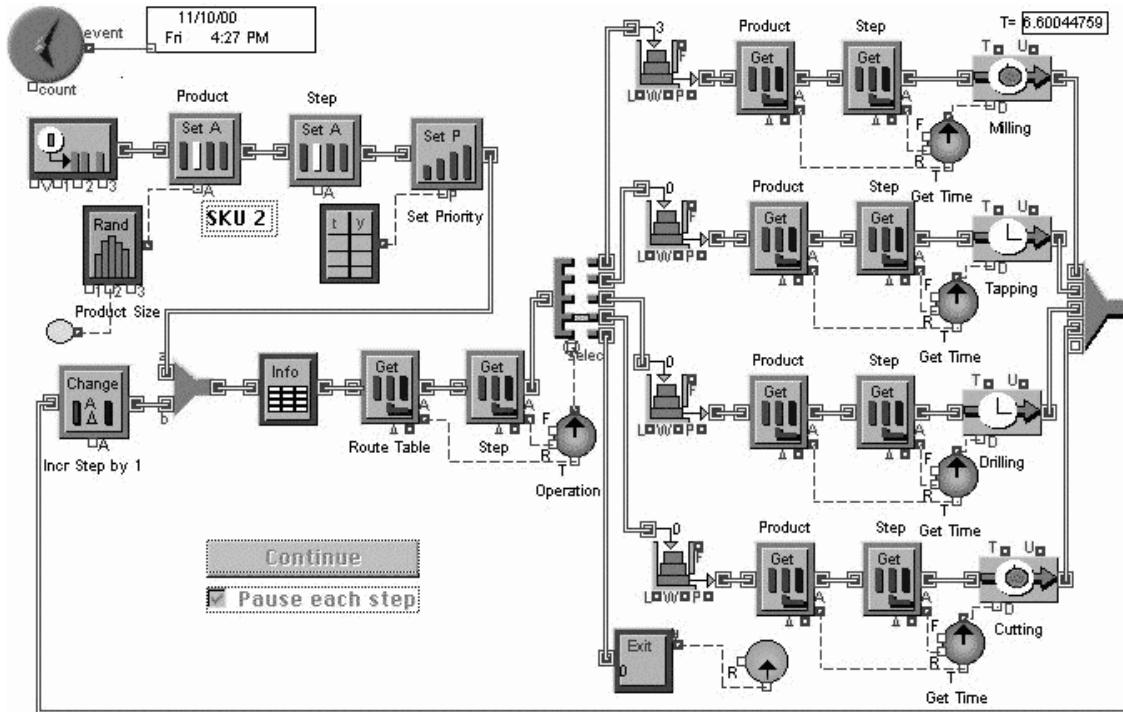
Products table

Repeat these steps for each of the ‘Operation’ Fields in the ‘Routing SKU “X” tables using the ‘Operations Table’ table as the parent table, and ‘Operation’ as the parent field. The database for this model is now complete. Restructure the tables using the DB menu function. Check the ‘Check child fields’ box. Use the menu *DB:Text Output* to export the DB Text file into the folder where the Extend model is located. Open up the model and import the DB Text file into the database by opening the Database Manager and clicking on the ‘Import Database’ button.

Setting up the Model Worksheet

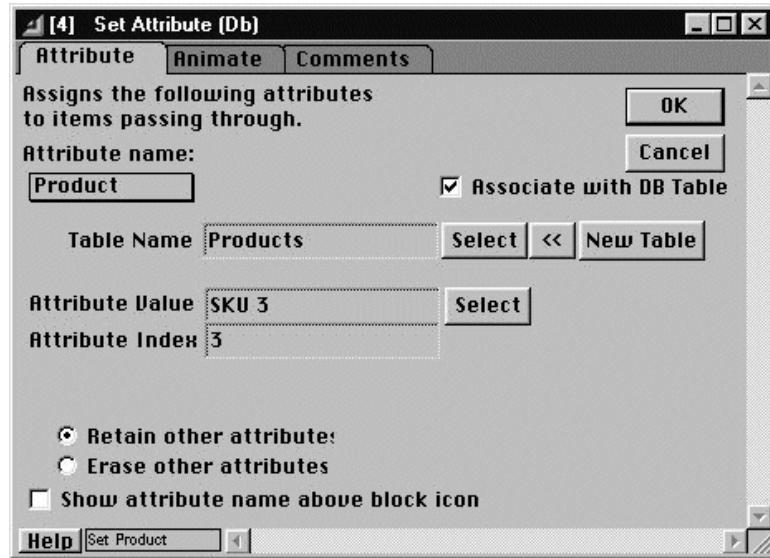
We will use 3D Lookup blocks along with the other database-aware blocks. A Set Priority block (DE Library) has also been added so that high priority products can be routed through the processes quickly. In order to do this the Queues must be changed from FIFO to Priority. Notice that the blocks between the Select DE Output (5) and the Combine (5) blocks (MFG Library) contain four identical configurations. Make one set and duplicate it. The Change Attribute block is found in the DE Library.

Modify the Tutorial 2 model to look like the following, or simply follow along by opening the completed model located at Extend5\SDI Industry\Tutorial\Tutor3.



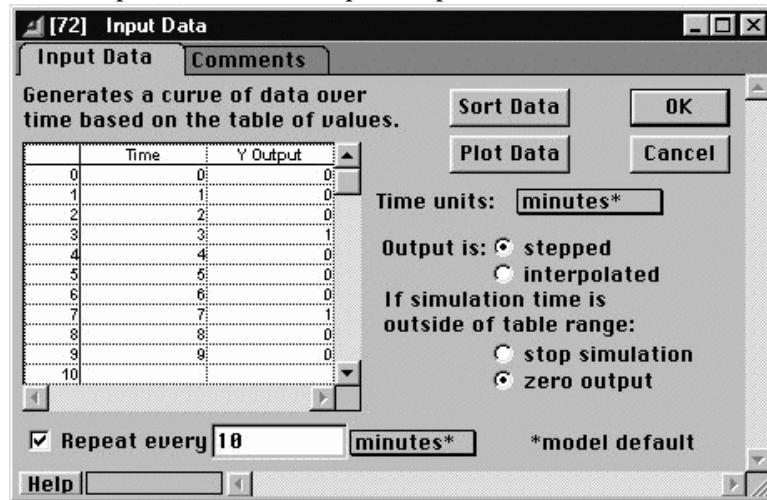
Tutorial 3 Model worksheet

The Generator and Random Input blocks are set up the same as in Tutorial 2. Set the DB Specs block to the 'Products' table. Configure the first Set Attribute (DB) block as shown in the dialog.



Set Attribute (DB) dialog box

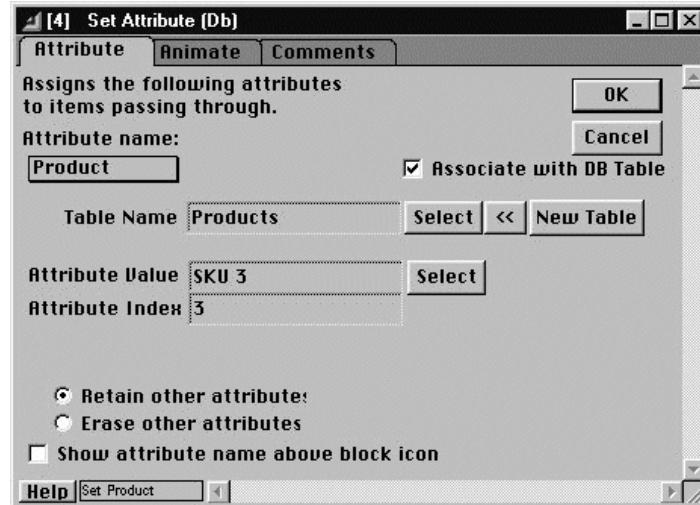
Configure the second Set Attribute (DB) block with an attribute ‘Step’ having a value of 1. This block is not associated with a table. Configure the Set Priority block to ‘0’ and attach an Input Data block to its ‘P’ Input connector. Set up the Input Data block as follows:



Input Data block dialog

Note: a Value Schedule block along with a DB Lookup block could substitute for the Input Data block if you wanted to access a database table for this data.

Configure the Get Attribute (DB) block labeled ‘Route Table’ as follows.



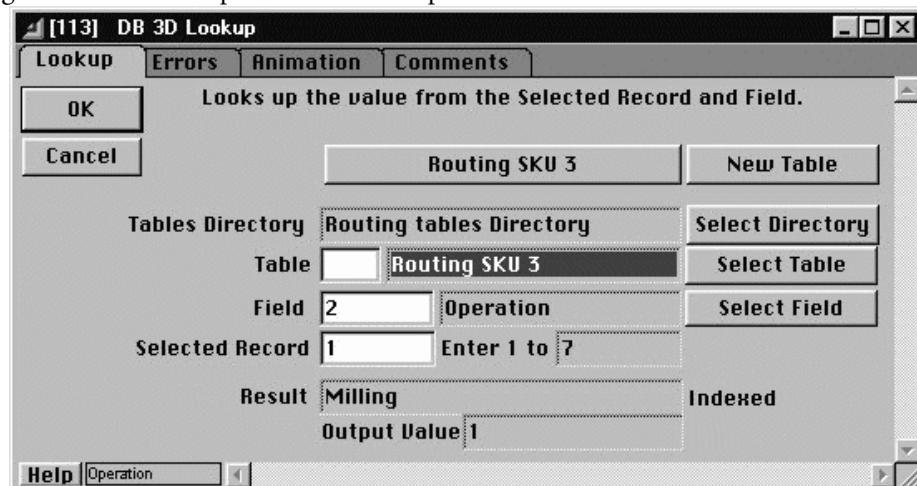
Get Attribute (DB) dialog

The Routing Table is indexed and will report the number of the table referenced. Configure the Get Attribute (DB) labeled ‘Step’ to read the attribute ‘Step’. Check the Select DE Output (5) blocks radio button to select the option ‘Item enters Select block before decision is made’. Set the ‘top Output is chosen by “select” =’ to ‘1’. Configure the Information (DB) block to look up the attributes ‘Product’ and ‘Step’, then clone the table onto the model worksheet.

Industry

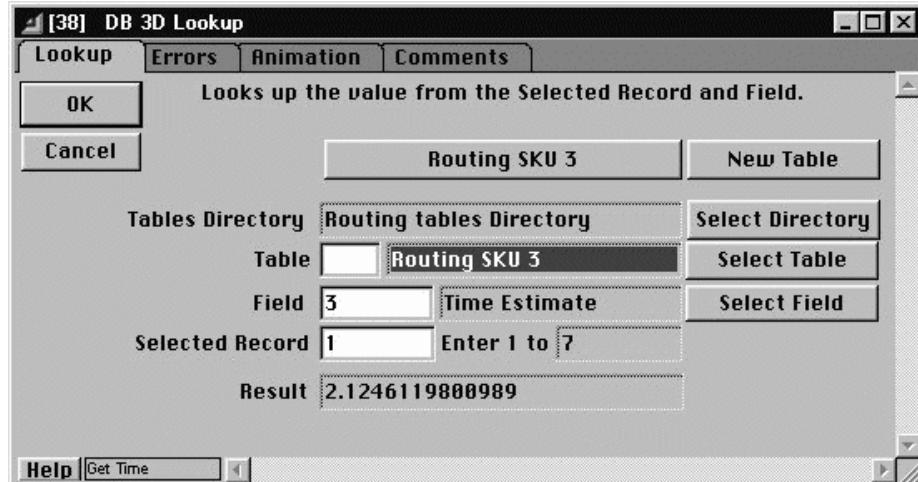
Using the 3D Lookup Blocks

Configure the 3D Lookup block labeled ‘Operation’ as follows:



3D Lookup block dialog

Next, configure the 3D Lookup blocks labeled ‘Get Time’ as follows:



3D Lookup block dialog

The 3D Lookup block first selects a table from a table directory. After the table is located the look up field is found within that table. The record is then set by the value received in the ‘R’ input connector. The 3D Lookup block labeled ‘Operation’ is set to the field ‘Operation’ in the ‘Routing SKU X’ table. This field is a child field of ‘Operation’ in the ‘Operations’ table. It will return the value of the current operation. Similarly, the 3D Lookup block labeled ‘Time’ returns the ‘time estimate’ which is the field referenced by the block. For more details on parent-child relationships refer to the on-line Help topic “Indexed Fields”.

The model is now complete. Run it to check that it works properly.

Industry

Flow Architecture

Materials flowing through manufacturing models are often represented by:

- Items, as with trains, buses, or automobiles
- Rates of flow, as with chemical processing
- Combinations of the two as with a product that is handled as item batches at one stage and then as continuous material flow in another

SDI Industry provides the capability to model **ALL** of these systems of material flow.

Extend's Discrete Event architecture represents material as discrete units, objects or items, while *SDI Industry* Flow blocks represent material as rates of flow. In *SDI Industry*, you can flexibly model systems of material flow using combinations of Discrete Event Item and Flow blocks.

This manual discusses the Flow system tool that is provided with *SDI Industry*. All Flow blocks are located in the library **Flow.LIX**. To get a firm grasp of Flow modeling, one needs background knowledge about Extend's simulation architectures, especially how time and items are handled in a Discrete Event model and a Flow model.

Flow modeling employs a simulation architecture that is very different from a classic discrete event architecture. However, Flow modeling is fully compatible with Extend's discrete event blocks. In fact, the Flow library provides conversion blocks that allow an easy transition from Item to Flow and Flow to Item architectures. One can imagine that a Flow to Item block might simulate some kind of "palletizer" at the end of a packaging line. Flow architecture handles both line blocking and line starving dynamics, which is somewhat similar in concept to how discrete event blocks work. For every category of discrete event block, there is an analogous Flow block. By understanding both systems, one can grasp where each fits best, resulting in very flexible models.

Why Handle Materials Using Flow Blocks?

When people want a logistics-oriented model of an operation where material movement is continuous, or where the movement can easily be thought of as continuous, Flow provides a perfect fit where classic discrete event item or continuous architectures are a mismatch for the problem. In cases where either of the latter systems are unnatural, cumbersome, or result in slow running models, Flow often enables a natural modeling approach, and faster running models.

At its most basic level, a simulation program architecture must answer two questions. The first is "how is time dealt with?" The second is "how does material move in the model?" Different simulation architectures answer these questions in different ways.

Logistics-oriented simulation tools address these questions with a classic system called discrete event. By working with Extend, and particularly the blocks from the Discrete Event, Manufacturing (MFG), or BPR libraries, you can get a good feel for what discrete event is all about. There is an Executive block that jumps the clock to the next time something happens; i.e., time steps are not predetermined. Material movement is handled by items, containing attributes and a priority, which essentially represent objects. Extend's discrete event system is true to what is a fairly well defined simulation paradigm.

A not so well defined paradigm in simulation is termed continuous. Typically, though not always, time intervals are predetermined. Material flow is represented by rates. Flow constituents (e.g., particulate size, and particle size) and properties (e.g., specific heat, temperature, and pressure) are often modeled. To build models, one must often deal with differential equations. Continuous simulation packages are often used for engineering process simulations, or what is often referred to as "first principles" models.

The classic discrete event simulation paradigm is appropriate for building models that focus on the "big picture," logistics-oriented problems or issues of operations, such as long-term capacity, scheduling, changeovers, reliability, and labor. However, discrete event packages are applied with great difficulty in process or consumer products industries, because the continuous nature of the

problem does not fit well with the item approach for handling material flow. Attempts to make it fit are unnatural, or they yield slow running models.

The continuous simulation paradigm is appropriate for building models that focus on the more narrow scope or time focus of the engineering-oriented problems or issues of equipment. Such problems deal with the dynamics of temperature, pressures, or constituents, and with unit operations that convert those constituents and properties. The continuous simulation paradigm is not a good fit for logistics-oriented models, because it is too focused on unnecessary technical detail, and it yields slow running models.

Flow allows you to build a model or a piece of a model that is best represented as a network of continuous material flow. Material is assumed to be homogenous, but simple rate conversion blocks can simulate aspects like moisture addition or subtraction, and conversions like pounds to cartons, and cartons to cases.

What is meant by Flow?

Flow implies the continuous movement (flow) of a material rather than individual objects or things. Truly continuous flow can be envisioned as the gallons of water per minute through a pipe, barrels of oil per day through a pipeline, or pounds of a fine powder by a conveyor system. Less continuous flow might be the extruding of dog food pellets or bones at the rate of a specific number of bones or pellets per hour. Even less continuous in nature is the high-speed packaging of small objects such as cereal boxes. To understand how Flow is important, we must consider the question, “What is the alternative to Flow?” An alternative method for representing material is to use discrete event items or things.

When Flow is used to model something that is less continuous, such as the high-speed packaging of cereal boxes, we do so because that flow can be thought of in the same terms as water moving through a pipe. Boxes per carton per hour are similar in context to gallons of water per hour. And, for the extruding of dog food, we sense that it's probably not important to model the movement of each and every pellet or bone as a thing or item. When we discretely model the flow of consumer products in a high-speed packaging line, we are able to discern the difference between 1 can or jar and another. However, that can or jar may be moving at such a speed through a filling line that visually, it looks like a blur. We know that flow is not continuous, but we also know that it might as well be (jars or cans per hour), in the context of this example.

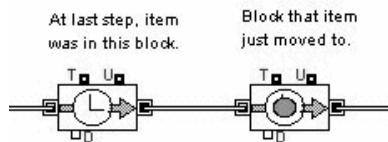
Industry

How are Items Different?

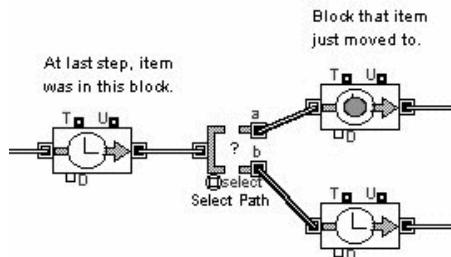
An item (or entity) is a thing that moves from place to place within a model. It may represent a product piece, a tool, a part, a person (either a customer or a worker), or even a space shuttle. In a model that uses items to represent material, there can be many items simultaneously moving inside a model, and they can represent anything the model-builder wants them to represent.

An Item cannot be in Two Blocks at Once

In an Extend discrete event model, item-handling blocks are used to set up a network to control the movement of items. Think of this network as a conduit or set of pipes that provide a path along which items move. When the model runs, items move along the path from one item block to another in the network. Often, an item will move to the next connected item block; e.g., the one that is connected just to the right of the current block.



Items usually move through many blocks from left to right before reaching its planned destination. The following example shows how an item might move from one block, through a selection block, and then on to another block.



Industry

When an item moves, it does so instantaneously when transiting “pass-through” item blocks (no simulation time is used here) and on to the next item block that is a residence. A **residence block** seems to allow an item to reside for some period of time (it uses simulation time). Items move in a network from residence to residence. A discrete event model can be conceptualized as a network of pipes, where balls roll through the pipes to reach certain holding points. When an item moves, it does so **from** one residence **to** another, and therefore cannot exist at two residences at once.

Item Movement Happens at Steps

To simulate the time that it takes to perform work in a discrete event operation, items use simulation time while in residence at Activity blocks. In a discrete event model, the duration of each time step is not predetermined; rather the activities themselves actually determine when the time step occurs. Activities (a residence type) are called event-posting blocks, because they actually tell the model Executive when to schedule events. What causes an Activity Delay block (or other event posting block) to schedule events is the arrival of items. Items cause the scheduling of steps based on how long they remain (simulated) in each residence block. Shorter activities mean more steps are scheduled, and more item movement occurs.

Though you cannot see this happening, the Activity blocks communicate behind the scenes with the Executive block, to post the clock times when the event is to occur. All event-posting blocks in

the model performs this action and the Executive block executes and sends messages at the earliest posted time as the next model step. This time is passed to each block as the current global simulation time, allowing each to react accordingly. When the Executive block has no more posted events, it advances the clock time to End Time, which is specified by the user in the Simulation Setup. This is always the last step in a simulation run.

How is this Behavior Different from Flow?

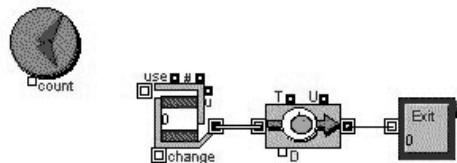
Using Flow architecture, it is possible for material to be "on the move," through a whole network of Flow blocks. For example, material can be "flowing" through several sequential pipe segments simultaneously wherein all Flow blocks "feel" the same material flow rate, and this rate may persist over a specific time duration.

This aspect of "flow" can never be truly captured by classic discrete event modeling. Once an item moves from one residence to another, it's gone from the previous residence, and the material moved in either a single chunk or in smaller chunks that together make up the total. Another way to understand this concept is to envision two buckets. If we pour the contents of one bucket into the other, both buckets are involved throughout the course of the movement. One could even determine points in time when one bucket was $\frac{3}{4}$ full and the other $\frac{1}{4}$ full, or where both are $\frac{1}{2}$ full. However, modeling such an "analog" transaction using discrete event blocks calls for compromises that result in loss of accuracy or an unacceptably slow model execution. Increased accuracy in determining amounts processed or transferred over time demands that the granularity of material-per-item be very small. This approach usually equates to excessive item generation and movement, or slow model execution. The solution to slow model execution usually takes the form of fewer items in the model resulting in decreased value granularity and ultimately in loss of accuracy. This action, however, can be modeled precisely and efficiently using Flow architecture.

Comparing Items and Flow Using a Bucket Problem

Suppose we want to simulate a bucket that starts with 100 pounds of water, and then empties at a rate of 16 pounds per minute. To model this problem using discrete event items, we must decide on how much water will be represented by an item. Let's say that an item represents one pound of water. We could modify the model you have created, by doing the following:

In the dialog of the Resource block, change the Initial Number value from one to 100. Delete the second and third Activities, and connect the end of the first Activity to the Exit. Your model should now look like this:



To simulate a rate of 16 pounds per minute, we need to set up the value of "Delay (time units)" inside the Activity block. What should this value be? Since we have set the item equal to one

pound, our “aggregate size” is one pound. If each item is to represent one pound, we need to calculate how long it will take for one pound of material to empty out of the bucket. The easy answer is one minute per 16 pounds, or 1/16 minutes, or 0.0625 minutes. Enter 0.0625 into the Activity block for “Delay (time units).” If you still have debugging (*Run* menu) turned on, you can use the “Step” button to advance execution at each step. You will see that steps occur at

Step	Time
1	0.0625
2	0.125
3	0.1875
4	0.25

and so on. When you get tired of this, hit the "Resume" button to finish the model run. You might want to save this model for use in a later demonstration.

Model Speed and Accuracy Considerations

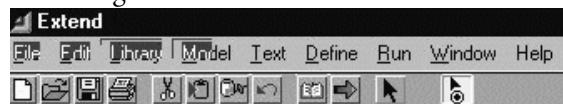
Discrete event models may execute very slowly when using the aggregating technique applied in the bucket model because each movement of an item involves a time step. These models may produce inaccurate results due to the selection of an incorrect aggregate size. If 1/2 pound of material actually moved (rather than our 1 pound aggregate size), the model would ignore or round up this 1/2 pound to 1 pound. If more accuracy is desired, we might decide to make the item represent a smaller amount of material. But by making the aggregate size smaller, it makes the model slower. For example, if we wanted to have a 1/2 pound aggregate size, it would now take 0.03125 minutes for an item to pour out of the bucket. It would now take twice as many time steps to simulate the same amount of time.

The aggregating technique is troublesome in other ways. To gain more speed, you might decide to make the items represent a larger amount of material. However, such a decision decreases processing accuracy. You might think that a happy medium could be reached, but the process is often long and tiresome to test. Each time aggregate size is changed, all of the delays in your model must also be changed. Not only that, but the starting levels and capacities must be changed also. For example, if the aggregate size is halved, you must now change the Initial Number of Items in the Resource block from 100 to 200, to truly represent 100 pounds in the bucket.

Flow Tutorial 1: Simulating the Bucket Problem Using Flow

1. Create a new model worksheet using the menu *File>New Model*.
2. Open the libraries SDI Tools.LIX and Flow.LIX using the menu *Library:Open Library*.

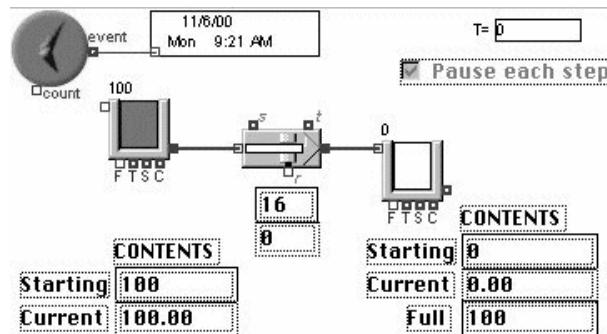
3. In this model, place the Database Manager along with the Executive block onto the worksheet using the menu *Library:SDI Tools.LIX*.
4. Select and position two “Store” blocks and one “Constraint” block using the menu *Library:Flow.LIX*.
5. Select and position a “Pause Simulation” block using the menu *Library:SDI Tools.LIX*.
6. Connect the blocks as shown below. Make sure that the Database Manager and Executive are at the far left of the model, and that the Pause Simulation block is at the far right.
7. Clone out the parameters of Starting Contents and Current Contents from both store blocks and the Maximum Rate from the Constraint Block.(Open the dialog, click the Clone Layer Tool, and pull out the parameters from the dialog and onto the worksheet).
8. Click on the main cursor tool (the pointer icon to the left of the clone icon in the picture to the right, to deselect cloning.



9. In the first Store block’s dialog, set the “Starting Contents” to 100. Also, click on the “Animation High Point” button, and choose “Starting Contents.” Note that Max Rate is set to 103.
10. In the Constraint block’s dialog, enter a value of 16 for "Maximum Rate."
11. Turn on animation using the menu *Run>Show Animation*, and run the model using the menu *Run:Run Simulation*.

To advance the simulation from step to step, click on the “Resume” button. The “Pause Simulation” block is causing the model to pause at the end of each step. If you have sound on your computer, you should hear a click or bell tone at each step.

Industry



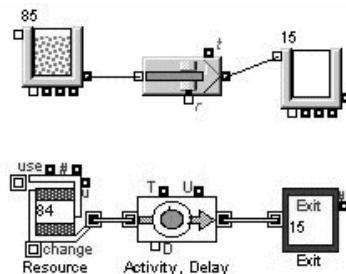
Flow Tutorial 1 Worksheet

Observations of Flow Model Behavior

When you run the model, notice the following:

- The model progresses through only three steps, with clock times of: T=0, T=6.25, and T=10.
- Material moves to the other bucket, even though it does not animate the full range of the bucket. The Pause Simulation block has a check box inside it that toggles the pausing on and off. For now, keep pausing turned on.
- When you run the model the bucket will be empty at Time=6.25. The next time of 10 is simply the End Time specified in the Simulation Setup. Now, you may be wondering is happening? Are the blocks automatically sizing an item to be worth 100 pounds and then transferring it in 6.25 minutes? The answer is no. Instead, the blocks are communicating a rate of flow between them. The first Store block posted a time event at 6.25 which was the next earliest time that an event occurred.
- Had other blocks posted any other events between 0 and 6.25, both Store blocks would calculate and display accurate contents at the intermediate time steps. This is what Flow blocks do at the Extend System Simulate Event - they update any throughput or contents. To demonstrate, try this interesting experiment. Copy and paste the Item bucket model into this same worksheet and run with the Pause Simulation turned on. As it runs, watch the Flow model update correctly for each step in the discrete event model.

This figure shows how the model will look after 15 steps:



Industry

How Flow Works

The bucket model example demonstrates precisely how Flow works. Since there are no discrete event items there is no need to deal with the size of a discrete event item. The flow model was simply configured with a specified rate by entering that rate in the Constraint block.

The Flow architecture works by communicating rate information between Flow blocks and updating block throughput and contents at each step. Although Constraint blocks are analogous to Activity blocks, they do not schedule events to occur as Activity blocks do. Rather, the Store blocks schedule events to occur when the contents are expected to be full or empty. Since Store blocks know the incoming and outgoing rates, they also know the rate of change for their contents, and so they can easily calculate the time when they will be full or empty. For our bucket problem, there is only one event that happens, and that is when the bucket becomes empty.

Flow Material wants to get Out and get Done

One might ask, "What happened in the all-Flow model between Time=6.25 and Time=10?" The answer is: **nothing**. The model simply jumped to the end time specified in the Simulation Setup because there was no more material available to process or flow. Think of the Flow architecture as a network that allows material to flow forward at a measured rate. Once there is no more material to flow, the model run is done. For example, had there been zero Starting Contents in the first bucket, there would have been less steps in the model; just a start step at Time=0 and an end step at Time=10. The flow of contents or materials schedule events. Once material or contents starts flowing it flows at the dialog rate to fill or empty buckets (create events) so that, once all of it has gone to an ending bucket, the model run is complete.

Flow Breakthroughs and Benefits

The Flow architecture simulates continuous material flow in a model environment that handles simulation time in a discrete event fashion. Time is handled just as it is in a standard discrete event paradigm, not as a finite number of equal time steps. Flow posts the times when events will occur and then jumps to the next earliest event scheduled following the completion of the previous step. However, the item aspect of the discrete event paradigm is not used to represent material. Instead, rate information is communicated between Flow blocks, and block material content and throughput is updated each simulation step as scheduled by model Flow or discrete event blocks.

The Execution Time Breakthrough

There are powerful implications to this simple but unique modeling technique. There is no need for equal, always-executed time steps, as in continuous modeling approaches. The Flow architecture enjoys the full benefit of discrete event time jumps to the next earliest scheduled event. If events caused by other blocks are posted for execution, each block's throughput and contents are updated. However, without any event scheduling by set points in our Store blocks or events caused by other blocks, the only other event is when the first bucket becomes empty.

Industry

By setting the master simulation clock to the time at which the next event occurs, we save significant execution time. The result is something like a continuous model that runs at the optimized speed of a discrete model, but without the need to deal with discrete event items.

The major activities that post an event are:

- A Store block hitting Empty, Full, or having reached a dialog entered set point.
- A Rate Change simulating a breakdown or other scheduled event.

Depending upon the kind of system being modeled, the Flow approach can offer execution speeds faster, by several orders of magnitude, than discrete event item approaches.

The Accuracy Breakthrough

By scheduling events when Store blocks become empty or when contents reach the level of dialog entered set points, the model simulates the flow of material very precisely. When events occur at posted master clock times, material throughput, and incoming or remaining block contents are recalculated based on the time expired since the last event and rate of material flow that was entered. The result is extremely high accuracy results; there are no accuracy problems similar to those associated with the use of discrete event items to represent a continuous flow of material.

The Naturalness Breakthrough

Consider the problem of simulating a continuous process model such as the mixing of concrete or jello. The discrete event approach would be to create a set of items with attributes that represent a quantity of water, a quantity of other materials, and a time delay that represents the process step of mixing. However, a more natural and intuitive method is to consider the process of adding water and other materials as a rate of flow.

The Compatibility Breakthrough

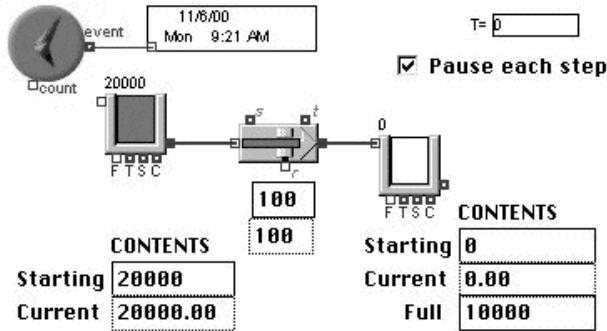
The discrete event time clock that Flow uses is compatible with the Executive block that item blocks use. Therefore, Item blocks and Flow blocks can be used in the same model. Also, the **Flow.LIX** has conversion blocks for going from Item to Flow, and vice versa. This is a major benefit, which allows you to represent parts of a process using items, and other parts using Flow.

The following tutorials help to illustrate many of the concepts we have been discussing. This simple model expands on the bucket problem by adding some Constraints and some downtime processes.

Flow Tutorial 2a: Simple Buckets with Blocking

1. Start with the Flow Tutorial 1 model.
2. Set the first Store block's Starting Contents to 20,000 (Do this from the cloned out boxes). Set its Animation High to Full Set point.
3. Set the second Store block's Starting Contents to 0, and its Full Set Point to 10,000. Set its Animation High to Full Set point. Clone out the Full Set point of 10,000.
4. Set the Constraint blocks Maximum Rate to 100. Clone out the Actual Throughput Rate.
5. Set the ending time of the model to 150 minutes (Use the menu *Run:Simulation Setup*).
6. Run the model (Use the menu *Run:Run Simulation*).

The worksheet should look like this:



Click through the steps (Click on the Resume Simulation button at the bottom of the screen, until the buttons go away).

Explanation of Flow Tutorial 2a

When you run the model, here is what happens:

1. As the model pauses at Time=0, you notice that the Actual Rate is 100, the Constraint block is satisfied with a color of medium blue, and the model is ready to go. Click the Resume button to advance to the next step.
2. The model pauses at Time=100. At this event, the second Store block's Current Contents is full and reads 10000, the Constraint block's Actual Rate is zero and its animation state is black to show that it is completely blocked. Click Resume.
3. The model pauses at Time=150. This is the last event in the model because it is the end time entered in the simulation setup. The state is the same as before.

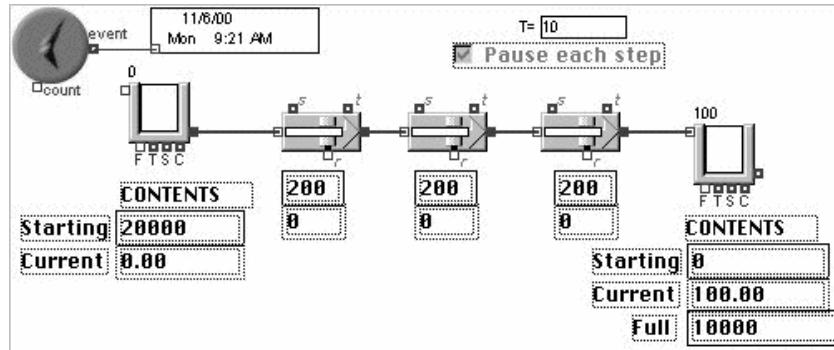
Industry

Flow Tutorial 2b: Additional Constraints

In this next step, we will add additional Constraints. Do the following:

1. Delete the connection lines between the Store blocks and the Constraint block (Click on the segments of the lines, and use the menu *Edit:Clear*, or the Delete key).
2. Duplicate the Constraint block twice, to add two new Constraints.
3. Position the new Constraints to the left and right of the original Constraint.
4. Set up clones below both of these Constraints in the same manner as before.
5. Set the Maximum Rate of these 2 Constraints to 200.

The model should now look like this:



Run the model.

Explanation of Flow Tutorial 2b

This time when you run the model, you will see that at Time=0, the Actual Rate for all of the Constraint blocks is 100. The first Constraint block's animation state shows that it is partially blocked, and the second Constraint block's animation state shows that it is partially starved. The middle Constraint block is satisfied.

- On the second step at Time=100, all Constraint blocks become blocked as before.
- On the third step at Time=150, the model is done.

Flow Tutorial 2c: Adding a Downtime Process

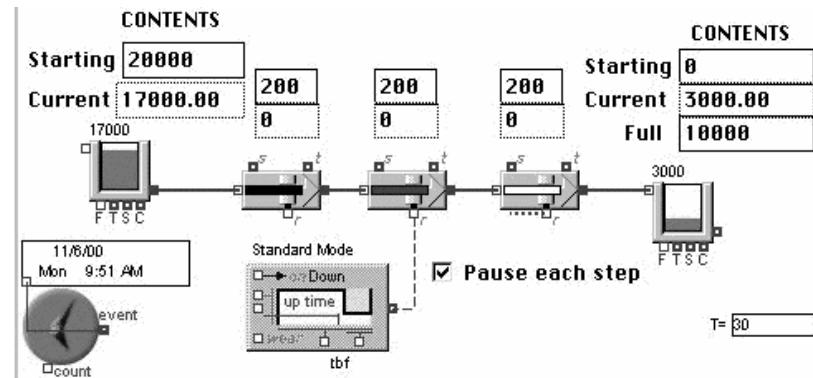
In this next step, we add a downtime process. Do the following.

1. Bring in an Equipment Controller (Use the menu *Library:SDI Tools.LIX*).
2. Connect the output of the Equipment Controller to the "r" input of the middle Constraint. For neatness, you might want to move all Constraint block clones above their icons.
3. In the Equipment Controller dialog, set the TBF to 30, the TTR to 1, and the Up = Rate parameter to 100. Uncheck the checkbox labeled TBF = Fail to Fail. Run the model.

Explanation of Flow Tutorial 2c

- At Time=0, the first Constraint block is partially blocked, and the third is partially starved.
- At Time=30, the first Constraint is totally blocked, the second is down, and the third is completely starved.
- At Time=31, the second Constraint is up again, and the situation is the same as at Time=0.
- At Time=61, the second Constraint is down again.
- At Time=62, the second Constraint is back up, and so on.

The picture below depicts the state of the model at Time=30:



Flow Starving & Blocking Behavior

Simulation software that models logistics supply and demand problems must deal with process blocking and starving behavior. Starving is a condition where there is a lack of block incoming material. Blocking is a physical or policy constraint, which says that material cannot be accepted or passed because the downstream operation cannot accept more. A model should not be able to start running an operation or process when there is no material to process (starving behavior). On the other end, a model or model section must stop sending material downstream (or stop) when the downstream operation cannot accept additional material (blocking behavior).

Starving & Blocking for Items- An All or Nothing Proposition

The item architecture built into the Extend + Manufacturing libraries **Discrete Event.LIX** and **MFG.LIX** handles these issues in a unique and powerful way. If an item is not available from upstream to pull, you cannot start an Activity or Machine delay. Similarly, if a block's item is done with its delay, it cannot proceed downstream to a one-at-a-time delay until the downstream operation becomes idle and empty. This automatic handling of blocking is an interesting aspect of the Item architecture. Other software packages handle things differently. For some, you have to be more explicit about how this blocking takes place. For example, to model a one-at-a-time delay, the activity must explicitly request from a resource pool that is defined to have one resource.

Industry

The key thing to understand about how the Item architecture handles starving and blocking is that the behavior is by nature an **all-or-nothing** proposition. An operation is either completely starved for material or completely blocked. There is no in-between.

Starving & Blocking for Flow- A Matter of Degree

The Flow architecture handles these issues in a way that is very analogous to the way the items work. However, the difference is that starving and blocking can be complete or partial. Imagine a Filling machine on a packaging line that has a maximum rate of 100 cans per minute. You can

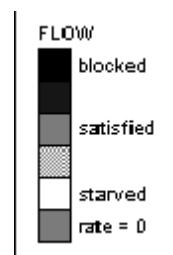
model this machine using a Constraint block that has a Maximum Rate of 100. If a condition arises in the model in which cans are supplied to the Filler at 80 cans per minute by the upstream operations, the Filler will run at 80 cans per minute (barring some other kind of control arrangement). Simulating the line in this manner is automatic. The Constraint will run at an “Actual Rate” of 80, and be in a partially starving state. Of course, if you want to, you can override this “natural” behavior of Flow by way of other controllers.

Flow handles blocking in a similar manner. Say there is a Labeling operation immediately downstream from the Filler, and there is no buffer between the Filler and the Labeler (if there were a buffer, you would use a Store block.). If a condition arises in the model in which the Labeler shuts down, the Filler’s Actual Rate would become completely blocked, and have an Actual Rate of 0. If the Labeler’s rate drops to 70, then the Filler would become partially blocked, and have an Actual Rate of 70, as long as the possible flow rate upstream from the Filler were greater than or equal to 70.

This behavior is exactly analogous to the way the Extend item blocks handle blocking automatically. It is troublesome to some modelers who want to explicitly model every control operation. However, for most modelers this is a timesaving feature, because this behavior is fairly universal in systems like packaging lines. Of course, if you want to, you can override this “natural” behavior of Flow by using other controllers. For example, it would be a simple matter to wire things so that the blocked material goes to the floor, as is the case in some operations.

Flow blocks have a very helpful visual feature to indicate starving and blocking behavior. In animation mode, they display colors to indicate their state. The legend to the right explains these various colors:

If you are seeing this printed in black & white, the colors are black, 3 shades of blue (dark blue, medium blue, light blue), white, and red. The middle blue shade means that the block is satisfied, in that the Actual Rate corresponds to the Maximum Rate exactly. We refer to the ‘satisfied’ state as ‘happy blue’. The color Red indicates that the Maximum Rate has been set to 0.



Starving & Blocking for Store Blocks

Store blocks handle starving and blocking in a unique way, but again the behavior is very analogous to the way that the Extend item architecture works. A Store block must always define a level for Empty and Full. Empty is almost always zero. Full can be infinity if you want. You are able to define a maximum rate out for a Store block. Most of the time this is infinity, because the downstream operations control the rate of pull.

If you configure the Store block to have a Maximum Rate Out of infinity, and a condition arises in the model where the level is greater than zero, then the next downstream operation will be satisfied. However, if the level goes to zero, then the rate possible for the next downstream operation becomes equal to the rate coming into the Store block. At this point, the downstream operation

will likely become partially starved for material. We term this condition **on-stream at empty**, because material is simply streaming through the Store block at an equal rate in and out.

On the other side of the equation, the operation just upstream from a Store block can never be blocked as long as the Store block is not at its Full level. If you have configured the Full level to be finite, and a condition arises in the model where the Full level has been reached, the allowable rate into the Store block becomes equal to the rate at which material is leaving. At this point, it is likely that the upstream operation will become partially blocked. We term this condition **on-stream at full**, because material is simply streaming through the Store block at an equal rate in and out.

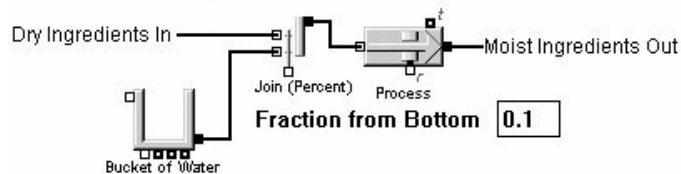
Handling Material Additions and Losses

A common need in operations modeling is to handle material losses and additions. In some operations, material may swell due to water or ingredient additions. In other operations, material may incur shrinkage due to moisture loss, as in a dryer or roaster. Material loss may also occur due to periods where off-grade product is made, or due to the fact that off-grade is made continuously at a certain percent.

All of these needs are handled by Join or Split blocks of some kind. For example, a process can pick up water or moisture in a continuous fashion, such that coming out of the process, we have 10% water and 90% original ingredients. We handle this by using a bucket of water and a Join block; e.g.,

Example: Handling Moisture Addition

Use the Join (Percent) block



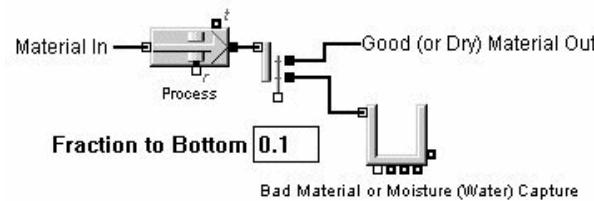
Industry

In this case we would probably configure the bucket to have an infinite amount of Starting Contents. If we run out of water in this configuration, the process will starve completely for material, even if there is plenty of dry ingredients available. Note that this arrangement significantly changes the blocking and starving behavior of the model. In the example, if the Process Maximum Rate is 10, a process upstream from the one shown that wants to send material at a rate greater than 9 will be partially blocked. For more information about the Join (Percent) block, refer to the on-line help by pressing F1 when the block's dialog window is selected.

Continuous material losses (e.g., continuous off-grade) are handled in a similar fashion by using a Split (Percent) block. In this case the bucket should probably be configured to have zero Starting Contents, and an infinite Full set point; e.g.

Example: Handling Moisture Loss or Continuous Off-Grade

Use the Split (Percent) block



In this case, we would set the Fraction to Bottom with a value between zero and one. Here again it is important to have an infinite Full set point. If the contents reach full in this example, the process would become completely blocked due to the nature of the Split (Percent) block.

To handle spurts of whole material losses, you would use this same block, and direct whole streams to the losses bucket. To do this, you would use the Value input connector to override the Fraction to Bottom. You would make it one during periods where material is to go completely to waste, and zero during periods where all good material is being produced.

Handling Unit Conversion

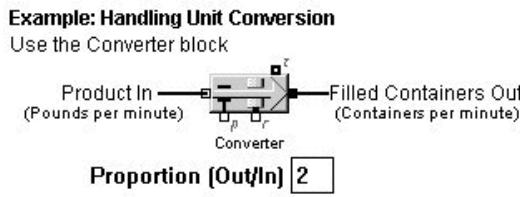
In all kinds of systems, unit conversion is a typical need. Particularly in packaging systems, we see the most common examples. A packaging line Filler typically takes in pounds of materials, and then produces packaged units of cans, tubes, jars, etc. Somewhere downstream from this, those units may be put into cases. Downstream from this, those cases may be put into palletized units.

Industry

Flow Rate Conversion

Different operations in a packaging line usually require the conversion of input material flow to a modified state of output flow. For example in a filler operation, processed fruit jam may flow into a filler machine (constraint) at the rate of 12 ounces per second, and the filler machine may output filled jars at the rate of one 12 ounce jar per second. Note that we are not thinking about the container material that is also coming into the filler machine. For a true filler operation, we would employ both rate conversion and the Join (Minimum) block to combine both jam and jars as incoming materials.

Using Flow architecture, flow rates can be converted, so that in a given model segment, rates and volumes are whatever they need to be. The Converter block in **Flow.LIX** handles this job. For example, the following model segment would handle the conversion for a filler machine that produces two filled units for every incoming pound of material.



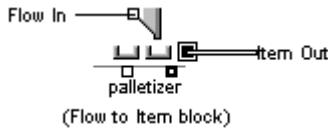
If the operation downstream from this conversion wants to input at a rate of 10, then an operation upstream from the conversion will be partially blocked if it can produce at a rate above five. Conversely, if the upstream operation is producing at a rate of five, downstream operations will be starved if they want to produce at a rate greater than 10.

Flow to Item Conversion

The discussion of converting flow rates in a packaging line model leads to some provocative questions. At first glance, the application of Flow architecture to model a packaging line might appear questionable because the model represents the production of discrete items such as filled jam jars or containers of jam jars which is not a continuous operation. Or is it? In reality, this is a modeling area where Flow techniques are appropriate because the movement of jam in and filled jars or cartons out is just like the flow of water. When a river flows into a freezing zone, chunks of ice flow out. Using rate conversion techniques and blocks, Flow modeling becomes the perfect technique for modeling high speed packaging lines. These Flow models execute very fast compared to similar purpose models that use discrete event items design.

However, to get to the provocative question, continue following the product from the packaging line forward. After material is converted by weight to filled containers, and then from containers to cases, and then from cases to pallets, it is probably put on a truck. The trucks might get loaded onto a ship, and at some point, we should get concerned about making a half of a unit. Modeling the movement of ships is difficult because it is impossible to move half of a ship. The realization is that at some point, Flow techniques become unrealistic because the material movement is not at all continuous, and the nature of this reality is that discrete event modeling is required. When this happens is dependent on the nature of the process being modeled. But it is also dependent upon the nature of the problem that the model is addressing. There is no strict rule for when a model changes from DE to Flow and back.

When the material that is moving aggregates into larger and larger units, it changes character from continuous material flow into the movement of discrete items. Luckily, the need to convert continuously flowing material into discrete event items is not difficult to model. Because the Flow architecture works perfectly with the discrete event clock, DE item blocks can be used in the same model as Flow blocks. When it is necessary to convert from Flow to Items or Items to Flow, use blocks from the **FLOW.LIX** such as the Flow to Item, Item Empty, Item Fill, or the Store (Item to Flow) blocks.



The Flow to Item block allows the specification of the amount of material required into the block in order for it to be released as a discrete event item. This is quite similar to an item-batching block. For Item to Flow, use a Store (Item to Flow) or the Item Empty block. Refer to the on-line help for information on these blocks.

Knowing What the Material Is

In most operations logistics processes or problems, schedules play a critical role. It is uncommon to find a situation where the problem can be solved by modeling one long run of the same material. Material-related characteristics are such that it is possible for bottlenecks to actually move around dynamically in a plant as different materials are produced. The “moving bottleneck” problem is where simulation modeling becomes the power diagnostic tool of choice.

If a model is designed to handle a single type of material, there is no problem of knowing the type of material. Rates, conversion factors, and other parameters can be directly entered into blocks that model operations. However, if a model is designed to model the production of different kinds of material, then knowing what we are making becomes critical. It's important to know how fast we can make it, to know the parameters for converting production units to packages or containers, and containers to cases.

A major difference between discrete event items in Extend and Flow architecture is that items can store attributes carrying information about themselves, whereas Flow cannot attach attributes.

When that item reaches a routing point or branch in the road, item attributes can be interrogated to determine which path it should take. When an item is done with production, it can be interrogated to determine what SKU it is. On the other hand, Flow does not store information about the material, and it cannot be interrogated for any information.

Industry

Categories of Flow Blocks

There are categories of Flow blocks to handle most common plumbing chores. We have talked a lot about the Constraint block, which models an instantaneous flow constraint. It does not handle resident material, as in a dryer or roaster. We deal with this kind of operation with other approaches and other blocks. We have not discussed the use of Set Points for Store blocks, and we have not covered the Split and Join blocks in detail. Further documentation on each category of Flow block is covered in the online Help.

Rules for Using Flow Blocks

The library **Flow.LIX** provides not just a set of blocks, but a new simulation architecture. The Flow blocks follow a blueprint for dealing with time, material movement, and data handling. This section explains some important assumptions and rules for the use of Flow blocks.

The Flow architecture makes the following assumptions:

1. Flow rate changes in stepped fashion.

Typically, to represent a machine with downtime, we use a Constraint or Conveyor type of block, and we control its rate to change instantaneously from zero to its full speed. If you wanted to, you could represent a ramp-up phase for turning a machine on, but you would use a block that would schedule an event at each step change.

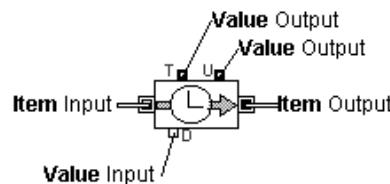
2. Flow material does not store attributes.

Unlike items, Flow does not store attributes about itself, such as constituent information or other properties. Tracking of Flow material and production is handled in *SDI Industry* without the need for attributes. SDI Plant Builder allows detailed handling of schedules comprised of multiple products.

To build a Flow model, start by placing the Database Manager and the Executive Block at the far-left side of the model worksheet, then connect them together. The Database Manager is in the library **SDI Tools.LIX**. In Flow models, the Database Manager works with the Executive block and is responsible for handling the database.

Flow Connectors and Information Connectors

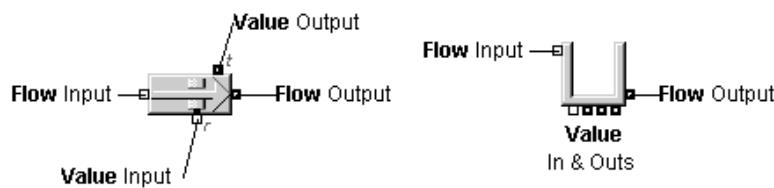
Model blocks use connectors to link to and to communicate with other blocks. Different type connectors are for different purposes, with the major connectors' categories being material flow and information flow. Material connectors convey material flow, and Value connectors share information.



Industry

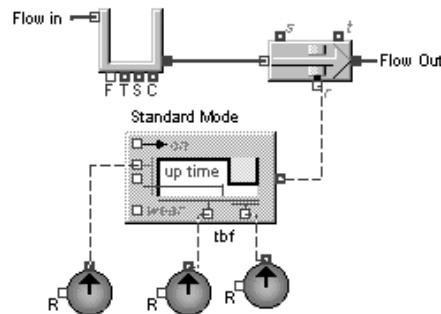
For the discrete event blocks in the libraries of **Discrete Event.LIX**, **MFH.LIX**, and **BPR.LIX**, there is a clear, visual distinction between Material (Item) and Value Connectors, with the item connectors having double walled outlines, while value connectors have single walls.

For Flow blocks there is no visual distinction between Material (Flow) connectors and value connectors.



The guidelines to be used to determine whether a connector is of the Flow or Value type is that the Flow connectors will be on the left and right sides of the block icons, whereas the value connectors will be on the top or bottom of the icons.

At the primitive block level of *SDI Industry*, often the connection line features of Extend are used to change the way connection lines look. This allows a visual identification of the different flow connections. To change the connection line look, use the menu *Model:Connection Lines*. The following is an example.

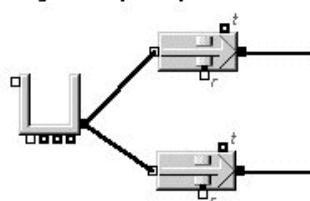


Note that the solid lines are used to depict material flow, while the thin dashed lines are used to depict information. In the example, Flow moves through the Store and then the Constraint block, in a left to right direction. The “t” output reports throughput, which can be plotted. The “r” input is used to override the “Maximum Rate” value in the dialog of the Constraint. Essentially, the rectangular green Equipment Controller is for providing values equal to the proper rate or zero, depending on whether the equipment is up or down, respectively.

Flow cannot split or join without a Split or Join Block

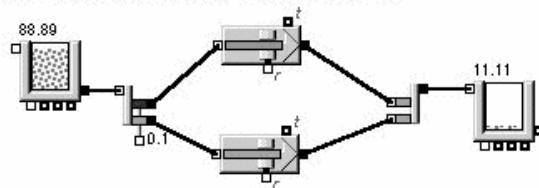
You cannot split or join flow without a Split or Join Block of some kind. For example, this model will give an error message when you try to run it.

Wrong: Attempt to split the Flow.



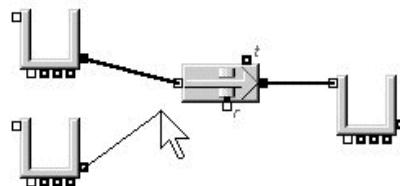
To do this correctly you must use a Split or Join block of some kind.

Right: Splitting Flow using Split (Percent)

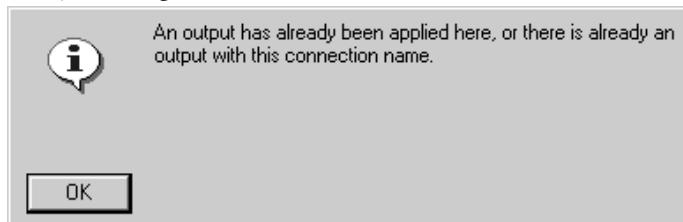


Likewise, you cannot join flow paths without some kind of Join block. The attempt might be to try to connect two Store blocks into one Constraint block.

Attempt to connect 2 lines to the same input connector.



If you try to do this, you will get an immediate error:



Industry

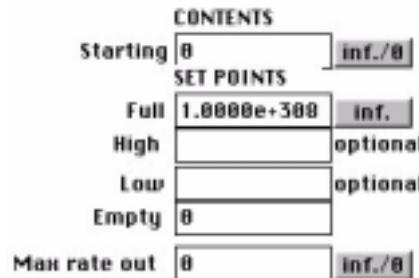
This is a basic rule for all Extend blocks, not just Flow blocks. With Flow blocks you must use some kind of Join block in order to make this work properly.

Start with a Store and end with a Store

When you build networks to convey material represented by Flow, you should always start with a Store block, and end with a Store block (or a Flow to Item block). The Store block can represent starting material, ending product, or an in-process surge or buffer. You start with a Store block by having a huge amount of material available. All you have to do is bring in a Store block, and set the Starting Contents to infinite. This is what the “nf./0” button is for. Press it to set the contents to infinite. This is the set up for a Starting Store Block:

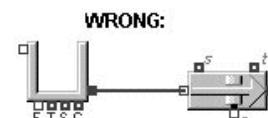
CONTENTS
Starting 1.0000e+308 inf./0 Max rate out 1.0000e+10 inf./0

You can end a network of Flow with a Store block. In this case, simply set the Starting Contents to zero. Also be sure to set the Full level to infinite. Below is the setup for an Ending Store Block.

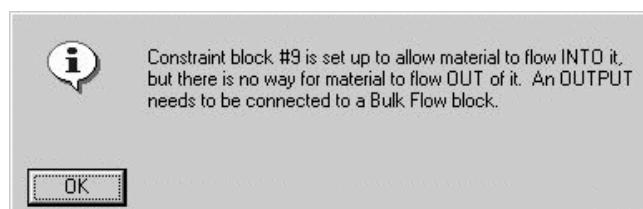


Do Not Leave Flow Blocks Hanging

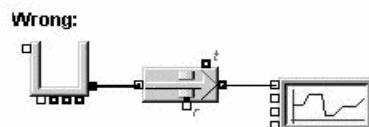
The corollary to the previous rule is that you cannot leave a Flow Block hanging. For example, in our bucket example we followed all of the rules. We started with a Store and ended with one. This is okay. On the other hand, it does not work to simply start with a Store, have a Constraint and not connect it.



If you attempt to run this model, you will get this error:

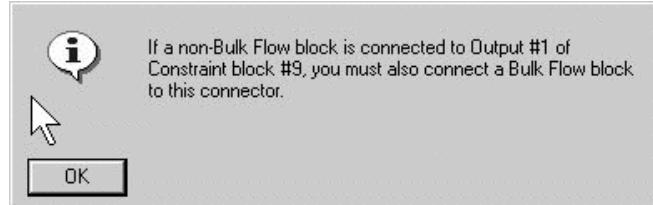


Note that you will get a similar error if you hang a Plotter block off the end of a Constraint:



This is because the Plotter is not a Flow block. This violates the first rule of connectors because the Constraint output is a Material (Flow) output, and the Plotter input is a Value input.

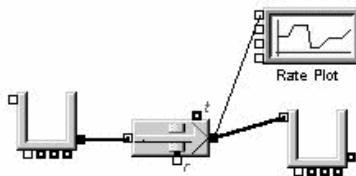
If you try to run it, you will get the following error:



Flow Connectors also Report Rate Information

While it is not permitted to split the flow out of a flow connector, you can tie one Flow block and multiple informational blocks from the end of a Flow output. The information reported is always the rate of flow.

For example, if the Constraint is properly wired to allow flow out, it is now OK to make one or more Value connections.



Note that the rate reported at the output is the instantaneous actual flow rate. This may be different than the Maximum Rate, which is a dialog input.

Industry

Rate Override does not query for Values

The Constraint block has a Value input connector labeled "r." You can connect blocks to this input which supplies a value treated by the Constraint block as the maximum rate. During the model run, the dialog parameter labeled "Maximum Rate" will update to display the values piped in through this input. You can view this action by either opening the dialog during the run, or by cloning the dialog parameter to the main worksheet for easier viewing.

The relationship between the "r" input and the "Maximum Rate" dialog parameter is termed **overriding**. Overriding is a powerful way to model dynamic relationships in models. You can use this to set up control mechanisms that are based on schedules or even some kind of equation whose inputs are the status of other blocks in the model. You control the rate by either scheduling values to be generated, or by having some calculation occur due to a model status change. In either of these cases, we are talking about a proactive calculation, which results in messages being fired into the rate input.

As you learn more about Extend, you will become comfortable with the concept that a block's value input connector can be either *passive* or *querying*. A *querying* input connector messages the

connected block's output to get a new calculated value, whereas a *passive* input connector does not. Using this lexicon, the "r" input on the Constraint block is passive; i.e., the Constraint block does not query blocks connected to "r" except at Time=0. While there are detailed technical explanations for how a block's input connectors might behave; you need a general rule to help you identify how a block most likely treats its neighbors. The following is such a rule:

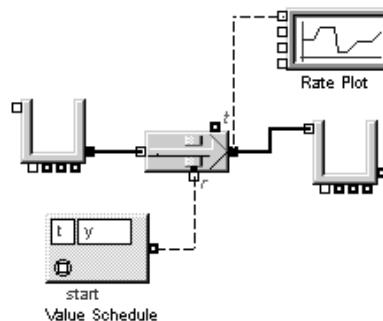
If there is a natural reason and point in time in which a block should ask for a new value from a block connected to its input connector, then it probably will cause the block supplying the input to calculate at those points in time. Otherwise it probably will not.

Since "stuff" is always flowing for Flow blocks, there are no "edges" to the material (as for items), and therefore, there is no natural point in time to ask for a new rate, for example.

If a block "asks for a value," a message will be sent through the input connector proactively causing the connected block to calculate at the time the natural event occurs. The important corollary is that if there is no such natural event, then the block will not cause calculations to occur. Thus, the next rule:

If a block does not ask other blocks to calculate an input value, then new values will enter that input only due to events caused by other blocks.

This shows a way we could configure the Constraint block's rate to change through time:

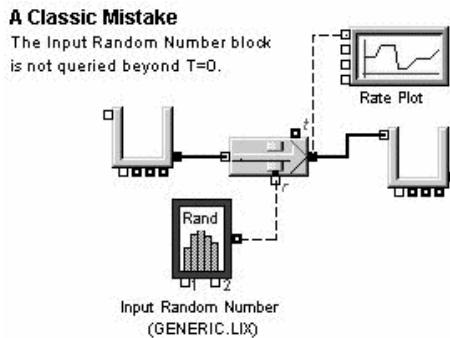


The "Value Schedule" block is from the library **SDI Tools.LIX**. To set up the rate schedule, enter rates at various times in the block dialog. The following sets up a simple rate schedule:

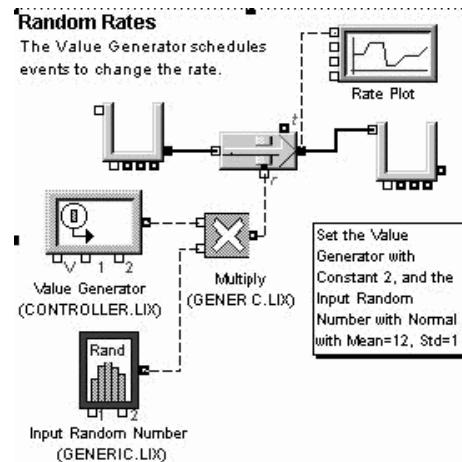
	Output Time	Value
0	0	20
1	3	10
2	7	0

This sets the starting rate to 20, and changes the rate to 10 at Time=3. It is interesting to run the model and watch the actual throughput rate change on the Plotter. If you start with 100 pounds in the starting Store block, why does the actual throughput rate go to 0 at Time=7? It could be

because the buffer becomes empty at that point! The reason this works is because the Value Schedule block causes its own events to occur at the times entered in the table.



This is a classic mistake. The intent is to model random rates by using the Input Random Number block from the **Generic.LIX**. All blocks in this library are passive; they do not schedule events. You may recognize that there is a logical problem in this. If we are to have random rates, we need to know *when* the rate is to change! The proper way to model random rates is to use a Value Generator, from **SDI Tools.LIX**. This is an example:



Industry

In this example, the Value Generator schedules an event every two minutes. The value generated is always a one. It is important to have the Value Generator schedule the events that cause the one to be fired. This works because the Value Generator schedules events, and those events cause connector messages to chain backwards, causing the Input Random Number to calculate, and forward, causing the Multiply and the Constraint to calculate.

In each of the examples so far, we have illustrated the use of blocks from the library **SDI Tools.LIX** that proactively post events. Such events can also be fired by way of status updates from other sections of the model.

This is an example of how to control a rate based on status. In this case, the status has to do with a block right next door. We use the Equation block from the library **GENERIC.LIX**. Here again, the Equation block is passive; it does not schedule events. However, our Store block does schedule events for when it hits empty, full, or any other set point that you think is important. If you enter a High set point of 50, an event will be scheduled when this level is hit. Associated with this event will be an update to the status of the level output. For details on this, refer to the Store block help. You can see the basics of what is going on from the example- when the Store block's level is above 50, set the downstream rate to 20. When the level is below 50, set it to 10. Run the model to see how this works. Put in a Pause Simulation block to click through the events.

Flow Block Optimization Techniques

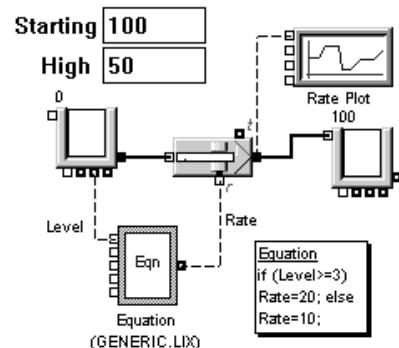
The Flow block architecture employs several innovative optimization technologies to ensure that simulations run as fast as possible. It is helpful to understand the basics behind these techniques. The essential ideas are as follows:

First-Order Assumption and Tentative Event Posting

Flow blocks which post events do so tentatively. For example, Store blocks post events based on the relative rate that their level is changing. This is based on a “first-order” assumption, which is to say that rates are expected to change in stepped or square fashion. Therefore, they are able to predict the clock time at which the level will hit certain set points, or empty. When incoming or outgoing flow changes, the old event for the Store block is cancelled, and a new one is posted. In older simulation architectures, this cancellation issue was quite complex for a number of reasons, whereas in Extend’s block-related posting system, event cancellation is simple. The goal is to enable the modeling of continuous material flow systems using discrete event time advance, which allows simulations to run extremely fast. Discrete event time enables jumping over wide spans in time without the need for intermediate time steps, which are unnecessary because “nothing is changing.” In the past, this technique was closely tied to the concept of material flow being items (alternatively termed “entities” or “orders”). With Flow, we have revised the classic “nothing is happening” argument by saying that “nothing is really changing, except that material is flowing (!)”. There can be wide spans of time in which nothing is happening, except that material is flowing at given rate(s) through the system. The important events are when rate changes occur. This drastically reduces the number of events that are dealt with in a model to those that are associated with control actions, breakdowns, schedule changes, changeovers, etc.

Controlled Rate

The Store block schedules an event at the High set point. The Equation controls the rate based on the level.



Propagation of Potential Rates Using a Block Connector Message System

Flow blocks leverage Extend's unique block connector message system to ensure that changes in the state of the blocks in a Flow network automatically propagate to other blocks on a "need to know" basis. This makes it possible to build arbitrary networks of Flow, with complex controls, and to account for material flow and to ensure mass balance. This also makes it possible for any Flow block in the network to know whether downstream or upstream conditions are responsible for actual flow rate being less than maximum potential flow rate. In the classical continuous or combined discrete event continuous approach, there was a need for artificial time steps designed to ensure that the model "detects" change. By leveraging Extend's block message architecture, this need is completely bypassed.

Deferred Throughput and Contents Calculation

Due to these optimizations, it is not necessary for Flow blocks to update their throughput or contents during 'SIMULATE' steps that do not affect the state of their control (e.g., Flow rate). Flow blocks that display cumulative throughput in their dialogs (most all of them) and ones that display resident contents (some of them) update these displays as the model runs. These displays are often found in the blocks' "Results" tab. As a default, Flow blocks avoid intermediate updating of these displays during 'SIMULATE' steps that do not affect the state of their control via block messages. This is an optimization that helps the simulation to run as fast as possible. The disadvantage of running in this mode has to do with the desire on the part of the user to observe intermediate changes associated with each step, for example, in Store block level animation. Over sizeable time jumps associated with steps, many Flow blocks will appear "dormant" with respect to contents animation. In addition, if you have cloned one of the throughput or contents displays to the worksheet, it will appear to be lagging by staying fixed on a past value. For a throughput display, this can be disconcerting, when in truth you know that vast amounts of material are truly flowing during these spans of time! To aid in the presentation of results during a run, Flow blocks automatically begin to update contents and throughput on every step, in the following conditions and configurations:

- a) If the Flow block's dialog is opened.
- b) If the Flow block's "T" or "C" output connector is connected. (This feature goes beyond presentation concerns- it also ensures accuracy in simulating controls that rely on the output!)
- c) If the user pauses the simulation, all Flow blocks update.
- d) If the check box "Update Every Step" is checked in the individual Flow block.

Global Override

In the Database Manager dialog, in the Simulation tab, there is a checkbox to "Update Flow Blocks on Simulate." By checking this checkbox, all Flow blocks will calculate on every 'SIMULATE' step of the model, regardless of whether the step affects state of control for that Flow block. If there are many events in the model, then the user will see more intermediate calculation points,

and thus smoother animation of level changes in Store blocks. This setting is not recommended, as it will significantly slow down the simulation.

Appendix A: Menu Command Reference

A reference section containing a description of each menu item in Extend

*“I wish it, I command it. Let my will
take the place of a reason.”*
—Juvenal

This chapter describes the commands and menus that you use in Extend. As you have seen in earlier chapters, you only need to know a few commands in order to run or create models. This chapter explains all the commands that appear in the menus and the circumstances in which you might use them.

Some of the menus in Extend are *hierarchical menus*. A hierarchical menu has a sub-menu for one or more of its main menu items and is indicated by a triangle pointing to the right (►). To use a hierarchical menu, drag the cursor down the menu to the desired command. As the command is selected, a second menu appears to the right of the first one. While still holding down the mouse button, slide the cursor to the right, then down the hierarchical menu to the desired choice, and let go of the mouse button.

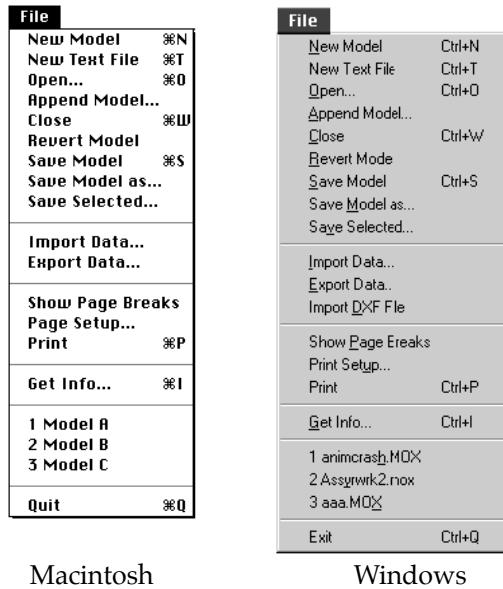
Apple menu (Macintosh only)

The Apple menu, labeled with a , holds many system-wide items available to you. The additional choice, About Extend, tells you about the version of Extend that you are using. If you open the About Extend window, click on the window to continue.

You may also use the Chooser desk accessory to change printers while using Extend. See your Macintosh documentation for more information on standard desk accessories such as the Chooser.

File menu

The File menu lets you open, save, import, export, and print model and text files. Note that library files are opened from the Library menu described later. Most of the commands in this menu act just like they do in other applications.



Macintosh

Windows

File menu

New Model

Opens an untitled model window.

New Text File

Opens an untitled text file window. You can use this to create text files for the File Input block, as input to sensitized blocks, or for any other Extend feature that uses a text file as input. See “Importing and exporting with text files” on page E167 for information about text files.

Appendix

Open

Opens an existing model or a text file. When Extend opens the model, it also opens any libraries that are used by the model. Thus, you do not need to manually open libraries if you simply want to run a model. Of course, if the library is already open, Extend does not reopen it. To open a library directly, use the Library menu.

Usually, the process whereby Extend automatically opens libraries when the model is opened works fine. However, if you have moved either the model or a library since the model was closed, Extend may not be able to find the library. You will see the message “Searching for library *xxx...*” as Extend searches for the library. As described in “Searching for libraries and blocks” on

page E142, if Extend cannot find all of the blocks used in the model, the missing blocks will be replaced with text blocks. When this occurs, Extend will open the model as “Model - x” to prevent accidentally saving over the old model.

Append Model

Adds another model to the current model. This is faster and uses less memory than using the Clipboard when you want to merge two models. The command prompts you for the model file to open. Appending models is described in “Appending models” on page E167.

Close

Closes the active window. You can also close a file by clicking on the window’s close box in the upper left corner of the window.

Revert Model

Reverts the selected model or text file to the version saved on disk, discarding any changes since the last save. Extend warns you before it completes this command.

Save and Save As

Saves the selected model or text file to disk. Choose Save to save the file under the current name or Save As to give a new file a name or to save a file under a new name.

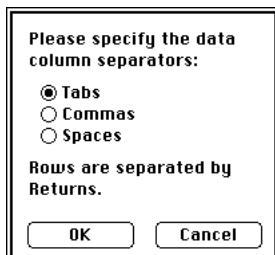
To protect against file corruption, Extend makes a copy of the previously saved version of the model and saves it with the extension *.BAK* prior to saving any new changes to the model. If a crash occurs during the save process, your original file may be corrupt. However, the backup file *filename.BAK* will contain the previously saved version of the model. If the save process is successful, you may choose to have Extend automatically delete the backup file or keep the file around (see “Preferences” on page A10).

Save Selected

Saves just the selected items into a new model. This is faster and uses less memory than using the Clipboard to copy portions of the model. The items selected (blocks and text, drawing items, etc.) depend on the selection tool used to make the selection.

Import Data

Copies data from a text file into the selected table. *A table from a dialog or plotter must be selected before this command can be used.* After choosing the file to be imported, you see:



Column separator dialog

You must specify how the columns in the text file are delimited (separated); rows are automatically separated by returns. Most text files that are exported from other applications have columns delimited by tabs. Check the format of the text file before choosing Import Data. For more information, refer to "Importing and exporting with text files" on page E167.

Export Data

Copies data from a *selected* table to a text file. It works the opposite of the Import Data command. *A table from a dialog or plotter must be selected before this command can be used.* After you give the file name, Extend puts up the column separator dialog (as shown in the previous section), so you can specify what type of separator to use in the text file. You can read the text file in Extend or in a word processing or spreadsheet application. For more information, refer to "Importing and exporting with text files" on page E167.

Import DXF File (Windows only)

Imports CAD drawings in standard DXF format from AutoCAD or other CAD programs. The drawing becomes a graphic image which can be used as a background picture in the model, the notebook, or on an icon. In AutoCAD, the file must be a version 12 DXF file.

Appendix

Show Page Breaks

Causes Extend to draw a set of page boundaries on the model and place page numbers in the upper left hand corner. These page boundaries show you where page breaks will occur if you print the worksheet. Since the size of a page is dependent upon the settings in the Page Setup (Macintosh) or Print Setup (Windows) command, it is recommended that you make your Page or Print Setup choices before showing page breaks.

When you choose Show Page Breaks, this menu item has a check mark next to it. To hide page breaks, select this command again. The Show Page Breaks command also causes page numbers to display in the upper left corner of each page on the model. This can help you decide which pages to print if you are only printing a portion of the whole model.

Notebooks and dialog windows always show page breaks. Notebooks also always show page numbers.

Page Setup (Macintosh only)

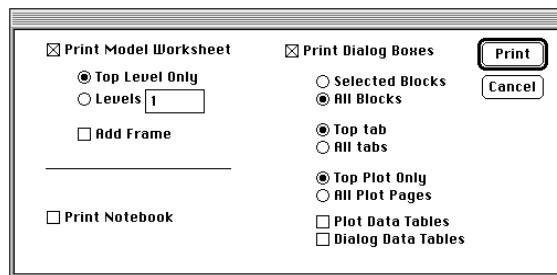
Sets the printing parameters (paper orientation, reduction or enlargement, etc.) for the printer you are using. Choose this command after changing printers with the Chooser desk accessory or whenever you want to change the printing parameters.

Print Setup (Windows only)

Sets the printing parameters (paper orientation, reduction or enlargement, etc.) for the printer you are using. Choose this command whenever you want to change the printing parameters.

Print

Prints various Extend documents. If the worksheet, dialog, or plotter is the active window, Extend first displays the following dialog:



Extend Print dialog

As discussed in “Printing” on page E162, you can choose to print the model itself, its Notebook, and/or the dialogs of the blocks in the model. Selecting Add Frame causes Extend to print a border around the worksheet. If you choose Show Block Numbers or Show Simulation Order from the Model menu before you choose Print, the blocks will print with that information on them.

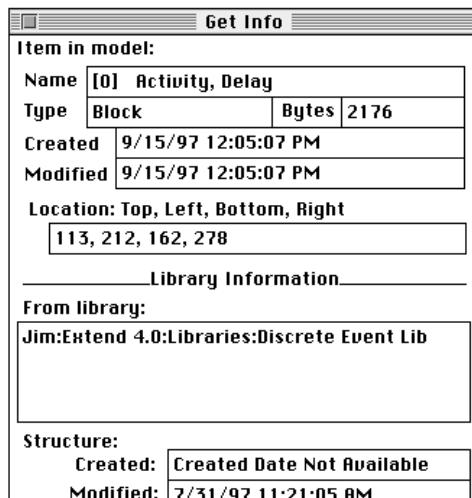
If you choose to print the block dialogs, you can specify to print the dialogs for just the selected blocks or for all of the blocks. You can also choose to print just the top tab or all of the tabs in the dialog. Since some dialogs have long data tables, you can specify whether or not to print the entire data table. For plotters, you can print just the top plot page or all pages, and you can also print the (usually lengthy) data tables from the plotters.

The structure and dialog windows have their own print dialog that lets you choose to print the ModL code, the dialog window, the connector names, the icon, or the help text.

The Print command also lets you print individual windows, such as the Notebook, the dialog, or the plotter window. When one of these windows is the active window, the Print command will print the contents of that window.

Get Info

Shows information about the selected block or blocks, including Controls and hierarchical blocks.
The window looks like:



Get Info dialog (Macintosh)

Three most recent models or text files

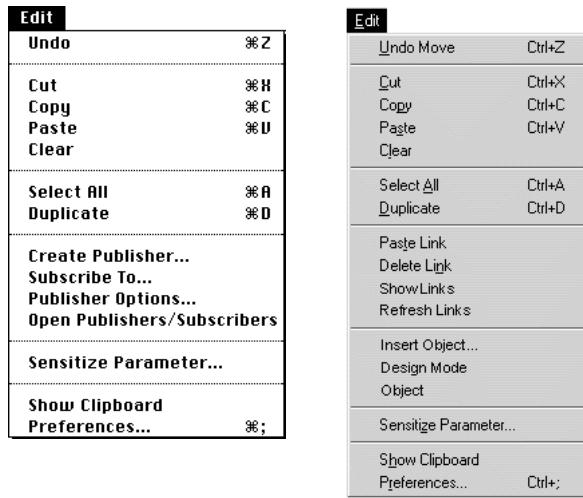
The three model or text files with which you have most recently worked will be listed towards the bottom of the File menu. To open, select one of the files.

Quit or Exit

Leaves Extend. If there are any model files with unsaved changes, you are first prompted whether you want to save them.

Edit menu

The Edit menu contains the standard Cut, Copy, and Paste commands as well as commands for directly linking with other applications, Sensitivity Analysis, and Preferences.



Macintosh

Windows

Edit menu

Undo

Reverses the most recent action. You can undo commands, moving blocks, and so on.

Cut

Removes the selected item (such as a block, some text, or numeric data from a data table) and places it on the Clipboard. You can see the current contents of the Clipboard by choosing Show Clipboard in the Edit menu.

Copy

Copies the selected item to the Clipboard. You can see the current contents of the Clipboard by choosing Show Clipboard in the Edit menu. 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 the Notebook or dialog box as a picture. *The items copied (blocks and text, drawing objects, etc.) depend on the selection tool used to make the selection.* Copying data and pictures is discussed in “Cut, Copy, and Paste with the Clipboard” on page E165.

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,

then click on the model worksheet and give 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 window.

Clear

Removes the selected item.

Select All

Selects all the items, such as all the blocks in a model or all the text in a field. The items selected (blocks and text, drawing items, and so on) depend on the selection tool chosen in the tool bar.

Duplicate

Makes a copy of the selected item(s) and puts it near the original item(s). This is faster and often more convenient than copying and pasting an item.

Paste Link (Windows only)

Copies the contents of the Clipboard to the selected parameter fields or cells. This action creates a Hot Link so that data from another application is dynamically linked to Extend. The Paste Link command will only be active if you have data in the clipboard, you have selected a location to paste to, and the other application supports Hot Links. For more information, see “Interprocess communication” on page E177.

Delete Link (Windows only)

Unlinks the selected parameter or cell so that it is no longer Hot Linked. See also the Paste Link command, above.

Show Links (Windows only)

Opens any block dialogs that have links in them.

Refresh Links (Windows only)

If linked applications are open and links appear to be working incorrectly, this command will attempt to reestablish existing links.

Appendix

Insert Object (Windows only)

Brings up a dialog list of registered embedded objects that can be inserted into an Extend worksheet or block's dialog box.

Design Mode (Windows only)

Causes the owning application to open when an embedded object is double-clicked. Also will allow an object to be moved if activated by a click in non-Design mode.

Object (*Windows only*)

Selecting an embedded object activates this menu. The contents of this menu depend upon the type of embedded object.

Create Publisher (*Macintosh only*)

Makes the selected plot or data table cells a publisher. See “Publishing data tables and plots” on page E183 for more details.

Subscribe To (*Macintosh only*)

Gets the information for the selected data table cells from a publisher. See “Subscribing to data tables” on page E184 for more details.

Publisher/Subscriber Options (*Macintosh only*)

Sets the options for a selected publisher or subscriber. See “Publisher/Subscriber options” on page E185 for more details.

Open Publishers/Subscribers (*Macintosh only*)

Locates and opens the blocks that contain publishers or subscribers. See “Locating publishers and subscribers” on page E186 for more details.

Sensitize Parameter

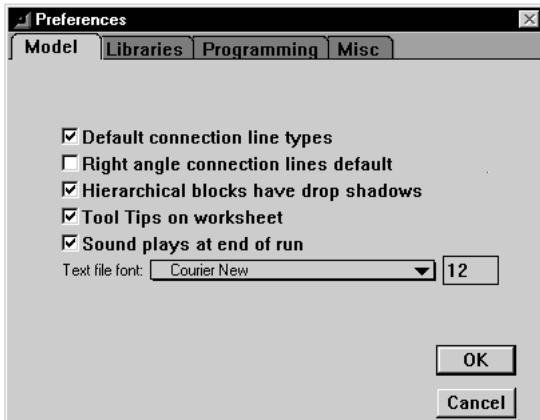
If a dialog parameter is selected, this command opens the Sensitivity Setup dialog. The Sensitivity Setup dialog lets you set values for sensitivity analysis. An alternate method of opening the Sensitivity dialog is to click on the dialog parameter once while holding down the Command (⌘ Macintosh) or Control (⌃ Windows) key. Sensitivity analysis is discussed in “Sensitivity analysis” on page E224.

Show Clipboard

Shows the contents of the Clipboard in its own window.

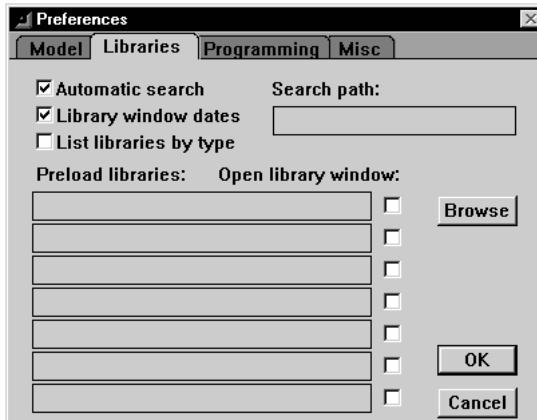
Preferences

The Preferences command from the Edit menu lets you specify how you want Extend to run. Note that this is different from the Simulation Setup command which only affects a single model. The Preferences command controls actions for all models and has four tabs: Model, Libraries, Programming, and Miscellaneous.

Model tab*Model tab of Preferences dialog (Windows)*

Option	Description
Default connection line types	Specifies whether or not Extend uses the default connection lines styles when creating new connections (see "Connection line characteristics" on page E73).
Right angle connection lines default	Sets the default for the Connection Lines command from the Model menu to be right-angle connections instead of diagonal connections.
Hierarchical blocks have drop shadows	Determines whether there is a shadow around hierarchical blocks. (Note that this does not apply if you are running Extend under Windows 3.1. Hierarchical blocks do not have drop shadows in Windows 3.1.)
Tool Tips on worksheet	If selected, displays the block name and number for blocks on the model worksheet. Note that help captions for the tool bar are not turned off if this choice is unselected.
Sound plays at end of run	Causes Extend to play the default system sound at the end of every simulation run.
Text file font	Lets you specify the font and size of the characters for viewing and printing text files.

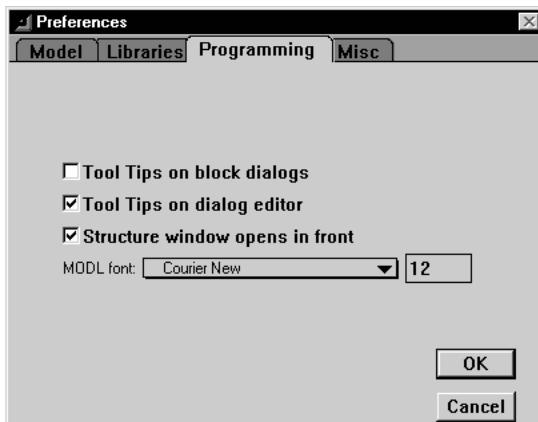
Libraries tab



Libraries tab of Preferences dialog (Windows)

Option	Description
Automatic library search	Causes Extend to automatically find and open the libraries used in a model. If this is not selected, Extend prompts you for the location of each library that a model uses.
Library window dates	If selected, the modified and compiled dates for each block will be displayed in the library windows.
List blocks by type	Causes blocks in each library to be listed in hierarchical menus by type.
Search path	Allows you to specify a default path for library searches.
Preload libraries	Enter names of libraries that are to be automatically opened when Extend is started. You can type in the name or use the Browse button to locate a library and cause its name to be entered in the selected field.
Open library window	If selected, the window of the library listed directly to the left of the checkbox will be opened when Extend is started.
Browse	Allows you to locate and select a library file. The library name will be entered in whichever "Preload libraries" field you have selected.

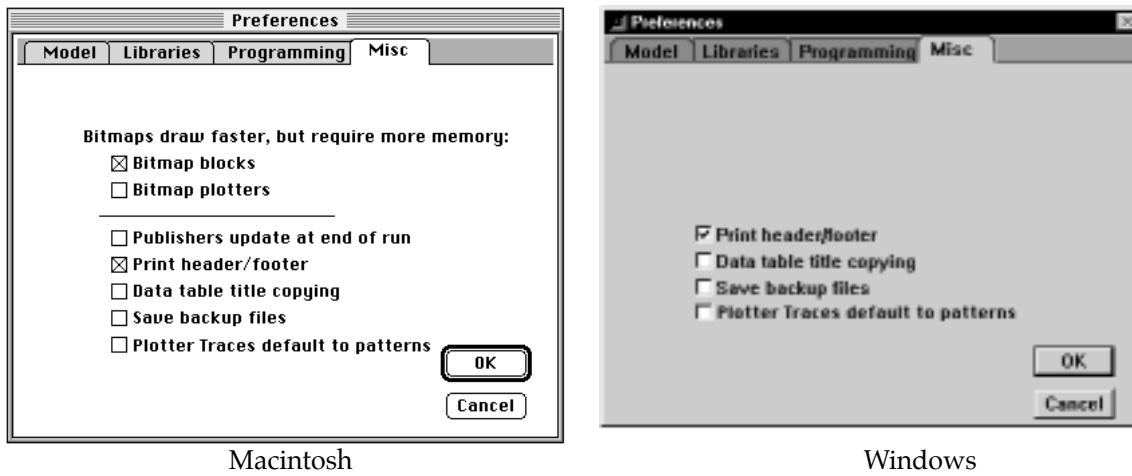
Programming tab



Programming tab of Preferences dialog

Option	Description
Tool Tips on block dialogs	If selected, displays the variable/message dialog names for dialog items.
Unsupported XCMD callbacks beep	(<i>Macintosh only</i>) Causes Extend to beep and show an error message if the XCMD you use has callbacks that are not supported in Extend. If this choice is not selected, Extend will ignore unsupported calls. This should be turned on when you are trying out a new XCMD, and turned off after you have determined that the unsupported callbacks are not necessary. See “XCMDS (Macintosh only)” on page P117 for a list of supported and unsupported callbacks.
Tool Tips on dialog editor	If selected, displays the variable/message dialog names for dialog items in the dialog structure window.
Structure window opens in front	When you open a block, Extend opens the block’s structure window in front of its dialog window. Deselect this choice if you want the dialog to open in front.
ModL font	Lets you specify the font and size of the characters for viewing and printing the code pane of the structure window.

Miscellaneous tab



Macintosh

Windows

Miscellaneous tab of Preferences dialog

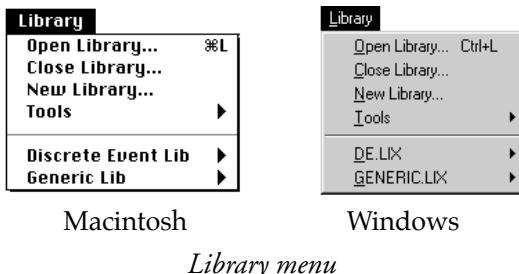
Option	Description
Bitmap blocks	(<i>Macintosh only</i>) Causes the block icons to be saved as a bitmapped image the first time they are drawn in the model window. This speeds up subsequent redrawing of the screen but uses more memory.
Bitmap plotters	(<i>Macintosh only</i>) Causes the plot pane to be saved as a bitmapped image. This speeds up subsequent redrawing of the plot but uses more memory. Note that the redraw speed increases only when the plotter has not changed from the saved image, for example when another window moves in front of it and then is moved away again. There is no speed increase when the plotter has been resized or modified.

Option	Description
Publishers update at end of run	(<i>Macintosh only</i>) Causes an update of publisher information, for all publishers in the model, when the simulation run ends. Note that this departs from Apple's standard interface, where publishers are only updated when the model is saved.
Print header/footer	If selected, prints the header and footer information for all files.
Data table title copying	Specifies whether or not the Copy command copies row and column titles when copying from a data table. Select this option if you want to copy the titles, such as when you are copying to another program. Do not select this if you are copying into another table within Extend.
Save backup files	If selected, a backup file (filename.BAK) will be saved into the same folder or directory as the original file. The backup file will contain the previously saved version of the model. It will not contain any new changes made to the model.
Plotter Traces default to patterns	If selected, plotter traces are drawn with a pattern, allowing traces to be visible on a black and white monitor.

Library menu

Extend opens libraries automatically when you open models. To open or close a library manually, you use the Library menu.

You add blocks to your model by selecting them from the open libraries listed at the bottom of the Library menu. The first choice in each list is Open Library Window, which opens a window listing the blocks in the library. See “Library usage and maintenance” on page E140 for more information about libraries and library windows.



Open Library

Opens a library file. This causes the library's name to appear in the Library menu in alphabetical order. To view the blocks in the library, click the Library menu and drag down to the name of the library. When the library name is selected, the names of all the blocks in the library, or submenus that contain the blocks by type, are listed to the right of the menu. The first choice in the blocks

list for each library in the Library menu is Open Library Window. When you select this command, Extend opens a library window for that library as discussed in “Library windows” on page E144. Blocks within each library will be either listed alphabetically or by type, depending on an option in the Libraries tab of the Preferences command. Block types are discussed in “Listing blocks by type or alphabetically” on page E146.

Close Library

Closes an open library. This command displays a dialog of 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.

New Library

Creates a new library.

Tools

Allows you to protect the code of blocks, set library versions, convert libraries to RunTime format and edit the RunTime Startup Screen.



Conversion Tools choices

Protect Library

Removes the ModL code from all the blocks in a library so users cannot access block code. This is discussed in “Protecting libraries” on page P241. You would only use this command to protect libraries of blocks you build yourself; do not use it with the libraries that come with Extend.

Set Library Version

Allows you to set the long and short version strings for the library. This is useful if you are programming your own libraries and are concerned about version control.

Convert to RunTime Library

Changes a copy of a selected library to RunTime format. The library can then be read by a RunTime or Player version of Extend, but not by regular versions of Extend.

As mentioned in “Extend RunTime and Player versions” on page E91, RunTime and Player versions of Extend allow users to run models and change parameters but do not allow users to build models or blocks. The RunTime and Player versions of Extend only read libraries that have been converted to RunTime format. Converting libraries to RunTime also removes their ModL code, as discussed in “Protecting libraries” on page P241.

RunTime Startup Screen Editor

Allows you to customize the startup screen of a RunTime version of Extend so that users know who to contact if they have questions. To use this command, you must have an Extend RunTime application installed on your hard drive. Note that you cannot personalize the startup screen of the Player version.

Open All Library Windows

Opens all the library windows for libraries currently open in Extend.

Compile Open Library Windows

Causes all libraries whose library windows are open to be recompiled. To activate the command, open the library window for the desired library.

Compile Selected Blocks

Causes the selected blocks in the library to be recompiled. To activate the command, open the library window for the desired library and select the desired blocks.

Remove Debug Code in Open Library Windows

Causes all libraries whose library windows are open to be recompiled without Debugger code. To activate the command, open the library window for the desired library.

Add Debug Code to Open Library Windows

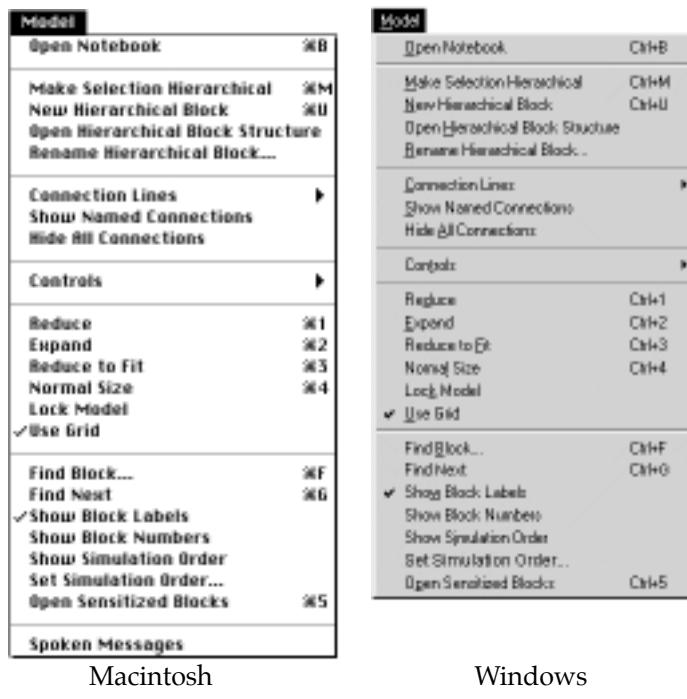
Causes all libraries whose library windows are open to be recompiled with Debugger code. To activate the command, open the library window for the desired library and select the desired blocks.

Libraries

As libraries are opened, they will be listed at the bottom of the Library menu. The libraries are organized in hierarchical menus by library name, then by block type (if applicable), and finally by block name. See “Listing blocks by type or alphabetically” on page E146 to see how to cause the libraries to list blocks by types.

Model menu

The commands in this window affect the parts of the model window and the way that the window is viewed.



Macintosh

Windows

Model menu

Open Notebook

Opens the Notebook for the model. A Notebook can be used to document the model and modify model parameters. If the Notebook has anything in it, the Open Notebook command has a check mark next to it; if there is nothing in the Notebook, the check mark does not appear. “Notebooks” on page E160 describes Notebooks in detail.

Make Selection Hierarchical

Changes the selected blocks into a hierarchical block and replaces them on the model with a single block. This is described in “Hierarchy” on page E252.

New Hierarchical Block

Starts a new hierarchical block. This prompts you for the name of the new hierarchical block, then opens a blank hierarchical block structure window. This is described in “Hierarchy” on page E252.

Open Hierarchical Block Structure

Opens the structure of the selected hierarchical block. The command is equivalent to double clicking a hierarchical block in the library window or on the model worksheet while holding down the Option (Macintosh) or Alt (Windows) key.

Rename Hierarchical Block

Changes the name of a hierarchical block. The block must be either selected in the library window or its structure window must be the active window.

Connection Lines

Sets the format of the connection lines. These are described in detail in “Connection line characteristics” on page E73. The choices are:

diagonal		Ctrl-Shift-A
right angle		Ctrl-Shift-B
right arrow		Ctrl-Shift-E
left arrow		Ctrl-Shift-F
default line types		Ctrl-Shift-G
thin		Ctrl-Shift-H
medium		Ctrl-Shift-J
thick		Ctrl-Shift-K
hollow		Ctrl-Shift-M
solid		Ctrl-Shift-N
dotted		Ctrl-Shift-O
✓ Black Connections		Ctrl-Shift-Q
Color Connections		Ctrl-Shift-T

Connection Lines choices

The View Using Defaults choice draws connections depending on their type. Item connections are drawn hollow, Value connections are drawn solid, and Diamond connections are drawn dashed (see “Connection line characteristics” on page E73).

Show Named Connections

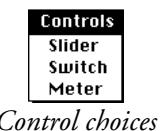
Shows the connections between named connections. This is useful to show data flow in complex models with many named connections. Named connections are discussed in “Additional ways of connecting blocks” on page E59.

Hide All Connections

Hides the connecting lines between blocks. This is a cosmetic change that is mostly used to enhance presentations.

Controls

Allows you to add a control block to the model. You can add a Slider, a Switch, or a Meter. These are described in “Control blocks” on page E88.



Reduce

Scales the image of the model by half. You can use the Expand command or Normal Size command to reverse the action of this command. The Reduce command is useful if your model is too large to fit on the screen and you want to see the blocks without scrolling. You can choose Reduce several times to display a large model at a very reduced scale. You cannot edit text or change the values of cloned dialog items while the model is reduced.

Expand

Doubles the size of the model. This works the opposite of the Reduce command. Extend shows a rectangle; click and drag it to the desired view. When you release the mouse button, the view is expanded. You cannot edit text or change the values of cloned dialog items while the model is expanded.

Reduce to Fit

Scales the model so that it fits in the active window. You can use this instead of using the Reduce command many times. You cannot edit text or change the values of cloned dialog items while the model is reduced.

Normal Size

Enlarges a previously-reduced model to its original size. Extend shows a rectangle; click and drag it to the desired view. When you release the mouse button, the view is expanded.

Lock Model

Prevents any modification of a model other than changing dialog values. This command also hides most of the tools in the tool bar so that the user cannot add or change connection lines, drawing elements, and so on. Locking models is useful if you are giving the model to others who are unfamiliar with Extend’s features and may accidentally move or delete blocks. To unlock the model, simply choose the Lock Model command again.

Use Grid

Creates an invisible snap grid on the model worksheet, Notebook, or icon pane to help you draw, move, and resize objects and blocks. The grid spacing is 4 pixels. While the grid is enabled, you can snap the upper left hand corner of an item to the grid. You can bypass the grid by holding down the Option (Macintosh) or Alt (Windows) key.

Find Block

Selects a block by its global block number, block label, name, type, or text block contents. Block numbers are unique, permanent identifiers for blocks. Block labels are user defined in the block dialog and are especially useful to find classes of blocks. *Name* means the block name in the library menu (e.g. Activity Delay). *Type* is the library type (e.g. Queues). The text means the actual text of a text block. This command is useful on large models when you are looking for blocks.

Find Next

Finds the next block meeting the above criteria.

Show Block Labels

Shows the block labels below the blocks in the model. For more information, see “Labeling the blocks and adding comments” on page E54.

Show Block Numbers

Puts the block numbers in brackets on the blocks in the model. Block numbers are unique, permanent identifiers for blocks. Hierarchical blocks and the blocks inside them show two numbers. The first number is their global block number and the second is their local block number.

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.

Set Simulation Order...

Sets the number of the block’s order of execution in the model. See “Custom order” on page E212.

Appendix

Open Sensitized Blocks

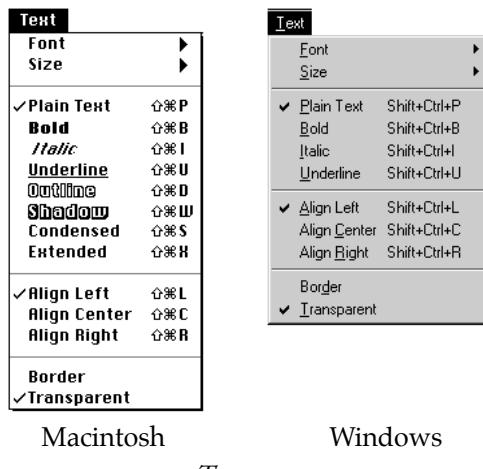
Opens the dialogs of all blocks in which sensitivity analysis has been activated. This is useful for finding which blocks are used in the scenario. For more information see “Sensitivity analysis” on page E224.

Spoken Messages (Macintosh only)

Causes all messages to be spoken if you have a speech synthesis module (the Apple Speech Manager) in your Macintosh System folder. As Apple comes out with additional speech synthesis technologies, Extend will automatically work with them.

Text menu

The Text menu is used to set the style of text in the model and to temporarily set the style of text in text files. The choices are the same as in most applications. The Border command draws a shadowed border around text in the window and a black border around drawing objects like the *rectangle*. The Transparent command causes the background of the text to be transparent. If the Transparent command is not selected, the background of the text is white. For more information about using text, see “Text” on page E69.



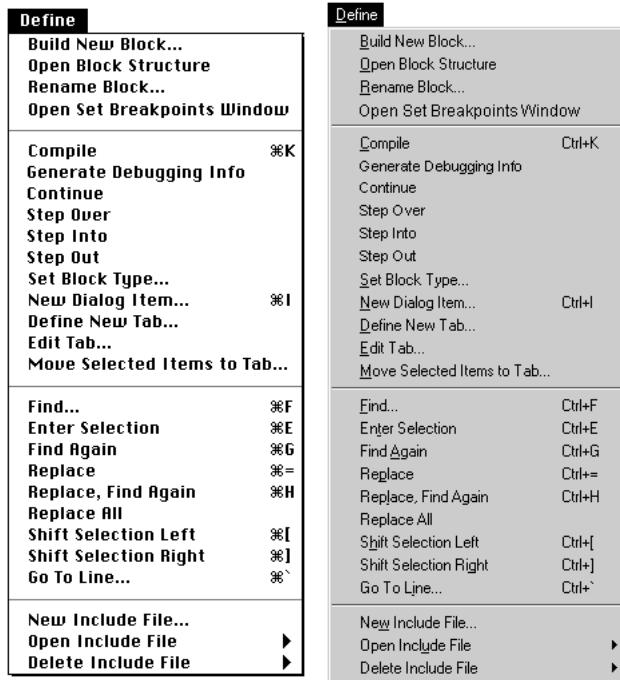
Macintosh

Windows

Text menu

Define menu

The Define menu is used when creating or modifying blocks in libraries. Chapters 9 through 12 discuss building and modifying blocks.



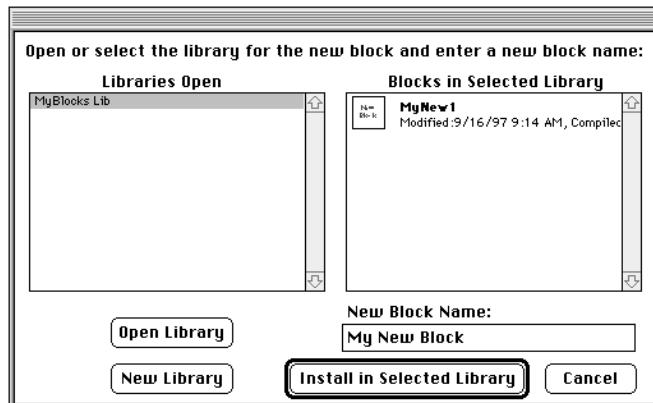
Macintosh

Windows

Define menu

Build New Block

Creates a new block. In the initial dialog, you specify the library in which you want to create the block and the block's name. Use the Open Library or New Library buttons to open or create a library for the new block.



Build New Block dialog (Macintosh)

After naming the block and selecting the library, you are presented with the default dialog and structure windows. For a quick overview on building a new block, see “Creating a block” on page P38.

Open Block Structure

Opens the structure of the selected blocks. This command is equivalent to double clicking a block in the library window on the model worksheet while holding down the Option (Macintosh) or Alt (Windows) key.

Rename Block

Changes the name of the block, including hierarchical blocks. The block must be either selected in the library window or its structure window must be the active window.

Open Set Breakpoints Window

Opens the *Set Breakpoints* window for the selected block or brings it forward into view if it is already open. This window shows the source code for the block, along with a breakpoint margin on the left of the window. If it is needed, the block is automatically recompiled in debugging mode.

Compile

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 structure or dialog windows. The Compile command is only available when the structure window of a block is the active window.

Generate Debugging Info

If this menu item is checked, the compiler will generate debugging information for this block when it is compiled. Blocks with debugging code run slower and are shown in red in the open library window.

Continue

Continues execution from a breakpoint.

Step Over

Steps over a function call when stepping after a breakpoint.

Step Into

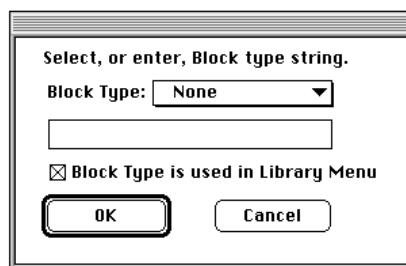
Steps into a function call when stepping after a breakpoint.

Step Out

Steps out of a called function to return to the caller when stepping after a breakpoint.

Set Block Type

Allows you to set the type of a block. The Set Block Type dialog is:



Set Block Type dialog

A block's type serves two functions:

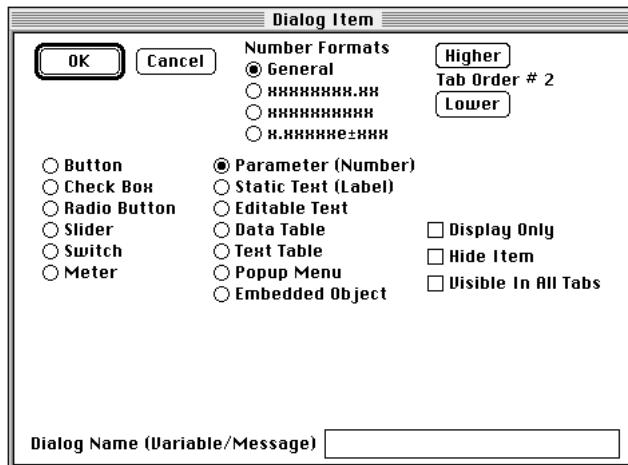
- To organize blocks within the Library menu by functionality.
- To organize blocks within statistical reports by functionality.

Appendix

The Set Block Type command is only available when the structure window of a block is the active window. If "Block Type is used in Library Menu" is not checked in the Set Block Type dialog, the block name will be listed in alphabetical order directly under the library name in the Library menu. See "Organizing blocks in libraries" on page E146, "Model reporting" on page E187, and "Block types" on page P56 for further discussions on block types.

New Dialog Item

Adds a dialog item (such as a button, some text, and so on) to the block's dialog window. This command is only enabled when a block's dialog window is the active window. Dialog items, and the use of this command, are covered in detail in "Dialogs" on page P8. Your choices are:



New Dialog Item dialog

Define New Tab

Adds a new tab to the block's dialog window. This command is only enabled when a block's dialog window is the active window. Dialog tabs are covered in detail in "Dialogs" on page P8.

Edit Tab

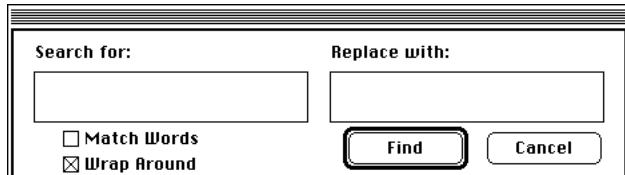
Allows you to rename or delete a tab. This command is only enabled when a block's dialog window is the active window. Dialog tabs are covered in detail in "Dialogs" on page P8.

Move Selected Items to Tab

Allows you to move the selected dialog item to a specific tab in the block's dialog. This command is only enabled when a block's dialog window is the active window and a dialog item is selected. Dialog tabs are covered in detail in "Dialogs" on page P8.

Find

Searches for text in the help, a text file, or in the ModL code of a structure window 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. The Find dialog is:



Find dialog

Enter the text you want to search for in the *Search for:* edit box. The Match Words option tells Extend to match whole words only (e.g. *boxer* would not be found by a search specifying *box* as the *Search for:* string). The Wrap Around choice tells the command to begin searching from the top of the ModL code if it did not find the text before the end.

The *Replace with:* edit box is used for finding and replacing at the same time (see Replace).

Enter Selection

Chooses the selected text to be the search string. This puts the text in the *Search for:* edit box and also determines the search string to be used for the Find Again command. This is useful if you want to find the next instance of some text that is already in the code.

Find Again

Repeats the most recent find operation using the same search string.

Replace

Replaces the selected text with the text in the *Replace with:* edit box in the Find dialog. Note that the currently selected text need not necessarily be the same text as that contained in the *Search for:* box in the Find dialog. If there is no text in the *Replace with:* edit box, the selected text is deleted.

Replace, Find Again

Replaces the selected text with the contents of the *Replace with:* box in the Find dialog, then performs another Find command. If there is no text in the Replace with: edit box, the selected text is deleted.

Replace All

Replaces every instance of the text in the *Search for:* box with the text in the *Replace with:* edit box in the Find dialog. This starts at the current selection point, replaces every instance, and should therefore be used with caution.

Shift Selection Left

Moves the selected lines of the ModL code one tab stop to the left.

Shift Selection Right

Moves the selected lines of the ModL code one tab stop to the right.

Go To Line

Allows you to quickly move to a specific line in the block code.

New Include File

Creates a new Include file window. See “Include files” on page P183.

Open Include File

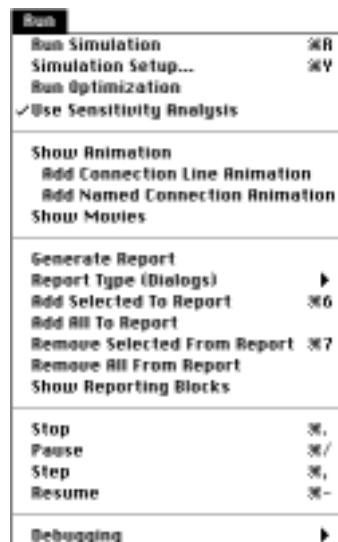
Opens an existing Include file window. See “Include files” on page P183.

Delete Include File

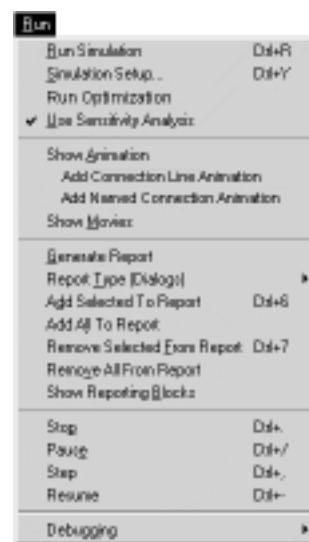
Deletes an Include file. See “Include files” on page P183.

Run Menu

The Run menu lets you modify the way your simulation runs.



Macintosh



Windows

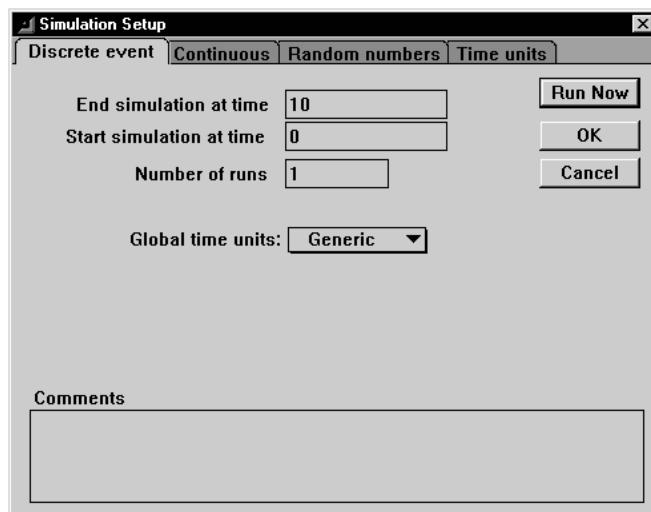
Run menu

Run Simulation

Starts a simulation. The command first checks every block in the model to see that it has been compiled.

Simulation Setup

Modifies the parameters of the simulation. Its dialog is:



Simulation Setup dialog

This dialog is described in detail in “Simulation Setup command” on page E204.

Run Optimization

Runs an optimization. Alerts the user if there is no optimization block on the model. See “Optimization Tutorials” on page E231.

Appendix

Use Sensitivity Analysis

Causes Extend 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 E83.

Show Animation

Causes blocks with animation to animate when the simulation is run. Animation is discussed in “Animation” on page E81. Note that some blocks can show some animation, such as text on the icon reporting final values, even if Show Animation is not selected.

Add Connection Line Animation

This option controls whether or not connection line animations will be displayed. If this is on, the blocks will display their block to block animations as discussed in “Animation between block icons” on page E82. If it’s off, only on block animations will be displayed.

Add Named Connection Animation

If the connection line is a named connection (connection line does not appear on the worksheet), this option will cause the 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, reappear at the matching text label, and continue traveling to the next block.

Show Movies (Macintosh only)

Causes blocks that have QuickTime movies to show their movies when you run the simulation. This is only available if you have QuickTime installed. You do not have to have Show Animation selected to show movies.

Generate Report

Causes Extend to generate a report file when the simulation is run. Extend will prompt for a name for the new report file when the model is run. Report files show final results of the simulation for selected blocks. They are described in “Model reporting” on page E187.

Report Type

Allows you to choose which report type, Dialogs or Statistics, to generate. The current selection is shown in parenthesis following the command. Report files show final results of the simulation for selected blocks. They are described in “Model reporting” on page E187.

Add Selected To Report

Causes blocks that have been selected to be included in the report.

Add All To Report

Causes all blocks to be included in the report.

Remove Selected From Report

Causes blocks that have been selected to be removed from the report.

Remove All From Report

Causes all blocks to be removed from the report.

Show Reporting Blocks

Causes blocks that have been included in a report to show the word “Report” on them.

Stop

Stops the simulation. This is the same as the Stop button in the status bar. As an alternative, you can hold down the Command (Macintosh: ⌘) or Control (Windows: CTRL) key while pressing the period (.) key.

Pause

Temporarily halts the simulation. This is the same as the Pause button in the status bar.

Step

Steps the simulation depending on what mode is selected in the Debugging menu. See the Debugging menu, below. This is the same as the Step button in the status bar.

Resume

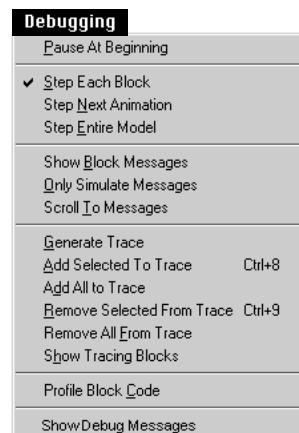
Restarts a paused simulation. This is the same as the Resume button in the status bar.

Debugging

This hierarchical menu lets you modify the way your simulation runs and facilitates finding a modeling problem. The “Step...” commands in this menu determine how the Step commands in the Run menu perform during a simulation run. The Trace commands are used to generate a Trace file of the values for each selected block at each step of the simulation. For debugging tips see “Debugging hints” on page E217.



Macintosh



Windows

Debugging menu

Pause At Beginning

If this is enabled, the simulation pauses when started so that the user can step through the run from step zero.

Step Each Block

Controls the behavior of the Step command or button so that you can step through a simulation block by block. This command is only active when the Pause command or Pause button have been activated.

Step Next Animation

Controls the behavior of the Step command or button so that you can step through a simulation, pausing only at animation changes. This command is only active when the Pause command or Pause button have 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 blocks in the model. Each Step command causes the simulation run to continue until the execution order returns to the original block. This command is only active when the Pause command or Pause button have been activated.

Show Block Messages

Used in conjunction with the Step Each Block, Step Next Animation, or Step Entire Model commands to show system messages on the blocks. When you choose Run, the simulation automatically pauses so you can step through one cycle of the simulation or block by block. The block that is active is highlighted and the current message name is written on it. If that block is not visible, the window scrolls to the block.

Only Simulate Messages

When the Show Block Messages command is active, causes only the messages that occur during the Simulate stage of the run to be shown. This saves time by not showing all the initial and final messages.

Scroll To Messages

This causes the window to scroll automatically, following the path of messages so that the user can quickly step through the model.

Generate Trace

Causes Extend to generate a trace file when the simulation is run. Extend will prompt for a name for the new trace file when the model is run. For more information, see “Model tracing” on page E218.

Add Selected To Trace

Causes blocks that have been selected to be included in the Trace when the model is run.

Add All To Trace

Causes all blocks to be included in the Trace when the model is run.

Remove Selected From Trace

Causes blocks that have been selected to be removed from the Trace.

Remove All From Trace

Causes all blocks to be removed from the Trace. 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.

Profile Block Code

When the command is active, generates a text file showing the percentage of time blocks execute during a simulation. See “Model profiling” on page P233 for more information.

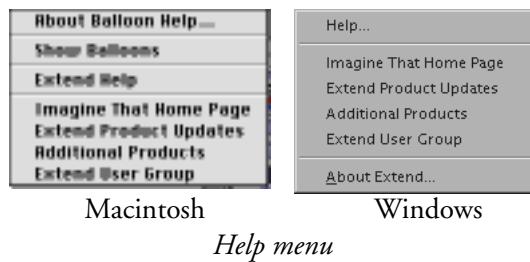
Show Debug Messages

This menu item allows the user to turn on and off the messages created using the debugMsg() function.

Window menu

The Window menu lists all of the Extend windows which are open. To bring a window to the top of your workspace, select it from the menu. This list includes all models, dialogs, and library windows open.

Help menu

**About Balloon Help (Macintosh only)**

Displays a description of Balloon Help. Balloon Help is used to help you learn about the items on your screen. As you point your cursor to items on your screen, help balloons are displayed about the items.

Show Balloons (Macintosh only)

Activates Balloon Help.

Extend Help (Macintosh) / Help (Windows)

Shows a list of help topics available. Select a topic from the list and click Help for more information. You can search for key words within any Help topic.

Imagine That Home Page

This uses your current browser to open the Imagine That, Inc. home page at:
<www.imaginethatinc.com>.

Extend Product Updates

This uses your current browser to open the Imagine That, Inc. updates web page.

Additional Products

This uses your current browser to open the Imagine That, Inc. products web page.

Extend User Group

This uses your current browser to open the current Extend user's group homepage.

About Extend (Windows only)

Tells you the version of Extend that you are using. Click on the banner window to continue.

Appendix B: Upper Limits

*A list of the maximum numbers of things
that you can do at one time*

“The thing I am most aware of is my limits.”
— André Gide

Like every program, Extend has its limits. It is unlikely that you will find them in your normal work, but it is good to know what they are in case you come across them.

Blocks in a model	2 billion
Output connectors in a model (nodes)	2 billion
Connectors per block	255
Length of a block's ModL code (characters)	4 megabytes
Dialog items in a block	1024
Dynamic arrays (each array)	2 gigabytes
Number of array dimensions	5
Maximum index for array dimensions	2 billion
Dynamic arrays per block	255
Text files open at one time	200
Blocks in a library	200
Libraries open at one time	30
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
Steps in a simulation run	2 billion
Block name length, characters	31
Variable name length, characters	63
Number of attributes for discrete event item	300
User defined function arguments	127
Nested loops	32
Number of simulations in a multiple run	32K
Maximum, minimum of real numbers	$\pm 1E\pm 308$
Maximum, minimum of integer numbers	$\pm 2,147,483,647$
Significant figures in real calculations	Macintosh: 20 (IEEE extended) PowerMacintosh: 16 (double) Windows: 16 (double)

Appendix C: Cross-Platform Considerations

*File conversion, file name comparisons, and keyboard shortcuts
for the Macintosh and Windows systems*

“There never were, since the creation of the world, two cases exactly parallel.”
— Lord Chesterfield

Libraries

- **Macintosh:** library names can be up to 31 characters long and usually end in “Lib”, although this is not required. Libraries are stored in the Libraries subfolder within the Extend folder.
- **Windows:** Extend supports file names up to 64 characters for Windows 95, 98, ME, 2000, and Windows NT. Library names must end in the “.LIX” extension. Libraries are located in the *Libraries* subdirectory within the Extend directory.

Models

Macintosh model names can be up to 31 characters. For Windows 95 and NT, Extend supports long file names. All Windows models must end in the “.MOX” extension.

Menu and keyboard equivalents

The following table lists some common actions and keyboard shortcuts under the Macintosh and Windows systems. “Appendix A: Menu Command Reference” on page A1 contains pictures of the Extend menus, including the keyboard equivalents for the menu commands.

Action	Command (in... menu)	Macintosh keyboard	Windows keyboard
Open a model	Open... (File menu)	Command-O (⌘-O)	Control-O (CTRL+O)
Open a library	Open Library (Library menu)	Command-L (⌘-L)	Control-L (CTRL+L)
Save a model	Save Model (File menu)	Command-S (⌘-S)	Control-S (CTRL+S)
Close the active window	Close (File menu)	Command-W (⌘-W)	Control-W (CTRL+W)
Run a simulation	Run (Run menu)	Command-R (⌘-R)	Control-R (CTRL+R)
Stop a simulation	Stop (Run menu)	Command-period (⌘-.)	Control-period (CTRL+.)
Stop a library search		Command-period (⌘-.)	Control-period (CTRL+.)

Action	Command (in... menu)	Macintosh keyboard	Windows keyboard
Select a parameter for sensitivity analysis	Sensitize Parameter (Edit menu) <i>(Select parameter first)</i>	Hold down the Command (⌘)key and click once on the parameter	Hold down the Control (CTRL) key and click once on the parameter
Open a hierarchical block's structure to edit the icon, etc.	Open Block Structure (Define menu) <i>(Select block icon first)</i>	Hold down the Option key while double-clicking on the block's icon	Hold down the Alt key while double-clicking on the block's icon
Open a library block's structure to edit the Modl code or icon	Open Block Structure (Define menu) <i>(Select block icon first)</i>	Hold down the Option key while double-clicking on the block's icon	Hold down the Alt key while double-clicking on the block's icon
Remove the grid when aligning drawing objects, connectors, etc.		Hold down the Option key while moving the object	Hold down the Alt key while moving the object
Proportionately scale drawing object		Hold down the Shift key while resizing the object	Hold down the Shift key while resizing the object

Transferring files between operating systems

The Windows and Macintosh versions of Extend are cross-platform compatible. For example, if you build a model or a library on your Windows computer, you can move it to a Macintosh computer and Extend will read it, and vice versa.

There are three considerations when transferring Extend files between Windows and Macintosh systems: file name adjustments, physical transfer, and file conversion.

Note Models and libraries developed in an older version and different platform may not transfer successfully. It is strongly recommended that you upgrade to the latest version on the source platform, resave models, recompile libraries, and resave the structure of hierarchical blocks in libraries (see “Saving hierarchical blocks in a library” on page E259) before transferring your files.

Appendix

File name adjustments

If you transfer files from a Macintosh to a Windows computer, you may need to change the name of your file before you transfer it. Extend supports file names of up to 64 characters for Windows 95, 98, ME, 2000, and Windows NT. File names must end in a three (3) character extension (the extensions are “.MOX” for Extend model names, “.LIX” for library names, and “.TXT” for text file names). To make your file names be in Windows format, change the name of the file *on the*

Macintosh computer using Extend's Save As command (if the file is open) or using the Finder (if the file is not open).

If you are transferring files from a Windows computer to a Macintosh, you do not need to change the file name or delete the extension; the Macintosh system can read names up to 31 characters long.

Note It is important that you do not delete the .MOX extension from a Windows model file name before transferring the model to your Macintosh system. The .MOX extension is required so Extend can identify the file as a Windows model. After Extend has converted the Windows model to Macintosh format, you can save the model under the same name or a new name.

Physically transferring files

Once you have made any necessary file name adjustments, you need to physically transfer the files between Macintosh and Windows computers. This process depends on your system resources and is independent of Extend. For example, you might copy the file onto a floppy disk to be read by a program that is designed to read disks from other operating systems (such as PC Access on the Macintosh or Conversion Plus for Windows). If you have the two computers networked, you could send the files directly from one to the other. If you are using Macintosh System 8.6+, it can read and write disks formatted for Windows systems.

Note Libraries and extensions that you build on a Macintosh computer need to be converted *before* being physically transferred to the Windows system, as discussed below.

File conversion

Depending on the type of file, file conversion may be handled automatically by Extend or may involve using a conversion application. As discussed below, Extend automatically converts model files when they are opened on a different platform. If you program your own blocks, the Extend for Macintosh package includes a file conversion utility which you can use to convert libraries and picture resource extensions from one operating system format to another. Other extensions that you build, such as QuickTime movies, DLLs, and XCMDs require more extensive conversion. Include files are, of course, already cross-platform compatible.

Model files

The Windows version of Extend can read Extend model files created on the Macintosh as long as the name format is correct, as discussed above. The Macintosh version of Extend can read Extend model files created under Windows without file name modification (in fact, as discussed in the Note above, the .MOX extension should not be removed.) When you open a model file that was created on another operating system, Extend will notify you that it is converting the file from that system to the current one. Once you save the model file, it will be in the format of your current operating system.

If your models (including hierarchical blocks) use libraries that you have created yourself, and you have changed the name of those libraries, Extend will not be able to locate the library. In this case,

Extend will ask you to find and select the correct library as described in “Searching for libraries and blocks” on page E142. Keeping all your libraries in the Libraries folder will make this search process easier. Saving the model will cause the new libraries to be used from then on.

For model files that have blocks that access text files (such as the File Input block from the Generic library) you may need to change the name of the text file that is being read to conform to platform requirements, as discussed above. Be sure to also change the name of the file in the File block’s dialog to correspond to the new file name.

Note The first time you run a model that has been transferred from one operating system to another, any Equation blocks in the model will recompile to the format of the new system at the beginning of the simulation run.

Hierarchical blocks in libraries

If you have a hierarchical block saved in a library and you have renamed any of the libraries of the blocks *inside* the hierarchical block (for example, to comply with Windows format), you need to update the hierarchical block’s information so that it can locate the renamed libraries. The easiest way to do this is to drag hierarchical blocks from their libraries, place them on a worksheet, and update their structure, as discussed below.

Note This is only required for hierarchical blocks saved in libraries; hierarchical blocks saved only in a model get updated with the model.

When you add a hierarchical block from a library to a model worksheet, the hierarchical block causes Extend to open the libraries of the blocks inside it. Since you have renamed those libraries, Extend will not be able to locate them. In this case, Extend will ask you to find and open the correct libraries. Note: keeping all your libraries in the Libraries folder will make this search process easier.

If you save the model worksheet that contains the hierarchical block, the location of the renamed libraries is saved for the model only. Before you close the model worksheet, you also need to update the hierarchical block’s library information. To do this, open the hierarchical block’s *structure window* and then close it, causing the hierarchical block Save Dialog to appear. In the dialog, choose “Also Save to Library”. This process is described in “Results of modifying hierarchical blocks” on page E263.

Appendix

Libraries

The libraries that come with your Extend package are already formatted correctly for your operating system. However, if you build your own libraries, and want to transfer them to a computer running a different operating system, you must convert them to the appropriate operating system format. You do this **on the Macintosh computer** using the Extend conversion utility, MacWin Converter. This utility converts libraries to the specified operating system format, ensures that the file names are properly formatted, and so forth.

After the libraries have been converted to Windows or Macintosh format, physically transfer them to the target computer (that is, keep them on the Macintosh or transfer them to a Windows computer). Then recompile the libraries under the target computer's operating system using Extend's Compile Open Libraries command.

The MacWin Converter converts libraries to either Macintosh or Windows format. After conversion, you must recompile the library on the target computer. When you do this, the library will compile to native code for the target system. For example, if you compile on a Windows computer, the library will be in native Windows mode. If you compile on a Power Macintosh computer, the library will be in native Power Macintosh mode.

Blocks that use the equation functions

If you build blocks that use the equation functions, your code needs to detect if the model is being opened on a different platform. See "Equations" on page P96 for more information.

Extensions and drivers

Extensions are files (such as pictures and DLLs) that can be accessed by Extend to fulfill specialized tasks. Drivers ( *Macintosh only*) are code segments that allow you to access external devices and functions. Like libraries, the extensions that come with your Extend package are formatted correctly for your operating system. However, if you build your own extensions or use Extend's driver calls, and you want to transfer your extensions or blocks to a computer running a different operating system, you will need to do some conversion:

- *Pictures:* As discussed in "Picture and movie files" on page P241, Extend for Windows accepts three kinds of pictures: .WMF (Windows MetaFiles), BMP (Bitmap), and a Macintosh picture resource file that has been converted to Windows format. Extend for the Macintosh accepts only picture resources. To convert Macintosh picture resource files to Windows format, use Extend's MacWin Converter utility on the Macintosh. To convert Windows pictures to Macintosh format, use a graphics conversion application.
- *Movies:* When you create a QuickTime movie file, save it in multi-platform format. You can then move the movie file between operating systems without having to do any conversion. Note that Extend for Windows requires the 32-bit compatible version of QuickTime for Windows.
- *Sounds:* Shareware utilities are available to convert Macintosh sound resources (SNDs) to Windows sound files (.WAV) and vice versa.
- *DLLs, XCMDs, and XFCNs:* If you use either the DLLXCMD, DLLParam, DLLArray ( *Windows*) or XCMD, XCMDPARAM, and XCMDARRAY ( *Macintosh*) functions in your blocks, the block Modl code will not need to be rewritten when you transfer blocks between Macintosh and Windows systems. However, the extensions will have to be rebuilt on the platform to which you are transferring them. These extensions are discussed fully in "XCMDs and XFCNs" on page P237. If your blocks call DLLs, XCMDs, or XFCNs, their code needs to detect if they are being used on a different platform, as discussed in the Note, below.

- If you build blocks that use the driver functions, your code needs to detect if the blocks are being used on a different platform, as discussed in the Note, below.

For more information, see “Extensions” on page P236 or “Other drivers (Macintosh only)” on page P116.

Note The following ModL constants return TRUE or FALSE depending on the platform: PLATFORMMACINTOSH; PLATFORMWINDOWS; PLATFORMPOWERPC. If your code calls XCMDs, DLLs, or other drivers, use these constants in an “if” statement to make the code be cross-platform capable. For example:

```
if (PLATFORMWINDOWS)
    use DLL TypeLanguage calls;
else if (PLATFORMMACINTOSH)
    use Driver calls;
```


Appendix D: Generic Library Blocks

*A detailed description of the building blocks
in the Generic library*

*“It is wise to learn;
it is God-like to create.”*
— John Saxe

This chapter provides an alphabetical table of the blocks in the Generic library.

Each Generic 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. Extend icons are color coded: green-bordered icons provide items to the model, blue-bordered icons represent computation or operations, and red-bordered icons either output information about items or remove items from the model.

Table of Generic blocks

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 Extend's on-line help for the block's name
- Double-click the block on your model and click on the block's Help button in the lower left of the dialog.

Submenus

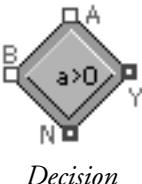
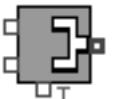
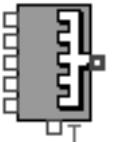
The block tables that follow are categorized by types, and displayed in hierarchical submenus of the Generic Library menu:

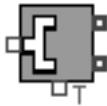
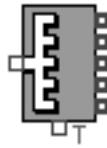
- Arrays - Storing, accessing global data
- Decisions - Routing or deciding which value to use
- Holding -Accumulating or storing values
- Input/Output - Reading and writing files, or generating values
- Math - calculating values
- Statistics - Calculating Mean, Variance

Arrays - Storing, accessing global data

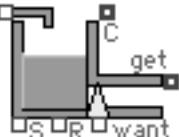
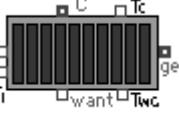
Arrays	Function
 <i>Global Array</i>	Allows access to Extend's global arrays. Global arrays are general purpose arrays available anywhere in the model. Global Array values can be set, reset, or modified. Any number of Global Array blocks can access the same array. One can be used to set the values in the array and many others can read those values at different points in the model. Row and column connectors specify the cell in the array for the value input or output connector.
 <i>Global Array Manager</i>	Creates, resizes, deletes and views Extend's global arrays. Global arrays are general purpose arrays available anywhere in the model. They can be used to transfer values from one point in the model to another or to accumulate information. The global arrays managed by this block can be either integer or real. Each global array is referenced by a unique name and can be any number of rows, and up to 255 columns.

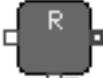
Decisions - Routing or deciding which value to use

Decisions	Function
 <i>Decision</i>	This block makes a decision based on the inputs and internal logic you define. 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 compares the two inputs (A and B). If only the A input is connected, the block compares that value to the B value in the dialog. The block can also use hysteresis. Examples of using the Decision block are contingency planning, pass/fail tests, and routing.
 <i>Select Input</i>	Selects its output to be either of two input values based on a threshold test. The block acts like a switch. The value to be output is determined by comparing the value of the T connector to a critical value set in the dialog. When the value of the T connector is less than the critical value, the top input is used; when the value of the T connector is greater than or equal to the critical value, the bottom input is used. This block is useful for any sorting procedure. The default for the critical value, 0.5, works well for boolean logic since Extend booleans have true equal to 1 and false equal to 0.
 <i>Select Input (5)</i>	Selects its output to be one of five inputs according to the value of the T connector. The top input is selected if the T connector is 1, and the bottom input is selected if the T connector is 5. The dialog also lets you specify an output value of either noValue or 0 if the value at T is out of the range of 1 to 5.

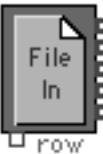
Decisions	Function
 <i>Select Output</i>	Passes the input value to one of two output connectors based on a threshold test. The output connector is selected by comparing the value of the T connector to a critical value specified in the dialog. When the value of the T connector is less than the critical value, the top output connector is used; when the value of the T connector is greater than or equal to the critical value, the bottom output connector is used. For the unselected output connector, the dialog lets you specify an output value of either noValue, 0, or a repeat of the last value it output. The default for the critical value, 0.5, works well for boolean logic since Extend booleans have true equal to 1 and false equal to 0.
 <i>Select Output (5)</i>	Passes the input value to one of five outputs according to the value of the T connector. The top output is selected if the T connector is 1 and the bottom output is selected if it is 5. For connectors that are not selected, the dialog lets you specify an output value of either noValue, 0, or a repeat of the last value they output.

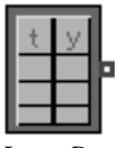
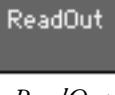
Holding -Accumulating or storing values

Holding	Function
 <i>Accumulate</i>	Adds up the values at the input connector from each step or event and shows the total contents at the output connector. This is essentially a sum operation with no delay. The Accumulate block may be used as a storage bin, reservoir, stock, or any other function that involves adding many different values together. Examples of where you would use an Accumulate block are to determine bank balance, billable hours, or number of customers served.
 <i>Holding Tank</i>	Accumulates the total of the input values, allows you to request an amount to be removed, and outputs that requested amount, if it is available. The Holding Tank block is similar to the Accumulate block except that it allows you to request an amount to be removed at each step or event. The amount to be removed is specified at the want connector; the amount is actually output at the get connector, up to the amount available. You can choose to allow output that would make the contents go negative (such as an overdraft); the default is not to allow withdrawals that make the contents go negative.
 <i>Holding Tank (Indexed)</i>	This block represents a series of holding tanks. It accumulates input values, allows the current contents to be reduced by a certain amount, and reports the contents for the tanks. The specific tank used is controlled by its toggle connector. The Holding Tank (Indexed) differs from a Holding Tank in that it contains a series of holding tanks. The active tank is controlled by input connectors which specify the index of the tank to be used. Up to 100 holding tanks can be referenced.

Holding	Function
 <i>Retain</i>	Outputs the initial value set in its dialog until the T connector is TRUE (greater than 0.5), then outputs the value of its input while T is TRUE. This block retains the current value at its input when T becomes FALSE. While T is FALSE, the retained value is output. If T becomes TRUE again, the block outputs and retains the current value at its input. The initial value must be specified in the dialog (the default initial value is 0). The value retained and output is changed to the value of the input only when the value at the T connector is TRUE (greater than 0.5).
 <i>Wait Time</i>	Holds its inputs for a specified amount of simulation time (the delay) before passing them to the output. Note: this block should not be used in a discrete event model. 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. The number of slots is equal to the delay.

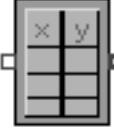
Input/Output - Reading and writing files, or generating values

Input/Output	Function
 <i>Constant</i>	Generates a constant value at each step. You specify the constant value in the dialog (the default constant is 1). If the input is connected, the input value is added to the constant in the dialog. This block is typically used for setting the value for the inputs to other blocks. For example, you can use it for a steady flow of fluid, cash, or a delay time value.
 <i>File Input</i>	Reads data from a text file and writes it into the block's table. Once the data is in the table, you can use it in your model. You select the file to be read by typing its path name, or you can leave the name blank so Extend prompts you and fills in the path name. You can create the text file using Extend's text file features, in a word processor, or a spreadsheet application. This block can be used to control parameters for many blocks in a model or many runs of the model.
 <i>File Output</i>	Writes data from the block to a text file. You can type or paste data into the block, or model data can be input through the input connectors during the simulation. You select the file either by typing its path name or you can leave the name blank so Extend prompts you and fills in the path name. You specify in the dialog whether the file is written at the end of the simulation or when you click the Write button in the dialog.
 <i>Help</i>	This block is obsolete. Use a new hierarchical block with a text description in it. Shows help text. You can use this block to document your models. Include information about what the model does, what impact specific blocks or parameters have on the model, authorship, cross-references to other models, and so on.

Input/Output	Function
 <i>Input Data</i>	<p>Generates a curve of data over time from a table of values and acts as a lookup table. This block is the same as the Conversion Table block except that the variable is always the current time of the simulation.</p> <p>You can use this block to represent items which change over time, such as traffic volume, sales volume, production, and so on.</p>
 <i>Input Function</i>	<p>Generates a function over time. This works the same as the Conversion Function block except that the variable is always the time in the simulation. A delay parameter is included for most functions so the output may be delayed for a specified length of time. In addition to the functions in the Conversion Function block, this block also includes impulse, ramp, and repeated pulse functions, however it does not contain the modulo and multiplier functions.</p>
 <i>Input Random Number</i>	<p>Generates random integers or real numbers based on the selected distribution. You can use the dialog or the three inputs, 1, 2, and three to specify arguments for the distributions. You can select the type of distribution: Uniform (integer or real), Beta, Binomial, Erlang, Exponential, Gamma, Geometric, HyperExponential, LogLogistic, LogNormal, Neg. Binomial, Normal, Pearson type V, Pearson type VI, Poisson, Triangular, Weibull, and Empirical. The Empirical distribution uses a table of up to 50 values to generate a discrete, stepped, or interpolated empirical distribution.</p>
 <i>Prompt</i>	<p>Prompts for a value to output if the inputs meet a test. Until the test is met, the block outputs a value specified in the dialog. When the test is met, the block prompts for a new output value, then outputs that value. You may enter your own prompt message in the dialog by typing over the default prompt message. If you click the cancel button in the prompt, the simulation is stopped. Note that the output of this block can be any value, whereas the Decision block outputs only 1 or 0.</p> <p>You would use this block to pause a simulation, request a value from the user, then continue the simulation using the new value.</p>
 <i>ReadOut</i>	<p>Displays the value at the input connector at each simulation step. This block comes to the front of the screen during a simulation run if the “Open dialog” option is selected, even in front of any plotters you may have open. This is useful for debugging models and scripts because you can see the value of another block’s value output connector at any time. The NTicks field in the dialog allows you to set a delay in ticks (60ths of a second) for the block to pause while displaying each number.</p>
 <i>Sound</i>	<p>Plays a sound when the input value goes above or below a critical value amount specified in the dialog. The sound name is identified in the dialog (the default is Click). You can enter the sound name for a system sound, or for any sound in Extend’s Extensions folder. Click the Play Sound button to hear the named sound. To see the System sound names, use the Sounds item in the Control Panel.</p>

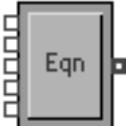
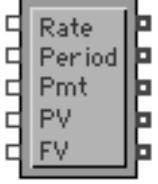
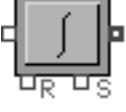
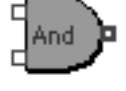
Input/Output	Function
 <i>Stop</i>	Stops the simulation when its input goes above or below a critical value that is set in the dialog. A message may be displayed when the simulation is stopped. This block is useful for ending a simulation before the end time set in the Simulation Setup command.
 <i>System Variable</i>	Allows you to regulate some aspect of the model based on the status of the simulation. 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, and time step.
 <i>TimeOut</i>	Outputs a true value (1.0) at specified times, and a false (0.0) value at all other times. In the dialog, you specify the time between outputting true values (the delay or timeout); the dialog value is overridden by the D connector. The R connector resets the block back to the beginning of the delay period.

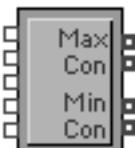
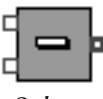
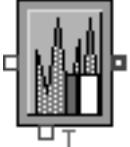
Math - calculating values

Math	Function
 <i>Add</i>	This block adds the three inputs on the left of the block and outputs the total.
 <i>Conversion Function</i>	This block outputs a value that is the input modified by a function. The functions that you can select from the dialog are: absolute value, arccosine, arcsine, arctangent, ceiling, cosine, cosh, exponential, floor, lognormal, log base 10, modulo, multiplier, nearest integer, pulse, sine, sinh, square root, step, tangent, tanh, and user defined.
 <i>Conversion Table</i>	This block acts like a lookup table. It contains a table of values (x in and y out) that are used to calculate what the output value would be for the given input. The table of values defines a curve; the output is calculated based on where the input occurs on the curve. The table may be typed into the dialog, imported from a file, or pasted from the Clipboard using the commands in the Edit menu. The values in the table do not need to be in sorted order. You can use conversion tables as a lookup table for price/unit or customers/server.

A52 | Appendix D: Generic Library Blocks
Table of Generic blocks

Appendix

Math	Function
 <i>Divide</i>	Divides the top input by the bottom input. You can choose whether a bottom input of 0 yields an output of noValue or stops the simulation with an error message.
 <i>Equation</i>	Outputs the results of an equation entered in the dialog. The equation must be of the form "Result = formula;". You can use Extend's built-in operators and functions, and some or all of the input values as part of the equation. Each input must be named in the dialog in order to use it in the equation. You can use the default input value names (Var1, Var2, etc.) or specify new names. Extend will warn you if an input is used in the equation but it is not connected.
 <i>Exponent</i>	Raises the bottom input (x) to the power of the top input (y). You can specify y either through the connector or dialog. The connector overrides the dialog value.
 <i>Financials</i>	Calculates either the rate, period, payment, present value, or future value based on the other four values. You may specify the values in the dialog, or connect four of the input connectors to the known values. If you connect the input connectors, you must run the simulation to calculate the fifth value.
 <i>Integrate</i>	Integrates the input over time. If present, a starting value is added to the outputs. This block may be used in a discrete event model.
 <i>Limits</i>	Allows you to limit the output value to a range of values. The range is specified by the maximum and minimum set in the dialog. The block passes the input through to the output unless the input exceeds the given maximum limit or falls below the given minimum limit, in which case the output is that limit. You can use the block to specify volume ranges, salary draw, temperature tolerances, and so on.
 <i>Logical AND</i>	Performs logical AND operation on the inputs. If each of the two inputs is greater than 0.5, the output is 1; if none or only one of the inputs is greater than 0.5, the output is 0.

Math	Function
 <i>Logical NOT</i>	Performs logical NOT operation on the inputs. If the input is greater than 0.5, the output is 0; otherwise the output is 1.
 <i>Logical OR</i>	Performs logical OR operation on the inputs. If either of the inputs is greater than 0.5, the output is 1; if neither of the inputs is greater than 0.5, the output is 0.
 <i>Max & Min</i>	Determines the maximum and minimum values from among the five values input. The dialog shows the maximum and minimum values and the input connectors they came from. The block outputs the maximum and minimum values and their respective connector numbers.
 <i>Multiply</i>	This block multiplies the inputs.
 <i>Subtract</i>	Subtracts the bottom input from the top input.
 <i>Threshold</i>	Has an output of 0 until the input exceeds the threshold value; when the input exceeds the threshold, the output is the difference between the input and the threshold. The threshold value comes from the dialog unless the T connector is used.
 <i>Time Unit</i>	Converts the value at the input connector from one time unit to another.

Statistics - Calculating Mean, Variance

Statistics	Function
 <i>Mean & Variance</i>	<p>Calculates the mean, variance, and standard deviation of the values input during the simulation. If an input is a noValue, it is ignored and does not affect the statistics.</p> <p>Depending on the choice in the dialog, the variance is computed as: Sum of $(\text{valid inputs} - \text{mean})^2 / (\text{number of inputs})$ or Sum of $(\text{valid inputs} - \text{mean})^2 / (\text{number of inputs} - 1)$</p> <p>The variance is computed using $1/(N-1)$ as an averaging factor.</p>

Appendix E: Discrete Event Library Blocks

*A detailed description of the building blocks
in the Discrete Event library*

*“The causes of events are ever more interesting
than the events themselves.”*

— Cicero

This chapter provides an alphabetical table of the blocks in the Discrete Event library.

Each Discrete Event 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. Extend icons are color coded: green-bordered icons provide items to the model, blue-bordered icons represent computation or operations, and red-bordered icons either output information about items or remove items from the model.

Table of Discrete Event blocks

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 Extend's on-line help for the block's name
- Double-click the block on your model and click on the block's Help button in the lower left of the dialog.

Submenus

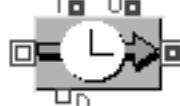
The block tables that follow are categorized by types, and displayed in hierarchical submenus of the Discrete Event Library menu:

- Executive - Needed in every discrete event model to handle events
- Activities - Processing items
- Attributes - Giving items an identity
- Batching - Joining and dividing items
- Generators - Creating items, scheduling, shifts
- Information - Getting information about items
- Queues - Holding, sorting, and ranking items
- Resources -Representing items as resources
- Routing - Moving items to the correct place

Executive - needed in every model

Executive	Function
	This block is the heart of each discrete event model and must be placed to the left of all other blocks in the model. It allows the duration of the simulation to be controlled by the end time or by another number specified in the dialog. Generally you will have no reason to change the default values in the dialog.

Activities - Processing items

Activities	Function
	Holds an item for a specified amount of delay time, then releases it. The delay time is the value in the dialog or, if connected, the value at the D connector when the item is received (the connector overrides the dialog). This block can be used for any kind of service delay. For example, you can use this block to represent red lights in traffic, the time it takes a clerk to wait on a customer, or multitasking CPU time.
	Works the same as the Activity Delay block except it interacts with an item's attributes. You can use the attribute value as the delay and/or modify the attribute value. The delay time is (in order of decreasing priority): D (delay) connector (if it is connected), the value of the chosen attribute (if that option is selected in the dialog), the delay (time units) value specified in the dialog
	Holds many items and passes them out based on the delay and arrival time for each item. The item with the smallest delay and earliest arrival time is passed out first. The delay time for each item is set through the D connector or, if nothing is connected there, can be specified in the dialog. For example, this block can be used to represent a supermarket where customers arrive at different times and take a varying amount of time to shop. Customers who arrive earlier or only shop a little will leave first; customers who arrive later or shop a long time will leave last.

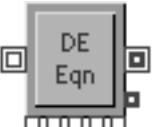
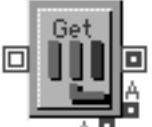
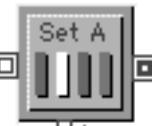
Appendix

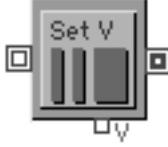
Attributes - Giving items an identity

Attributes	Function
	Changes an item's attribute value then passes the items through. You specify the name of the attribute to change, an operation to use in the change, and a modifier value to change with. The operations are: increment, decrement, multiply, divide The values you can use as modifiers are: a constant, the value from another attribute, the value from the A connector

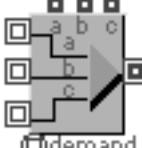
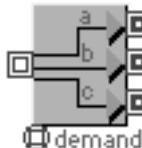
A58 | Appendix E: Discrete Event Library Blocks
Table of Discrete Event blocks

Appendix

Attributes	Function
 <i>DE Equation</i>	Calculates an equation when an item passes through. The inputs to the equation can be from the item's attributes, value, priority, or from one of the five value input connectors. The result of the equation can optionally be assigned to an attribute.
 <i>Get Attribute</i>	Displays and/or removes attributes on items, then passes the items through. The attribute value is shown in the dialog and output at the A connector. You can also use this block to clone the item based on the number in an attribute. As items are passed through the block, the block can either read or remove an attribute, and that attribute can be specified as the first attribute in the list or a named attribute. If the attribute is found, its value is reported in the dialog and sent through the A connector. The D connector outputs 1 if the attribute value has changed since the last value was read in.
 <i>Get Priority</i>	Reads the priority of items then passes them through. The priority is shown in the dialog and output at the P connector. The priority number is usually used to route items. The D connector outputs 1 if the priority value has changed since the last item was read in.
 <i>Get Value</i>	Reads the value of items then passes them through. The value is shown in the dialog and output at the V connector. The D connector outputs 1 if the value has changed since the last item was read in.
 <i>Set Attribute</i>	Sets the attributes of items passing through the block. Up to seven attribute names and values may be assigned to an item with each Set Attribute block. The attributes may add to or replace existing item attributes. You can specify the value of one of the attributes with the A connector. The value at the A connector overrides the corresponding value in the dialog. If the attribute name is already present on the item passing through, the old value will be stripped off, and the new value will be substituted for it.
 <i>Set Attribute (5)</i>	Sets the attributes of items passing through the block. Up to five attribute names and values may be assigned to an item with this block. The attributes may add to or replace existing item attributes. You can specify the value of each of the attributes with the value input connectors; these connectors override values set in the dialog. This works similarly to the Set Attribute block except that you can use the value input connectors to set the value for each of the five attributes rather than for just one. If the attribute name is already present on the item passing through, the old value will be stripped off, and the new value will be substituted for it.

Attributes	Function
 <i>Set Priority</i>	Assigns a priority to items that pass through. The priority value may be set at the P connector or, if no connection is made there, in the dialog. Note that the lowest value (including negative numbers) has the top priority.
 <i>Set Value</i>	Assigns a value to items. This is used to change an item's initial value of 1 to a different value. Items originally have a value of 1 unless the value is changed by the Set Value, Generator, Get Attribute, Program, or Schedule blocks. You would use this to have one item represent a group of items as it travels through the simulation, or to have an item with a value greater than 1 to cause a demand at the Activity Service block. Note: queues split items with a value greater than one into clones. For example, if you set a value of 5 on an item, when it goes into a queue, it will become five distinct items.

Batching - Joining and dividing items

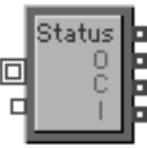
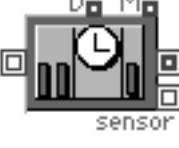
Batching	Function
 <i>Batch</i>	Allows items from several sources to be joined as a single item. This is useful for synchronizing resources and combining various parts of a job ("kitting"). In the dialog, you specify the number of items from each of the inputs that is required to produce one output item (one "kit"). You can also specify that items at one or more inputs will not be brought into the block until one or more of the other inputs has its requirements filled.
 <i>Unbatch</i>	Produces several items from a single input item. The number of items produced at each output are specified in the dialog. By default, this block holds its inputs until its outputs are used or another demand occurs at the connector. The attributes and priorities of the input item are copied to each output. If you selected preserve uniqueness in the Batch block and here, items will be output with their original properties (attributes and priorities) restored. This block can be used to break a message packet into component messages, route the same message to several places, or distribute copies of invoices.

Generators - Creating items, scheduling

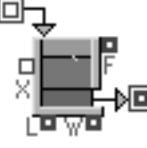
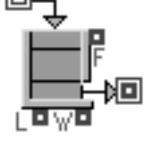
Generators	Function
 <i>Generator</i>	Provides items for a discrete event simulation at specified interarrival times. Choose either a distribution on the left, or choose the empirical distribution and enter probabilities in the table. Items can be created with a random distribution or at a constant rate of arrival. You can also specify the number of items output at each event in the dialog or at the V connector. The parameters for the distribution arrival times are set in the dialog. The random distributions include: beta, binomial, constant, empirical, Erlang, exponential, gamma, hyperexponential, log normal, normal, Pearson type V, Pearson type VI, Poisson, Triangular, uniform integer, uniform real, and Weibull. The empirical distribution may have up to 20 points and may be interpreted as a discrete, stepped, or interpolated distribution. The input connectors 1, 2, and 3 allow you to change the parameters of the random distribution as the simulation progresses.
 <i>Program</i>	Provides items by scheduling many items to be output into the model. This is similar to the Generator block, except the arrival times of the items are scheduled rather than random. Also, you can assign a value, a priority, and attributes to each item generated. These items (with a given output time, item value, priority, attribute name, and attribute value) may repeat on a regular basis. This block is useful for repetitive or timed needs.

Information - Getting information about items

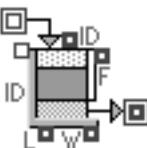
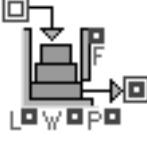
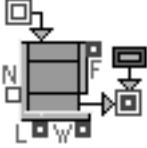
Information	Function
 <i>Count Items</i>	Passes items through and reports the total number of items passed in its dialog and at the # connector. Item values are usually but not always 1. If you have item values other than 1, and do not want to increment the count by item values, and instead want the count to increment by only 1 when an item with a value greater than 1 passes by, choose Count by 1 only. A value greater than 0.5 at the r connector will cause the count to be reset back to zero.
 <i>Information</i>	Views and displays information about the items that pass through it. The first column in the table in the dialog is the time the item arrived in the block, the second is the priority of the item, and the remaining columns are attribute values for the named attributes. You must specify the names of attributes you wish to view. If an attribute is not found, its value is blank.
 <i>Show Times</i>	Displays when the next event will occur for each block in the model. The block displays the requested next event times and their index numbers. To see events as they are posted, leave this block's dialog open when running a simulation. You can specify in the dialog how long the block should pause before updating the display. This block must be placed to the right of all blocks to display events from all blocks.

Information	Function
 <i>Status</i>	<p>Views and displays information about an item or values. For items: Connect the Status block's item input to any block's item output connection to view the items moving through the connection. The Status block's dialog lists a great deal of information about the items at that connector. If this block is left open during a simulation, it will stop the simulation at each event and you can specify a length of time to pause before updating the display.</p>
 <i>Timer</i>	<p>Displays the time that it takes an item to pass between two parts of the model. You can find how long it takes for the item to get from this block to another one (the target block) by attaching the sensor connector to the output of the target block. You can also specify to time only items with a particular attribute. How you connect this block in the model determines the section in which the item will be timed. The Timer is the first block in the timing section; the target block is the last block in the timing section. The output of the target block should also be connected to the sensor of the Timer.</p>

Queues - Holding, sorting, and ranking items

Queues	Function
 <i>Queue, Attributes</i>	<p>A queue where items with a particular attribute have a higher priority than other items. If there are no attributes to prioritize, this becomes a simple FIFO queue. Any items that do not have the attribute at all, will fall to the back of the queue. Up to four different attribute names may be listed in the dialog.</p>
 <i>Queue, FIFO</i>	<p>A first-in-first-out (FIFO) queue. The maximum length, which determines how many items the queue can hold, can be set in the dialog. You can specify that the simulation should stop when the queue is full (reaches the maximum length). You can also see the average queue length, average wait time, and utilization of the queue in the dialog.</p>
 <i>Queue, LIFO</i>	<p>A last-in-first-out (LIFO) queue. The maximum length, which determines how many items the queue can hold, can be set in the dialog. You can specify that the simulation should stop when the queue reaches the maximum length (is full). You can also see the average queue length, average wait time, and utilization of the queue in the dialog.</p>

A62 | Appendix E: Discrete Event Library Blocks
Table of Discrete Event blocks

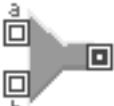
Queues	Function
 <i>Queue, Matching</i>	A queue in which items are released only if they have the specified attribute and the attribute's value matches the value at the ID connector. The block searches an item entering the queue to find the attribute named in the dialog. If the attribute's value matches the value at the ID connector, the item is released. If there is more than one item with that attribute name and value, only the first one that entered the block is released unless the "Release multiple items" option in the dialog is selected. This block is useful for being sure that items in a simulation have a particular characteristic at a particular time.
 <i>Queue, Priority</i>	A queue that releases items by priority. The item in the queue with the lowest numerical value for its priority will be released first. If the items in the queue all have the same priority, it becomes a first-in-first-out (FIFO) queue. The maximum length, which determines how many items the queue can hold, can be set in the dialog. You can specify that the simulation should stop when the queue is full (reaches the maximum length).
 <i>Queue, Resource Pool</i>	A queue for resource pool units. Items wait until the specified number of resource pool units become available. The order of items in the queue is determined by the ranking rule in the dialog of the Resource Pool block. The maximum length, which determines how many items the queue can hold, can be set in the dialog. You can also see the average queue length, average wait time, and utilization of the queue in the dialog.

Resources -Representing items as resources, shifts

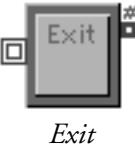
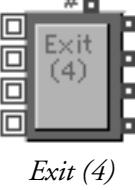
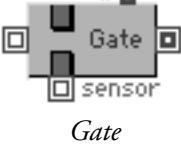
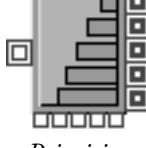
Resources	Function
 <i>Release Resource Pool</i>	Releases a resource pool as the item passes through. This pool of resource units can be released by either: <ul style="list-style-type: none"> - Choosing the "Release Resource Pool by name" radio button and entering the name of the Resource Pool block and the number of units to be released. - Choosing the "Release resource pool by attribute" radio button and specifying an attribute which has been set by a Queue, Resource Pool block.
 <i>Resource</i>	This block holds and provides items (cars, workers, orders, etc.) to be used in a simulation. It can be used as part of an open or closed system. If you use it in place of a Generator or Program block to provide items for the simulation, be sure that there are sufficient items in initial number to satisfy item requirements for the duration of the simulation. This block is similar to a queue. Items can be pulled from the resource through the item output connector as long as they are available. If the block's contents become negative, the block will not output any values until the contents become a positive number. The change connector modifies the number of items stored in the resource by the value of the item at change.

Resources	Function
 <i>Resource Pool</i>	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. Unlike the Resource block, the resource pool units are not items. They are variables which indicate how much of a constraining factor is available. The Resource Pool block works with the Queue, Resource Pool to allocate the pool of resources to items and it works with the Release Resource Pool block to release the pool of resources.
 <i>Shift</i>	Generates a schedule over time which can be used to change the capacity of other blocks in the model. Shifts can be either ON/OFF or represented by a number. If a shift is ON/OFF, blocks which use this shift will be suspended when the shift is off and operate normally when the shift is on. If a shift is numerical, blocks which use this shift will change their capacity based on the current value of the shift. Only blocks with a multiple unit capacity (such as the Activity Multiple and Resource blocks) will utilize the numerical shift. One shift block can control the shift schedule of any number of other blocks in the model. Shifts are repeated at regular intervals if the “Repeat every” checkbox is selected and a time is entered in the adjacent dialog item. This is useful for modeling the many types of shifts that have a regular pattern. An example would be working 8 hours each day of the week or breaks that occur every 4 hours.

Routing - Moving items to the correct place

Routing	Function
 <i>Activity, Service</i>	Passes an item only when the demand connector is connected and certain conditions exist at the demand input (either demand's value is true [greater than 0.5] or it pulls in an item). If an item output connector is attached to the demand connector, the block accumulates the value of the items coming into it and passes that many items through. The block accumulates demand: two items pulled in at the demand connector with values of 4 and 3 will cause this block to accumulate a demand for 7 items at its item input connector. If required items are not immediately available, the block will remember the demand amount. This block serves as a conditional wait. Examples are: multiple food orders, customers waiting in line, flight arrivals.
 <i>Catch</i>	This block “catches” items sent by Throw blocks even though the blocks aren't connected by connection lines. Any number of Throw blocks can send items to a Catch block. The connection between the blocks is made at the Throw block by specifying in its dialog the label and block number of the 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.
 <i>Combine</i>	Combines the items from two different sources into a single stream of items. This is different from the batch blocks which join items from several sources into one item. The items in the Combine block retain their separate identities and are not batched together. Examples of use are: merging traffic, customers coming from two entrances to form a single line.

A64 | Appendix E: Discrete Event Library Blocks
Table of Discrete Event blocks

Routing	Function
 Exit	Passes items out of the simulation. The total number of items absorbed by this block is reported in its dialog and at the # connector.
 Exit (4)	Passes items out of the simulation from many inputs. The total number of items absorbed by this block is reported in its dialog and at the # connector. The number absorbed for each input is reported in the dialog and at the value connectors on the right of the block.
 Gate	Allows only a specified number of items to be in a section of the model (the restricted section). Use this block to restrict the passing of items into a system that can only have a specific number of items in the system at a time. You specify in the dialog the number of items which can be in the restricted section at one time. The first items to arrive are allowed through, up to the number specified. New items are only allowed to pass through the block when one or more of those original items have left the restricted section.
 Prioritizer	Prioritizes the outputs, allowing items to be sent into parallel process activities based on a user-defined priority. Items arrive through the item input connector. They are made available at one of the five item output connectors based on two things: the specified priorities, and whether or not the items are needed at the output. In order of its priority, each output will be checked to see if an item is needed. The item will leave through the highest available prioritized output. If two or more outputs have the same highest priority, the output closest to the top will be checked first to see if the item is needed; if it isn't needed there, the next lower output with that same priority will be checked.
 Select DE Input	Selects one input to be output based on a decision. The item that is present at the selected input is passed through the output. The dialog has options for choosing based on the top priority, changing which input is selected after a given number of items have passed, or choosing based on the select connector. For example, you can use this block to represent traffic signals at an intersection, candidate selection, CPU interrupt access, and so on.

Routing	Function
 <i>Select DE Output</i>	Selects the input item to be output at one of two output connectors based on a decision. The item at the input is passed through the selected output. The dialog has options for changing the outputs after a given number of items have passed and selecting based on the select connector. If the select connector is not used, you can have 1 out of every specified number of items go to the top connector or a random probability for each item to go to the top connector. If the select connector is used, the dialog has options for toggling (choosing the outputs sequentially each time select is activated), choosing the output based on the value at the select connector or specifying the probability of the top connector.
 <i>Throw</i>	This block “throws” items to a Catch block without using an output connector or connection lines. Any number of Throw blocks can send items to a single Catch block. The connection between the blocks is made by specifying the label and block number of the Catch block in the Throw block’s dialog. You could use the Throw and Catch blocks instead of using Combine blocks, even from inside one hierarchical block to inside another one. Choose either a specific Catch block to throw to or a Catch block based on an attribute value.

Appendix F: Blocks in the Manufacturing Library

This chapter provides an alphabetical table of the blocks in the Manufacturing library (MFG).

Each Manufacturing 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. Extend icons are color coded: green-bordered icons provide items to the model, blue-bordered icons represent computation or operations, and red-bordered icons either output information about items or remove items from the model.

Table of Manufacturing blocks

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 Extend's on-line help for the block's name
- Double-click the block on your model and click on the block's Help button in the lower left of the dialog.

Submenus

The block tables that follow are categorized by types, and displayed in hierarchical submenus of the Manufacturing Library menu:

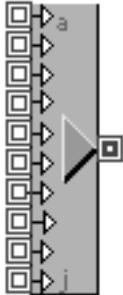
- Activities - Processing items
- Batching - Joining and dividing items
- Generators - Generating items, schedules
- Queues - Holding and ranking items
- Resources - Representing items as resources
- Routing - Moving items to the correct place

Activities - Processing items

Activities	Function
 Conveyor Belt	<p>A linear conveyor for materials handling. You can set the belt capacity, length, and speed in the dialog; the speed connector overrides the value in the dialog.</p> <p>By default, this conveyor is non-accumulating; you can choose in the dialog to have it be accumulating. If it is non-accumulating and the conveyor jams, empty slots will not be filled. If it is accumulating, when the conveyor jams empty slots can be filled.</p>
 Conveyor, Carousel	<p>A non-accumulating rotating conveyor for materials handling. This is similar to the Conveyor Belt except that it is a rotating disk and has a specified number of slots that can pull in items. The speed (circulation time), number of slots between the input and the output, and the capacity are all specified in the dialog. The Δt connector overrides the circulation time set in the dialog. The use connector and the dialog tell how many available spaces on the conveyor are in use at any time.</p>
 Crane	<p>Simulates the movement of items by a crane. Items to be moved are pulled one at a time through the input connector and held for pickup at the output connector when the trip is finished. The path length and speed can be specified in the dialog; the speed connector overrides the value in the dialog. Note that the Crane does not interact with the Route block.</p>
 Machine	<p>Simulates a machine operating on a single item for a specified processing or delay time. When the machine is ready, it pulls an item from a resource or operation (buffer, conveyor, transporter, batch, and so on) and processes it for the time specified. Once an item is processed, it is held until it is picked up by another block. The machine is only ready to process the next incoming item when the processed item is taken by another block.</p>
 Machine (Attribute)	<p>Simulates a machine which can interact with item attributes. This block works the same as the Machine block except you can specify the processing or delay time as an attribute value and modify that attribute value after the work is performed. The processing time is (in order of decreasing priority):</p> <ul style="list-style-type: none"> • D (delay) connector (if it is connected) • the value of the chosen attribute (if this option is selected in the dialog) • the processing time value specified in the dialog
 Process, Preemptive	<p>Holds many items and passes them out based on the process time and arrival time for each item. Items can be prematurely released (preempted) from the block by an item outside the block sending a value greater than 0.5 to the I value input connector.</p> <p>This block could be used to represent a process which can be interrupted to perform higher priority jobs and then resume the lower priority job at a later time.</p>

Activities	Function
 <i>Route</i>	Simulates a route for either AGV and ASR vehicles and the items they transport. The vehicles are pulled into the route through the AGV/ASR input connector, while items to be transported enter through the unlabeled item input connector. The unlabeled item output connector holds items to be picked up by other blocks (such as another Route or a batch block). Multiple vehicles may be allowed on the route at the same time. You can set the maximum number of AGVs on the route at once in the dialog.
 <i>Route (Delay)</i>	Simulates a route for AGV and ASR vehicles when they are not transporting items (such as before pickup or after dropoff). Unlike the Route block, this block is not used for transporting items, only for delaying a vehicle, such as when the vehicle is returning to the source. The input and outputs are vehicles Route Delay blocks can be cascaded by connecting the output of one to the input of the next.
 <i>Station</i>	Simulates a workstation operating on a single item with a specified operation time. It is identical to the Machine block. When the workstation is ready, it pulls an item from a resource or operation (buffer, conveyor, transporter, batch, and so on) and processes it for the time specified. Once an item is processed, it is held until it is picked up by another block. The workstation is only ready to process the next incoming item when the processed item is received by another block.
 <i>Station (Attributes)</i>	Simulates a workstation that can interact with the attributes of the item it is processing. It is identical with the Machine (Attribute) block. This block works the same as the Station block except you can specify the processing or delay time as an attribute value and modify that attribute value after the work is performed. The processing time is (in order of decreasing priority): <ul style="list-style-type: none">• D (delay) connector (if it is connected)• the value of the chosen attribute (if this option is selected in the dialog)• the processing time value specified in the dialog
 <i>Transporter</i>	Simulates the movement of items by a cart along the ground, and the return trip of the cart. You can use this to move parts, items, or work orders through the manufacturing plant. It delays the item being moved and takes an amount of time for the return trip, based on the values entered in the dialog. The Transporter does not begin its return trip until the item it is carrying is picked up. Note that the Transporter does not interact with the Route block.

Batching - Joining items and dividing items

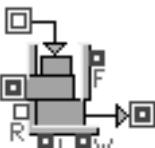
Batching	Function
 Batch (10)	Allows items from up to ten sources to be joined into a single item. More than one item may be required at each input before an output is created. In the dialog, you specify the number of items from each of the inputs that is required to produce each output item. This block is useful when you have many parts that are attached in a process and you want to join them into a single assembly. The block works on demand if the demand connector has input, otherwise it works continuously.
 Batch (Demand)	Accumulates items until the demand connector is activated. It then outputs a single item that is the combination of the accumulated items at that time. This could be used to "kit up" parts until a particular time.
 Batch (Variable)	Allows a variable number of items to be joined into a single item. It then outputs an item that is the combination of the accumulated items. The required quantity of items to be batched into the single item is specified in the dialog or at the n connector. You can specify in the dialog how many input items are needed for each batch; the dialog value is overridden by the n connector.
 Matching	This block batches items based on attributes and releases the batch when it has reached a specified number of items; items that are not released are stored in the block. The block pulls in items and matches them with other items which have the same attribute name(s) and value(s), then releases the matched items as a batch when the quantity of items demanded for the batch is met.
 Unbatch (Variable)	Turns a single item into a stream of identical items (clones or copies). You can specify in the dialog or at the n connector how many copies will be created from each input item. This is useful for unbatching identical items (such as parts coming out of a box) or to break up an invoice into many copies.

Generators - Generating items, schedules

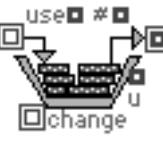
Generators	Function
 <i>Downtime (Unscheduled)</i>	Generates downtime for machines in the Manufacturing Library. Connect the output to the down connector on another block. An item is generated at random intervals according to the Time Between Failure (TBF) distribution. This is a distribution of the durations between the start time of consecutive failures. Each time a failure occurs and an item is generated, a random duration of downtime is selected according to the Time To Repair (TTR) distribution. The duration is assigned as the “value” of the item sent.
 <i>Schedule</i>	Provides items by scheduling up to five repeatable events at specified times. These events (represented by a name, an item and its value) may repeat on a regular basis. You must enter information for at least the first row's Output Time and Value. This block is useful for repetitive or timed needs such as shift changes and known down times. The output from the Schedule block is often directly connected to the change connector on resource blocks to affect the amount of available resources.

Queues - Storing and ranking items

Queues	Function
 <i>Buffer</i>	Simulates a first-in-first-out (FIFO) queue for buffering items needed by machines, conveyors, or batching operations. The maximum length, which determines how many items the buffer can hold, can be set in the dialog. The instantaneous buffer length and wait time can be monitored at the L and W connectors, respectively. The L and W connectors output a noValue when data is not valid.
 <i>Holding</i>	Pulls in items when they are available and accumulates them until the required quantity is satisfied. Then it outputs the items one at a time. When the block is empty, it begins accumulating items again. This is different than the Batch (Variable) block which accumulates items but releases them as one item; the Holding block releases items individually.
 <i>Queue, Decision</i>	This queue releases (ranks) items according to the results of the primary equation defined on the Queue tab. When the queue can release an item downstream, the equation is evaluated for each item in the queue. Depending of the setting of the “Release items with...” popup menu, the item with the highest, lowest, or first TRUE value is released. If multiple items have the same result value, the secondary equation defined on the Tie Break tab is evaluated to determine which item is to be released.

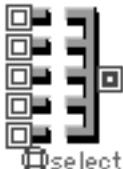
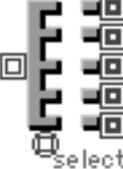
Queues	Function
 <i>Queue, Reneging</i>	<p>Provides a priority queue with reneging (items will leave the queue if their wait time reaches a specified value). Unless reneging occurs, items leave based on priority (the item with the top priority has the lowest priority value). If items in the queue all have the same priority, this becomes a first-in-first-out (FIFO) queue. Items leave through different outputs depending on whether or not they renege: items that do not renege leave through the standard item output connector on the right of the block; items that renege leave through the item output connector on the left of the block.</p> <p>This queue represents the common situation where someone waits in line, but gets discouraged by the wait and leaves after a while.</p>

Resources - Representing items as resources

Resources	Function
 <i>AGV</i>	<p>Provides auto-guided vehicles for use with the Route block. The AGVs are output into the simulation one at a time. Feed the AGVs back to the block when they reach the end of their route.</p> <p>This block is usually used in a closed system where AGVs are recycled back when no longer in use. AGVs can be pulled from the output as long as there are AGVs available.</p>
 <i>ASR</i>	<p>Provides auto-storage and retrieve vehicles for use with the Route block. This vehicle would move horizontally to a point along some shelving, and then move vertically to the desired shelf. The ASRs are output into the simulation one at a time. Feed the ASRs back to the block when they reach the end of their route..</p> <p>This block is usually used in a closed system where ASRs are recycled back when no longer in use. ASRs can be pulled from the output as long as there are ASRs available.</p>
 <i>Bin</i>	<p>Provides a storage location for collecting and retrieving items such as parts and tool bits. Use the change connector to modify the number of items in the bin in cases such as when a parts lot has gone bad.</p> <p>The change connector is usually used to change the number of items available in the bin by the value of the item input at change.</p>
 <i>Fixture</i>	<p>Provides fixtures for holding parts and tools. A Schedule block connected to the change connector allows you to take a specified number of fixtures out of service at regular intervals for repair or to show the result of adding newly-purchased fixtures to your system. This block may be used in an open system such as when fixtures are consumed, or in a closed system such as when you return fixtures to inventory. Fixtures can be pulled from the output as long as there are fixtures available. You can add attributes and priorities to fixtures passing through the block.</p>

Resources	Function
 Labor	<p>Provides labor expressed as people or as work units. You can use this block to generate items that are workers or work units like people/hours. The change connector is useful for common occurrences such as coffee breaks or shift changes.</p> <p>This block is usually used in a closed system such as when a clerk waits on customers then returns to the labor pool to be available for the next customer. Workers can be pulled from the output as long as there are workers available. You can add attributes and priorities to workers passing through the block.</p>
 Pallet	<p>Provides pallets to hold items. The change connector is useful for common occurrences such as when some pallets are broken or a new load of pallets is delivered. It is common for pallets to be joined with the items that would be stacked on them using a batch block, then unbatched when the items are taken off of them. This block may be used in an open system such as when pallets are discarded, or in a closed system such as when you return pallets to stock. Pallets can be pulled from the output as long as there are pallets available. You can add attributes and priorities to the pallets passing through the block.</p>
 Stock	<p>Provides and stores stockroom items such as raw materials, work in process, and so on. This block may be used in an open system such as when items are shipped, or in a closed system such as when you exchange parts in spares inventory. Items can be pulled from the output as long as there are items available. You can add attributes and priorities to items passing through the block.</p>
 Tool	<p>Provides tools that can be used by laborers. You can combine tools with laborers doing the work by using a batch block. This block may be used in an open system such as when tools are disposed of, or in a closed system such as when the laborer returns the tool to the supply room. Tools can be pulled from the output as long as there are tools available. You can add attributes and priorities to tools passing through the block.</p>

Routing - Moving items to the correct place

Routing	Function
 <i>Combine (5)</i>	Combines the items from five different sources into a single stream of items. This is different from the batch blocks which join items from several sources into one item. The items in the Combine block retain their separate identities and are not batched together. Examples of use are: merging traffic, customers coming from two entrances to form a single line.
 <i>Select DE Input(5)</i>	Selects one input connector based on a decision. The item at the selected input is passed through to the output.
 <i>Select DE Output(5)</i>	Selects one output connector out of the five available, based on a decision. The item at the input is passed through the selected output. By default, the block chooses outputs sequentially (toggles outputs); you can also choose based on the select connector.
 <i>Shutdown</i>	Can prevent items from passing through. For example, you would use it to shut down an assembly line. The block passes items through unless the down connector is activated (sees a value greater than 0.5). Whenever the down connector is activated, the item is not pulled into the block. This is conceptually the opposite of the Activity Service block in the Discrete Event library, which allows items to pass only when the demand connector is activated.

Appendix G: Blocks in the Statistics Library

Table of Statistics blocks

This appendix describes each block in the Statistics library, alphabetically.

This 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 Extend's on-line help for the block's name
- Double-click the block on your model and click on the block's Help button in the lower left of the dialog

Statistics	Function
 <i>Activity Stats</i>	Place this block anywhere in the model and it will report the following statistics for all activity-type blocks in the model: Block number (or label, if a label is entered in the block), Block Name, Arrivals, Departures, Utilization, and Time of Observation. The statistics are reported for the following types of blocks: Activity (except Activity, Service), Conveyor, Crane, Machine, Operation (except Operation, Reverse), Process, Station, Transaction, Transporter
 <i>Clear Statistics</i>	Clears the statistics in various blocks in a model. When an item enters this block or the clear time is reached, the statistical accumulators in other blocks in the model are reset. The actual blocks affected depend on the check boxes selected in the dialog of the Clear Statistics block. This block can be used to clear statistics at periodic intervals or in response to a system event. To clear at specific intervals, attach a Generator block to the Clear connector. To clear in response to a system event connect the Clear connector to a connector which will return a 1 when that system event has occurred.

A76 | Appendix G: Blocks in the Statistics Library
Table of Statistics blocks

Statistics	Function
 <i>Cost by Item</i>	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. Note: You must rerun the model when you change table types in the dialog.
 <i>Cost Stats</i>	Place this block anywhere in the model and it will report the following statistics for all costing blocks in the model: Block Number (or label, if a label is entered in the block), Block Name, Cost Per Item, Cost Per Time Unit, and Total Cost.
 <i>Mean and Variance Stats</i>	Place this block anywhere in the model and it will report the following statistics for all Mean & Variance blocks in the model: Block number (or label, if a label is entered in the block), Block Name, Mean, Variance, Standard Deviation, and Time of Observation.
 <i>Queue Stats</i>	Place this block anywhere in the model and it will report the following statistics for all queue-type blocks in the model: Block number (or block label, if a label is entered in the block), Block Name, Average Queue Length, Maximum Queue Length, Average Wait Time, Maximum Wait Time, and Time of Observation. The statistics are reported for the following types of blocks: Buffer, Matching, Queue FIFO, Queue LIFO, Queue Matching, Queue Priority, Queue Reneging, Queue Attributes, Stack
 <i>Random Seed Control</i>	Controls whether the random number seed is reset at the beginning of each simulation run. This only has an effect if the number of runs (set in the "simulation setup" menu item of the run menu) is greater than 1. Extend used a pseudo-random number generator which is based on a seed value. Each time a new random number is generated, a new seed is generated as well. Each Input Random Number and Generator block begins with its own unique seed value based on the block number. If the same seed value is used for different simulation runs, the random values calculated in that block will be identical. This reproducibility be useful when debugging a model or when comparing different systems where a precise comparison is desirable.
 <i>Resource Stats</i>	Place this block anywhere in the model and it will report the following statistics for all resource-type blocks in the model: Block number (or block label, if a label is entered in the block), Block Name, the number of items Available, Utilization, and Time of Observation. The statistics are reported for the following types of blocks: Resource, Bin, Fixture, Labor, Labor Pool, Pallet, Stock, Tool, AGV, ASR, Repository

Appendix H: Blocks in the BPR Library

This chapter provides an alphabetical table of the blocks in the Manufacturing library.

Each Manufacturing 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. Extend icons are color coded: green-bordered icons provide items to the model, blue-bordered icons represent computation or operations, and red-bordered icons either output information about items or remove items from the model.

Table of BPR blocks

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:

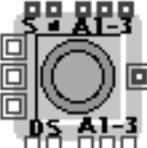
- Look in the index of Extend's on-line help for the block's name
- Double-click the block on your model and click on the block's Help button in the lower left of the dialog.

Submenus

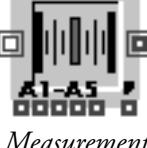
The block tables that follow are categorized by types, and displayed in hierarchical submenus of the BPR Library menu:

- Activities - Processing items
- Attributes - Reading attribute values
- Batching - Joining and dividing items
- Generators - Generating items, schedules
- Queues - Holding and ranking items
- Resources - Representing items as resources
- Routing - Moving items to the correct place

Activities - Processing items

Activities	Function
 <i>Operation</i>	Used to represent an activity, process, action, delay, transformation, and so on, within a model. The Operation block takes N items from up to three input streams, holds them for a specified period of time, and releases only one output item. The Operation block allows you to: <ul style="list-style-type: none">• statically set the input requirements to the block• statically or dynamically set the processing time• statically or dynamically set or modify the attribute values• dynamically interrupt and shutdown the operation
 <i>Operation Variable</i>	Used to represent an activity, process, action, delay, transformation, and so on, within a model. The Operation Variable block takes N items from up to three input streams, holds them for a specified period of time, and releases one or more output items. The Operation Variable block gives you the following capabilities not found in the Operation block: <ul style="list-style-type: none">• dynamically vary the input requirements to the block• statically or dynamically vary the number of items output by the block• automatically sets the attribute values to the input requirements
 <i>Transaction</i>	The Transaction block performs transactions on multiple items at the same time. The block takes in all available items (up to a specified maximum capacity) and performs a transaction (see below). The items exit one at a time. This block can also be used as a simple operation on one item at a time. To do this, set the maximum number of transactions to 1 and specify a transaction time in the dialog.
 <i>Transaction Preemptive</i>	Performs transactions on multiple items at the same time. This works the same as the Transaction block except items can be preempted (interrupted or prematurely released) if the value at the "I" input is true. The block takes in all available items (up to a specified maximum capacity) and performs a transaction (see below). The items exit one at a time from either of the two item outputs: the top output under normal conditions; the bottom output if the item is preempted. For example, this block could be used to represent a group of people performing office tasks. When the phone rings, a worker must interrupt what he is doing and answer the phone. When the phone call ends, the worker will resume the tasks.

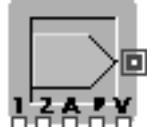
Attributes - Reading attribute values

Attributes	Function
 <i>Measurement</i>	For each item that enters, the block looks for the item's priority and up to five attributes named in the dialog. If it finds the priority and named attributes, it reports their values in the dialog and at the value output connectors at the top of the block. If the item does not have a priority, the block reports a NoValue (blank). If the named attribute is not found, the block reports a NoValue (blank) or other number, as specified in the dialog. Items just pass through this block; they are not delayed or modified in any manner. Use this block to read attributes and priorities and use their values to control other parts of the model.

Batching - Joining and dividing items

Batching	Function
 <i>Operation Reverse</i>	<p>Separates an incoming item into multiple copies and outputs them one at a time. Used to initiate parallel paths, make copies of items, or to separate an item into its original components.</p> <p>You can connect one, two, or all three of the item output connectors located on the right of the block. You can set the number to be output in the dialog or through the Q1, Q2, and Q3 input connectors at the bottom of the block (these input connectors override the dialog choices). You must specify a number greater than 0 for each output connector that is connected.</p>

Generators - Generating items, schedules

Generators	Function
 <i>Import</i>	<p>Represents the arrival of items from outside the scope of a model at a constant or a random rate. Select the distribution for providing inputs from the dialog. The first "(1)" and second "(2)" parameters (arguments) for the specified distribution are on the right side of the dialog; "unused" means that there is no second argument for the chosen distribution.</p> <p>Note that the Import block outputs items based on the time between arrivals of items. For instance, an exponential distribution with a mean of 4 would provide items approximately every 4 time units. You can also specify an attribute value, the priority, and the Value (the size of a batch) for the items output.</p>

Queues - Holding and ranking items

Queues	Function
 <i>Stack</i>	<p>Takes in items and holds them until they are requested by other blocks in the model. The stack holds items on a First-In-First-Out (FIFO), Last-In-First-Out (LIFO), Priority, or Reneging basis, as specified in the dialog. This block also provides the capability of setting the priority for each item that passes through.</p> <p>If the stack receives an item with a Value greater than 1 (for example, as set by the Import block), the stack treats the item as a number of items which will exit the stack one at a time. For example, an item with a Value of 4 becomes 4 items in the stack.</p>

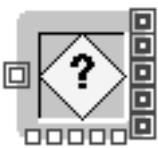
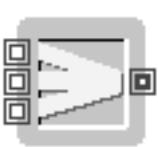
Appendix

Resources - Representing items as resources

Resources	Function
 <i>Labor Pool</i>	<p>Provides a pool of human labor (workers) for the model. An initial number of workers is set in the dialog. The block holds workers in FIFO order until they are required by the model and keeps count of the remaining number of workers available. When workers finish their projects, they are returned to the block to replenish the labor pool. If more workers are required by the model than are available, projects and processes will be required to wait until a worker is returned to the pool and becomes available.</p>

Resources	Function
 <i>Repository</i>	Provides a stock or resource of items for the model. This block is most often used when a pool of resources is required for the model or to represent situations where processes already have a backlog when the model starts (such as a pile of work already on a worker's desk). Repository items typically represent tasks, orders, equipment (such as computer time), and so on.

Routing - Moving items to the correct place

Routing	Function
 <i>Decision (2)</i>	Selects one of two paths based on one or more conditions calculated in an equation. This block is the familiar flow-charting symbol. The item exits at the Y output connector if the equation determines that Path = YesPath; the item exits at the N output connector if the equation determines that Path = NoPath. Model conditions determine which output connector the item takes: the value inputs at the bottom of the block give information regarding model conditions.
 <i>Decision (5)</i>	Selects one of five paths based on one or more conditions as calculated in an equation. This is similar to the Decision (2) block except there are five possible paths rather than just two, and you can name the output paths. Each input item is routed to one output path depending on model conditions and the logic specified in the equation. The value inputs at the bottom of the block give information regarding model conditions.
 <i>Export</i>	Passes items out of the simulation, for instance to other processes out of the scope of model interest. The total number of items exiting from this block is reported in its dialog, at the # connector, and on the icon.
 <i>Merge</i>	Receives items from up to three sources and merges them into a single stream. Note that the items remain individual and unique: they are not joined or batched together, they are just funneled from multiple flows into one flow. Items enter the stream in the order received by the block and flow directly to the output.

Index

Symbols

* E214

_cost attribute M181, M185, B27, B147, B150, B154
_rate attribute M181, M185, B147, B150, B154
 Δ connector M90, M133

Numerics

3D Lookup Blocks S30

A

A output connector M122
ABC M177, B26, B54, B143
About Extend command A34
Accumulate block M207, A48
accumulated cost M11, M18
accumulating data M206, M207
Accuracy S36, S40
ActiveX
 barchart E181
 controls E179
 design mode E182
 dialog item E181
 Object command E182
 worksheet E180
Active-X control S4
activities M4, M66, B60
 processing time B105–B109
 scheduling M135, B109
activity based costing M57
activity based costing (ABC) E272
Activity Costing model B26
Activity Multiple block **M44**, M126, M158, M159, M170, B137
 scheduling M170, B137
Activity Service block M118, M136, B98, B109
Activity Stats A75
Activity Stats block M16, **M61**, **M70**, M202, B53, B63, B165
activity time M66
Activity, Delay (Attributes) block A57
Activity, Delay block E31, E34, E126, A57
Activity, Multiple block A57
Activity, Service block A63
activity-based costing M57, M177, B25–B27, B143

_cost attribute M185, M188, B150, B154
_rate attribute M185, M188, B150, B154
accumulating costs M178, B144
activity costs M182, M189, B148, B155
batching M182, M184, B148
changing attributes M185, B151
combining cost accumulators M190, B156
cost accumulator M178, B144
cost accumulator rates M180, B146
cost array M188, B154
cost per item M179, B146
cost per time unit M180, B146
cost per use M179, B146
cost tabs M177, M179, B143, B145
costs M179, B145
defining costs M179, B145
direct materials cost M180, B146
example B54, B144
fixed cost M179, M188, B146, B154
gathering cost data M187, B152
generator costs M189, B154
item costs M180, B146
item types M178, B144
labor M178, M179, B144, B145
rates M179, B145
release cost resources M183, B149
resource costs M181, M190, B147, B155
resource pools M184, B150
resources M179, M182, B145, B148
statistical analysis B152
storage costs M180, M189, B146, B155
total cost M188, M189, B153, B154
variable cost M180, M188, B146, B154
viewing attributes M185, B151
waiting costs M180, M189, B146, B155
ad hoc experiments E228
Add All To Report command E188, A30
Add All To Trace command E218, A32
Add block E21, E50, A51
Add Connection Line Animation command A30
Add Debug Code to Open Library Windows command A17
Add Named Connection Animation command A30
Add Selected To Report command E188, A30

- Add Selected To Trace command E218, A32
- Adding a Downtime Process S42
- aggregating technique S36
- AGV A72
 - AGV block **M69**, M147
 - alerts E94
 - All layers tool E68, E161
 - AllTables.txt S22
 - Alt key E197, E254, E258, E260, E261, E262, A19, A24
 - analysis M203, B167
 - anchor points E46, E52, E59
 - Animate Attribute block E268
 - Animate Item block E197, E262, E268
 - Animate Value block E197, E262, E268
 - animation E197, M56, M204, B17, B34, B53, B167
 - between blocks E82
 - Building a simple proof animation E280
 - built-in E82
 - custom E83, E197, E267
 - debugging with E83, E217
 - functions E83
 - hierarchical blocks E262
 - in discrete event models E82
 - introduction E81
 - library E83, E197
 - object E263
 - on a block's icon E82
 - on hierarchical blocks E198
 - pictures E82, E199, E267
 - Proof E278
 - showing A29
 - slowing E23, E31
 - stepping E218
 - Animation library E9, E83, E198, E268, B34
 - animation pictures E82
 - custom E267
 - default E267
 - defining and changing E267
 - Append Model command E167, E199, A4
 - Apple menu A2
 - Apple Speech Manager A21
 - AppleEvents E177, E178
 - arguments M81, M195, B73, B160
 - arrival times M80–M87, B72–B77
 - custom intervals M83, B74
 - random M80, B72
 - scheduled M85, B76
 - as is process B9, B23, B38
 - ASCII files E167
 - ASR A72
 - ASR block **M69**, M147
 - Assembly-Rework model M11
 - assumptions
 - changing E24, E32
 - attributes M87–M92, M115, B77–B80, B95
 - _cost M181, M185, B147, B150
 - _rate M181, M185, B147, B150
 - Δ connector M90
 - accumulating data M206–M208
 - accumulating incorrectly M208
 - changing E122
 - changing values M89
 - costing M185, B150
 - cycle time tracking M211
 - error when combining B170
 - errors when using M208
 - examples E128
 - getting M89
 - item types B30
 - limits E115
 - managing M91
 - modifying values M89
 - names E115
 - overview E114
 - preserving M160
 - processing time M130, B107
 - reading values M89
 - removing M164
 - resource blocks M164
 - setting M87
 - shutdown B16
 - stripping M164
 - system M185, B150
 - using M91
 - authoring features E91
 - AutoCAD files A5
 - automated process B23
 - Automatic library search E141, E142, A12
 - Autoscale tools E156
 - Autoscaling E57, E157
 - Autostep E207
 - axis E154
 - changing E57, E153
 - label E153

- scaling E156
- B**
- backlog M11, M12, B57
 - backup files A15
 - bad blocks E145
 - balking B34
 - definition M97, B84
 - rate M18
 - Balloon Help A33
 - Bank Line model E14
 - Batch (10) A70
 - Batch (10) block **M67**
 - Batch (Demand) A70
 - Batch (Demand) block **M67**, M153, M184
 - Batch (Variable) A70
 - Batch (Variable) block **M50**, **M67**, M154, M159, M184
 - Batch block A59
 - preserving uniqueness E272
 - batching M67, M108, M151–M156, M163, B50, B90, B119, B120, B128
 - "Delay kit at..." M155
 - binding M152
 - demand connector M155
 - identical items M153
 - kitting M152
 - preserve uniqueness M159, M160, M184
 - simple M153
 - variable number of items M159
 - Benefits of the SDI Database S4
 - beta distribution M197, B160
 - Bin A72
 - Bin block **M69**
 - binding M152, M156, B120
 - binomial distribution M197, B160
 - bitmap blocks A14
 - bitmap pictures A42
 - bitmap plotters A14
 - Black Connections E75
 - Block Connector Message System S57
 - Block Models S6
 - block number E222, A21
 - block numbers
 - local E221
 - block type E146, A12
 - Blocking S40, S43
 - blocking M27, M50, M112, M114, M122, M126, M146, B20, B104
 - definition M97, B84
 - using buffers to prevent M43, M122
 - using queues to prevent B93, B95, B104
 - blocks
 - adding E24, E32, E42
 - adding (steps) E41
 - bad E145
 - bitmap A14
 - changing E117
 - compiling E147, A17, A24
 - connecting E44
 - copying E199
 - corrupted E145
 - creating A24
 - customizing E197
 - definition E17
 - deleting E24, E32
 - dialog items A26
 - dialogs E19
 - duplicating E58
 - duplicating in libraries E145
 - finding A21
 - help text E197
 - icons E197
 - information A7
 - labels E54, E221, A21
 - modifying A24
 - moving E43, E202
 - moving in libraries E145
 - names E221
 - new A24
 - numbers E221, E222, A21
 - organizing in libraries E140
 - removing E24, E32
 - removing from libraries E145
 - renaming A24, A25
 - searching for E142
 - structure A24
 - substituting E143
 - types E146, A12, A25
 - books B10
 - border E72, A22
 - Border command A22
 - bottleneck M11, M27, M29
 - BPR library E139, B8, B9
 - blocks B4, B59, A77, A77–A80
 - licensing B10
 - Browse option A12

Bucket Problem S35
 Bucket Problem Using Flow S36
 Buffer A71
 buffer B99
 Buffer block **M40**, **M68**, M96, M97, M122, M146
 buffer length M29
 buffers (also see "Buffer block") M119
 Build New Block command A24
 Building a Database from inside of Extend S8
 business process reengineering B5
 changes required B6
 goals B6
 process cycles B7
 steps B7
 buttons (toolbar) E67
 Copy E67
 Cut E67
 new files E67
 Open Notebook E67
 opening files E67
 Paste E67
 printing E67
 Run simulation E67
 saving files E67
 Undo E67

C

CAD files (importing) A5
 calandar S14
 Call Center model M22
 Cancel E19
 capacity M27
 Car Wash model E126
 case sensitivity E60
 Catch block M108, M109, M117, B89, B90, B97, A63
 groups M110
 labels M110
 causal loops B34–B36
 cause and effect B34
 cellular systems M10
 Change Attribute block E122, M87, M89, M91, M207, B78, A57
 change connector B110
 negative values M167, B134, B135
 Change the tab name S23
 changing assumptions E24, E32
 changing libraries E141, A12
 changing text A27
 Chooser desk accessory A2, A6
 Clear command E197, A9
 Clear Statistics A75
 Clear Statistics block **M58**, **M60**, **M71**, B64
 client application E177
 Clipboard E165, E259, A8, A10
 clone B53
 Clone Layer Tool S37
 Clone layer tool E68, E77
 cloned dialog parameters B24, B36
 clones E76, E160, M40, M156, M203, B35, B167
 deleting E78
 finding original dialog item E78
 moving E78
 resizing E78
 unlinked E78
 clones of items B122
 cloning dialog items E76, E261
 Close command A4
 text files E169
 Close Library command E141, A16
 closed systems M161, B126
 color E68, E153
 HSV E69
 Color Connections E75
 color palette E68
 column separators A5
 column width (resizing) E152
 Combine (5) A74
 Combine (5) block **M48**, **M69**, M108
 Combine block M108, M115, M122, A63
 combining resources with cost accumulators M188, B154
 Command key E23, E84, E142, E225, E226, E229, A10
 command reference A2
 comments E20, E54
 communication networks M28
 Compatibility S40
 Compile command E147, A24
 Compile Open Library Windows command E147, A17
 Compile Selected Blocks command E147, A17
 compiling
 blocks E147, A24
 libraries A17

computer networks M28
 conditional routing M118, B24, B33
 confidence interval estimation M202, B165
 confidence intervals M60, M203, B166
 confidence level M202, B165
 connecting blocks E44, E52, A19
 connection establishment delay M29
 Connection Lines command E73, A19
 connections E17
 anchor points (see anchor points)
 arrows E74
 checking E202
 dashed E75
 default A11
 definition E20
 deleting E44, E53
 diagonal E73
 hiding A20
 incomplete E202
 line types E74
 multiple E44
 multi-segment E46, E52
 named M148
 named (see named connections)
 right angle E59, E73, A11
 segmented E52
 selecting E75
 simple parallel M48
 straight E73
 unconnected E46, E75
 connector text object E258
 connectors E17, S49
 compatible E200
 connecting E44
 definition E20
 description E43
 diamond E201
 hierarchical blocks E257
 item E43, E114, E201, E203
 multiple E202
 naming E258
 types E43
 universal E44, E201
 value E43, E203
 conserving resources M155, B123
 Constant block E26, E106, E276, M134, A49
 constant distribution M82, M197, B40, B74, B160
 constant values M194, B42, B158
 constraints E251
 optimization E239, E246
 constraints on resources B32, B126, B131
 contention queues M29
 Contents Calculation S57
 Continue command A25
 continuous S32
 continuous blocks
 in discrete event models E265
 Continuous Improvement (CI) B3
 continuous modeling E98
 defined E98
 delta time E276
 example E105
 timing E209
 control blocks E88, A20
 Control key E23, E84, E142, E225, E226, E229, A10
 control panel M22, B33
 controlling rate S53
 controls (for interactive changes) M202, B165
 Controls command E88, A20
 conversion A39
 factors S48
 flow rate S46
 unit S46
 Conversion Function block A51
 Conversion Table block E110, A51
 Convert to RunTime Library command A16
 Converter block S46
 Conveyor Belt A68
 Conveyor Belt block M44, M66, M131, M147
 Conveyor Carousel block M66, M131, M147
 Conveyor, Carousel A68
 Copy button E67
 Copy command E160, E199, A8
 Copy To Picture command E161
 copying
 data table titles E165
 notebooks E161
 plotter data E160
 text E70
 text to Notebooks E70
 with tools E68
 correlation B24
 corrupted blocks E145
 cost accumulator M178, B144

- cost array M188, B154
 - Cost by Item A76
 - Cost By Item block M58, **M58**, **M70**, M187, B27, B56, B63, B152
 - cost equation E234, E242, E249
 - cost rates M179, B145
 - Cost Stats A76
 - Cost Stats block M58, **M70**, M187, B55, B152
 - cost tabs M177, B55, B143
 - costing E272, M57, B25–B27
 - (see also "activity-based costing") B143
 - example B54
 - fixed cost B154
 - see also activity-based costing
 - variable cost B154
 - costs M179
 - activity-based costing M177
 - defining B145
 - direct materials M180, B146
 - fixed M177, M179, B143, B146
 - fixed cost M188
 - storage M180, B146
 - variable M177, M180, B143, B146
 - variable cost M188
 - waiting M180, B146
 - Count Items block A60
 - Crane A68
 - Crane block **M66**, M131, M147
 - create duplicate items M183, B149
 - Create Publisher command E183, A10
 - Creating a database in Excel S23
 - Creating a DB Text File from Excel S22
 - Credit Application model B38
 - cross-platform compatibility A39
 - CSMA/CD network M30
 - cursor
 - drawing pen E44
 - hand E43, E53
 - main E68
 - tools E68
 - custom simulation order E211
 - customer support E10
 - Cut button E67
 - Cut command A8
 - cycle time E271, B18–B21, B171
 - attributes method M211
 - average B172
 - machine M11
 - service systems M18
 - system (in manufacturing) M11
 - Timer block method M210
 - using attributes B173
 - using the Timer block B172
 - cycling M139, B112
 - fixed number of items M139, B112
 - fixed period of time M140, B114
- D**
- D connector M134
 - daisy-chaining E203
 - data
 - accumulating M206, M207
 - data fitting B160, B162
 - data pane E152
 - data sharing E169
 - Data table title copying E165, A15
 - data tables
 - copying E165
 - exporting data E168
 - importing data E168
 - plotter E152
 - printing E163, A6
 - resizing column widths E152
 - row and column titles E165, E200, A15
 - selecting cells E200
 - database E167
 - database connectivity E174
 - Excel E174
 - importing data E175
 - ODBC E174
 - Database Manager Block S5
 - Database Manager Block, SDI Industry S5
 - Database, Industry S3
 - Database, SDI Industry
 - exporting S21
 - importing S23
 - Database-Aware
 - attributes S15
 - blocks S5
 - Date/Time setup S13
 - days in a month E209
 - days in a week E209
 - days in a year E209
 - DB Specs block S17
 - DB Workbook S23
 - DB Write block S19
 - DDE (dynamic data exchange) E178

DDEAdvise E177
 DDEExecute E177
 DDEPoke E177
 DDERequest E177
 DE Equation block M87, M89, A58
 debugger
 Continue A25
 Generate Debugging Info A25
 open window A24
 Step Into A25
 Step Out A25
 Step Over A25
 debugging E217, E218, M77, M204, B69, B167
 Debugging command A31
 Decision (2) A80
 Decision (2) block B44, B62, B92, B96
 for routing B92
 Decision (5) A80
 Decision (5) block B62, B92
 equalized parallel processing B92
 for choosing a process B96
 for routing B92
 Decision block E269, M118, B98, A47
 for preemption M142, B115
 Default connection lines option E74, A11
 default time unit M73, B65
 deferred Throughput S57
 Define menu A23
 Define New tab command A26
 Delay kit at M155, M158
 delay time M66, M127–M133, B18, B105
 Delete Include File command A28
 Delete Link command E178, A9
 delimiters A5
 delivery time M27
 delta time E209, E210, E276, E277
 DeltaTime variable E207
 demand connector E121, M119, M136, M139,
 M153, M155, B109, B113
 Design Mode command A9
 deterministic models M194, B40, B158
 device drivers E190
 devices E190
 dialog items
 cloning E76
 dialog window
 printing E164
 dialogs
 changing values in E25, E33
 copying E166
 items E76, E261
 notebooks E79, E160
 printing E162, E163
 Dialogs report E187, A30
 diamond connectors E201, E257
 Discrete Event library E9, E100, B37
 and continuous models E200
 attributes E114
 block list E121
 block reference A56
 connectors E114, E119
 description E112
 events E114
 example E125
 items E113
 model layout E113
 moving items E117
 pitfalls to avoid E118
 priorities E116
 servers E127
 values E116
 discrete event modeling
 animation E82, E267
 continuous blocks in E265
 defined E98
 example E125
 hints E269
 overview E113
 preprocessing E269
 restricting items E270
 timing E209
 discrete Event tutorial E28
 distributed computing E175
 Distributed simulation library E176
 distributions M194, M195, M197, B158, B159
 arguments M81, B73
 beta M197, B160
 binomial M197, B160
 constant M82, M197, B40, B74, B160
 custom B18
 empirical M196, M197, B95, B108, B160,
 B161
 Erlang M197, B161
 exponential M81, M83, M197, M201, B73,
 B74, B161, B165
 fitting M196, M199, B162

- gamma M197, B161
 geometric M197, B161
 guide to using M197
 hyperexponential M197, B161
 integer, uniform M198, B162
 list B160
 location M195, B159
 loglogistic M197, B161
 lognormal M198, B161
 negative binomial M198, B161
 normal M198, B161
 pearson type V M198, B161
 pearson type VI M198, B161
 Poisson M198, B162
 real, uniform M198, B47, B162
 shape M81, M195, B159
 skewness M195, B159
 spread M195, B159
 theoretical M196, B160
 triangular M42, M198, B46, B162
 user-defined M196, B160
 Weibull M198, B162
- Divide block E109, A52
 DLLs E95, E191, A42
 Do not show plot option E157
 Documentation Revision model B22
 documents E14
 Don't continue line to Endsim E157
 dotted line E46
 down connector M141, M143, M145, M146
 downtime M67, M83, M143, S42
 Downtime (Unscheduled) A71
 Downtime (Unscheduled) block M67, M83
 drag and drop editing E71, E167
 Draw layer tool E68
 drawing pen cursor E44
 drawing tools E71
 drivers E190
 cross-platform A42
 drop shadows
 hierarchical blocks A11
 dt E56, E209, E210, E276
 Duplicate command E58, A9
 duplicates M157
 duration E209
 DXF files A5
 dynamic data exchange (DDE) E178
 dynamic values M201, B42, B46, B164
- dynamic-link libraries (DLLs) E191, A42
- E**
- Edit menu A8
 Edit Tab command A26
 Editing Tables in Excel S22
 edition file E186
 Electronic engineering libraries E9, E138
 Embedded Database S4
 embedding objects E179
 barchart E181
 design mode E182
 dialog item E181
 Object command E182
 worksheet E180
- empirical distribution M19, M21, M112, M197, B95, B108, B161
- EN29000 B31–B32
 end time M75, B67
 ending time E205, E206
 Enter Selection command A27
 Equation block E86, M134, B20, A41, A52
 equations E222
 Erlang distribution M197, B161
 error bars M203, B166
 error rate M18
 event-posting blocks S34
 events E20, E99, E114, M4, M65, M79, M80, M86, B8, B59, B71, B72, B77
 evolutionary optimizer E232
 Excel E167
 Execution Time S39
 Executive block E30, E113, E126, M39, M91, B40, A57
 executive interface M22, B33
 Exit (4) block E31, M153, A64
 Exit block E126, M42, A64
 Exit command A7
 Expand command A20
 experiments M53
 ExpertFit M196
 explicit shutdown M146
 Exploring model E66
 Exponent block A52
 exponential distribution M81, M83, M197, M201, B73, B74, B161, B165
- Export A80
 Export block B42, B62
 Export Data command E168, A5

- Extend
 document types E14
 opening a model E15
 starting E15
 Extend Suite E139
 Extend+BPR (introduction) B8
 extensions
 converting A42
 external devices E190
- F**
 F connector M146
 Fast Food model M18
 Faster button E23, E31, E217
 Ferrari Agency model M20
 FIFO queue M97, B46, B83
 file conversion A39, A40
 File Input block E133, E168, A49
 File menu A3
 file names E14, A38, A39
 File Output block E168, A49
 FileMaker E167
 files
 closing E15
 copying E199
 exporting E168
 exporting data A5
 importing E168
 importing CAD A5
 importing data A5
 importing DXF A5
 name sizes A38
 opening E15, E16, E28, E41
 opening most recent A7
 saving as backup A15
 transferring A39
 types E14
 Financials block A52
 Find Again command A27
 Find Block command E222, A21
 Find command A27
 text files E169
 Find Next command A21
 first-in-first-out (FIFO) queue M97, B46, B83
 First-Order Assumption and Tentative Event Posting S56
 fixed cost M188, B154
 fixed path M147
 Fixture A72
- Fixture block **M69**
 flexible manufacturing systems (FMS) M10
 Flow S33
 architecture S31
 attributes S48
 Block Optimization Techniques S56
 Blocks - do not leave 'hanging' S52
 blocks- Rules for Using S48
 blocks update their throughput S57
 Breakthroughs and Benefits S39
 connectors S49, S50, S53
 definition S33
 Flow to item conversion S47
 How does it work? S38
 Items and Flow S35
 legend S44
 On Simulate global override S57
 rate changes S49
 rate conversion S46
 Split or Join S50
 starving & blocking behavior S43
 flow order, simulation E211
 Flow Rate Conversion S46
 flow systems M10
 Flow Tutorial 2
 additional Constraints S41
 fonts E169
 fonts for ModL code A13
 footers E164
 format, numbers E154
 formatting text E70
 full-duplex M27
- G**
 gamma distribution M197, B161
 Gate block E270, A64
 gathering information M203, B167
 Generate Debugging Info command A25
 Generate Report command E188, A30
 Generate Trace command E219, A32
 generating items M80–M87, B72–B77
 Generator block E30, E35, E113, E126, M69, B126, A60
 Generic library E8, E100
 block list E101
 block reference A46
 blocks in discrete event models E200
 description E101
 example E40, E105

- hints E275
- generic time unit B65
- geometric distribution M197, B161
- Get Attribute (DB) block S18
- Get Attribute block E129, M87, M133, M157, M208, B78, A58
- Get Info command E260, A7
- Get Priority block M93, A58
- Get Value block A58
- Global Array block E169, A47
- Global Array Manager block E169, A47
- global arrays
 - data sharing E169
- global numbers E221
- global time unit E205, E207, E209, E213, B65
- Go To Line command A28
- goal seeking E231
- graphing B57
- grid
 - icon A21
 - model worksheet A21
 - notebooks A21
- Grid density
 - tool for plotters E155
- H**
- half-duplex M27
- hand cursor E43, E53
- headers E164
- help E10, E19, A33
 - hierarchical blocks E262
 - printing E162
- Help block A49
- Help button E19
- Help menu E10
- Hide All Connections command A20
- hierarchical blocks M18, B16, B22, B24
 - animation E262
 - building E255, A18
 - changing E260
 - cloning dialog items E261
 - connecting E259
 - connectors E257
 - connectors pane E255
 - creating E254, A18
 - definition E18
 - drop shadows A11
 - help E262
 - icon pane E255
- icons E262
- introduction E79
- layout E255
- layout pane E253, E255
- library (converting) A41
- library (new) E260
- library (saving in) E259
- modifying E260
- named connections E61
- pictures, adding E197
- printing E163
- renaming E262, A19, A24, A25
- Save Block As command E260
- saving E259
- structure A19
- structure window E253, E255, E260
- submodels E256
- using E252
- viewing E81
- hierarchical menus A2
- histogram E158
- Histogram block E158
- Histograms M203, B166
- historical information M19, M21
- Holding A71
- holding E117
- Holding block **M68**, M96, M97, M105
- Holding Tank (Indexed) block A48
- Holding Tank block E50, E106, E277, A48
- hot links E177, E178, A9
- hours in a day E209
- HSV E69
- hyperexponential distribution M197, B161
- hysteresis M139, B112
- I**
- icon grid A21
- icons
 - custom E197
 - grid A21
 - hierarchical blocks E262
- idle time B19
- Imagine That, Inc. E10
- Import A79
- Import block B40, B61, B78
 - generating items B72
- Import Data command E168, A5
- Import DXF Files command A5
- include files A28, A40

- Indexing Specification S26
 Indexing Tables S26
 Industry E139
 Industry Suite E139
 Information (DB) block S19
 Information block E220, M154, M205, B169, A60
 Information Connectors S49
 information gathering M203, B167
 initial bias M76, B68
 initial conditions M75, B67
 input connector (passive) S53
 input connector (querying) S53
 Input Data block E21, E26, E48, S29, A50
 modifying a distribution M83, B74
 scheduling delay time B106
 shutting down operations B116
 specifying cost rates M186, B152
 Input Function block E276, A50
 Input Random Number block E21, E46, **M41**, M113, M199, B45, B163, A50
 "stop simulation" choice E49
 "zero output" choice E49
 equalized parallel processing B92
 for random delay time B107
 interpolated output E49
 setting an attribute value B108
 stepped output E49
 used for routing B95
 inputs, multiple E202
 Insert Object command A9
 installation M6, B10
 integer, uniform distribution M198, B162
 Integrate block A52
 integration E277
 interactive controls M202, B165
 interactive simulation E20, E88, E91
 inter-arrival time M96, B40, B46, B72–B77
 Interprocess communication (IPC)
 definition E177
 hot links E177, E178
 Publish and Subscribe E177
 with spreadsheets E265
 interrupting operations M141, M143–M146, B15, B114–B118
 Invoice Approval model B19
 ISO9000 B31–B32
 item behavior different than flow S35
 item connectors E43, E114, E201, E203, E257, S49
 Item Messages block M205
 Item Movement S34
 item Values M79, M82, M85, B71, B73, B76
 items E113, M4, M79, B8, B71, S33
 as cost accumulators M178
 as resources M68, M179, B128
 balking M97, B84
 batching M108, M152, B42, B50, B119, B120
 blocking M97, M114, B20, B84, B93, B104
 clones B122
 cloning M156
 costing M178, B144
 delay time M127
 disposing B42
 duplicates M157, B122
 generating M80–M87, B72–B77
 importing B40, B72
 joining M108, B90, B120–B123
 merging M108, B44, B89
 parallel processing M110
 preempting M141, B114
 processing B42
 processing time M127, B105
 reneging M98, B85
 routing M108, M111, B89, B91
 routing and merging M107–M111
 storing B41
 timing M210, B171
 tracking E274
 transacting B42
 unbatching M111, M156–M160, B52, B121, B122
 Values E35
 Items and Flow (Comparing) S35
- J**
- JIT M146
 - job shops M10
 - jockeying M96, M99, B86
 - joining items B90, B120–B123
 - just-in-time system M146
- K**
- kanban system M146
 - key tool E155
 - keyboard shortcuts A38
 - kitting M152, B120

L

labels E19, E54, E69
 Labor A73
 Labor block **M69**, M157
 Labor Pool A79
 Labor Pool block B50, B126
 Lake Pollution model E40, E226
 landscape tool E156
 LANs M30
 last-in-first-out (LIFO) queue M97, B46, B83
 lead time M11
 left to right order E211
 libraries B37
 block types A12
 changing E141, A12
 closing E15, E140, A16
 compiling E147, A17
 converting A41
 creating A16
 definition E8, E18
 documents E14
 file names A38
 included with Extend E9, E138
 maintaining E145, A15
 new E140, A16
 opening E15, E41, E140, A12, A15
 optimization E234, E242
 other E139
 preload E19, E140, A12
 printing E162
 protecting A16
 Run Time conversion A16
 saving blocks in E147
 search path A12
 searching for E141, E142, A12
 substituting E143
 transferring between platforms A41
 types E146
 uses E18
 version strings A16
 window E145
 Library menu A15
 library window E144, E145, A15
 dates option A12
 opening at startup A12
 printing E162
 licensing M6, B10
 LIFO queue M97, B46, B83

limits A36

Limits block A52
 line balancing M119, B99
 line types E74
 linear processing M45
 List blocks by types option A12
 load capacity M27
 load movement time M27
 local area networks M30
 local block numbers E221
 local time units E213
 location M195, B159
 Lock Model command E91, A20
 log tool E154
 logic B92
 Logical AND block A52
 Logical NOT block A53
 Logical OR block A53
 loglogistic distribution M197, B161
 lognormal distribution M198, B161
 long file names A38

M

M/M/1 M96
 Machine A68
 Machine (Attribute) A68
 Machine (Attributes) block **M66**, M91, M207
 Machine block **M66**, M110
 machine cycle time M11
 Machining Operations model M14
 Macintosh A39
 file conversion A40
 file names E14
 keyboard shortcuts A38
 MacWin Converter A41
 magnify tool E156
 mailslots E175
 Main cursor E68
 mainframes E167
 Make Selection Hierarchical command E197, E253, E254, A18
 Make Your Own block E117
 makespan M11
 managing attributes M91
 manual overview E10
 Manufacturing library E139, M5
 blocks M66, A67–A74
 licensing M6
 manufacturing systems M9–M16

- considerations M10
- performance measures M10
- types M10
- Matching A70
- Matching block **M67**, M87
- matching items M87, M104
- Material - Knowing what it is S48
- Material Additions and Losses S45
- material handling systems M25–M27
 - considerations M26
 - fixed path (routed) blocks M147
 - independent (transporter) blocks M147
 - performance measures M26
 - types M25
- Max & Min block M109, M120, B99, A53
- Maximum Rate override S53
- Mean & Variance block A54
- Mean & Variance Stats block **M70**, B63
- Mean and Variance Stats A76
- Measurement A78
- measurement
 - performance B13–B18
- Measurement block B18, B61, B78, B79
- memory E195
- menu command shortcuts A38
- menus A2
- Merge A80
- Merge block B44, B62, B89
 - alternatives to using B90
 - merging B89
- message delay M29
- message latency M29
- messages to users E92
- meter E90
- metrics B16
- minicomputers E167
- Model menu A18
- model reporting (see reporting)
- Model Speed and Accuracy S36
- model tracing (see tracing)
- models
 - animation E81
 - Appending A4
 - Bank Line E14
 - building E40
 - closing E15, A4
 - continuous E98
 - converting A40
 - copying E68, E166
 - cross-platform A38
 - default time unit E214
 - definition E4
 - discrete event E98
 - documents E14
 - examples E63
 - file names A38
 - grid A21
 - Lake Pollution E40
 - listing A38
 - locking E91, A20
 - Macintosh A38
 - names A38
 - opening E15, A3
 - parts E17
 - Predator/Prey E105
 - printing E163
 - refining E195
 - reverting A4
 - running (see simulations, running)
 - saving E53, A4
 - size A20
 - starting A3
 - steps in creating E194
 - techniques E194
 - time units E213
 - window E17
 - Windows A38
- models/modeling
 - activities M66, B60
 - activity-based costing M177, B143
 - attributes B77
 - balking M97, B84
 - batching M67, M152, B119
 - blocking M97, M126, B84
 - bringing a system on line M139–M140, B112–B114
 - closed and open systems B126
 - confidence interval M202, B165
 - costing M177, B143
 - deterministic M194, B158
 - distributions M195, B159
 - files in this package M5
 - initial conditions M75, B67
 - interactive M202, B165
 - interrupting processes M141, B115
 - items M79, B71

- Monte Carlo simulations M202, B166
multiple runs M77, M202, B69, B166
nonterminating systems M75, B67
number of runs B67
plotting M203, B166
preemption M141, B114
priorities M92, B81
queueing B83
queues M68
randomness M194, B158
reneging M98, B85
resources M68, B62
routes M107, B89
run length M75, B67
sensitivity analysis M202, B166
sources B89
statistical analysis M193, B157
stochastic M194, B158
terminating systems M75, B67
timing items M210, B171
unbatching M67, B119
validation M52, M77, B69
values M79, B71
verification M52, M77, B69
warm-up period M71, M76, B64, B68
modem E189, E190
ModL
font A13
Monte Carlo simulation E36, E47, E158, M53, M194, M202, B158, B166
most recent files A7
Move Selected Items to Tab command A26
movies A30, A42
moving bottleneck S48
multi-dimensional analysis E230
multiple inputs E202
multiple scenarios M202, B166
multiple servers E127
multiple simulations E23, E83
Multiply block E51, E106, A53
multi-tasking B104
- N**
n connector M154
named connections E46, E59, E201, E202, M148, B52, A19
hierarchical blocks E61
negative binomial distribution M198, B161
network systems M27–M34
- considerations M28
models M30, M33
performance measures M28
types M28
New DB Workbook S24
New Dialog Item command A26
new Extend features E6
Manufacturing M6
new features, V5 E6
New Hierarchical Block command E253, E255, A18
New Include File command A28
New Library command E140, A16
New Model command E41, A3
New Table Wizard S8
New Text File command E168, A3
new worksheet S25
non value-adding activities M18
nonterminating M75, B67
normal distribution M198, B161
Normal Size command A20
notebooks E79, E160, M211, B173
as control panels E161
cloning items to E161
copying E161, E166
copying text to E70
creating E79
debugging E220
general E160
grid A21
items E161
opening E161, A18
pages E161
printing E162, E163
numbers
constant M194, B158
display only E34
format E154
random M194, B158
- O**
object
border E72, A22
Object command A10
objective function E232, E234, E242, E249
objects
ActiveX E179
drawing E71
embedding E179

- OLE E179
 ODBC E169, E174, S5
 importing data E175
 OLE
 barchart object E181
 Object command E182
 object design mode E182
 object dialog item E181
 objects E179
 worksheet E180
 on-line help E10, S6
 Only Simulate Messages command A32
 on-time-delivery M11
 on-time-service M18
 Open All Library Windows command A17
 Open Block Structure command E197, A24
 Open command A3
 models E16, E28
 text files E168
 Open Debugger Window command A24
 Open Dialog tool E155
 Open Hierarchical Block Structure command A19
 Open Include File command A28
 Open Library command E140, A15
 Open library window option A12
 Open Notebook button E67
 Open Notebook command E79, E161, A18
 Open Notebook tool E67
 Open Publishers/Subscribers command E186, A10
 Open Sensitized Blocks command E225, A21
 open systems M161, B126
 Operation A78
 Operation block B42, B50, B60, B78, B119
 attributes for processing time B108
 custom processing time B108
 shutting down B117
 used for batching B121
 Operation Reverse A79
 Operation Variable A78
 Operation, Reverse block B52, B61, B91, B121
 unbatching B123
 Operation, Variable block B60, B119
 operations M66
 interrupting M141, B114, B115
 processing time B105–B109
 optimization E85
 adding variables E235, E243
 algorithms E231
 approaching a constant E250
 constraints E239, E246, E251
 cost equation E234, E242, E249
 library E234, E242
 maximum samples E250
 objective function E234, E242, E249
 parameters E249, E250
 profit equation E249
 steps for using E232
 terminate if best and worst E251
 tutorials E231, E233, E241
 variables table E249
 Option key E197, E254, E258, E260, E261, E262, A19, A24
 Order Processing model B25
 oval E71
- P**
- page breaks A5
 page numbers A5
 Page Setup command A6
 Pallet A73
 Pallet block **M69**
 parallel activities M126–M127
 parallel connections M48
 parallel processing E127, E203, M44, M48, M110, M126–M127, B15, B48, B91, B104
 buffering M119, B99
 explicit ordering M114
 line balancing M119, B99
 routing B91–B100
 simple parallel connections M127
 successive ordering M113
 parameter arguments M81, M195, B73, B160
 parameters E24, E32, B164
 parameters, changing dynamically M201
 Parent/Child relationships S26
 partially closed system M162, M169, B127, B135
 passive source M69
 Paste button E67
 Paste command E199, A8
 Paste Link command E178, A9
 pattern E68
 pattern palette E68
 Pause at Beginning command E222, A31
 Pause button E23
 Pause command A31
 Pause Simulation block S37

- pearson type V distribution M198, B161
- pearson type VI distribution M198, B161
- performance measurement B13–B18
- physical hierarchy E254
- pictures E166, A40, A42
 - bitmap A42
 - Windows MetaFiles A42
- planning B28
- PLATFORMMACINTOSH A43
- PLATFORMPOWERPC A43
- PLATFORMWINDOWS A43
- Plot every nth point option E157
- plot pane E151
- plot tools E152
- plotter
 - adding E54
 - autoscale (automatic) E157
 - autoscale (manual) E156
 - axis E154
 - bitmap A14
 - clearing data E160
 - closed during simulation E155, E157
 - color E153
 - copying E165
 - copying data E160
 - data pane E152
 - data storage E157
 - data table E57
 - definition E23, E31
 - description E150
 - dialogs E157
 - duplicating E156
 - Grid density tool E155
 - input to other models E275
 - Key on-off tool E155
 - landscape tool E156
 - lines as reference E275
 - log menu tool E154
 - number format E154
 - open dialog tool E155
 - open during simulation E155, E157
 - panes E151
 - plot pane E151
 - point style E154
 - printing E162, E163, A6
 - push plot tool E156
 - Redraw trace tool E156
 - saving pictures E166
- second plotter E58
- show trace E154
- split bar E151
- tools E152
- Trace properties tool E153
- Traces default to patterns A15
- two Y axes E61
- types E157
- viewing E57
- Y2 axis E61
- Zoom in tool E156
- Zoom out tool E156
- Plotter library E9, E138
- Plotter Traces default to patterns A15
- Plotter, DE Error Bars block E158
- Plotter, DE MultiSim block E158
- Plotter, Discrete Event block E127, E159, **M51**, M204, B20, B57, B167
- Plotter, Error Bars block E159
- Plotter, FFT block E159
- Plotter, I/O block E54, E159, E275
- Plotter, MultiSim block E159, E227, E276, M203, B166
- Plotter, Scatter (4) block E160
- Plotter, Scatter block E159
- Plotter, Strip block E160
- Plotter, Worm block E160
- plotting M204, B57, B167
 - error bars M203, B166
 - histograms M203, B166
 - multiple simulation runs M203, B166
- point style E154
- Poisson distribution M198, B162
- polygon E71
- popup menus
 - in block dialogs E35, E47
- Potential Rates S57
- Predator/Prey model E105
- preemption M141, B114–B118
- preface E1
- Preferences command A10
 - bitmap blocks E217
 - Data table title copying E165, E200
 - Libraries tab A12
 - Miscellaneous tab A14
 - Model tab A11
 - Programming tab A13
 - text font E169, A11

Preload libraries option A12
 preprocessing E269
 preserve uniqueness M159, M160
 Print command E162, A6
 print dialog E163, A6
 Print header/footer option A15
 Print Setup command A6
 printing E162, A6
 headers and footers E164, A15
 hierarchical blocks E163
 Page Setup E164
 Print Setup E164
 priorities B81
 examples E131
 getting M92
 introduction E116
 preserving M160
 Prioritizer block E204
 Queue, Priority block M105
 setting M92
 Prioritizer block E204, M93, M111, M114, B94, A64
 priority queue M97, B46, B81
 probability distributions M195, B159
 Process, Preemptive A68
 Process, Preemptive block M66, M126, M141
 processes B7
 as is B23
 automated B23
 changing B6
 costs M182
 defined B7
 definition M3
 examples M4, B8
 interrupting M141, B115
 linear M45
 parallel M44, M126–M127, B15, B43, B48, B51, B91, B104
 serial M45, M126, B15, B103
 to be B23
 processing by type M122
 processing time M11, M41, M66, B18, B105–B109
 attributes M46
 cumulative M131
 custom M130, B107
 fixed M127, B105
 implied M131
 random M129, B107
 scheduled M128, B106
 setting M127–M133
 setup time M133
 time sharing M131
 time units M74, B66
 processor delay time M29
 productivity B18
 Profile Block Code command A33
 profiling A33
 profit equation E249
 Program block E113, E269, M39, M69, M85, M143, M154, B76, B126, A60
 scheduling arrivals M85, B76
 programming
 profiling A33
 protecting block code A16
 Prompt block E94, A50
 prompts E94
 Proof Animation E278
 Proof animation
 Adding paths E282
 Building a simple proof animation E280
 Protect Library command A16
 protocols M30
 Publisher Options command E185, A10
 Publishers update at end of run option E184, A15
 publishing A10
 publisher E183
 subscriber E183
 updating publisher A15
 pulling E118
 pure hierarchy E254
 push plot tool E156
 pushing E117

Q

queue
 attributes M97
 choosing M97, B84
 concepts M95–M105
 FIFO M97, B46, B84
 LIFO M97, B46, B84
 Manufacturing library M68
 matching M97, M104
 priority M97, M105, B46, B84, B87
 Queue, Resource Pool block M165
 reneging B46, B84
 queue length M18

queue size M27
 Queue Stats A76
 Queue Stats block M16, **M61**, **M70**, B63
 Queue, Attribute block M87, M95, B78
 Queue, Attributes block A61
 Queue, Decision A71
 Queue, Decision block M87, M96
 Queue, FIFO block E30, E36, E126, M95, A61
 Queue, LIFO block M95, A61
 Queue, Matching block M87, M95, B78, A62
 Queue, Priority block E131, M93, M95, A62
 Queue, Reneging A72
 Queue, Reneging block M21, **M68**, M96, M97
 Queue, Resource Pool block M96, M97, M164, B131, A62
 queueing disciplines M95, M97, B83
 queues E14
 jockeying M96, M99, B86
 QuickBlocks library M6, M83
 QuickTime A30
 Quit command A7

R

Random Details and Plot S12
 random distributions M194, M195, B158, B159
 random elements E36, E84
 random number generator E208, E214, E228, M194, B159
 recommended E208
 random number stream M194, B159
 random numbers E36, E214, M194, B24, B45, B57, B159
 random rates S55
 Random sampling parameters S12
 random seed E208, E214, E215, E228, M194, M202, B45, B58, B159, B166
 Random Seed Control A76
 Random Seed Control block **M71**, M202, B64, B166
 random shutdown M145
 random values M42, M194, B42, B158
 randoms (from inside of Extend) S11
 rate
 based on status S56
 change S54
 information reporting S53
 of flow S38
 Read Database S23
 ReadOut block E220, M207, B24, A50
 real, uniform distribution M198, B47, B162
 Record Item Messages block E220
 Record Message block E220, M205
 rectangle E71
 Redraw trace tool E156
 Reduce command A20
 Reduce to Fit command A20
 redditio-ad-absurdum M77, B69
 reference line E152, E275
 references M7, B10
 Refresh Links command E178, A9
 Regional Call Processing model B33
 registration number E10
 release cost resources M183, B149
 Release Resource Pool block M164, B131, A62
 Remove All From Report command A30
 Remove All From Trace command E219, A33
 Remove Debug Code in Open Library Windows command A17
 Remove Selected From Report command E188, A30
 Remove Selected From Trace command E219, A33
 Rename Block command E145, E262, A24
 Rename Hierarchical Blocks command A19
 renaming
 blocks A24
 hierarchical blocks E262
 reneging B34
 definition M98, B85
 rate M18
 reneging queue M97, M98, B46
 Repeat the program every... M40
 repeatable results M195, B159
 Replace All command A27
 Replace command A27
 Replace, Find Again command A27
 Report command M204, B168
 Report Type command E188, A30
 reporting M203, B57, B167
 commands E187
 Dialogs report E187
 general E187
 notebook E160
 report types A30
 Statistics report E187
 steps E188
 Repository A80
 Repository block B41, B126
 requirements M6, B10

residual error rate M29
 resizing E17
 Resource block A62
 resource constraints B32–B34
 Resource Pool block M164, M174, B131, B141, A63
 Resource Stats A76
 Resource Stats block **M70**, B63
 resources M4, M68, M179, M182, B32, B62, B126, B145, B148
 allocating B129–B131
 attributes assigned to B128
 attributes for tracking information M164
 batching method M163, B128
 conserving M155, B123
 constraining flow of items M164
 constraints B126, B131
 costing M181
 definition B125
 explicit B125
 from different resource pools M166
 implicit B125
 limited M163, M164, B126
 modeling M162, B127
 number available M163, M164
 reducing M167, B134
 resource pool method M164, B131
 reusing B126
 scheduling B134
 scheduling using Change connector M167
 scheduling using Shift block M169, M174, B136, B141
 unlimited B126
 used in multiple places M166
 response time M27
 restraints on resources M164
 restructure the Indexed Fields S26
 Restructure ALL Tables S22
 Resume button E23, E25, E33
 Resume command A31
 Retain block A49
 Revert Model command A4
 Right angle connection lines default option A11
 rightsizing B37
 Roulette model B35
 rounded rectangle E71
 Route A69
 Route (Delay) A69
 Route (Delay) block **M66**, M147
 Route block **M66**, M131, M147
 routing M107–M123, B89
 based on attributes M115, B95
 conditional M118, B24, B33, B91, B98
 explicit B94
 extended M120, B100
 parallel B24, B91–B100
 rules B24, B91
 sequential B24, B91
 simple M111, B92
 using Throw and Catch blocks M109, M117, B90, B97
 Routing Tables S25
 rules B24
 run length M75, B67
 Run menu A28
 Run Optimization command A29
 Run simulation button E67
 Run Simulation command A29
 Run Time version E91
 Run Time commands A16
 RunTime Startup Screen Editor command A17
S
 S connector M143
 Save and Save As commands A4
 Save backup files option A15
 Save Block As command E255, E260
 Save Selected command E167, E199, A4
 Save Text File As command E169
 scaling E274, A20
 axes E156, E157
 scaling models B30
 scattergram E159
 scenario analysis M53
 Schedule A71
 Schedule block **M67**, M69, **M69**, M143, M168
 scheduled shutdown M143
 scheduling
 activities M135
 fixed arrival times M15
 labor M167, M169, B136
 repeating M40
 resources M167, M169, B136
 scheduling activities B109
 scheduling algorithms M95, B83
 scheduling resources B134
 scrap M11, M113

Scrapbook E165
 screen size A20
 Scroll To Msgs command A32
 scrolling E17
 SDI Database (exporting) S21
 Search path option A12
 searching A27
 seed E208, E215, E228, M194, M202, B45, B58, B159, B166
 Select All command A9
 select connector E273, M109, M111, M113, M116, M120, M122, M132
 Select DE Input (5) block **M69**, M93, M108
 Select DE Input block M93, M108, A64
 Select DE Input(5) A74
 Select DE Output (5) block **M69**, M111, M112, M113, M122
 Select DE Output block E129, M111, M113, M115, M132, A65
 Item enters Select block before... M116
 Select DE Output(5) A74
 Select Input (5) block A47
 Select Input block A47
 Select Output (5) block A48
 Select Output block A48
 sensitivity analysis M53, M203, M204, B166, B168
 command key E225
 control key E225
 disabling E228
 enabling E228
 example E226
 introduction E83
 multiple dimensions E230
 number of runs E228
 opening sensitivity blocks A21
 Sensitivity Setup dialog E84, E225
 Sensitize Parameter command E225, A10
 settings E228
 steps E225
 Sensitize Parameter command A10
 sensor connector on Timer block B172
 serial number E10
 serial ports E189
 serial processing M45, M126, B15, B103
 server application E177
 servers E14, E127
 service rate M18
 service systems M16–M22
 considerations M17
 performance measures M17
 types M17
 service time M18
 Set Attribute (5) block M87, B78, A58
 Set Attribute block E128, M87, B78, A58
 Set Block Type command A25
 Set Library Version command A16
 Set Priority block E131, M93, A59
 for preemption M143
 Set Simulation Order command A21
 Set Value block A59
 setup time M14, M133
 shape M195, B159
 Shift block M137, M143, M169, B111, B116, B136, A63
 Shift key E145, E254
 Shift Selection Left command A28
 Shift Selection Right command A28
 Show Animation command E82, A29
 Show Block Labels command A21
 Show Block Messages command A32
 Show Block Numbers command E164, A21
 Show Clipboard command E165, A10
 Show Debug Messages command A33
 Show instantaneous queue length checkbox E157
 Show Links command E178, A9
 Show Movies command A30
 Show Named Connections command E61, E201, E202, A19
 Show Page Breaks command E164, A5
 Show plot at end of simulation option E157
 Show plot button E157
 Show plot during simulation option E157
 Show Reporting Blocks command E188, A30
 Show Simulation Order command E204, E221, A21
 Show Times block E220, A60
 Show Tracing Blocks command E218, A33
 shuffle graphics tool E72
 Shutdown A74
 shutdown M143–M146, B16, B114–B118
 explicit M146
 random M145, B117
 scheduled M143, B116
 Shutdown block **M69**
 simplex M27

Simulate A32
 simulation order E207, E211, A21
 Simulation Setup command E55, E204, A29
 simulations
 beep A11
 definition E4
 multiple E83, M76, M202, B68, B166
 number of E205, E206
 order E211, E221
 pausing E23, A31
 resuming E23
 running E14, E22, E55, A29, A41
 setup E55, A29
 slowing E217
 speeding up E215
 status E22, E274
 stopping E23, A31
 time E56, E209
 time units E213
 skewness M195, B159
 slider E88, E107
 Slider control M127, M202, B105, B165
 slot time M30
 Slower button E23, E31, E217
 sorting B30
 Sound block E93, E220, A50
 Sound plays at end of run option E22, A11
 sounds A42
 speech synthesis module A21
 speed M147
 split bar E151
 Spoken Messages command A21
 spread M195, B159
 spreadsheets E5, E167
 Stack A79
 Stack block B41, B46, B61, B84, B85, B87
 standard deviation E37, E47
 standard line E275
 start connector E269, M86, M139, B77, B113
 start time
 absolute vs. relative M86, B77
 Starting Contents S51
 Starting Store Block S51
 starting time E205, E206
 StatFit M196, M199, B162
 static values M201, B42, B164
 Station A69
 Station (Attributes) A69
 Station (Attributes) block **M47, M66, M91, M131, M207**
 Station block **M41, M66**
 stationery E200
 statistical analysis M70, M193, M203, B63, B157, B167, A75
 confidence interval M202, B165
 statistical bias M71, M205, B64, B168
 statistical data fitting B160, B162
 statistics M70, M193, B53, B63, B157, A75
 Statistics library E220, M5, M204, B9, B168
 blocks **M70, B4, B59, B63, A75–A76**
 licensing M6, B10
 Statistics report E187, A30
 status bar E22, E78, E274
 Status block E134, E220, E274, M204, B167, A61
 steady-state systems M75, B67
 Step command A31
 Step Each Block command E222, A32
 Step Entire Model command E222, A32
 Step Into command A25
 Step Next Animation command E218, A32
 Step Out command A25
 Step Over command A25
 Steps S34
 steps E20, E210, E276
 StepSize E276
 stepsize E210, E276
 stepwise refinement E4
 stochastic models M194, B41, B158
 Stock A73
 Stock block **M69, M69**
 Stop block E92, E220, A51
 Stop button E23
 Stop command A31
 Stop Message block E266
 storage costs M180, M189, B146
 Store block S51
 strategic planning B27–B31
 structure window A13
 hierarchical blocks E260
 printing E162, E164
 Structure window opens in front option A13
 submodels E79, E256
 Subscribe To command E183, E184, A10
 subscriber E183, A10
 Subscriber Options command E186, A10

substituting blocks E143
 substituting libraries E143
 Subtract block A53
 Suite E139
 Industry E139
 Support Planning model B29
 Support Services model B14
 Switch control E90
 system attributes M185, B150
 system cycle time M11
 System Variable block M21, A51
 systems (defined) M3

T

tab delimited files E169, A5
 tables
 adjusting column widths E200
 in Input Data block E48
 tabs E27, E30
 deleting A26
 moving dialog items to A26
 new A26
 renaming A26
 take last B121, B123
 task time B18, B105
 technical support E10
 technology insertion B21–B24
 template E167
 templates M5
 terminating systems M75, B67
 Testing Process model B31
 text
 adding E69
 as a connector in hierarchical blocks E258
 as a named connection E60, E201
 box E69
 copying E70
 copying to Notebooks E70
 copying with drag and drop E71
 deleting E70
 drag and drop E71, E167
 duplicating E60
 formatting E70, A22
 labels E54, E60, E201
 moving E70
 moving with drag and drop E71
 resizing E69
 tool E69
 transparent A22
 text clippings E71, E167
 text file font option A11
 text files
 closing E169, A4
 creating E168, A3
 exporting E167, A5
 File Input block E133
 Find command E169
 font A11
 importing E167, A5
 introduction E15, E168
 opening E15, E168, A3
 printing E162
 reverting A4
 saving E169, A4
 text labels E54, E60, E201
 Text menu E70, E262, A22
 Text Output S22
 Threshold block A53
 throughput M11, M27, M29
 Throw block M108, M109, M111, M117, B89, B90, B97, A65
 popup list of Catch blocks M110
 time M127–M133, S32
 time between arrivals M80, B40, B46, B72
 time between failures M143, M145
 time per step E206
 time sharing M131
 time steps S32
 time to repair M143, M145
 Time Unit block A53
 time units E208, E213, M38, M73, M84, B65
 default E214
 default time unit M73, B39, B65
 generic E213, B65
 global E213
 global time unit B39, B65
 local E213
 processing time M74
 TimeOut block A51
 Timer block E220, E271, M210, B20, B172, A61
 timing E208
 to be process B23, B38
 Tool A73
 Tool block **M69**
 Tool Tips
 on block dialogs A13
 on dialog editor A13

- on worksheets E196, A11
- Tool Tips on block dialogs option A13
- Tool Tips on dialog editor option A13
- Tool Tips on worksheet option A11
- toolbar E67
 - buttons E67
 - model window E67
 - plotter E152
 - structure window E257
- tools
 - All layers E68
 - autoscale plotter manually E156
 - Clone layer E68, E77
 - Color E68
 - color E68
 - copying with E68
 - Draw layer E68
 - drawing E71
 - editing E67
 - files (opening and closing) E67
 - files (saving and printing) E67
 - Grid density (plotter) E155
 - introduction E67
 - Key on-off(plotter) E155
 - log (plotter) E154
 - Main cursor E68
 - open dialog (plotter) E155
 - Open Notebook E67
 - Pattern E68
 - plotter E152
 - push plot (plotter) E156
 - Redraw trace (plotter) E156
 - Run Simulation E67
 - selecting E68
 - Shuffle graphics E72
 - Text E69
 - Trace properties (plotter) E153
 - Zoom in (plotter) E156
 - Zoom out (plotter) E156
- Tools command A16
- total cost M188, M189, B153, B154
- Total Quality Management (TQM) B3
- trace
 - color E153
 - name E153
 - pattern E153
 - width E153
- tracing
- commands E218
- general E218
- steps E219
- tracking information M92
- tracking items E274
- Transaction A78
- Transaction block B51, B60
 - parallel processing B104
 - setting the processing time B106
- Transaction Preemptive A78
- Transaction, Preemptive block B60, B114, B115
- transit delay M29
- transmission rate M29
- Transparent command A22
- transportation blocks M147
- Transporter A69
- Transporter block **M66**, M131, M147
- triangular distribution M42, M198, B46, B162
- Tutorial B37–B58
- tutorials E16
 - discrete event E28
- type A46, A56, A67, A77
- types of blocks A12, A25
- U**
- Unbatch (Variable) A70
- Unbatch (Variable) block **M51**, **M67**, M157, M159, M184
- Unbatch block M157, M158, A59
 - preserving uniqueness E272
- unbatching M67, M111, M152, M156–M160, B52, B121, B122
 - create duplicate items M183, B149
 - preserve uniqueness M159, M160, M184
 - release cost resources M183, B149
 - variable number of items M159
- Undo button E67
- Undo command A8
- uniform (integer) distribution M198, B162
- uniform (real) distribution M198, B162
- Unit Conversion S46
- units of time E205, E207, E209, E213
- universal input connectors E44, E120, E201, E257
- Unsupported XCMD callbacks A13
- Update Flow Blocks on Simulate S57
- updating S57
- upper limits A36
- Use Grid command A21
- Use Sensitivity Analysis command E84, A29

- user messages E92
- user prompts E94
- Utilities library E266, M205
- utilization M11, M18, M27, B16, B19, B57
 - networks M29
- V**
 - V5 new features E6
 - validation M52, M77, B69
 - value connector messages E265
 - value connectors E43, E203, E257, S49
 - Value input connector S53
 - values
 - constant M194, B158
 - dynamic M201, B164
 - informational M79, B71
 - random M194, B158
 - static M201, B164
 - values (informational) E114
 - connectors E43
 - Values, item M40, M79, M82, M85, B71, B73, B76
 - examples E132
 - introduction E116
 - variable cost rate M188, B154
 - variables table E249
 - verification M52, M77, M203, M204, B69, B167
 - viewing E118
- W**
 - Wait Time block A49
 - waiting costs M180, M189
 - waiting time M18
 - WANs M30
 - warm-up bias M76, B68
 - warm-up period M71, M76, M205, B64, B68, B168
 - waste M11
 - Weibull distribution M198, B162
 - weight M206
 - wide area networks M30
 - Window menu A33
 - Windows A39
 - file conversion A40
 - file names E14
 - keyboard shortcuts A38
 - windows
 - model E17
 - Windows MetaFiles A42
 - WIP M11